WenQuanYi Micro Hei [Scale=0.9]WenQuanYi Micro Hei Mono song-WenQuanYi Micro Hei sfWenQuanYi Micro Hei "zh" = 0pt plus 1pt

nebulas Documentation

Release 1.0

nebulas

Jan 10, 2021

Category:

Nebulas is a next-generation public blockchain, aiming for a continuously improving ecosystem. Based on its blockchain valuation mechanism, Nebulas proposes future-oriented incentive and consensus systems, and the ability to self-evolve without forking.

Nebulas community is open and everyone can be a contributor and build a decentralized world with us.

The Nebulas wiki is a collaboration tool for the community to publish various documents in a collaborative manner.

CHAPTER 1

Use Wiki

1.1 Learn

1.2 Develop

1.3 Use Nebulas

1.4 Get Involved

1.4.1 WhatâĂŹs Nebulas

The Future of Collaboration

Nebulas is an open-source, public blockchain focused on creating a true Autonomous Metanet. NebulasâĂŹ focus utilizing on-chain data for users interactions and collaboration. Our core principal is **Let everyone get values from decentralized collaboration fairly through technical ways such as blockchain.**

Nebulas uses its innovative technology to realize its vision of creating a collaboration model with the help of unique innovative technologies to manage on-chain public assets and to realize the Decentralized Autonomous Organization (DAO) which will provide positive incentives and self-evolution.

There are four technical features:

- Quantifiable: measure the value of Blockchain data
- Self-evolving: low-cost instant upgrade capability

- Incentive: positive ecosystem incentives
- On-chain Governance: improved decentraalized autonomous organization (DAO)

Autonomous Metanet

We focus on on-chain data and interactions. Raw Data is such as users and smart contracts. Metadata is information that provides information about other data such as balance and address. Hypermapping refers to the raw data, then abstracts a layer of metadata to beetter describe itself. And Hyper-mapped Structural Metadata can handle increasingly complex on-chain data and describe these interactions. visit the Nebulas Technology Page on the official website to learn more about metadata.

For example, Nebulas Rank (NR) is a hyper-mapped structural metadata. It can measure the value of Blockchain data. Read the Yellow Paper - Nebulas Rank to learn more about the Nebulas Rank. Or visit NR page to learn more:

Nebulas Rank (NR)

Nebulas Rank (NR) is an open source ranking algorithm used to measure the influence of relationships among addresses, smart contracts, and distributed applications (DApps). It helps users utilize information within the ever-increasing amount of data on all blockchains, but it also helps developers to use our search framework directly in their own applications.

On Nebulas, we measure value regarding:

- **Liquidity**

Finance is essentially the social activities which optimize social resources via capital liquidity and in turn promotes economic development. Blockchains establish a value network in which the financial assets can flow. Daily volume of Bitcoin and Ethereum, which are most familiar to us, already exceeds $1 billion. From this data, we can see that the higher the transaction volume and transaction scale, the higher the liquidity. As a consequence of this, higher liquidity will increase the quantity of transactions and enhance the value. That will further strengthen the value of the financial assets, creating a complete positive feedback mechanism. Therefore liquidity, i.e. transaction frequency and scale, is the first dimension that NR measures.

- **Propagation**

Social platforms like WeChat and Facebook have almost 3 billion active users per month. Social platformsâĂŹ rapid user growth is a result of the reflection of existing social networks and stronger viral growth. In particular, viral transmission, i.e. speed, scope, depth of information transmission and linkage, is the key index to monitor the quality of social networks and user growth. In the blockchain world, we can see the same pattern. Powerful viral propagation indicates scope and depth of asset liquidity, which can promote its asset quality and asset scale. Thus, viral transmission, i.e. scope and depth of asset liquidity, is the second dimension that NR measures.

- **Interoperability**

During the early stages of the internet, there were only basic websites and private information. Now, information on different platforms can be forwarded on the network, and isolated data silos are gradually being broken. This trend is the process of identifying higher dimensional information. From our point of view, the world of blockchains shall follow a similar pattern, but its speed will be higher. The information on usersâĂŹ assets, smart contracts, and DApps will become richer, and the interaction of higher dimensional information shall be more frequent, thus better interoperability shall become more and more important. Therefore, the third dimension measured by the NR is interoperability.

Based on the aforementioned dimensions, we started constructing NebulasâĂŹ NR system by drawing from richer data, building a better model, digging up more diversified value dimensions, and establishing a measure of value in the blockchain world.

And a network includes hyper-mapped structural metada is the metanet.

New Consensus Incentives

Nebulas Incentives are the cornerstone of autonomy, which provide lasting positive incentives. Motivate developers through the **Developer Incentive Protocol (DIP)**, motivate communities through **Proof of Devotion (PoD)** algorithm. Read the Mauve Paper - DIP to learn more about DIP. And visit the node strategy page to learn more about Nebulas PoD Node Decentralization Strategy - Based on the Proof of Devotion (PoD) Mechanism.

New Upgrade Capabilities

Upgrade without hard forks, self-evolution is the future of autonomy. **Nebulas Force (NF)** provides the ability to upgrade without hard forks.

A series of basic protocols such as the NR, the PoD, and the DIP shall become a part of the blockchain data. With the growth of data on Nebulas, these basic protocols will be upgraded, which will avoid fractures between developers and community, as well as a âĂIJforkâĂİ. We call this fundamental capability of our blockchain âĂIJNebulas ForceâĂİ (NF).

As the Nebulas community grows, NF and basic protocolsâĂŹ update ability shall be open to the community. According to usersâĂŹ NR weight and the community voting mechanism, NebulasâĂŹ evolution direction and its update objectives will be determined by the community. With the help of NFâĂŹs core technology and its openness, Nebulas will have an ever-growing evolutive potential and infinite evolving possibilities.

Decentralized Collaboration with Smart Assets

- Redefining the token economy: Nebulas founder Hitters Xu launches the new Smart asset platform nextDAO (2019).
- Decentralization is the Essence of Blockchain (by Hitters Xu, 2018)

Visit nextDAO to learn more.

Learning Resources

Nebulas Vision: Let everyone get values from decentralized collaboration fairly. View the Nebulas Manifesto, which was written on the first block.

If you want to know more about Nebulas, please subscribe to the official blog, or visit our website: nebulas.io to follow basic news. Here are some useful categories:

- Weekly & Monthly Report
- Announcements
- AMA

Interviews

Interviews with Nebulas Team:

- Interview with the Founder of Nebulas Hitters Xu - Seeing Through The Blockchain Bubble
- The Nebulas That IâĂŹm Looking Forward to [Youtube]
- Why Join Nebulas by Ph.D Samuel Chen [Youtube]
- Nebulers' Thoughts on the Future of Blockchain [Youtube]
- One day in Nebulas [Youtube]
- The Inspiration Behind the Nebulas NOVA Design by Mengggo Liu
- Take the Lead to Set Up Nebulas Research Institute by Xuepeng Fan
- Let Nebulas Fly Higher and Farther! by Congming Chen
- My Heart Belongs to Nebulas, I Hope We Shine Together by Zaiyang Tang
- Exclusive Interview to Nebulas Technical Director Dr. Joel
- My First Job at Nebulas by Dr. Yulong Zeng
- Life Is A Challenge by Dr. Dai

Interviews with Members of the Community:

- Nebulas Incentive ProgramâĂŁâĂŤâĂŁInterview with the Champion of Week 1
- Interview with a Nebulas DApp Developer: Jason Mansfield
- DApp Development and Architecture DesignâĂŁâĂŤâĂŁInterview with Honey Thakuria

Events

Since June 2017, the Nebulas meetups and hackathons (more than 60 meetups) have been held in 20 cities, 9 countries around the world. We have visited the University of California, Berkeley, the New York University, Columbia University, Harvard University, the Singapore University of Social Sciences, Tsinghua University, Tongji University, and many others. View the events history . You are welcome to organize local meetups and participate in the history of Nebulas.

1.4.2 Using Nebulas

If you are a developer and want to develop a DApp or use the mainnet, please visit *the develop chapter* and *tutorials* to learn more about Nebulas technology and find develop resources. If you are an individual, there are four ways to use Nebulas:

- *1. Use an application built on Nebulas*
- *2. What's NAS and how to get it?*
- *3. What's a wallet and how to hold NAS?*
- *4. What's NAX and how to get it?*

1. Use an application built on Nebulas

View recommend DApps here. You are welcome to submit the form to recommend more DApps. And you can find more DApps in the The Nebulas DApps Store by the community member m5j.

2. What's NAS and how to get it?

NAS is the native (utility) coin of Nebulas, viable for payment of transaction fees and the computing service charge. Click here to view the distribution. The Nebulas blockchain provides native incentives to encourage developers and community members to build a healthy economy and ecosystem.

You can buy & sell NAS from exchanges, click here to view the exchanges list. You can also buy NAS from CoinSwitch and SWFT Blockchain.

You can also be a community contributor and earn NAS. Please visit nebulas community collaboration platform: Go.nebulas.

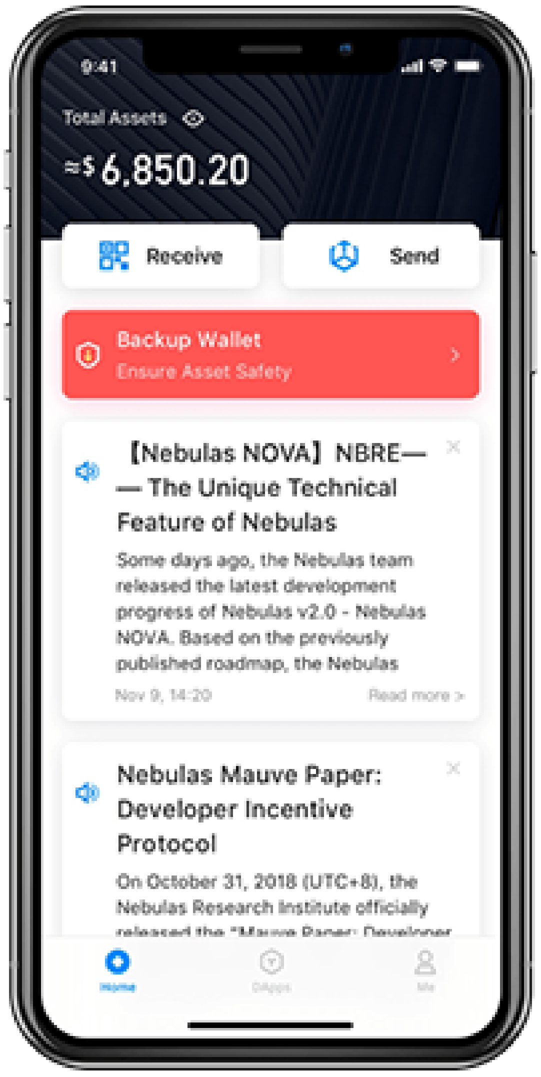
3. What's a wallet and how to hold NAS?

NAS nano pro

NAS nano pro is the official wallet, developed by the Nebulas team. You may download it here. It has a beautiful, easy-to-use interface, and implements all the features of a robust cryptocurrency wallet, as well as multiple security policies, so that users can easily manage their NAS assets without a steep learning curve.

The NAS nano pro wallet comes with four main features:

- Quickly and easily create, import, and manage wallets.
- Check the transaction progress in your wallet at a glance.
- Provide three kinds of wallet backups, including mnemonic, Keystore, private key backups, to minimize loss and theft of assets.
- Support NAS, as well as other NRC20 tokens, such as NAX and ATP. If you want to list your token on NAS nano pro, please click here.



【Nebulas NOVA】 NBRE—— The Unique Technical Feature of Nebulas

Some days ago, the Nebulas team released the latest development progress of Nebulas v2.0 - Nebulas NOVA. Based on the previously published roadmap, the Nebulas

Nov 9, 14:20 Read more >

Nebulas Mauve Paper: Developer Incentive Protocol

On October 31, 2018 (UTC+8), the Nebulas Research Institute officially released the "Mauve Paper: Developer

Nebulas Web Wallet

Click here to download NAS Wallet (Chrome Extension version). Click here to download Nebulas web wallet (local version). Nebulas web wallet tutorial is below:

- Part 1 - Creating A NAS Wallet
- Part 2 - Sending NAS from your Wallet
- Part 3 - Signing a Transaction Offline
- Part 4 - View Wallet Information

- Part 5 - Check TX Status
- Part 6 - Deploy a Smart Contract
- Part 7 - Call a Smart Contract on Nebulas Wallet

Other wallets

These following wallets support NAS, you can select the one you liked:

- Wallet.io
- Kaiser Wallet (an affordable cold wallet in a smart card form)
- Math Wallet
- SWFT Wallet
- BEPAL Wallet (with hardware wallet)
- Trust Wallet (a Secure Multi Coin Wallet, the official cryptocurrency wallet of Binance)

Click here to learn more details about these wallets.

4. What's NAX and how to get it?

This smart asset is generated by decentralized staking and is the first token on nextDAO. Users on the Nebulas blockchain can obtain NAX by dStaking NAS. NAX adopts dynamic distribution strategy where the actual issuance quantity is related to the global pledge rate, the amount of NAS pledged individually and the age of the dStake.

NAX is more closely related to its ecosystem and constitutes a positive-feedback economy. Visit nextdao.io to learn more.

There is only one way to mint NAX: dStaking NAS. But there are three ways to obtain NAX:

dStaking NAS and obtain NAX:

- Download NAS nano pro to dStake NAS.
- Online dStaking.
- Offline dStaking. Read cold wallet dStaking tutorial.

Buy & Sell NAX from Exchanges:

- gate.io
- MXC

Swap NAX: Swap NAX via many other tokens on SWFT Blockchain.

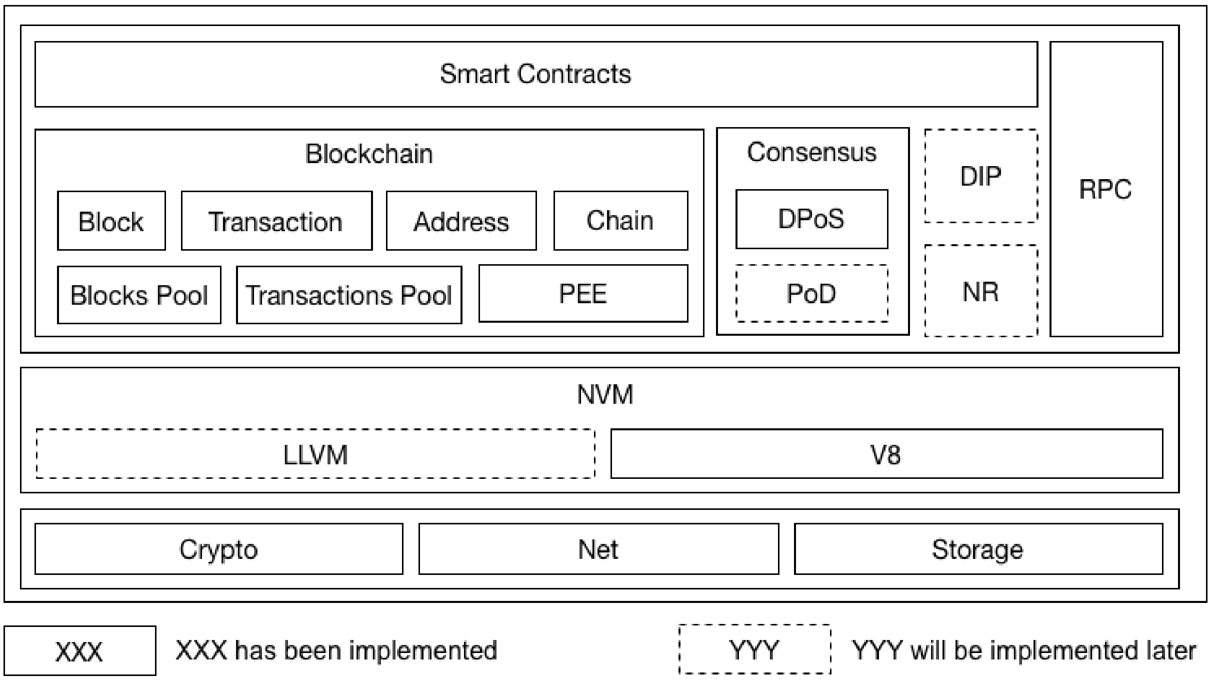
1.4.3 Development on the Chain

Getting started

To get the basic concepts of Nebulas visit the Nebulas homepage over at nebulas.io. If you want to get a deeper understanding, start by reading the technical whitepaper and non-technical whitepaper.To be a contributor, visit how to contribute.

For getting started guides and documents, see here:

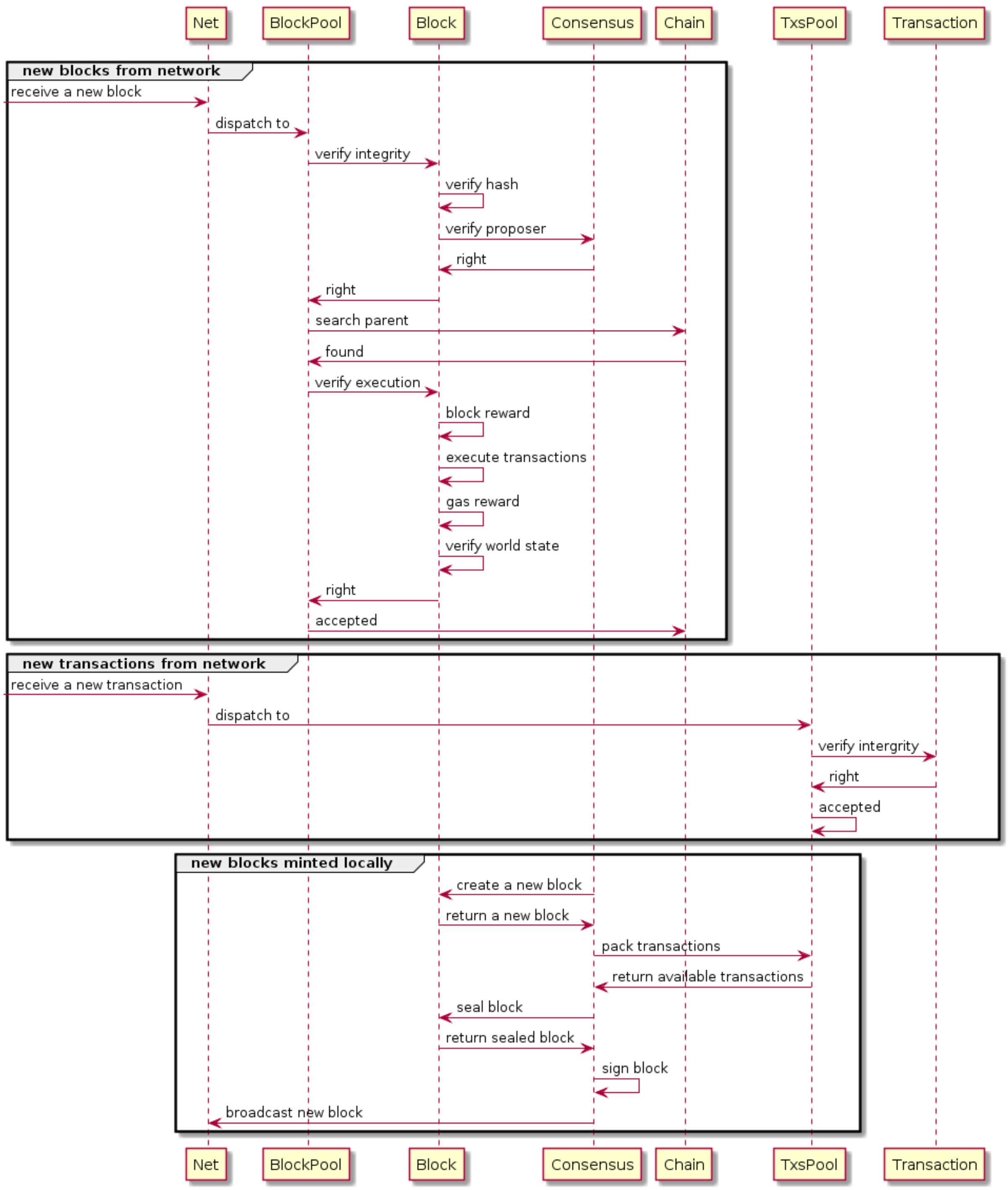
Design Overview



TODO: More features described in our whitepaper, such as NR, PoD, DIP and NF, will be integrated into the framework in later versions very soon.

Core Dataflow

Here is a core workflow example to explain how Nebulas works in current version. For each Nebulas node, it keeps receiving blocks or transactions from network and mining new block locally.



More Details

Blockchain

Model

Nebulas use accounts model instead of UTXO model. The execution of transactions will consume gas.

Data Structure

```
Block Structure
+---------------+----------------+--------------+
|  blockHeader  |  transactions  |  dependency  |
+---------------+----------------+--------------+
blockHeader: header info
transactions: transactions array
dependency: the dependency relationship among transactions

Block Header Structure
+-----------+--------+--------------+------------+-------------+----
↪---+--------+
|  chainid  |  hash  |  parentHash  |  coinbase  |  timestamp  | ␣
↪alg  |  sign  |
+-----------+--------+--------------+------------+-------------+----
↪---+--------+
+-------------+-----------+--------------+-----------------+
|  stateRoot  |  txsRoot  |  eventsRoot  |  consensusRoot  |
+-------------+-----------+--------------+-----------------+
chainid: chain identity the block belongs to
hash: block hash
parentHash: parent block hash
coinbase: account to receive the mint reward
timestamp: the number of nanoseconds elapsed since January 1, 1970␣
↪UTC
alg: the type of signature algorithm
sign: the signature of block hash
stateRoot: account state root hash
txsRoot: transactions state root hash
eventsRoot: events state root hash
consensusRoot: consensus state, including proposer and the dynasty␣
↪of validators


Transaction Structure
+-----------+--------+--------+------+---------+---------+----------
↪---+
|  chainid  |  hash  |  from  |  to  |  value  |  nonce  | ␣
↪timestamp  |
+-----------+--------+--------+------+---------+---------+----------
↪---+
+--------+------------+------------+
|  data  |  gasPrice  |  gasLimit  |
+--------+------------+------------+
chainid: chain identity the block belongs to
hash: transaction hash
from: sender's wallet address
to: receiver's wallet address
value: transfer value
```

```
nonce: transaction nonce
timestamp: the number of seconds elapsed since January 1, 1970 UTC
alg: the type of signature algorithm
sign: the signature of block hash
data: transaction data, including the type of transaction(binary␣
↪transfer/deploy smart contracts/call smart contracts) and payload
gasPrice: the price of each gas consumed by the transaction
gasLimit: the max gas that can be consumed by the transaction
```

Blockchain Update

In our opinion, **Blockchain** only needs to care about how to process new blocks to grow up safely and efficiently. What‘s more, **Blockchain** can only get new blocks in the following two channels.

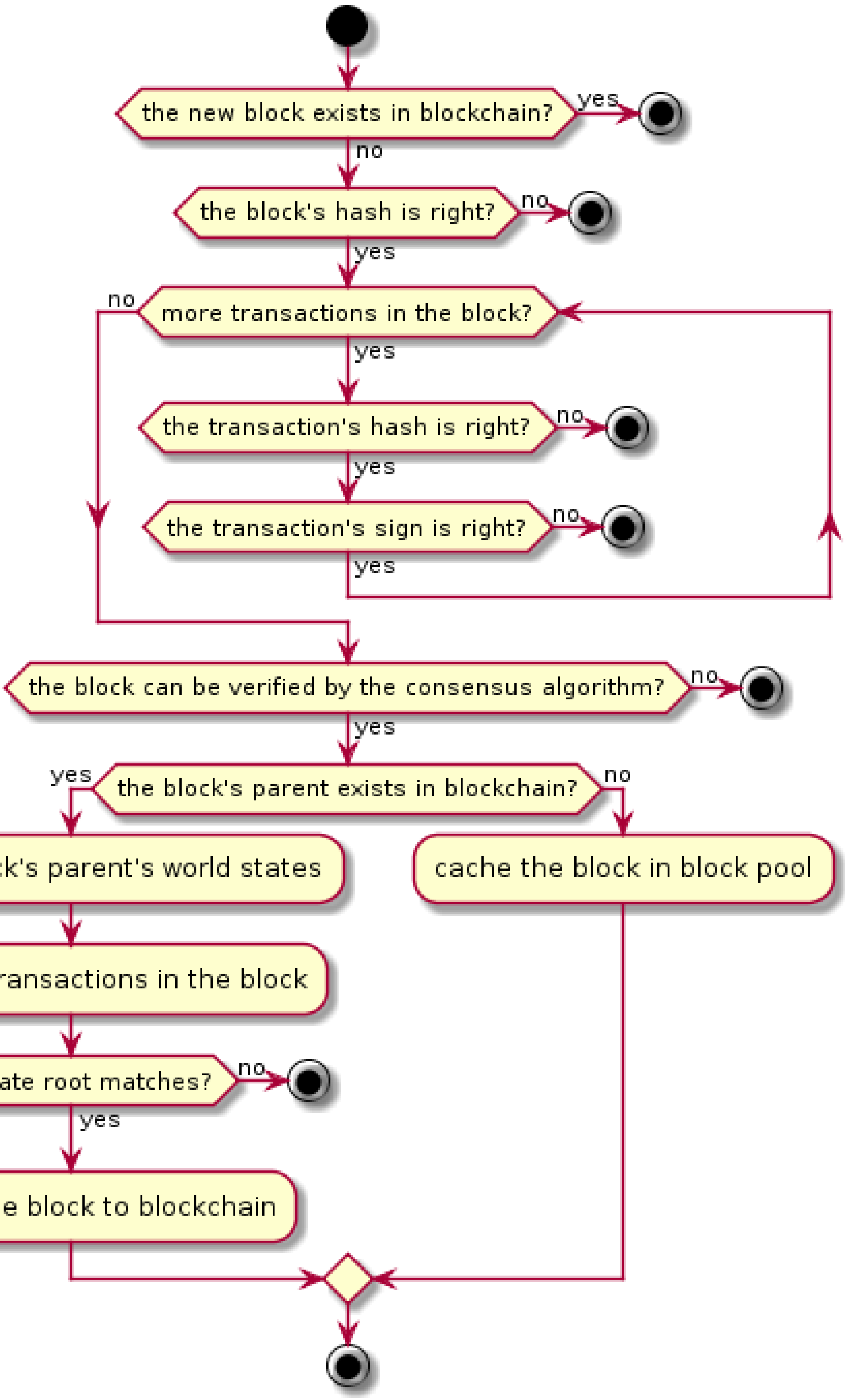
A new block from network

Because of the unstable network latency, we cannot make sure any new block received can be linked to our current **Chain** directly. Thus, we need the **Blocks Pool** to cache new blocks.

A new block from local miner

At first, we need the **Transactions Pool** to cache transactions from network. Then, we wait for a new block created by local **Consensus** component, such as DPoS.

No matter where a new block comes from, we use the same steps to process it as following.


the new block exists in blockchain?
yes
no
the block's hash is right?
no
yes
more transactions in the block?
no
yes
the transaction's hash is right?
no
yes
the transaction's sign is right?
no
yes
the block can be verified by the consensus algorithm?
no
yes
the block's parent exists in blockchain?
yes
no
clone the block's parent's world states
cache the block in block pool
execute all transactions in the block
the state root matches?
no
yes
append the block to blockchain

World State

Every block contains the current world state, consist of following four states. They are all maintained as Merkle Trees.

Accounts State

All accounts in current block are stored in Accounts State. Accounts are divided into two kinds, normal account & smart contract account.

Normal Account, including

- **wallet address**
- **balance**
- **nonce**: account's nonce, it will increment in steps of 1

Smart Contract AccountïijŇ including

- **contract address**
- **balance**
- **birth place**: the transaction hash where the contract is deployed
- **variables**: contains all variables' values in the contract

Transactions State

All transactions submitted on chain are storage in Transactions State.

Events State

While transactions are executed, many events will be triggered. All events triggered by transactions on chain are stored in Events State.

Consensus State

The context of consensus algorithm is stored in consensus state.

As for DPoS, the consensus state includes

- **timestamp**: current slot of timestamp
- **proposer**: current proposer
- **dynasty**: current dynasty of validators

Serialization

We choose Protocol Buffers to do general serialization in consideration of the following benefits:

- Large scale proven.

- Efficiency. It omits key literals and use varints encoding.
- Multi types and multilangue client support. Easy to use API.
- Schema is good format for communication.
- Schema is good for versioning/extension, i.e., adding new message fields or deprecating unused ones.

Specially, we use json to do serialization in smart contract codes instead of protobuf for the sake of readability.

Synchronization

Sometimes we will receive a block with height much higher than its current tail block. When the gap appears, we need to sync blocks from peer nodes to catch up with them.

Nebulas provides two method to sync blocks from peers: Chunks Downloader and Block Downloader. If the gap is bigger than 32 blocks, we'll choose Chunk Downloader to download a lot of blocks in chunks. Otherwise, we choose Block Downloader to download block one by one.

Chunks Downloader

Chunk is a collection of 32 successive blocks. Chunks Downloader allows us to download at most 10 chunks following our current tail block each time. This chunk-based mechanism could help us minimize the number of network packets and achieve better safety.

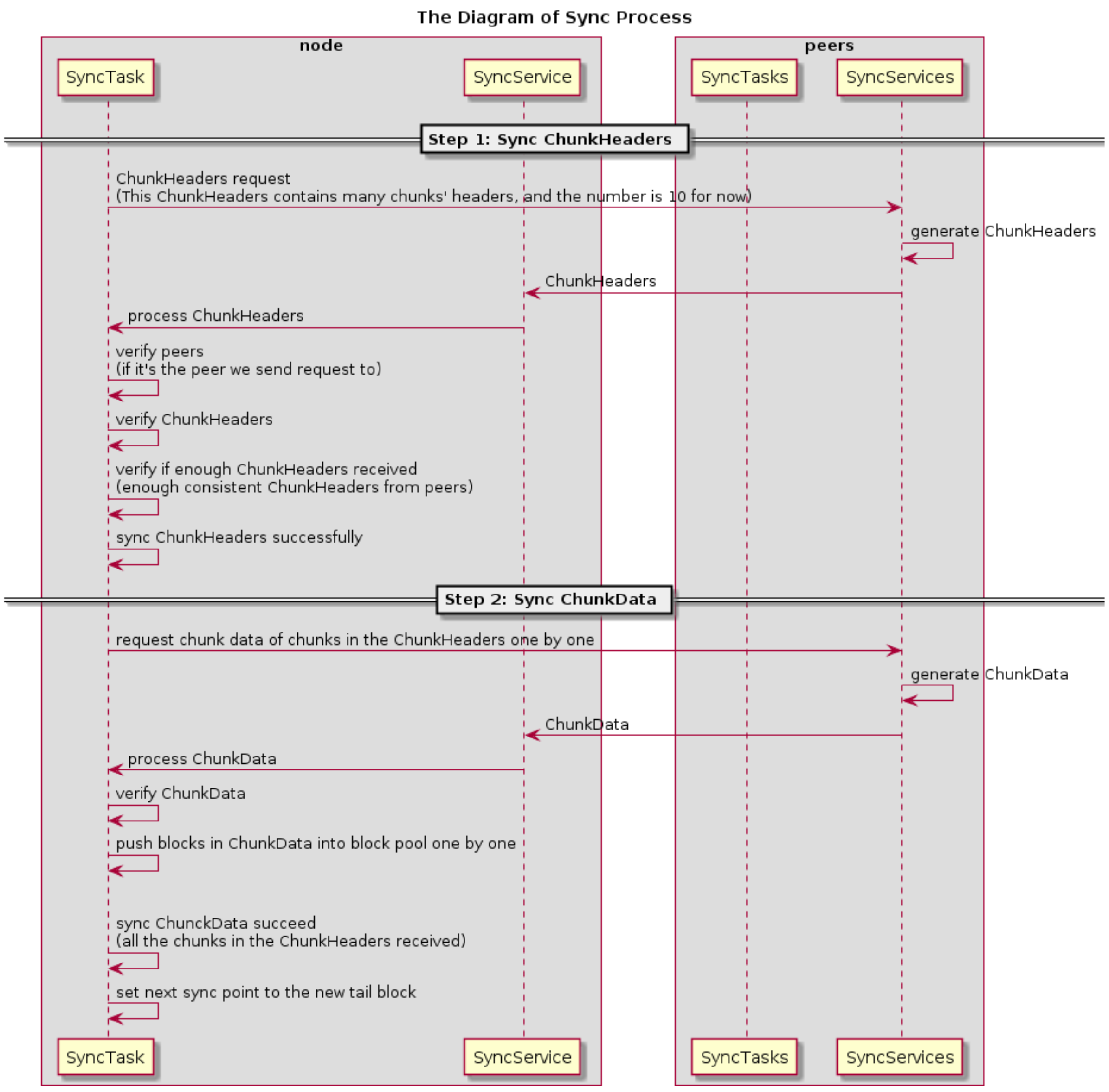
The procedure is as following,

```
1. A sends its tail block to N remote peers.
2. The remote peers locate the chunk C that contains A's tail block.
   Then they will send back the headers of 10 chunks, including the
↪chunk C and 9 C's subsequent chunks, and the hash H of the 10
↪headers.
3. If A receives >N/2 same hash H, A will try to sync the chunks
↪represented by H.
4. If A has fetched all chunks represented by H and linked them on
↪chain successfully, Jump to 1.
```

In steps 1~3, we use majority decision to confirm the chunks on canonical chain. Then we download the blocks in the chunks in step 4.

Note: `ChunkHeader` contains an array of 32 block hash and the hash of the array. `ChunkHeaders` contains an array of 10 `ChunkHeaders` and the hash of the array.

Here is a diagram of this sync procedure:


The Diagram of Sync Process
node
peers
SyncTask
SyncService
SyncTasks
SyncServices
Step 1: Sync ChunkHeaders
ChunkHeaders request
(This ChunkHeaders contains many chunks' headers, and the number is 10 for now)
generate ChunkHeaders
ChunkHeaders
process ChunkHeaders
verify peers
(if it's the peer we send request to)
verify ChunkHeaders
verify if enough ChunkHeaders received
(enough consistent ChunkHeaders from peers)
sync ChunkHeaders successfully
Step 2: Sync ChunkData
request chunk data of chunks in the ChunkHeaders one by one
generate ChunkData
ChunkData
process ChunkData
verify ChunkData
push blocks in ChunkData into block pool one by one
sync ChunckData succeed
(all the chunks in the ChunkHeaders received)
set next sync point to the new tail block

Block Downloader

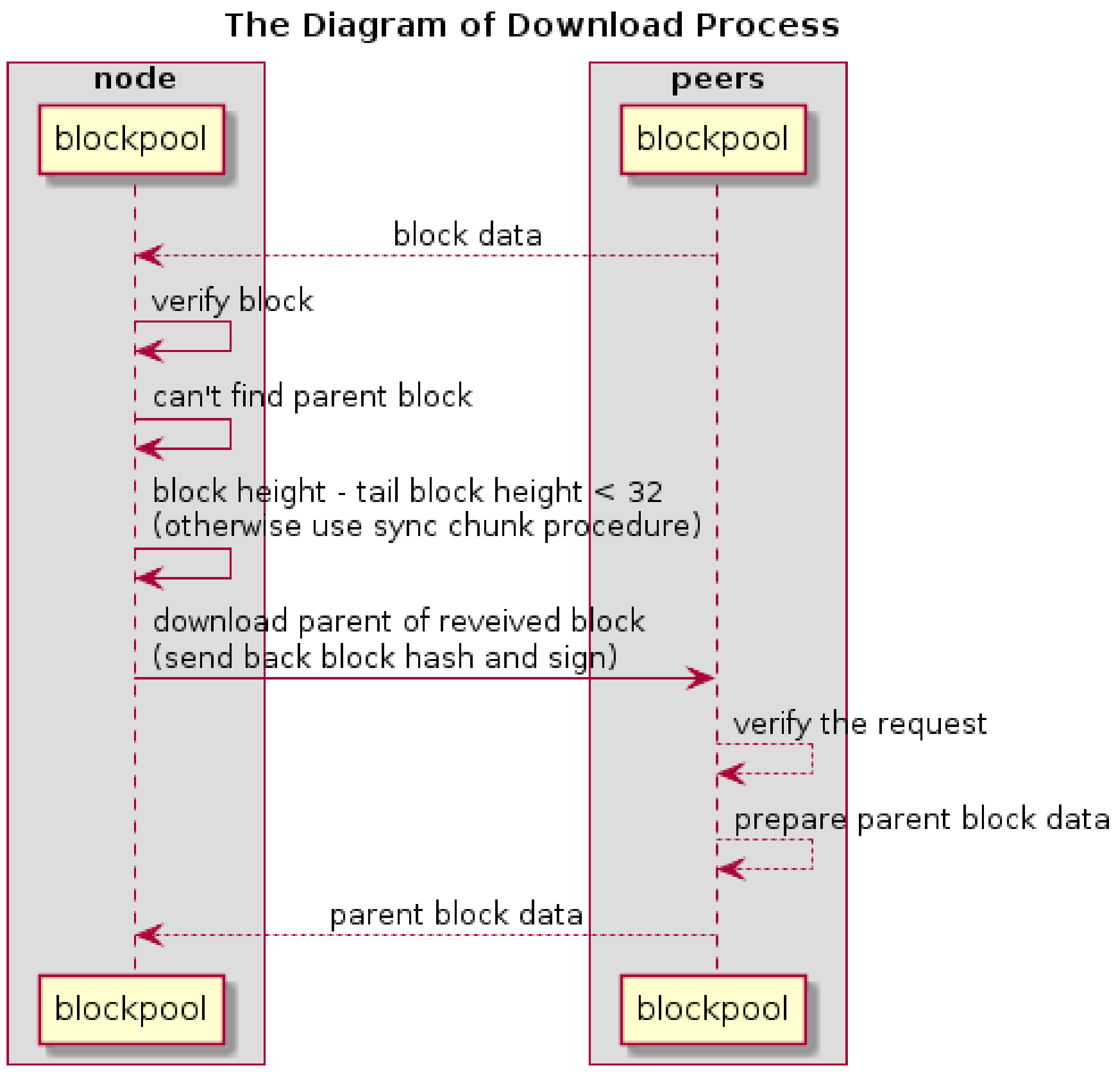
When the length gap between our local chain with the canonical chain is smaller than 32, we‘ll use Block downloader to download the missing blocks one by one.

The procedure is as following,

```
1. C relays the newest block B to A and A finds B's height is␣
↪bigger than current tail block's.
2. A sends the hash of block B back to C to download B's parent␣
↪block.
3. If A received B's parent block B', A will try to link B' with A
↪'s current tail block.
   If failed again, A will come back to step 2 and continue to␣
↪download the parent block of B'. Otherwise, finished.
```

This procedure will repeat until A catch up with the canonical chain.

Here is a diagram of this download procedure:


The Diagram of Download Process
node
peers
blockpool
blockpool
block data
verify block
can't find parent block
block height - tail block height < 32
(otherwise use sync chunk procedure)
download parent of reveived block
(send back block hash and sign)
verify the request
prepare parent block data
parent block data
blockpool
blockpool

Merkle Patricia Tree

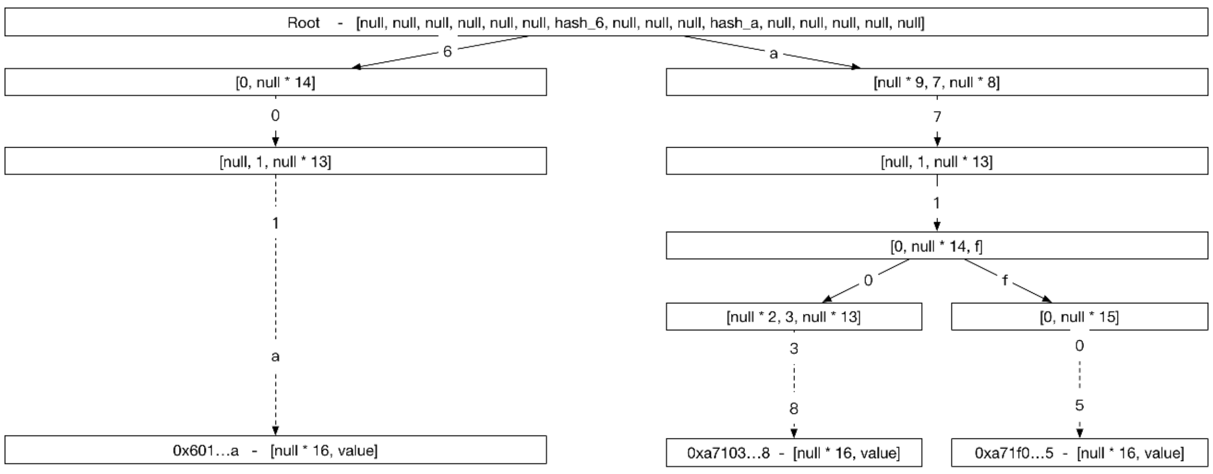
Basic: Radix Tree

Reference: https://en.wikipedia.org/wiki/Radix_tree

A Radix Tree using address as the key looks like below:

- Addresses are represented as Hex Characters
- Each node in the Tree is a 16-elements array, 16 branch-slots(0123...def)
- leaf node: value can be any binary data carried by the address
- non-leaf node: value is the hash value calculated based on the childrenâĂŹs data

As for a 160-bits address, the max height of the tree is 40


Root - [null, null, null, null, null, null, hash_6, null, null, null, hash_a, null, null, null, null, null]
6
a
[0, null * 14]
[null * 9, 7, null * 8]
0
7
[null, 1, null * 13]
[null, 1, null * 13]
1
1
[0, null * 14, f]
0
f
[null * 2, 3, null * 13]
[0, null * 15]
3
0
a
8
5
0x601…a - [null * 16, value]
0xa7103…8 - [null * 16, value]
0xa71f0…5 - [null * 16, value]

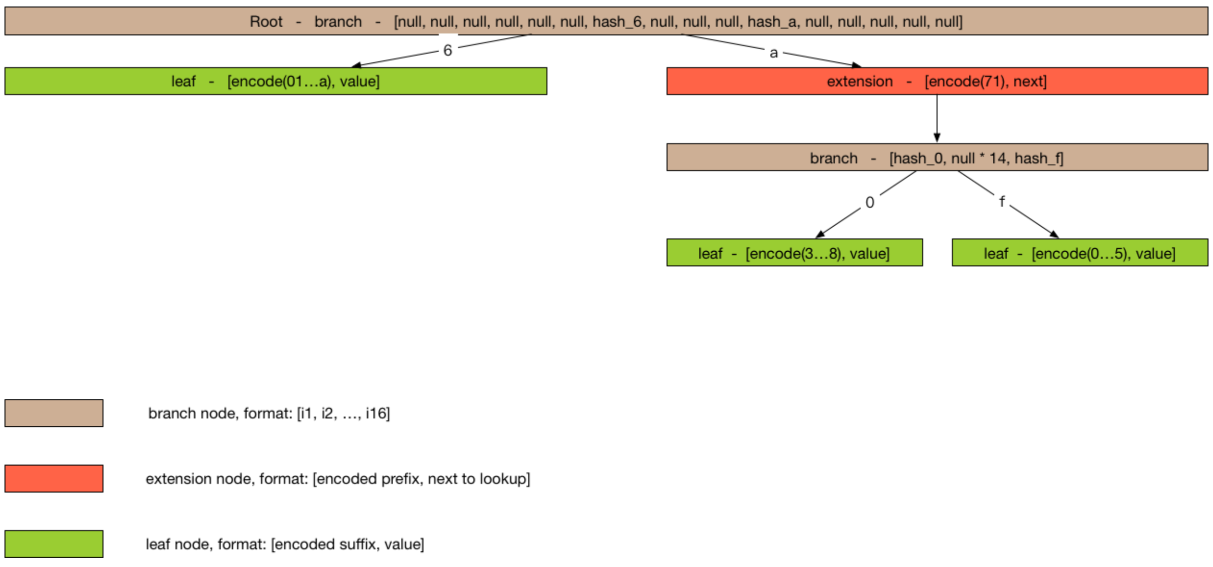
Problems: much space for a single entry 40 steps for each lookup

Advanced: Merkle Patricia Tree

Reference: https://github.com/ethereum/wiki/wiki/Patricia-Tree, http://gavwood.com/Paper.pdf

In order to reduce the storage of Radix Tree. The nodes in Merkle Patricia Tree are divided into three kinds,

- extension node: compress nodes using common prefix
- leaf node: compress nodes using unique suffix
- branch node: same as node in Radix Tree


Root - branch - [null, null, null, null, null, null, hash_6, null, null, null, hash_a, null, null, null, null, null]
6
a
leaf - [encode(01…a), value]
extension - [encode(71), next]
branch - [hash_0, null * 14, hash_f]
0
f
leaf - [encode(3…8), value]
leaf - [encode(0…5), value]
branch node, format: [i1, i2, …, i16]
extension node, format: [encoded prefix, next to lookup]
leaf node, format: [encoded suffix, value]

How to store Merkle Patricia Tree

Key/Value Storage

hash(value) = sha3(serialize(value))

key = hash(value)

How to update Merkle Patricia Tree

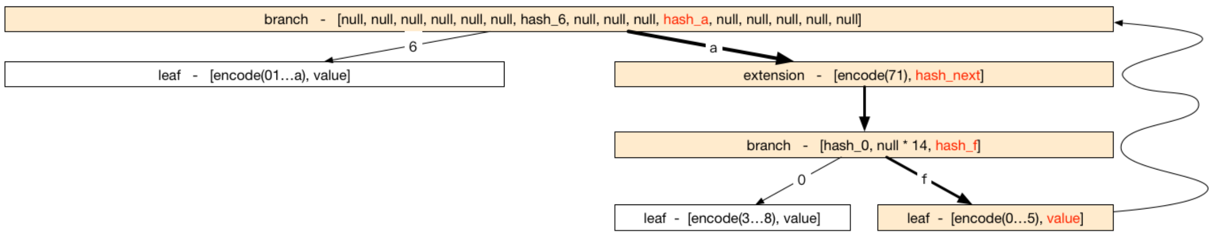
Query

DFS from top to bottom

Update, Delete or Insert

1.Query the node from top to bottom

2.update the hash along the path from bottom to top


branch - [null, null, null, null, null, null, hash_6, null, null, null, hash_a, null, null, null, null, null]
6
a
leaf - [encode(01...a), value]
extension - [encode(71), hash_next]
branch - [hash_0, null * 14, hash_f]
0
f
leaf - [encode(3...8), value]
leaf - [encode(0...5), value]

Performance Each operation costs O(log(n))

How to verify using Merkle Patricia Tree

Theorems

1.Same merkle trees must have same root hash.

2.Different merkle trees must have different root hash.

Using the theorems, we can verify the result of the execution of transactions.

Quick Verification

A light client, without sync huge transactions, can immediately determine the exact balance and status of any account by simply asking the network for a path from the root to the account node.

Consensus

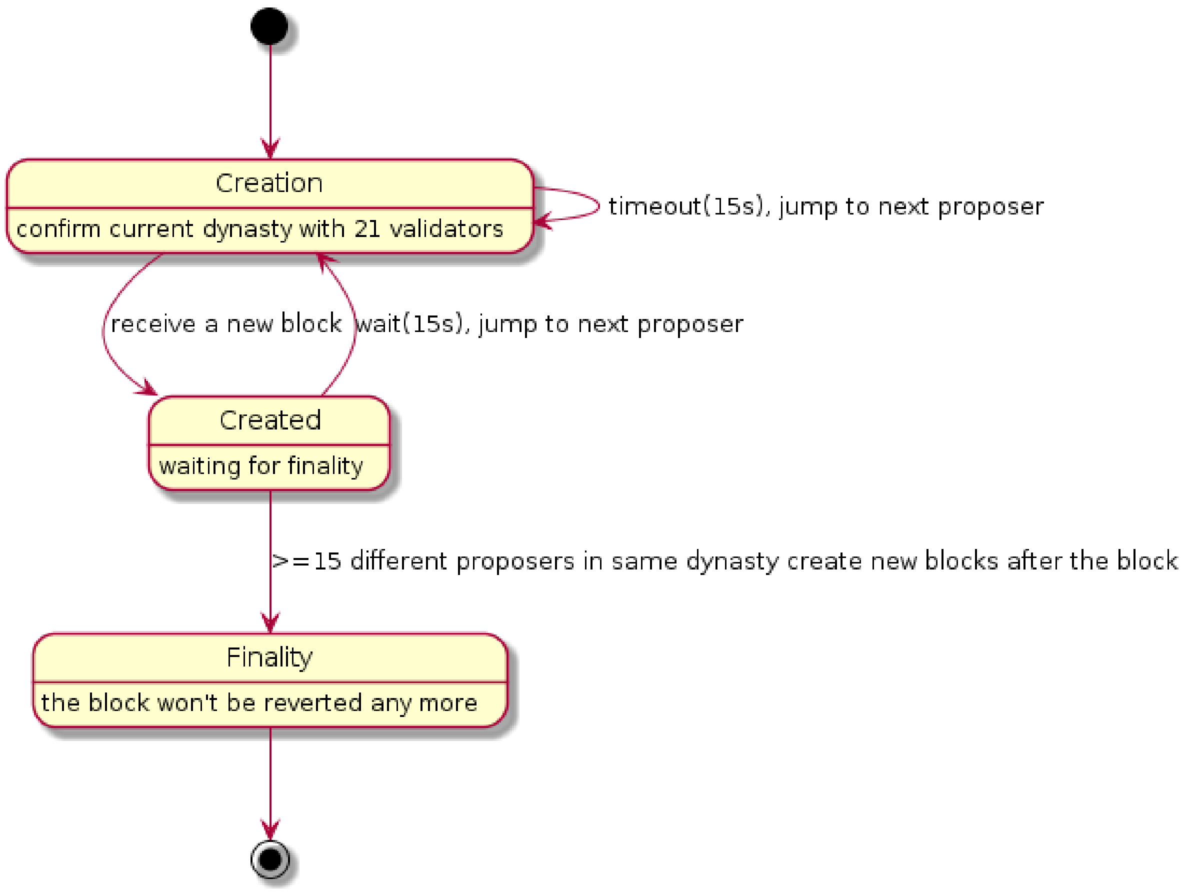
We think each consensus algorithm can be described as the combination of State Machine and Fork Choice Rules.

DPoS(Delegate Proof-of-Stake)

Notice For Nebulas, the primary consensus algorithm should be PoD, the DPoS algorithm is just a temporary solution. After the formal verification of PoD algorithm, we will transition mainnet to PoD. All witness (bookkeeper/miner) of DPoS are now accounts officially maintained by Nebulas. We will make sure a smooth transition from DPoS to PoD. We will create new funds to manage all the rewards of bookkeeping. And we will NOT sell those NAS on exchanges. All NAS will be used for building the Nebulas ecosystem, for example, rewarding DApp developers on Nebulas. And we will provide open access to all the spending of these rewards periodically.

As for the DPoS in Nebulas, it can also be decribed as a state machine.

State Machine


Creation
confirm current dynasty with 21 validators
timeout(15s), jump to next proposer
receive a new block
wait(15s), jump to next proposer
Created
waiting for finality
>=15 different proposers in same dynasty create new blocks after the block
Finality
the block won't be reverted any more

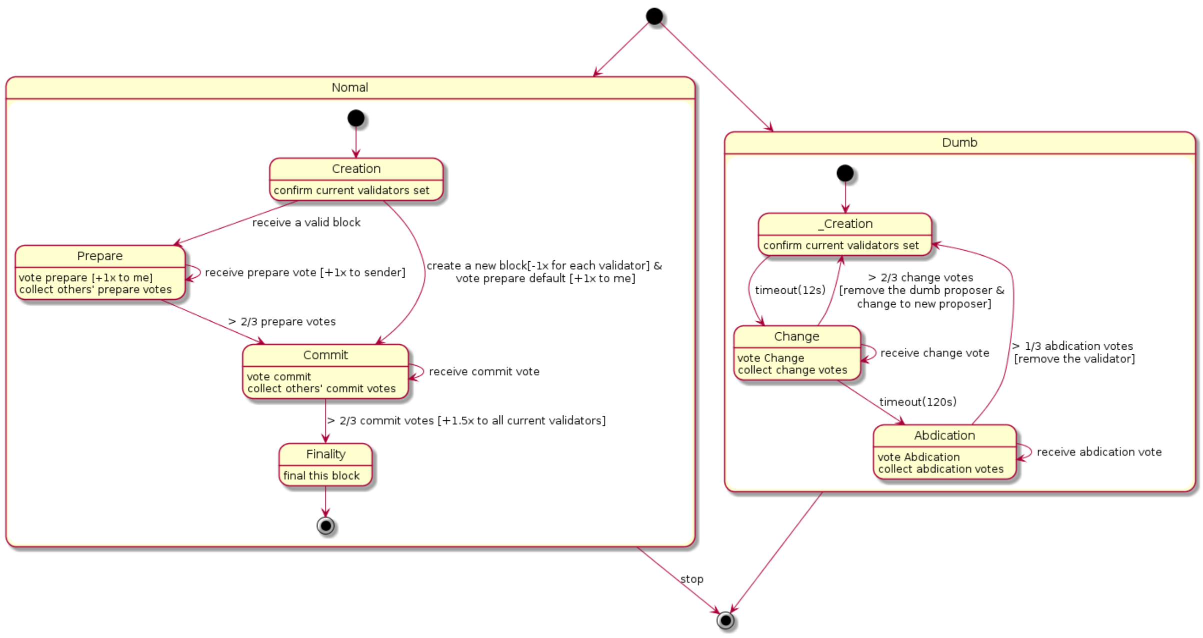
Fork Choice Rules

1. Always choose the longest chain as the canonical chain.
2. If A and B has the same length, we choose the one with smaller hash.

PoD (Proof-of-Devotion)

Here is a draft of PoD. The research on PoD is ongoing here.

State Machine



Fork Choice Rules

1. Always to choose the chain with highest sum of commit votes.
2. If A and B has the same length, we choose the one with smaller hash.

Transaction Process Diagram

When a transaction is submitted, it is necessary to check the chain in the transaction. Transactions that are submitted externally or have been packaged into the block are somewhat different when doing validation.

New Transaction Process (from network, rpc)

Transactions submitted through an RPC or other node broadcast.

- Api SendRawTransaction Verification below steps when exist fail, then return err
- check whether fromAddr and toAddr is valid (tx proto verification)
- check len of Payload <= MaxDataPayLoadLength (tx proto verification)
- 0 < gasPrice <= TransactionMaxGasPrice and 0 < gasLimit <= TransactionMaxGas (tx proto verification)
- check Alg is SECP256K1 (tx proto verification)
- chainID Equals, Hash Equals, Sign verify??; fail and drop;

- check nonceOfTx > nonceOfFrom
- check Contract status is ExecutionSuccess if type of tx is TxPayloadCallType, check toAddr is equal to fromAddr if type of tx is TxPayloadDeployType
- Transaction pool Verification
- gasPrice >= minGasPriceOfTxPool & 0 < gasLimit <= maxGasLimitOfTxPool??; fail and drop;
- chainID Equals, Hash Equals, Sign verify??; fail and drop;

Transaction in Block Process

The transaction has been packaged into the block, and the transaction is verified after receiving the block.

- Packed
- Nonce Verification: nonceOfFrom +1 == nonceOfTx ??; nonceOfTx < nonceOfFrom +1 fail and drop, nonceOfTx > nonceOfFrom +1 fail and giveback to tx pool;
- check balance >= gasLimit * gasPrice ??; fail and drop;
- check gasLimit >= txBaseGas(MinGasCountPerTransaction + dataLen*GasCountPerByte) ??; fail and drop;
- check payload is valid ??; fail and submit; gasConsumed is txBaseGas (all txs passed the step tx will be on chain)
- check gasLimit >= txBaseGas + payloasBaseGas(TxPayloadBaseGasCount[payloadType]) ??;fail and submit; gasConsumed is txGasLimit
- check balance >= gasLimit * gasPrice + value ??;fail and submit; gasConsumed is txBaseGas + payloadsBaseGas
- transfer value from SubBalance and to AddBalance ??;fail and submit; gasConsumed is txBaseGas + payloadsBaseGas
- check gasLimit >= txBaseGas + payloadsBaseGas + gasExecution ??;fail and submit; gasConsumed is txGasLimit
- success submit gasConsumed is txBaseGas + payloadsBaseGas + gasExecution
- Verify
- check whether fromAddr and toAddr is valid (tx proto verification) ??; fail and submit;
- check len of Payload <= MaxDataPayLoadLength (tx proto verification) ??; fail and submit;
- 0 < gasPrice <= TransactionMaxGasPrice and 0 < gasLimit <= TransactionMaxGas (tx proto verification)
- check Alg is SECP256K1 (tx proto verification) ??; fail and submit;
- chainID Equals, Hash Equals, Sign verify??; fail and drop;

- Next steps like Transaction Packed in Block Process.

Event functionality

The `Event` functionality is used to make users or developers subscribe interested events. These events are generated during the execution of the blockchain, and they record the key execution steps and execution results of the chain. To query and verify the execution results of transactions and smart contracts, we record these two types of events into a trie and save them to the chain.

Event structure:

```
type Event struct {
    Topic string // event topic, subscribe keyword
    Data  string // event content, a json string
}
```

After a event is generated, it will be collected for processing in eventEmitter. Users can use the emitter subscription event. If the event is not subscribed, it will be discarded, and for the event that has been subscribed, the new event will be discarded because of the non-blocking mechanism, if the channel is not blocked in time.

Events list:

- *TopicNewTailBlock*
- *TopicRevertBlock*
- *TopicLibBlock*
- *TopicPendingTransaction*
- *TopicTransactionExecutionResult*
- *EventNameSpaceContract*

Event Reference

TopicNewTailBlock

This event occurs when the tail block of the chain is updated.

- Topic:`chain.newTailBlock`
- Data:
  - `height`: block height
  - `hash`: block hash
  - `parent_hash`: block parent hash

  - `acc_root`: account state root hash
  - `timestamp`: block timestamp
  - `tx`: transaction state root hash
  - `miner`: block miner

TopicRevertBlock

This event occurs when a block is revert on the chain.

- Topic:`chain.revertBlock`
- Data: The content of this topic is like *TopicNewTailBlock* data.

TopicLibBlock

This event occurs when the latest irreversible block change.

- Topic:`chain.latestIrreversibleBlock`
- Data: The content of this topic is like *TopicNewTailBlock* data.

TopicPendingTransaction

This event occurs when a transaction is pushed into the transaction pool.

- Topic:`chain.pendingTransaction`
- Data:
  - `chainID`: transaction chain id
  - `hash`: transaction hash
  - `from`: transaction from address string
  - `to`: transaction to address string
  - `nonce`: transaction nonce
  - `value`: transaction value
  - `timestamp`: transaction timestamp
  - `gasprice`: transaction gas price
  - `gaslimit`: transaction gas limit
  - `type`: trsnaction type

TopicTransactionExecutionResult

This event occurs when the end of a transaction is executed. This event will be recorded on the chain, and users can query with RPC interface GetEventsByHash.

This event records the execution results of the transaction and is very important.

- Topic:`chain.transactionResult`
- Data:
  - `hash`: transaction hash
  - `status`: transaction status, 0 failed, 1success, 2 pending
  - `gasUsed`: transaction gas used
  - `error`: transaction execution error. If the transaction is executed successfully, the field is empty.

EventNameSpaceContract

This event occurs when the contract is executed. When the contract is executed, the contract can record several events in the execution process. If the contract is successful, these events will be recorded on the chain and can be subscribed, and the event of the contract will not be recorded at the time of the failure. This event will also be recorded on the chain, and users can query with RPC interface GetEventsByHash.

- Topic:`chain.contract.[topic]` The topic of the contract event has a prefix `chain.contract.`, the content is defined by the contract writer.
- Data: The content of contract event is defined by contract writer.

Subscribe

All events can be subscribed and the cloud chain provides a subscription RPC interface Subscribe. It should be noted that the event subscription is a non-blocking mechanism. New events will be discarded when the RPC interface is not handled in time.

Query

Only events recorded on the chain can be queried using the RPC interface GetEventsByHash. Current events that can be queried include:

- *TopicTransactionExecutionResult*
- *EventNameSpaceContract*

Transaction Gas

In Nebulas, either a normal transaction which transfer balance or a smart contract deploy & call burns gas, and charged from the balance of `from` address. A transaction contains two gas parameters `gasPrice` and `gasLimit` :

- `gasPrice`: the price of per gas.
- `gasLimit`: the limit of gas use.

The actual gas consumption of a transaction is the value: `gasPrice` * `gasUsed`, which will be the reward to the miner coinbase. The `gasUsed` value must less than or equal to the `gasLimit`. Transaction's `gasUsed` can be estimate by RPC interface estimategas and store in transaction's execution result event.

Design reason

Users want to avoid gas costs when the transaction is packaged. Like Bitcoin and Ethereum, Nebulas GAS is used for transaction fee, it have two major purposes:

- As a rewards for minter, to incentive them to pack transactions. The packaging of the transaction costs the computing resources, especially the execution of the contract, so the user needs to pay for the transaction.
- As a cost for attackers. The DDOS attach is quite cheap in Internet, black hackers hijack user's computer to send large network volume to target server. In Bitcoin and Ethereum network, each transaction must be paid, that significant raise the cost of attack.

Gas constitution

When users submit a transaction, gas will be burned at these aspects:

- `transaction submition`
- `transaction data storage`
- `transaction payload addition`
- `transaction payload execution`(smart contract execution)

In all these aspects, the power and resources of the net will be consumed and the miners will need to be paid.

Transaction submition

A transaction's submition will add a transaction to the tail block. Miners use resources to record the deal and need to be paid. It will burn a fixed number of gas, that would be defined in code as the following:

```
// TransactionGas default gas for normal transaction
TransactionGas = 20000
```

If the transaction verifies failed, the gas and value transfer will rollback.

Transaction data storage

When deploying a contract or call contract's method, the raw data of contract execution save in the transaction's data filed, which cost the storage of resources on the chain. A formula to calculate gas:

```
TransactionDataGas = 1

len(data) * TransactionDataGas
```

The `TransactionDataGas` is a fixed number of gas defined in code.

Different types of transactions' payload have different gas consumption when executed. The types of transactions currently supported by nebulas are as follows:

- `binary`: The `binary` type of transaction allows users to attach binary data to transaction execution. These binary data do not do any processing when the transaction is executed.
  - The fixed number of gas defined **0**.
- `deploy & call`: The `deploy` and `call` type of transaction allows users to deploy smart contract on nebulas. Nebulas must start `nvm` to execute the contract, so these types of transction must paid for the nvm start.
  - The fixed number of gas defined **60**.

Transaction payload execution(Smart contract deploy & call)

The `binary` type of transaction do not do any processing when the transaction is executed, so the execution need not be paid.

When a smart contract deploys or call in transaction submition, the contract execution will consume miner's computer resources and may store data on the chain.

- **execution instructions**: Every contract execution cost the miner's computer resources, the v8 instruction counter calculates the execution instructions. The limit of execution instructions will prevent the excessive consumption of computer computing power and the generation of the death cycle.
- **contract storage**: The smart contract's `LocalContractStorage` which storage contract objects also burn gas. Only one gas per 32 bytes is consumed when stored(`set`/`put`), `get` or `delete` not burns gas.

The limit of **contract execution** is:

```
    gasLimit - TransactionGas - len(data) * TransactionDataGas - TransactionPayloadGasCount[type]
```

Gas Count Matrix

The gas count matrix of smart contract execution

| Expression | Sample Code | Binary Opt. | Load Opt. | Store Opt. | Return Opt. | Call (inner) Opt. | Gas Count | | | | — | :— | —: | —: | —: | —: | —: | | CallExpression | a(x, y) | 0 | 0 | 1 | 1 | 1 | 8 | | | | AssignmentExpression | x&=y | 1 | 0 | 1 | 0 | 0 | 3 | | | | BinaryExpression | x==y | 1 | 0 | 0 | 1 | 0 | 3 | | | | UpdateExpression | x++ | 1 | 0 | 1 | 0 | 0 | 3 | | | | UnaryExpression | x+y | 1 | 0 | 0 | 1 | 0 | 3 | | | | LogicalExpression | x | | y | 1 | 0 | 0 | 1 | 0 | 3 | | MemberExpression | x.y | 0 | 1 | 0 | 1 | 0 | 4 | | | | NewExpression | new X() | 0 | 0 | 1 | 1 | 1 | 8 | | | | ThrowStatement | throw x | 0 | 0 | 0 | 1 | 1 | 6 | | | | MetaProperty | new.target | 0 | 1 | 0 | 1 | 0 | 4 | | | | ConditionalExpression | x?y:z | 1 | 0 | 0 | 1 | 0 | 3 | | | | YieldExpression | yield x | 0 | 0 | 0 | 1 | 1 | 6 | | | | Event | | 0 | 0 | 0 | 0 | 0 | 20 | | | | Storage | | 0 | 0 | 0 | 0 | 0 | 1 gas/bit | | |

Tips

In nebulas, the transaction pool of each node has a minimum and maximum `gasPrice` and maximum `gasLimit` value. If transaction's `gasPrice` is not in the range of the pool's `gasPrice` or the `gasLimit` greater than the pool's gasLimit the transaction will be refused.

Transaction pool gasPrice and gasLimit configuration:

- `gasPrice`
  - minimum: The minimum gasPrice can be set in the configuration file. If the minimum value is not configured, the default value is `20000000000`(2*10^10).
  - maximum: The maximum gasPrice is `1000000000000`(10^12), transaction pool's maximum configuration and transaction's `gasPrice` can't be overflow.
- `gasLimit`
  - minimum: The transaction's minimum gasLimit must greater than zero.
  - maximum: The maximum gasPrice is `50000000000`(50*10^9), transaction pool's maximum configuration and transaction's `gasLimit` can't be overflow.

Logs

Introduction

Nebulas provides two kinds of logs: console log & verbose log.

Console Log

Console Log(CLog) is used to help you understand which job **Neb** is working on now, including start/stop components, receive new blocks on chain, do synchronization and so on.

- CLog will print all logs to stdout & log files both. You can check them in your standard output directly.

Nebulas console log statements

```
// log level can be `Info`,`Warning`,`Error`
logging.CLog().Info("")
```

Startup specifications

Nebulas start service should give a console log, the logs should before the service start. The log format just like this:

```
logging.CLog().Info("Starting xxx...")
```

Stopping specifications

Nebulas stop service should give a console log, the logs should before the service stoped. The log format just like this:

```
logging.CLog().Info("Stopping xxx...")
```

Verbose Log

Verbose Log(VLog) is used to help you understant how **Neb** works on current job, including how to verifiy new blocks, how to discover new nodes, how to mint and so on.

- VLog will print logs to log files only. You can check them in your log folders if needed.

What‘r more, you can set your concerned level to VLog to filter informations. The level filter follows the priority as **Debug < Info < Warn < Error < Fatal.**

Hookers

By default, Function hookers & FileRotate hookers are added to CLog & VLog both.

FunctionNameHooker

FunctionHooker will append current caller‘s function name & code line to the loggers. The result looks like this, time="2018-01-03T20:20:52+08:00" level=info msg="node init success" file=net_service.go **func=p2p.NewNetManager** **line=137** node.listen="[0.0.0.0:10001]"

FileRotateHooker

FileRotateHooker will split logs into many smaller segments by time. By default, all logs will be rotated every 1 hour. The log folder looks like this,

neb-2018010415.log neb-2018010416.log neb.log -> /path/to/neb-2018010415.log

If you have any suggestions about logs, please feel free to submit issues on our wiki repo. Thanks!

Nebulas Address Design

Nebulas address system is carefully designed. As you will see below, both account and smart contract address are strings starting with a "n", which could be thought of as our faith Nebulas/NAS.

Account Address

Similar to Bitcoin and Ethereum, Nebulas also adopts elliptic curve algorithm as its basic encryption algorithm for Nebulas accounts. The address is derived from **public key**, which is in turn derived from the **private key** that encrypted with user's **passphrase**.Also we have the checksum design aiming to prevent a user from sending *Nas* to a wrong user account accidentally due to entry of several incorrect characters.

The specific calculation formula is as follows:

```
1.  content = ripemd160(sha3_256(public key))
    length: 20 bytes
                         +---------+--------+------------------+
2.  checksum = sha3_256( |  0x19   +  0x57  |      content     |␣
↪)[:4]
                         +---------+--------+------------------+
    length: 4 bytes

                        +---------+---------+-----------------+------
↪------+
3.  address = base58( |    0x19  |  0x57   |     content     | ␣
↪checksum  | ïijL'
                        +---------+---------+-----------------+------
↪------+
    length: 35 chars
```

0x57 is a one-byte "type code" for account address, **0x19** is a one-byte fixed "padding"

At this stage, Nebulas just adopts the normal bitcoin base58 encoding schema. A valid address is like: *n1TV3sU6jyzR4rJ1D7jCAmtVGSntJagXZHC*

Smart Contract Address

Calculating contract address differs slightly from account, passphrase of contract sender is not required but address & nonce. For more information, please check smart contract and rpc.sendTransaction. Calculation formula is as follows:

```
1.  content = ripemd160(sha3_256(tx.from, tx.nonce))
    length: 20 bytes
                         +--------+--------+-------------------+
2.  checksum = sha3_256( |  0x19  |  0x58  +      content      |␣
↪)[:4]
                         +--------+--------+-------------------+
    length: 4 bytes

                      +--------+---------+-----------------+--------
↪----+
3.  address = base58( |  0x19  |  0x58   |     content     | ␣
↪checksum  | ïijL'
                      +--------+---------+-----------------+--------
↪----+
    length: 35 chars
```

0x58 is a one-byte “type code“ for smart contract address, **0x19** is a one-byte fixed “padding“

A valid address is like: *n1sLnoc7j57YfzAVP8tJ3yK5a2i56QrTDdK*

DIP (TBD)

Infrastructure

Network Protocol

For the network protocol, there were a lot of existing solutions. However, the Nebulas Team decided to define their own wire protocol, and ensure the use of the following principles to design it:

- the protocol should be simple and straight.
- the messages can be verified before receiving all the packets, and fail early.
- the protocol should be debugging friendly, so that the developer can easily understand the raw message.

Protocol

In Nebulas, we define our own wire protocol as follows:

```
 0               1               2               3             ␣
↪(bytes)
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Magic Number                           |
+---------------------------------------------------------------+
|                        Chain ID                               |
+-----------------------------------------------+---------------+
|                        Reserved               |   Version     |
+-----------------------------------------------+---------------+
|                                                               |
+                                                               +
|                        Message Name                           |
+                                                               +
|                                                               |
+---------------------------------------------------------------+
|                        Data Length                            |
+---------------------------------------------------------------+
|                        Data Checksum                          |
+---------------------------------------------------------------+
|                        Header Checksum                        |
|---------------------------------------------------------------+
|                                                               |
+                        Data                                   +
.                                                               .
.                                                               .
|                                                               |
+---------------------------------------------------------------+
```

- Magic Number: 32 bits (4 chars)
  - The protocol's magic number, a numerical constant or text value used to identify the protocol.
  - Default: 0x4e, 0x45, 0x42, 0x31
- Chain ID: 32 bits
  - The Chain ID is used to distinguish the test network from the main network.
- Reserved: 24 bits
  - reserved field.
  - The first bit indicates whether the network message is compressed.
  - compressed: {0x80, 0x0, 0x0}; uncompressed: {0x0, 0x0, 0x0}
- Version: 8 bits
  - The version of the Message Name.

- Message Name: 96 bits (12 chars)
  - The identification or the name of the Message.
- Data Length: 32 bits
  - The total length of the Data.
- Data Checksum: 32 bits
  - The CRC32 checksum of the Data.
- Header Checksum: 32 bits
  - The CRC32 checksum of the fields from Magic Number to Data Checksum, totally 256 bits.
- Data: variable length, max 512M.
  - The message data.

We always use Big-Endian on the message protocol.

Handshaking Messages

- Hello

the handshaking message when a peer connects to another.

```
version: 0x1

data: struct {
    string node_id  // the node id, generated by underlying libp2p.
    string client_version // the client version, x.y.z schema, eg.␣
↪0.1.0.
}
```

- OK

the response message for handshaking.

```
version: 0x1

data: struct {
    string node_id // the node id, generated by underlying libp2p.
    string node_version // the client version, x.y.z schema, eg. 0.
↪1.0.
}
```

- Bye

the message to close the connection.

```
version: 0x1
data: struct {
    string reason
}
```

Networking Messages

- NetSyncRoutes

request peers to sync route tables.

```
version: 0x1
```

- NetRoutes

contains the local route tables.

```
version: 0x1
data: struct {
    PeerID[] peer_ids // router tables.
}

struct PeerID {
    string node_id  // the node id.
}
```

Nebulas Messages

TBD.

Crypto Design Doc

Similar to Bitcoin and Ethereum, Nebulas also adopted an elliptic curve algorithm as its basic encryption algorithm for Nebulas transactions. Users' private keys will be encrypted with their passphrases and stored in a keystore.

Hash

Supports generic hash functions, like sha256, sha3256 and ripemd160 etc.

Keystore

The Nebulas Keystore is designed to manage userâĂŹs keys.

Key

The Key interface is designed to support various keys, including symmetric keys and asymmetric keys.

Provider

The Keystore provides different methods to save keys, such as *memory_provider* and *persistence_provider*. Before storage, the key has been encrypted in the keystore.

- `memory provider`: This type of provider keeps the keys in memory. After the key has been encrypted with the passphrase when user setkey or load, it is cached in memory provider.
- `persistence provider`: This type of provider serializes the encrypted key to the file. The file is compatible with EthereumâĂŹs keystore file. Users can back up the address with its privatekey in it.

Signature

The Signature interface is used to provide applications with the functionality of a digital signature algorithm. A Signature object can be used to generate and verify digital signatures.

There are two phases, in order to use a Signature object for signing data :

- Initialization: with a private key, which initializes the signature for signing (see initSign() in the source code of go-nebulas).
- Signing of all input bytes.

A Signature object can recover the public key with a signature and the plain text that was signed (see function RecoverSignerFromSignature in go-nebulas). So just comparing the from address and the address derived from the public key can verify a transaction

> Similar to the Android Keystore, TPM, TEE and hardware low level security protection will be supported as a provider later.

NVM - Nebulas Virtual Machine

NVM is one of the most important components in Nebulas. As the name implies, it provides managed virtual machine execution environments for Smart Contract and Protocol Code.

go-nebulas now support two kinds of Virtual Machines:

- V8: Chrome V8
- LLVM: Low-Level Virtual Machine

Nebulas V8 Engine

In go-nebulas, we designed and implemented the Nebulas V8 Engine based on Chrome V8.

The Nebulas V8 Engine provides a high performance sandbox for Smart Contract execution. It guarantees user deployed code is running in a managed environment, and prevents massive resource consumption on hosts. Owing to the use of Chrome V8, JavaScript and TypeScript are first-class languages for Nebulas Smart Contracts. Anyone familiar with JavaScript or TypeScript can write their own Smart Contract and run it in Nebulas V8.

The following content is an example of Smart Contract written in JavaScript:

```
"use strict";

var BankVaultContract = function() {
    LocalContractStorage.defineMapProperty(this, "bankVault");
};

// save value to contract, only after height of block, users can␣
↪takeout
BankVaultContract.prototype = {
    init:function() {},
    save:function(height) {
        var deposit = this.bankVault.get(Blockchain.transaction.
↪from);
        var value = new BigNumber(Blockchain.transaction.value);
        if (deposit != null && deposit.balance.length > 0) {
            var balance = new BigNumber(deposit.balance);
            value = value.plus(balance);
        }
        var content = {
            balance:value.toString(),
            height:Blockchain.block.height + height
        };
        this.bankVault.put(Blockchain.transaction.from, content);
    },
    takeout:function(amount) {
        var deposit = this.bankVault.get(Blockchain.transaction.
↪from);
        if (deposit == null) {
            return 0;
        }
        if (Blockchain.block.height < deposit.height) {
            return 0;
        }
        var balance = new BigNumber(deposit.balance);
        var value = new BigNumber(amount);
        if (balance.lessThan(value)) {
            return 0;
        }
        var result = Blockchain.transfer(Blockchain.transaction.
↪from, value);
        if (result > 0) {
            deposit.balance = balance.dividedBy(value).toString();
            this.bankVault.put(Blockchain.transaction.from,␣
↪deposit);
        }
        return result;
    }
};

module.exports = BankVaultContract;
```

For more information about smart contracts in Nebulas, please go to Smart Contract.

For more information about the design of the Nebulas V8 Engine, please go to Nebulas V8 Engine.

LLVM

TBD.

Permission Control in Smart Contract

What Is Permission Control Of Smart Contract

The permission control of a smart contract refers to whether the contract caller has permission to invoke a given function in the contract. There are two types of permission control: owner permission control, and other permission control.

Owner permissions control: Only the creator of the contract can call this method, other callers can not call the method.

Other permission control: The contract method can be invoked if the contract developer defines a conditional caller according to the contract logic. Otherwise, it cannot be invoked.

Owner Permission Control

If you want to specify an owner for a small contract and wish that some functions could only be called by the owner and no one else, you can use following lines of code in your smart contract.

```
"use strict";
var onlyOwnerContract = function () {
    LocalContractStorage.defineProperty(this, "owner");
};
onlyOwnerContract.prototype = {
  init: function() {
          this.owner=Blockchain.transaction.from;
  },
  onlyOwnerFunction: function(){
          if(this.owner==Blockchain.transaction.from){
              //your smart contract code
              return true;
          }else{
              return false;
          }
  }
};
module.exports = BankVaultContract;
```

Explanation:

The function init is only called once when the contract is deployed, so it is there that you can specify the owner of the contract.The onlyOwnerFunctiuon ensures that the function is called by the owner of contract.

Other Permission Control

In your smart contract, if you needed to specify other permission control, for example, if you needed to verify its transaction value, you could write it the following way.

```
'use strict';
var Mixin = function () {};
Mixin.UNPAYABLE = function () {
   if (Blockchain.transaction.value.lt(0)) {
       return true;
   }
   return false;
};
Mixin.PAYABLE = function () {
   if (Blockchain.transaction.value.gt(0)) {
       return true;
   }
   return false;
};
Mixin.POSITIVE = function () {
   console.log("POSITIVE");
   return true;
};
Mixin.UNPOSITIVE = function () {
   console.log("UNPOSITIVE");
   return false;
};
Mixin.decorator = function () {
   var funcs = arguments;
```

```
    if (funcs.length < 1) {
        throw new Error("mixin decorator need parameters");
    }
    return function () {
        for (var i = 0; i < funcs.length - 1; i ++) {
            var func = funcs[i];
            if (typeof func !== "function" || !func()) {
                throw new Error("mixin decorator failure");
            }
        }
        var exeFunc = funcs[funcs.length - 1];
        if (typeof exeFunc === "function") {
            exeFunc.apply(this, arguments);
        } else {
            throw new Error("mixin decorator need an executable␣
↪method");
        }
    };
};
var SampleContract = function () {
};
SampleContract.prototype = {
    init: function () {
    },
    unpayable: function () {
        console.log("contract function unpayable:", arguments);
    },
    payable: Mixin.decorator(Mixin.PAYABLE, function () {
        console.log("contract function payable:",arguments);
    }),
    contract1: Mixin.decorator(Mixin.POSITIVE, function (arg) {
        console.log("contract1 function:", arg);
    }),
    contract2: Mixin.decorator(Mixin.UNPOSITIVE, function (arg) {
        console.log("contract2 function:", arg);
    }),
    contract3: Mixin.decorator(Mixin.PAYABLE, Mixin.POSITIVE,␣
↪function (arg) {
        console.log("contract3 function:", arg);
    }),
    contract4: Mixin.decorator(Mixin.PAYABLE, Mixin.UNPOSITIVE,␣
↪function (arg) {
        console.log("contract4 function:", arg);
    })
};
module.exports = SampleContract;
```

Explanation:

Mixin.UNPAYABLE,Mixin.PAYABLE,Mixin.POSITIVE ,Mixin.UNPOSITIVE are permission control functionãĂĆThe permission control function is as follows:

- Mixin.UNPAYABLE: check the transaction sent value, if value is less than 0 return true, otherwise return false
- Mixin.PAYABLE : check the transaction sent value, if value is greater than 0 return true, otherwise return false
- Mixin.UNPOSITIVE ïijŽoutput log UNPOSITIVE
- Mixin.POSITIVE ïijŽoutput log POSITIVE

Implement permission control in Mixin.decoratorïijŽ

- check arguments: if (funcs.length < 1)
- invoke permission control function: if (typeof func !== "function" || !func())
- if permission control function success ,invoke other function: var exeFunc = funcs[funcs.length - 1]

Permission control tests in smart contracts are as follows:

- The permission control function of the contract1 is Mixin.POSITIVE. If the permission check passes, the output is printed, otherwise an error is thrown by the permission check function.

```
    contract1: Mixin.decorator(Mixin.POSITIVE, function (arg)
↪{
        console.log("contract1 function:", arg);
    })
```

- The permission control function of the contract2 is Mixin.UNPOSITIVE. If the permission check passes, the output is printed, otherwise an error is thrown by the permission check function.

```
    contract2: Mixin.decorator(Mixin.UNPOSITIVE, function␣
↪(arg) {
              console.log("contract2 function:", arg);
    })
```

- The permission control function of the contract3 is Mixin.PAYABLE, Mixin.POSITIVE. If the permission check passes, the output is printed, otherwise an error is thrown by the permission check function.

```
    contract3: Mixin.decorator(Mixin.PAYABLE, Mixin.POSITIVE,␣
↪function (arg) {
              console.log("contract3 function:", arg);
    })
```

- The permission control function of the contract4 is Mixin.PAYABLE, Mixin.UNPOSITIVE. If the permission check passes, the output is printed, otherwise an error is thrown by the permission check function.

```
        contract4: Mixin.decorator(Mixin.PAYABLE, Mixin.
↪UNPOSITIVE, function (arg) {
                console.log("contract4 function:", arg);
        })
```

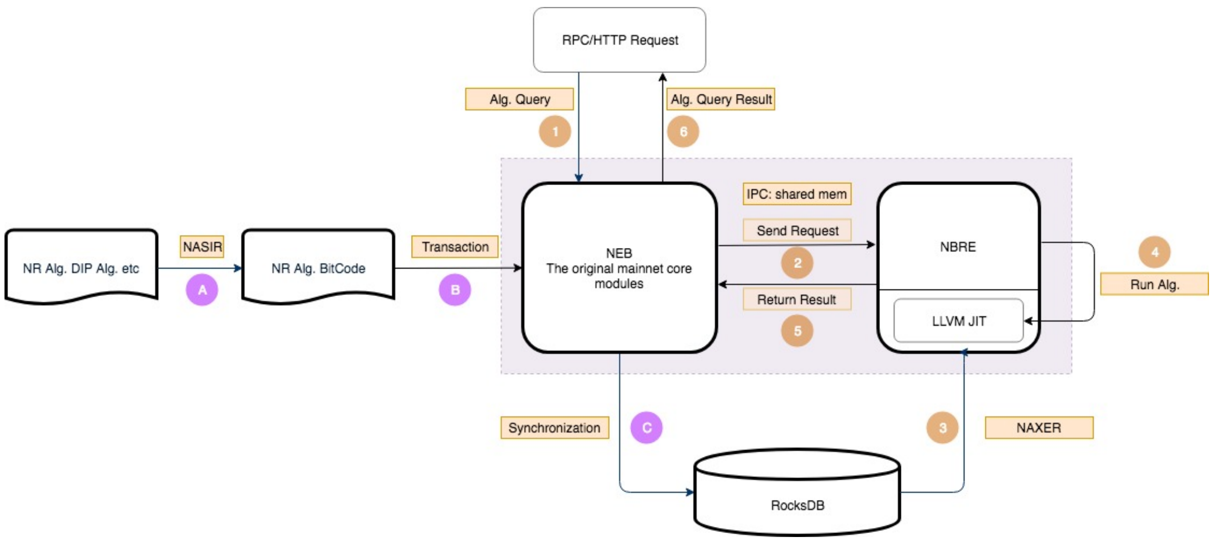
Tips:

With reference to the above example, the developer needs only three steps in order to implement other permission controls:

- Implement permission control functions.
- Implement the decorator function, and the permission check is completed by the conditional statement if (typeof func !== "function" || !func()).
- Refer to the contract1 function to implement other permission control.

NBRE Design Doc

NBRE (Nebulas Runtiome Environment) is the Nebulas chain execution environment. Its framework is shown as follows.



NBRE contains two main processes, which provide the methods how to update algorithms and how to execute algorithms.

The updating process provides how to upload algorithms and core protocols. It includes the following steps:

1. The algorithms are implemented with the languages supported by LLVM. Then, their codes are handled by the NASIR tool, which are translated to bitcode.
2. The bitcode streams are coded with base64, which are translated to payload of transaction data. The transaction data is uploaded to the online chain.
3. After that, the transaction data will be packed and varified. Then, the related bitcode will stored into the RocksDB.

The execution process exhibits the processes from request to results. The corresponding details are as follows.

1. User appries for algorithm call requests with the forms of RPC or RESful API.
2. After receiving the request, the core NEB forward it to NBRE.
3. NBRE starts JIT and loads the algorithm code into JIT.
4. The JIT executes the algorithm with specified parameters and the invoking method, and returns the execution result.
5. NBRE returns the execution result to NEB through IPC.
6. NEB returns the result to the user.

IPC

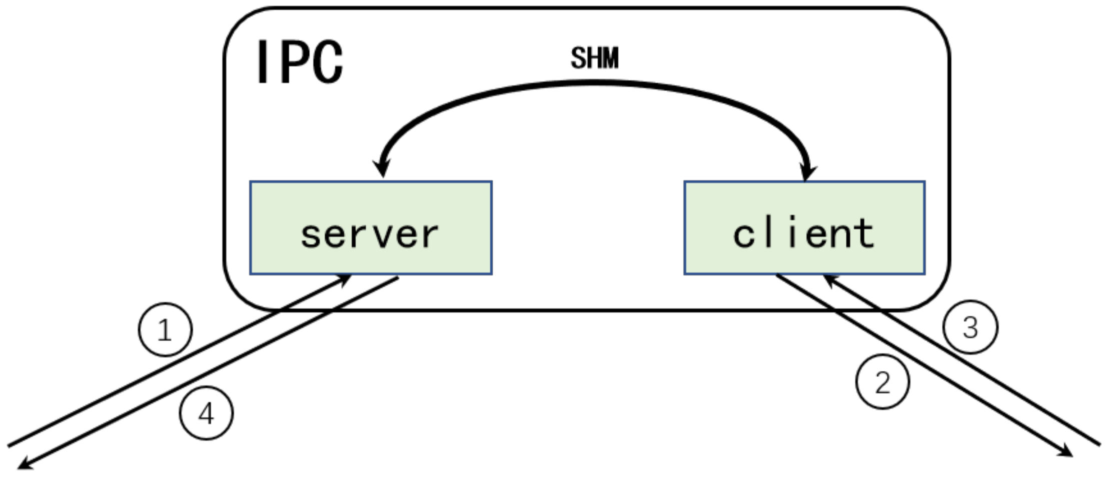
IPC is the messenger for NEB and NBRE interaction.

Features

IPC adopts shared memoty to communicate between NEB and NBRE to improve performance. There are two sub-threads, a server and a client, inside IPC. The server listens for the NEB request, and the client listens for the NBRE result. Also, there is communication interaction between the two threads.

Framework

The framework of IPC is shown as below.



1. NEB calls a function, and the server receives the request and sends it to the client.
2. The client sends the request to NBRE.
3. NBRE runs the corresponding program and returns the result to the client, the client sends the result to the server.

4. The server returns the result to the NEB.

JIT

JIT is a concurrent virtual machine based on LLVM, which runs ir programs providing algorithms and interfaces for NBRE. It is the key of the dynamic update for NBRE.

Features

Dynamic update

The dynamic update in NBRE contains two respects: - NBRE's own dynamic update - NBRE's new feature interfaces

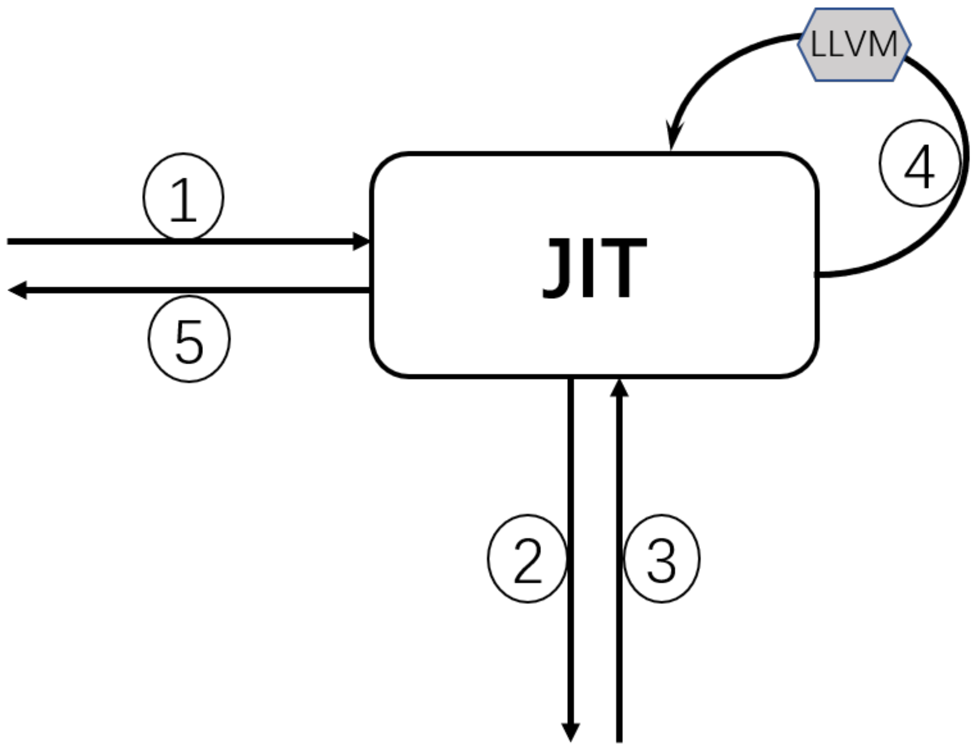
NBRE's updates are performed by adding algorithms and interface programs to the database. When a new function is updated or called, the corresponding program will be loaded into the JIT in the database.

Concurrent virtual machine

To improve performace, JIT is implemented based on a concurrent virtual machine mechanism. When one interface is called, the JIT first queries whether the corresponding program has been loaded. If the programs is loaded, sets its execution count to be 1800; otherwise, loads the program from database and sets its execution count to be 1801. Then runs the corresponding progrm. At regular intervals, the JIT decrements the corresponding count of each loaded function by one and releases the program with a count when its count less than zero.

Framework

The JIT framework is shown as below.



1. One interface is requested from outside.
2. JIT queries the corresponding function program from the database.
3. JIT loads the corresponding program.
4. Runs the program.
5. Returns the result.

Joining the Blockchain

Intent on joining the Nebulas Blockchain? Check out the following:

How to Join the Nebulas Mainnet

Introduction

The Nebulas Mainnet 3.0.0 (Nebulas Voyager) has been released. This tutorial will teach you how to join and work with the Nebulas Mainnet.

https://github.com/nebulasio/go-nebulas/tree/v3.0.0

Build

The Nebulas Mainnet‘s executable file and dependant libraries need to be built first. Several important modules are highlighted below:

- **NEB:** The main process of the Nebulas Mainnet.

  Details of building the modules can be found in tutorials.

Configuration

The Mainnet configuration files are in folder `mainnet/conf`, including

genesis.conf

All configurable information about genesis block is defined in genesis.conf, including

- **meta.chain_id:** chain identity
- **consensus.dpos.dynasty:** the initial dynasty of validators
- **token_distribution:** the initial allocation of tokens

  Attention: DO NOT change the genesis.conf.

config.conf

All configurable information about runtime is defined in config.conf.

Please check the `template.conf` to find more details about the runtime configuration.

Tips: the official seed node info is as follows,

```
seed:["/ip4/52.76.103.107/tcp/8680/ipfs/
↪Qmbi1NVTYHkeuST2wS3B3aHiTLHDajHZpoZk5EDpAXt9H2","/ip4/52.56.55.
↪238/tcp/8680/ipfs/QmVy9AHxBpd1iTvECDR7fvdZnqXeDhnxkZJrKsyuHNYKAh",
↪"/ip4/34.198.52.191/tcp/8680/ipfs/
↪QmQK7W8wrByJ6So7rf84sZzKBxMYmc1i4a7JZsne93ysz5"]
```

Miner config

Nodes can participate in mining and share rewards after signing up for mining. The miner node needs to turn on the mine switch and configure both the miner address and reward address(coinbase).

miner config example:

```
chain {
  # mainnet chainID
  chain_id: 1
  # mainnet datadir, should be different with private chain
  datadir: "mainnet/data.db"
  keydir: "keydir"
  # mainnet genesis.conf
  genesis: "mainnet/conf/genesis.conf"
  # mainnet dynasty.conf
  dynasty: "mainnet/conf/dynasty.conf"

  # start mine
  start_mine: true
  # receive the mining award, must change to your address
  coinbase: "n1XkoVVjswb5Gek3rRufqjKNpwrDdsnQ7Hq"
  # block signature address, needs to be placed in the node's␣
↪configuration `keydir`. Also make sure that the address is the␣
↪node address at the time of registration
  miner: "n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE"
  #
  passphrase: "passphrase"

  signature_ciphers: ["ECC_SECP256K1"]
}
```

Synchronization

Since Nebulas mainnet is running there for certain period of time, it will take quite some time to sync all the mainnet data from scratch.

For developers‘ convenience, we provided a offline data package, which already includes the data of more than 1 million blocks, you can download the package directly by following either link below (choose whichever is faster for you):

- `download from AWS s3`
- `download from Aliyun oss`

  Please note that, the data package should be put under the same path of "datadir" as specified in your `config.conf` file.

API List

Main Endpoint:

- GetNebState : returns nebulas client info.
- GetAccountState: returns the account balance and nonce.
- Call: execute smart contract local, don‘t submit on chain.
- SendRawTransaction: submit the signed transaction.
- GetTransactionReceipt: get transaction receipt info by tansaction hash.

More Nebulas APIs at RPC.

Tutorials

English

1. Installation (thanks Victor)
2. Sending a Transaction (thanks Victor)
3. Writing Smart Contract in JavaScript (thanks otto)
4. Introducing Smart Contract Storage (thanks Victor)
5. Interacting with Nebulas by RPC API (thanks Victor)

Contribution

Feel free to join the Nebulas Mainnet. If you have found something wrong, please submit an issue or submit a pull request to let us know, and we will add your name and URL to this page as soon as possible.

How to Join the Nebulas Testnet

Introduction

We are glad to release the Nebulas Testnet. It simulates the Nebulas network and NVM, and allows developers to interact with Nebulas without paying the cost of gas.

https://github.com/nebulasio/go-nebulas/tree/testnet

Build

The Nebulas Testnet's executable file and dependant libraries need to be built first. Several important modules are highlighted below:

- **NEB:** The main process of the Nebulas Testnet. `NEB` and `NBRE` run in standalone processes, and communicate through IPC.

Details of building the modules can be found in tutorials.

Configuration

The testnet configuration files are in the folder `testnet/conf` under `testnet` branch, including:

genesis.conf

All configurable information about the genesis block is defined in genesis.conf, such as:

- **meta.chain_id:** chain identity.
- **consensus.dpos.dynasty:** the initial dynasty of validators.
- **token_distribution:** the initial allocation of tokens.

  Attention: DO NOT change the genesis.conf.

config.conf

All configurable information about the runtime is defined in config.conf.

Please check the `template.conf` to find more details about the runtime configuration.

Tips: the official seed node info is as below,

```
seed:["/ip4/47.92.203.173/tcp/8680/ipfs/
↪QmfSJ7JUnCEDP6LFyKkBUbpuDMETPbqMVZvPQy4keeyBDP","/ip4/47.89.180.5/
↪tcp/8680/ipfs/QmTmnd5KXm4UFUquAJEGdrwj1cbJCHsTfPWAp5aKrKoRJK"]
```

Miner config

Nodes can participate in mining and share rewards after signing up for mining. The miner node needs to turn on the mine switch and configure both the miner address and reward address(coinbase).

miner config example:

```
chain {
  # testnet chainID
  chain_id: 1001
  # testnet datadir, should be different with private chain
  datadir: "testnet/data.db"
  keydir: "keydir"
  # testnet genesis.conf
  genesis: "testnet/conf/genesis.conf"
  # testnet dynasty.conf
  dynasty: "testnet/conf/dynasty.conf"

  # start mine
  start_mine: true
  # receive the mining award, must change to your address
  coinbase: "n1XkoVVjswb5Gek3rRufqjKNpwrDdsnQ7Hq"
  # block signature address, needs to be placed in the node's␣
↪configuration `keydir`. Also make sure that the address is the␣
↪node address at the time of registration
  miner: "n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE"
  #
  passphrase: "passphrase"

  signature_ciphers: ["ECC_SECP256K1"]
}
```

Synchronization

Since Nebulas testnet is running there for certain period of time, it will take quite some time to sync all the testnet data from scratch.

For developers‘ convenience, we provided a offline data package, which already includes the data of more than 1.2 million blocks, you can download the package directly by following either link below (choose whichever is faster for you):

- `download from AWS s3`
- `download from Aliyun oss`

  Please note that, the data package should be put under the same path of “datadir“ as specified in your `config.conf` file.

API List

Test Endpoint:

- GetNebState : returns nebulas client info.
- GetAccountState: returns the account balance and nonce.
- Call: execute smart contract local, don‘t submit on chain.
- SendRawTransaction: submit the signed transaction.
- GetTransactionReceipt: get transaction receipt info by tansaction hash.

More Nebulas APIs at RPC.

Claim Tokens

Each email can claim tokens every day here.

Tutorials

1. Installation (thanks Victor)
2. Sending a Transaction (thanks Victor)
3. Writing Smart Contract in JavaScript (thanks otto)

4. Introducing Smart Contract Storage (thanks Victor)
5. Interacting with Nebulas by RPC API (thanks Victor)

Contributing

Feel free to join Nebulas Testnet. If you did find something wrong, please submit an issue or submit a pull request to let us know, we will add your name and url to this page as soon as possible.

Configuration Files

There are four types of configuration files in Nebulas.

- Normal node.
- Miner node.(Miner - related configuration is increased relative to normal nodes)
- Super node.(Some connection limits are higher than normal nodes)
- Sign node. (Do not synchronize information with any node, only do signature and unlock)

Normal node

```
network {
  seed: ["/ip4/13.251.33.39/tcp/8680/ipfs/
↪QmVm5CECJdPAHmzJWN2X7tP335L5LguGb9QLQ78riA9gw3"]
  listen: ["0.0.0.0:8680"]
  private_key: "conf/networkkey"
}

chain {
  chain_id:1
  datadir: "data.db"
  keydir: "keydir"
  genesis: "conf/genesis.conf"
  signature_ciphers: ["ECC_SECP256K1"]
}

rpc {
    rpc_listen: ["0.0.0.0:8784"]
    http_listen: ["0.0.0.0:8785"]
    http_module: ["api","admin"]
    connection_limits:200
    http_limits:200
}

app {
    log_level: "debug"
    log_file: "logs"
    enable_crash_report: true
}

stats {
    enable_metrics: false
}
```

Miner node

```
network {
  seed: ["/ip4/13.251.33.39/tcp/8680/ipfs/
↪QmVm5CECJdPAHmzJWN2X7tP335L5LguGb9QLQ78riA9gw3"]
  listen: ["0.0.0.0:8680"]
  private_key: "conf/networkkey"
}

chain {
  chain_id: 1
  datadir: "data.db"
  keydir: "keydir"
  genesis: "conf/genesis.conf"
  coinbase: "n1EzGmFsVepKduN1U5QFyhLqpzFvM9sRSmG"
  signature_ciphers: ["ECC_SECP256K1"]
  start_mine:true
  miner: "n1PxjEu9sa2nvk9SjSGtJA91nthogZ1FhgY"
  remote_sign_server: "127.0.0.1:8694"
  enable_remote_sign_server: true
}

rpc {
    rpc_listen: ["127.0.0.1:8684"]
    http_listen: ["0.0.0.0:8685"]
    http_module: ["api","admin"]
    connection_limits:200
    http_limits:200
}

app {
    log_level: "debug"
    log_file: "logs"
    enable_crash_report: true
}

stats {
    enable_metrics: false
}
```

Super node

```
network {
  seed: ["/ip4/13.251.33.39/tcp/8680/ipfs/
↪QmVm5CECJdPAHmzJWN2X7tP335L5LguGb9QLQ78riA9gw3"]
  listen: ["0.0.0.0:8680"]
  private_key: "conf/networkkey"
  stream_limits: 500
  reserved_stream_limits: 50
}

chain {
  chain_id:1
  datadir: "data.db"
  keydir: "keydir"
  genesis: "conf/genesis.conf"
  signature_ciphers: ["ECC_SECP256K1"]
}

rpc {
    rpc_listen: ["0.0.0.0:8684"]
    http_listen: ["0.0.0.0:8685"]
    http_module: ["api"]
    connection_limits:500
    http_limits:500
    http_cors: ["*"]
}

app {
    log_level: "debug"
    log_file: "logs"
    enable_crash_report: true
    pprof:{
        http_listen: "0.0.0.0:8888"
    }
}

stats {
    enable_metrics: false
}
```

Sign node

```
network {
  listen: ["0.0.0.0:8680"]
  private_key: "conf/networkkey"
}
```

```
chain {
  chain_id:0
  datadir: "data.db"
  keydir: "keydir"
  genesis: "conf/genesis.conf"
  signature_ciphers: ["ECC_SECP256K1"]
}

rpc {
    rpc_listen: ["0.0.0.0:8684"]
    http_listen: ["127.0.0.1:8685"]
    http_module: ["admin"]
    connection_limits:200
    http_limits:200
}

app {
    log_level: "debug"
    log_file: "logs"
    enable_crash_report: true
    pprof:{
        http_listen: "127.0.0.1:8888"
    }
}

stats {
    enable_metrics: false
}
```

Node Environment

Introduction

We are glad to release Nebulas Mainnet here. Please join and enjoy Nebulas Mainnet.

https://github.com/nebulasio/go-nebulas/tree/master

Hardware configuration

The nodes of the nebulas have certain requirements for machine performance. We recommend the performance of the machine has the following requirements:

```
CPU: >= 4cores(recommand 8 cores)
RAM: >= 16G
Disk: >= 600G SSD
```

Environment

Node Installation Tutorial - review the Nebulas Technical Documentation: Nebulas 101 - 01 Compile Installation.

ItâĂŹs recommended to build and deploy nodes via docker:

- Install docker and docker-compose
- Execute the following docker command via root

```
sudo docker-compose build

sudo docker-compose up -d
```

```
System: Ubuntu 18.04(recommand), other Linux is ok.
NTP: Ensure machine time synchronization
```

NTP

Install the NTP service to keep system time in sync.

Ubuntu install steps:

```
#install
sudo apt install ntp
#start ntp service
sudo service ntp restart
# check ntp status
ntpq -p
```

Centos install steps:

```
#install
sudo yum install ntp
#start ntp service
sudo service ntp restart
# check ntp status
ntpq -p
```

Contribution

Feel free to join Nebulas Mainnet. If you did find something wrong, please submit a issue or submit a pull request to let us know, we will add your name and url to this page soon.

Tutorials

If you are a developer, here is all you need to dive into Nebulas. You can also visit the developer page to check all development tools.

Go-Nebulas

- Join the Testnet
- Join the Mainnet
- Explorer

Tutorials (Nebulas 101):

01 Compile and Install Nebulas

The current version of Nebulas Mainnet is 3.0.0, which is called Nebulas Nova.

Nebulas Nova aims to discover the value of blockchain data, and it also means the future of collaboration.

Check our Youtube Introduction for more details.

You can download the Nebulas source code to compile the private chain locally.

To learn about Nebulas, please read the Nebulas Non-Technical White Paper.

To learn about the technology, please read the Nebulas Technical White Paper and the Nebulas github code.

> At present, Nebulas can only run on Mac and Linux at this stage. The Windows version will be coming later.

Golang Environment

Nebulas is implemented in Golang and C++.

Mac OSX

Homebrew is recommended for installing golang on Mac.

```
# install
brew install go
```

> Note: GOPATH is a local golang working directory which could be decided by youself. After GOPATH is configured, your go projects need to be placed in GOPATH directory.

Linux

```
# download
wget https://dl.google.com/go/go1.14.1.linux-amd64.tar.gz

# extract
tar -C /usr/local -xzf go1.14.1.linux-amd64.tar.gz

# environment variables
export PATH=$PATH:/usr/local/go/bin
```

Compile Nebulas

Download

Clone source code with the following commands.

```
# enter workspace
cd /path/to/workspace

# download
git clone https://github.com/nebulasio/go-nebulas.git

# enter repository
cd go-nebulas

# master branch is most stable
git checkout master
```

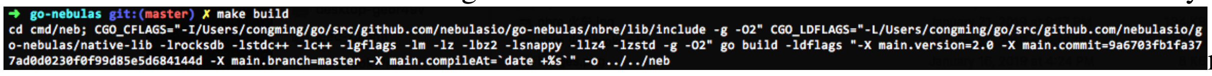
Build NEB

- Set up runtime environment

```
cd /path/to/workspace/go-nebulas
source setup.sh
```

- Build NEB You can now build the executable for Nebulas:

```
cd /path/to/workspace/go-nebulas
make build
```

Once the building is completeïijŇthere will be an executable file `neb` generated under the root directory. 

```
➜ go-nebulas git:(master) ✗ make build
cd cmd/neb; CGO_CFLAGS="-I/Users/congming/go/src/github.com/nebulasio/go-nebulas/nbre/lib/include -g -O2" CGO_LDFLAGS="-L/Users/congming/go/src/github.com/nebulasio/g
o-nebulas/native-lib -lrocksdb -lstdc++ -lc++ -lgflags -lm -lz -lbz2 -lsnappy -llz4 -lzstd -g -O2" go build -ldflags "-X main.version=2.0 -X main.commit=9a6703fb1fa37
7ad0d0230f0f99d85e5d684144d -X main.branch=master -X main.compileAt=`date +%s`" -o ../../neb
```

make build

Start NEB

Genesis Block

Before launching a new Nebulas chain, we have to define the configuration of genesis block.

Genesis Block Configuration

```
# Neb genesis text file. Scheme is defined in core/pb/genesis.proto.

meta {
    # Chain identity
  chain_id: 1
}

consensus {
    dpos {
    # Initial dynasty, including all initial miners
        dynasty: [
            [ miner address ],
            ...
        ]
    }
}

# Pre-allocation of initial tokens
token_distribution [
    {
        address: [ allocation address ]
        value: [ amount of allocation tokens ]
    },
    ...
]
```

An example genesis.conf is located in `conf/default/genesis.conf`.

Configuration

Before getting a neb node started, we have to define the configuration of this node.

Neb Node Configuration

```
# Neb configuration text file. Scheme is defined in neblet/pb/
↪config.proto:Config.
```

```
# Network Configuration
network {
    # For the first node in a new Nebulas chain, `seed` is not need.
    # Otherwise, every node need some seed nodes to introduce it␣
↪into the Nebulas chain.
    # seed: ["/ip4/127.0.0.1/tcp/8680/ipfs/
↪QmP7HDFcYmJL12Ez4ZNVCKjKedfE7f48f1LAkUc3Whz4jP"]

    # P2p network service host. support mutiple ip and ports.
    listen: ["0.0.0.0:8680"]

    # The private key is used to generate a node ID. If you don't␣
↪use the private key, the node will generate a new node ID.
    # private_key: "conf/network/id_ed25519"
}

# Chain Configuration
chain {
    # Network chain ID
    chain_id: 100

    # Database storage location
    datadir: "data.db"

    # Accounts' keystore files location
    keydir: "keydir"

    # The genesis block configuration
    genesis: "conf/default/genesis.conf"

    # Signature algorithm
    signature_ciphers: ["ECC_SECP256K1"]

    # Miner address
    miner: "n1SAQy3ix1pZj8MPzNeVqpAmu1nCVqb5w8c"

    # Coinbase address, all mining reward received by the above␣
↪miner will be send to this address
    coinbase: "n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE"

    # The passphrase to miner's keystore file
    passphrase: "passphrase"
}

# API Configuration
rpc {
    # GRPC API port
    rpc_listen: ["127.0.0.1:8684"]

    # HTTP API port
    http_listen: ["127.0.0.1:8685"]

    # The module opened
    http_module: ["api", "admin"]
}

# Log Configuration
app {
    # Log level: [debug, info, warn, error, fatal]
    log_level: "info"

    # Log location
    log_file: "logs"

    # Open crash log
    enable_crash_report: false
}

# NBRE configurations
nbre {
    # The root directory of NBRE, where the NBRE libraries located
    root_dir: "nbre"

    # NBRE log folder path
    log_dir: "conf/nbre/logs"

    # NBRE db location
    data_dir: "conf/nbre/nbre.db"

    # NBRE binary location
    nbre_path: "nbre/bin/nbre"

    # Administrator address used to submit tx and authorize␣
↪specific account
    # with the right of IR submission. For more details, please␣
↪check the NBRE
    # related documents.
    admin_address: "n1S9RrRPC46T9byYBS868YuZgzqGuiPCY1m"

    # Height when the DIP takes effect
    start_height: 2307000

    # NEB and NBRE inter-process communication socket
    ipc_listen: "127.0.0.1"
    ipc_port: 8688
}

# Metrics Configuration
stats {
    # Open node metrics
    enable_metrics: false

    # Influxdb configuration
    influxdb: {
        host: "http://localhost:8086"
        db: "nebulas"
        user: "admin"
        password: "admin"
    }
}
```

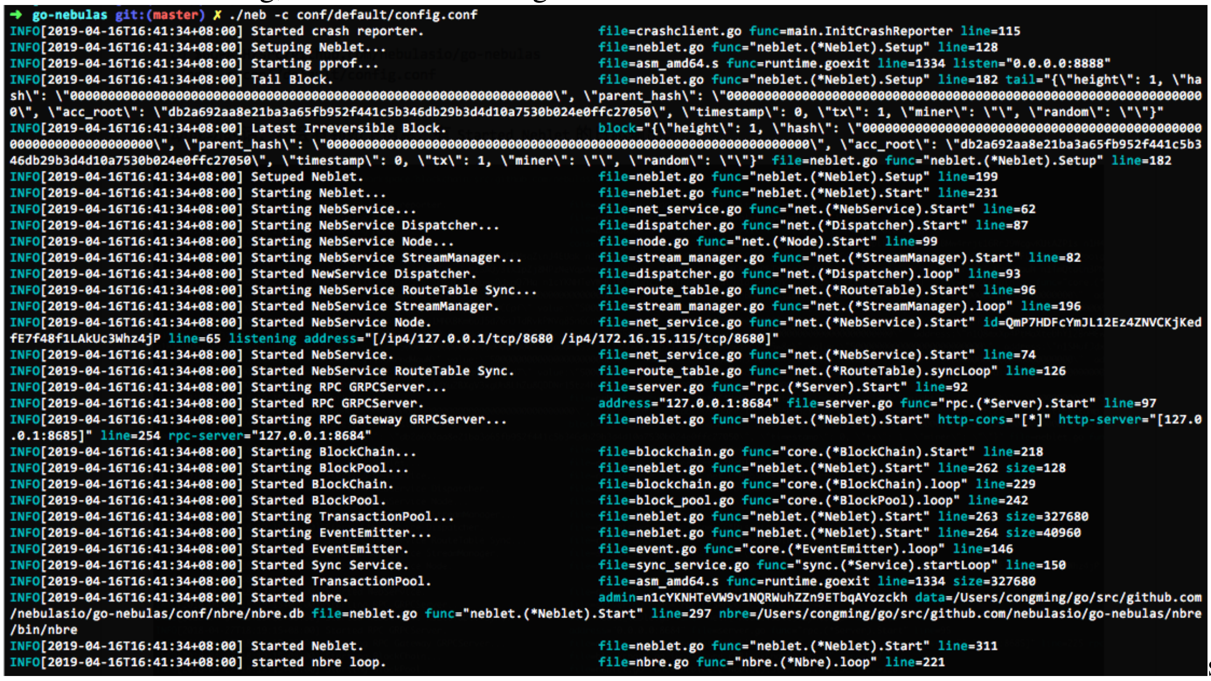
A lot of examples can be found in `$GOPATH/src/github.com/nebulasio/go-nebulas/conf/`

Run Nodes

> The Nebulas chain you are running at this point is private and is different with official Testnet and Mainnet.

Start your first Nebulas node with the following commands.

```
cd $GOPATH/src/github.com/nebulasio/go-nebulas
./neb -c conf/default/config.conf
```

After starting, the following should be visible in the terminal:


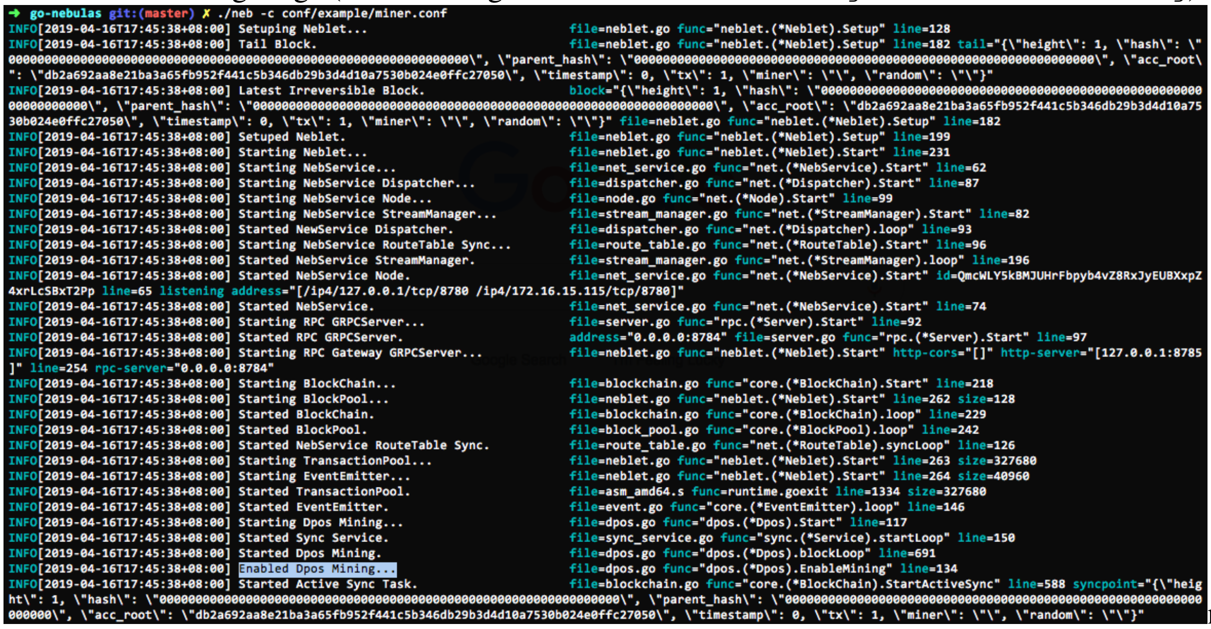
```
➜ go-nebulas git:(master) ✗ ./neb -c conf/default/config.conf
INFO[2019-04-16T16:41:34+08:00] Started crash reporter.                       file=crashclient.go func=main.InitCrashReporter line=115
INFO[2019-04-16T16:41:34+08:00] Setuping Neblet...                            file=neblet.go func="neblet.(*Neblet).Setup" line=128
INFO[2019-04-16T16:41:34+08:00] Starting pprof...                             file=asm_amd64.s func=runtime.goexit line=1334 listen="0.0.0.0:8888"
INFO[2019-04-16T16:41:34+08:00] Tail Block.                                   file=neblet.go func="neblet.(*Neblet).Setup" line=182 tail="{\"height\": 1, \"ha
sh\": \"0000000000000000000000000000000000000000000000000000000000000000\", \"parent_hash\": \"000000000000000000000000000000000000000000000000000000000000000
0\", \"acc_root\": \"db2a692aa8e21ba3a65fb952f441c5b346db29b3d4d10a7530b024e0ffc27050\", \"timestamp\": 0, \"tx\": 1, \"miner\": \"\", \"random\": \"\"}"
INFO[2019-04-16T16:41:34+08:00] Latest Irreversible Block.                    block="{\"height\": 1, \"hash\": \"00000000000000000000000000000000000000000000
00000000000000000000\", \"parent_hash\": \"0000000000000000000000000000000000000000000000000000000000000000\", \"acc_root\": \"db2a692aa8e21ba3a65fb952f441c5b3
46db29b3d4d10a7530b024e0ffc27050\", \"timestamp\": 0, \"tx\": 1, \"miner\": \"\", \"random\": \"\"}" file=neblet.go func="neblet.(*Neblet).Setup" line=182
INFO[2019-04-16T16:41:34+08:00] Setuped Neblet.                               file=neblet.go func="neblet.(*Neblet).Setup" line=199
INFO[2019-04-16T16:41:34+08:00] Starting Neblet...                            file=neblet.go func="neblet.(*Neblet).Start" line=231
INFO[2019-04-16T16:41:34+08:00] Starting NebService...                        file=net_service.go func="net.(*NebService).Start" line=62
INFO[2019-04-16T16:41:34+08:00] Starting NebService Dispatcher...             file=dispatcher.go func="net.(*Dispatcher).Start" line=87
INFO[2019-04-16T16:41:34+08:00] Starting NebService Node...                   file=node.go func="net.(*Node).Start" line=99
INFO[2019-04-16T16:41:34+08:00] Starting NebService StreamManager...          file=stream_manager.go func="net.(*StreamManager).Start" line=82
INFO[2019-04-16T16:41:34+08:00] Started NewService Dispatcher.                file=dispatcher.go func="net.(*Dispatcher).loop" line=93
INFO[2019-04-16T16:41:34+08:00] Starting NebService RouteTable Sync...        file=route_table.go func="net.(*RouteTable).Start" line=96
INFO[2019-04-16T16:41:34+08:00] Started NebService StreamManager.             file=stream_manager.go func="net.(*StreamManager).loop" line=196
INFO[2019-04-16T16:41:34+08:00] Started NebService Node.                      file=net_service.go func="net.(*NebService).Start" id=QmP7HDFcYmJL12Ez4ZNVCKjKed
fE7f48f1LAkUc3Whz4jP line=65 listening address="[/ip4/127.0.0.1/tcp/8680 /ip4/172.16.15.115/tcp/8680]"
INFO[2019-04-16T16:41:34+08:00] Started NebService.                           file=net_service.go func="net.(*NebService).Start" line=74
INFO[2019-04-16T16:41:34+08:00] Started NebService RouteTable Sync.           file=route_table.go func="net.(*RouteTable).syncLoop" line=126
INFO[2019-04-16T16:41:34+08:00] Starting RPC GRPCServer...                    file=server.go func="rpc.(*Server).Start" line=92
INFO[2019-04-16T16:41:34+08:00] Started RPC GRPCServer.                       address="127.0.0.1:8684" file=server.go func="rpc.(*Server).Start" line=97
INFO[2019-04-16T16:41:34+08:00] Starting RPC Gateway GRPCServer...            file=neblet.go func="neblet.(*Neblet).Start" http-cors="[*]" http-server="[127.0
.0.1:8685]" line=254 rpc-server="127.0.0.1:8684"
INFO[2019-04-16T16:41:34+08:00] Starting BlockChain...                        file=blockchain.go func="core.(*BlockChain).Start" line=218
INFO[2019-04-16T16:41:34+08:00] Starting BlockPool...                         file=neblet.go func="neblet.(*Neblet).Start" line=262 size=128
INFO[2019-04-16T16:41:34+08:00] Started BlockChain.                           file=blockchain.go func="core.(*BlockChain).loop" line=229
INFO[2019-04-16T16:41:34+08:00] Started BlockPool.                            file=block_pool.go func="core.(*BlockPool).loop" line=242
INFO[2019-04-16T16:41:34+08:00] Starting TransactionPool...                   file=neblet.go func="neblet.(*Neblet).Start" line=263 size=327680
INFO[2019-04-16T16:41:34+08:00] Starting EventEmitter...                      file=neblet.go func="neblet.(*Neblet).Start" line=264 size=40960
INFO[2019-04-16T16:41:34+08:00] Started EventEmitter.                         file=event.go func="core.(*EventEmitter).loop" line=146
INFO[2019-04-16T16:41:34+08:00] Started Sync Service.                         file=sync_service.go func="sync.(*Service).startLoop" line=150
INFO[2019-04-16T16:41:34+08:00] Started TransactionPool.                      file=asm_amd64.s func=runtime.goexit line=1334 size=327680
INFO[2019-04-16T16:41:34+08:00] Started nbre.                                 admin=n1cYKNHTeVW9v1NQRWuhZZn9ETbqAYozckh data=/Users/congming/go/src/github.com
/nebulasio/go-nebulas/conf/nbre/nbre.db file=neblet.go func="neblet.(*Neblet).Start" line=297 nbre=/Users/congming/go/src/github.com/nebulasio/go-nebulas/nbre
/bin/nbre
INFO[2019-04-16T16:41:34+08:00] Started Neblet.                               file=neblet.go func="neblet.(*Neblet).Start" line=311
INFO[2019-04-16T16:41:34+08:00] started nbre loop.                            file=nbre.go func="nbre.(*Nbre).loop" line=221
```

seed node start

By default, the node using `conf/default/config.conf` won't mine new blocks. Start your first Nebulas mining node with another commands.

```
cd $GOPATH/src/github.com/nebulasio/go-nebulas
./neb -c conf/example/miner.conf
```

After the node starts, if the connection with the seed node is successful, you can see the following log (detailed log can be found in: `logs/miner.1/neb.log`):



```
➜ go-nebulas git:(master) ✗ ./neb -c conf/example/miner.conf
INFO[2019-04-16T17:45:38+08:00] Setuping Neblet...                            file=neblet.go func="neblet.(*Neblet).Setup" line=128
INFO[2019-04-16T17:45:38+08:00] Tail Block.                                   file=neblet.go func="neblet.(*Neblet).Setup" line=182 tail="{\"height\": 1, \"hash\": \"
0000000000000000000000000000000000000000000000000000000000000000\", \"parent_hash\": \"0000000000000000000000000000000000000000000000000000000000000000\", \"acc_root\
": \"db2a692aa8e21ba3a65fb952f441c5b346db29b3d4d10a7530b024e0ffc27050\", \"timestamp\": 0, \"tx\": 1, \"miner\": \"\", \"random\": \"\"}"
INFO[2019-04-16T17:45:38+08:00] Latest Irreversible Block.                    block="{\"height\": 1, \"hash\": \"0000000000000000000000000000000000000000000000000000
00000000000\", \"parent_hash\": \"0000000000000000000000000000000000000000000000000000000000000000\", \"acc_root\": \"db2a692aa8e21ba3a65fb952f441c5b346db29b3d4d10a75
30b024e0ffc27050\", \"timestamp\": 0, \"tx\": 1, \"miner\": \"\", \"random\": \"\"}" file=neblet.go func="neblet.(*Neblet).Setup" line=182
INFO[2019-04-16T17:45:38+08:00] Setuped Neblet.                               file=neblet.go func="neblet.(*Neblet).Setup" line=199
INFO[2019-04-16T17:45:38+08:00] Starting Neblet...                            file=neblet.go func="neblet.(*Neblet).Start" line=231
INFO[2019-04-16T17:45:38+08:00] Starting NebService...                        file=net_service.go func="net.(*NebService).Start" line=62
INFO[2019-04-16T17:45:38+08:00] Starting NebService Dispatcher...             file=dispatcher.go func="net.(*Dispatcher).Start" line=87
INFO[2019-04-16T17:45:38+08:00] Starting NebService Node...                   file=node.go func="net.(*Node).Start" line=99
INFO[2019-04-16T17:45:38+08:00] Starting NebService StreamManager...          file=stream_manager.go func="net.(*StreamManager).Start" line=82
INFO[2019-04-16T17:45:38+08:00] Started NewService Dispatcher.                file=dispatcher.go func="net.(*Dispatcher).loop" line=93
INFO[2019-04-16T17:45:38+08:00] Starting NebService RouteTable Sync...        file=route_table.go func="net.(*RouteTable).Start" line=96
INFO[2019-04-16T17:45:38+08:00] Started NebService StreamManager.             file=stream_manager.go func="net.(*StreamManager).loop" line=196
INFO[2019-04-16T17:45:38+08:00] Started NebService Node.                      file=net_service.go func="net.(*NebService).Start" id=QmcWLY5kBMJUHrFbpyb4vZ8RxJyEUBXxpZ
4xrLcSBxT2Pp line=65 listening address="[/ip4/127.0.0.1/tcp/8780 /ip4/172.16.15.115/tcp/8780]"
INFO[2019-04-16T17:45:38+08:00] Started NebService.                           file=net_service.go func="net.(*NebService).Start" line=74
INFO[2019-04-16T17:45:38+08:00] Starting RPC GRPCServer...                    file=server.go func="rpc.(*Server).Start" line=92
INFO[2019-04-16T17:45:38+08:00] Started RPC GRPCServer.                       address="0.0.0.0:8784" file=server.go func="rpc.(*Server).Start" line=97
INFO[2019-04-16T17:45:38+08:00] Starting RPC Gateway GRPCServer...            file=neblet.go func="neblet.(*Neblet).Start" http-cors="[]" http-server="[127.0.0.1:8785
]" line=254 rpc-server="0.0.0.0:8784"
INFO[2019-04-16T17:45:38+08:00] Starting BlockChain...                        file=blockchain.go func="core.(*BlockChain).Start" line=218
INFO[2019-04-16T17:45:38+08:00] Starting BlockPool...                         file=neblet.go func="neblet.(*Neblet).Start" line=262 size=128
INFO[2019-04-16T17:45:38+08:00] Started BlockChain.                           file=blockchain.go func="core.(*BlockChain).loop" line=229
INFO[2019-04-16T17:45:38+08:00] Started BlockPool.                            file=block_pool.go func="core.(*BlockPool).loop" line=242
INFO[2019-04-16T17:45:38+08:00] Started NebService RouteTable Sync.           file=route_table.go func="net.(*RouteTable).syncLoop" line=126
INFO[2019-04-16T17:45:38+08:00] Starting TransactionPool...                   file=neblet.go func="neblet.(*Neblet).Start" line=263 size=327680
INFO[2019-04-16T17:45:38+08:00] Starting EventEmitter...                      file=neblet.go func="neblet.(*Neblet).Start" line=264 size=40960
INFO[2019-04-16T17:45:38+08:00] Started TransactionPool.                      file=asm_amd64.s func=runtime.goexit line=1334 size=327680
INFO[2019-04-16T17:45:38+08:00] Started EventEmitter.                         file=event.go func="core.(*EventEmitter).loop" line=146
INFO[2019-04-16T17:45:38+08:00] Starting Dpos Mining...                       file=dpos.go func="dpos.(*Dpos).Start" line=117
INFO[2019-04-16T17:45:38+08:00] Started Sync Service.                         file=sync_service.go func="sync.(*Service).startLoop" line=150
INFO[2019-04-16T17:45:38+08:00] Started Dpos Mining.                          file=dpos.go func="dpos.(*Dpos).blockLoop" line=691
INFO[2019-04-16T17:45:38+08:00] Enabled Dpos Mining...                        file=dpos.go func="dpos.(*Dpos).EnableMining" line=134
INFO[2019-04-16T17:45:38+08:00] Started Active Sync Task.                     file=blockchain.go func="core.(*BlockChain).StartActiveSync" line=588 syncpoint="{\"heig
ht\": 1, \"hash\": \"0000000000000000000000000000000000000000000000000000000000000000\", \"parent_hash\": \"0000000000000000000000000000000000000000000000000000000000
000000\", \"acc_root\": \"db2a692aa8e21ba3a65fb952f441c5b346db29b3d4d10a7530b024e0ffc27050\", \"timestamp\": 0, \"tx\": 1, \"miner\": \"\", \"random\": \"\"}"
```

node start

> Note: You can start many nodes locally. Please make sure the ports in your node configurations won't conflict with each other.

Next step: Tutorial 2

Sending Transactions on Nebulas

02 Sending Transactions on Nebulas

Youtube Tutorial

For this portion of the tutorial we will pick up where we left off in the Installation tutorial.

Nebulas provides three methods to send transactionsïijŽ

1. Sign & Send
2. Send with Passphrase
3. Unlock & Send

Here is an introduction to sending a transaction in Nebulas through the three methods above and verifying whether the transaction is successful.

Prepare Accounts

In Nebulas, each address represents an unique account.

Prepare two accounts: an address to send tokens (the sending address, called "from") and an address to receive the tokens (the receiving address, called "to").

The Sender

Here we will use the coinbase account in the `conf/example/miner.conf`, which is `n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE` as the sender. As the miner's coinbase account, it will receive some tokens as the mining reward. Then we could send these tokens to another account later.

The Receiver

Create a new wallet to receive the tokens.

```
$ ./neb account new
Your new account is locked with a passphrase. Please give a␣
↪passphrase. Do not forget this passphrase.
Passphrase:
Repeat passphrase:
Address: n1SQe5d1NKHYFMKtJ5sNHPsSPVavGzW71Wy
```

When you run this command you will have a different wallet address with `n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE`. Please use your generated address as the receiver.

The keystore file of the new wallet will be located in `$GOPATH/src/github.com/nebulasio/go-nebulas/keydir/`

Start the Nodes

Start Seed Node

Firstly, start a seed node as the first node in local private chain.

```
./neb -c conf/default/config.conf
```

Start Miner Node

Secondly, start a miner node connecting to the seed node. This node will generate new blocks in local private chain.

```
./neb -c conf/example/miner.conf
```

How long a new block will be minted?

In Nebulas PoD consensus algorithm, each miner will mint new block one by one every 15 seconds.

In current context, we have to wait for 315(=15*21) seconds to get a new block because there is only one miner among 21 miners defined in `conf/default/genesis.conf` working now.

Once a new block minted by the miner, the mining reward will be added to the coinbase wallet address used in `conf/example/miner.conf` which is `n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE`.

Interact with Nodes

Nebulas provides developers with HTTP API, gRPC API and CLI to interact with the running nodes. Here, we will share how to send a transaction in three methods with HTTP API (API Module | Admin Module).

The Nebulas HTTP Lisenter is defined in the node configuration. The default port of our seed node is `8685`.

At first, check the sender's balance before sending a transaction.

Check Account State

Fetch the state of sender's account `n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE` with `/v1/user/accountstate` in API Module using `curl`.

```
> curl -i -H Accept:application/json -X POST http://localhost:8685/
↪v1/user/accountstate -d '{"address":
↪"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE"}'

{
    "result": {
        "balance": "5000000000000000000000000",
        "nonce": "0",
        "type": 87,
        "height":"1",
        "pending":"0"
    }
}
```

Note Type is used to check if this account is a smart contract account. `88` represents a smart contract account and `87` a non-contract account. Height is used to indicate the current height of the blockchain when the API is called. Pending is used to show how many pending transactions your address has in the Tx Pool.

As you can see, the receiver has been rewarded with some tokens for mining new blocks.

Then let‘s check the receiver‘s account state.

```
> curl -i -H Accept:application/json -X POST http://localhost:8685/
↪v1/user/accountstate -d '{"address":"your_address"}'


{
    "result": {
        "balance": "0",
        "nonce": "0",
        "type": 87,
        "height":"1",
        "pending":"0"
    }
}
```

The new account doesn‘t have tokens as expected.

Send a Transaction

Now letâĂŹs send a transaction in three methods to transfer some tokens from the sender to the receiver!

Sign & Send

In this way, we can sign a transaction in an offline environment and then submit it to another online node. This is the safest method for everyone to submit a transaction without exposing your own private key to the Internet.

First, sign the transaction to get raw data.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/sign -d '{"transaction":{"from":
↪"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE","to":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5", "value":
↪"1000000000000000000","nonce":1,"gasPrice":"20000000000","gasLimit
↪":"2000000"}, "passphrase":"passphrase"}'

{"result":{"data":"CiAbjMP5dyVsTWILfXL1MbwZ8Q6xOgX/
↪JKinks1dpToSdxIaGVcH+WT/
↪SVMkY18ix7SG4F1+Z8evXJoA35caGhlXbip8PupTNxwV4SRM87r798jXWADXpWngIhAAAAAAAAAAA
↪"}}
```

Note Nonce is an very important attribute in a transaction. It‘s designed to prevent replay attacks. For a given account, only after its transaction with nonce N is accepted, will its transaction with nonce N+1 be processed. Thus, we have to check the latest nonce of the account on chain before preparing a new transaction.

Then, send the raw data to an online Nebulas node.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/rawtransaction -d '{"data":
↪"CiAbjMP5dyVsTWILfXL1MbwZ8Q6xOgX/JKinks1dpToSdxIaGVcH+WT/
↪SVMkY18ix7SG4F1+Z8evXJoA35caGhlXbip8PupTNxwV4SRM87r798jXWADXpWngIhAAAAAAAAAAAA
↪"}'

{"result":{"txhash":
↪"1b8cc3f977256c4d620b7d72f531bc19f10eb13a05ff24a8a792cd5da53a1277
↪","contract_address":""}}âĂŐ
```

Send with Passphrase

If you trust a Nebulas node so much that you can delegate your keystore files to it, the second method is a good fit for you.

First, upload your keystore files to the keydir folders in the trusted Nebulas node.

Then, send the transaction with your passphrase.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/transactionWithPassphrase -d '{
↪"transaction":{"from":"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE","to":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5", "value":
↪"1000000000000000000","nonce":2,"gasPrice":"20000000000","gasLimit
↪":"2000000"},"passphrase":"passphrase"}'

{"result":{"txhash":
↪"3cdd38a66c8f399e2f28134e0eb556b292e19d48439f6afde384ca9b60c27010
↪","contract_address":""}}
```

Note Because we have sent a transaction with nonce 1 from the account `n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE`, new transaction with same `from` should be increased by 1, namely 2.

Unlock & Send

This is the most dangerous method. You probably shouldnâĂŹt use it unless you have complete trust in the receiving Nebulas node.

First, upload your keystore files to the keydir folders in the trusted Nebulas node.

Then unlock your accounts with your passphrase for a given duration in the node. The unit of the duration is nano seconds (300000000000=300s).

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/unlock -d '{"address":
↪"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE","passphrase":"passphrase",
↪"duration":"300000000000"}'

{"result":{"result":true}}
```

After unlocking the account, everyone is able to send any transaction directly within the duration in that node without your authorization.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/transaction -d '{"from":
↪"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE","to":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5", "value":
↪"1000000000000000000","nonce":3,"gasPrice":"20000000000","gasLimit
↪":"2000000"}'

{"result":{"txhash":
↪"8d69dea784f0edfb2ee678c464d99e155bca04b3d7e6cdba6c5c189f731110cf
↪","contract_address":""}}âŔŐ
```

Transaction Receipt

We'll get a `txhash` in three methods after sending a transaction successfully. The `txhash` value can be used to query the transaction status.

```
> curl -i -H Accept:application/json -X POST http://localhost:8685/
↪v1/user/getTransactionReceipt -d '{"hash":
↪"8d69dea784f0edfb2ee678c464d99e155bca04b3d7e6cdba6c5c189f731110cf
↪"}'

{"result":{"hash":
↪"8d69dea784f0edfb2ee678c464d99e155bca04b3d7e6cdba6c5c189f731110cf
↪","chainId":100,"from":"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE","to":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5","value":"1000000000000000000
↪","nonce":"3","timestamp":"1524667888","type":"binary","data
↪":null,"gas_price":"20000000000","gas_limit":"2000000","contract_
↪address":"","status":1,"gas_used":"20000"}}âŔŐ
```

The `status` fields may be 0, 1 or 2.

- **0: Failed.** It means the transaction has been submitted on chain but its execution failed.
- **1: Successful.** It means the transaction has been submitted on chain and its execution successeed.
- **2: Pending.** It means the transaction hasn't been packed into a block.

Double Check

Let's double check the receiver's balance.

```
> curl -i -H Accept:application/json -X POST http://localhost:8685/
↪v1/user/accountstate -d '{"address":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5"}'

{"result":{"balance":"3000000000000000000","nonce":"0","type":87,
↪"height":"10","pending":"0"}}
```

Here you should see a balance that is the total of all the successful transfers that you executed.

Next step: Tutorial 3

Write and run a smart contract with JavaScript

03 Write and run a smart contract

YouTube Tutorial

Through this tutorial we will learn how to write, deploy, and execute smart contracts in Nebulas.

Preparation

Before entering the smart contract, first review the previously learned content:

1. Install, compile and start neb application
2. Create a wallet address, setup coinbase, and start mining
3. Query neb node information, wallet address and balance
4. Send a transaction and verify the transaction was successful

If who have doubts about the above content you should go back to the previous chapters. So lets do this. We will learn and use smart contracts through the following steps:

1. Write a smart contract
2. Deploy the smart contract
3. Call the smart contract, and verify the contract execution results

Write a smart contract

Like Ethereum, Nebulas implements NVM virtual machines to run smart contracts, and the NVM implementation uses the JavaScript V8 engine, so for the current development we can write smart contracts using JavaScript and TypeScript.

Write a brief specification of a smart contract:

1. The Smart contract code must be a Prototype object;
2. The Smart contract code must have a init() method, this method will only be executed once during deployment;
3. The private methods in Smart contract must be prefixed with _ , and the private method cannot be a be directly called outside of the contract;

Below we use JavaScript to write the first smart contract: bank safe. This smart contract needs to fulfill the following functions:

1. The user can save money from this bank safe.
2. Users can withdraw money from this bank safe.
3. Users can check the balance in the bank safe.

Smart contract example:

```
'use strict';

var DepositeContent = function (text) {
  if (text) {
    var o = JSON.parse(text);
    this.balance = new BigNumber(o.balance);
    this.expiryHeight = new BigNumber(o.expiryHeight);
  } else {
    this.balance = new BigNumber(0);
    this.expiryHeight = new BigNumber(0);
  }
};

DepositeContent.prototype = {
  toString: function () {
    return JSON.stringify(this);
  }
};

var BankVaultContract = function () {
  LocalContractStorage.defineMapProperty(this, "bankVault", {
    parse: function (text) {
      return new DepositeContent(text);
    },
    stringify: function (o) {
      return o.toString();
    }
  });
};

// save value to contract, only after height of block, users can␣
↪takeout
BankVaultContract.prototype = {
  init: function () {
    //TODO:
  },

  save: function (height) {
    var from = Blockchain.transaction.from;
    var value = Blockchain.transaction.value;
    var bk_height = new BigNumber(Blockchain.block.height);

    var orig_deposit = this.bankVault.get(from);
    if (orig_deposit) {
      value = value.plus(orig_deposit.balance);
    }

    var deposit = new DepositeContent();
    deposit.balance = value;
    deposit.expiryHeight = bk_height.plus(height);

    this.bankVault.put(from, deposit);
  },

  takeout: function (value) {
    var from = Blockchain.transaction.from;
    var bk_height = new BigNumber(Blockchain.block.height);
    var amount = new BigNumber(value);

    var deposit = this.bankVault.get(from);
    if (!deposit) {
      throw new Error("No deposit before.");
    }

    if (bk_height.lt(deposit.expiryHeight)) {
      throw new Error("Can not takeout before expiryHeight.");
    }

    if (amount.gt(deposit.balance)) {
      throw new Error("Insufficient balance.");
    }

    var result = Blockchain.transfer(from, amount);
    if (!result) {
      throw new Error("transfer failed.");
    }
    Event.Trigger("BankVault", {
      Transfer: {
        from: Blockchain.transaction.to,
        to: from,
        value: amount.toString()
      }
    });

    deposit.balance = deposit.balance.sub(amount);
    this.bankVault.put(from, deposit);
  },
  balanceOf: function () {
    var from = Blockchain.transaction.from;
    return this.bankVault.get(from);
  },
  verifyAddress: function (address) {
    // 1-valid, 0-invalid
    var result = Blockchain.verifyAddress(address);
    return {
      valid: result == 0 ? false : true
    };
  }
};
module.exports = BankVaultContract;
```

As you can see from the smart contract example above, `BankVaultContract` is a prototype object that has an init() method. It satisfies the most basic specification for writing smart contracts that we have described before. BankVaultContract implements two other methods:

- save(): The user can save money to the bank safe by calling the save() method;
- takeout(): Users can withdraw money from bank safe by calling takeout() method;
- balanceOf(): The user can check the balance with the bank vault by calling the balanceOf() method;

The contract code above uses the built-in `Blockchain` object and the built-in `BigNumber()` method. Let's break down the parsing contract code line by line:

save():

```
// Deposit the amount into the safe

save: function (height) {
  var from = Blockchain.transaction.from;
  var value = Blockchain.transaction.value;
  var bk_height = new BigNumber(Blockchain.block.height);

  var orig_deposit = this.bankVault.get(from);
  if (orig_deposit) {
    value = value.plus(orig_deposit.balance);
  }
  var deposit = new DepositeContent();
  deposit.balance = value;
  deposit.expiryHeight = bk_height.plus(height);

  this.bankVault.put(from, deposit);
},
```

takeout ():

```
takeout: function (value) {
  var from = Blockchain.transaction.from;
  var bk_height = new BigNumber(Blockchain.block.height);
  var amount = new BigNumber(value);

  var deposit = this.bankVault.get(from);
  if (!deposit) {
    throw new Error("No deposit before.");
  }

  if (bk_height.lt(deposit.expiryHeight)) {
    throw new Error("Can not takeout before expiryHeight.");
  }

  if (amount.gt(deposit.balance)) {
    throw new Error("Insufficient balance.");
  }

  var result = Blockchain.transfer(from, amount);
  if (!result) {
    throw new Error("transfer failed.");
  }
  Event.Trigger("BankVault", {
    Transfer: {
      from: Blockchain.transaction.to,
      to: from,
      value: amount.toString()
    }
  });

  deposit.balance = deposit.balance.sub(amount);
  this.bankVault.put(from, deposit);
},
```

Deploy smart contracts

The above describes how to write a smart contract in Nebulas, and now we need to deploy the smart contract to the chain. Earlier, we have introduced how to make a transaction in Nebulas, and we used the sendTransaction() interface to initiate a transfer. Deploying a smart contract in Nebulas is actually achieved by sending a transaction by calling the sendTransaction() interface, just with different parameters.

```
// transaction - from, to, value, nonce, gasPrice, gasLimit,␣
↪contract
sendTransactionWithPassphrase(transaction, passphrase)
```

We have a convention that if `from` and `to` are the same address, `contract` is not null and `binary` is null, we assume that we are deploying a smart contract.

- `from`: the creator's address
- `to`: the creator's address
- `value`: it should be `"0"` when deploying the contract;
- `nonce`: it should be 1 more than the current nonce in the creator's account state, which can ben obtained via `GetAccountState`.
- `gasPrice`: The gasPrice used to deploy the smart contract, which can be obtained via `GetGasPrice`, or using default values: `"20000000000"`;
- `gasLimit`: The gasLimit for deploying the contract. You can get the estimated gas consumption for the deployment via `EstimateGas`, and cannot use the default value. And you could also set a larger value. The actual gas consumption is decided by the deployment execution.
- `contract`: the contract information, the parameters passed in when the contract is deployed
  - `source`: contract code
  - `sourceType`: Contract code type, `js` and `ts` (corresponding to javaScript and typeScript code)
  - `args`: parameters for the contract initialization method. Use empty string if there is no parameter, and use JSON array if there is a parameter.

Detailed Interface Documentation API.

Example of deploying a smart contract using curl:

```
> curl -i -H 'Accept: application/json' -X POST http://
↪localhost:8685/v1/admin/transactionWithPassphrase -H 'Content-
↪Type: application/json' -d '{"transaction": {"from":
↪"n1H4MYms9F55ehcvygwWE71J8tJC4CRr2so","to":
↪"n1H4MYms9F55ehcvygwWE71J8tJC4CRr2so", "value":"0","nonce":1,
↪"gasPrice":"20000000000","gasLimit":"2000000","contract":{"source
↪":"\"use strict\";var DepositeContent=function(text){if(text){var␣
↪o=JSON.parse(text);this.balance=new BigNumber(o.balance);this.
↪expiryHeight=new BigNumber(o.expiryHeight);}else{this.balance=new␣
↪BigNumber(0);this.expiryHeight=new BigNumber(0);}};
↪DepositeContent.prototype={toString:function(){return JSON.
↪stringify(this);}};var BankVaultContract=function()
↪{LocalContractStorage.defineMapProperty(this,\"bankVault\",
↪{parse:function(text){return new DepositeContent(text);},
↪stringify:function(o){return o.toString();}});};BankVaultContract.
↪prototype={init:function(){},save:function(height){var␣
↪from=Blockchain.transaction.from;var value=Blockchain.transaction.
↪value;var bk_height=new BigNumber(Blockchain.block.height);var␣
↪orig_deposit=this.bankVault.get(from);if(orig_deposit)
↪{value=value.plus(orig_deposit.balance);} var deposit=new␣
↪DepositeContent();deposit.balance=value;deposit.expiryHeight=bk_
```

```
{"result":{"txhash":
↪"aaebb86d15ca30b86834efb600f82cbcaf2d7aaffbe4f2c8e70de53cbed17889
↪","contract_address":"n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM"}}
```

The return value for deploying a smart contract is the transaction's hash address `txhash` and the contract's deployment address `contract_address`. Get the return value does not guarantee the successful deployment of the contract, because the sendTransaction () is an asynchronous process, which need to be packaged by the miner. Just as the previous transfer transaction, the transfer does not arrive in real time, it depends on the speed of the miner packing. Therefore we need to wait for a while (about 1 minute), then you can verify whether the contract is deployed successfully by querying the contract address or calling this smart contract.

Verify the deployment of the contract is successful

Check the receipt of the deploy transaction via `GetTransactionReceipt` to verify whether the contract has been deployed successfully.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getTransactionReceipt -d '{"hash":
↪"aaebb86d15ca30b86834efb600f82cbcaf2d7aaffbe4f2c8e70de53cbed17889
↪"}'

{"result":{"hash":
↪"aaebb86d15ca30b86834efb600f82cbcaf2d7aaffbe4f2c8e70de53cbed17889
↪","chainId":100,"from":"n1H4MYms9F55ehcvygwWE71J8tJC4CRr2so","to":
↪"n1H4MYms9F55ehcvygwWE71J8tJC4CRr2so","value":"0","nonce":"1",
↪"timestamp":"1524711841","type":"deploy","data":
↪"eyJTb3VyY2VUeXBlIjoianMiLCJTb3VyY2UiOiJcInVzZSBzdHJpY3RcIjt2YXIgRGVwb3NpdGVDb
↪ZmFsc2U6dHJ1ZX07fX07bW9kdWxlLmV4cG9ydHM9QmFua1ZhdWx0Q29udHJhY3Q7IiwiQXJncyI6Ii
↪","gas_price":"20000000000","gas_limit":"2000000","contract_
↪address":"n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM","status":1,"gas_
↪used":"22016"}}
```

As shown above, the status of the deploy transaction becomes 1. It means the contract has been deployed successfully.

Execute Smart Contract Method

The way to execute a smart contract method in Nebulas is also straightforward, using the sendTransactionWithPassphrase() method to invoke the smart contract method directly.

```
// transaction - from, to, value, nonce, gasPrice, gasLimit,␣
↪contract
sendTransactionWithPassphrase(transaction, passphrase)
```

- `from`: the user's account address
- `to`: the smart contract address

- `value`: The amount of money used to transfer by smart contract.
- `nonce`: it should be 1 more than the current nonce in the creator's account state, which can ben obtained via `GetAccountState`.
- `gasPrice`: The gasPrice used to deploy the smart contract, which can be obtained via `GetGasPrice`, or using default values `"20000000000"`;
- `gasLimit`: The gasLimit for deploying the contract. You can get the estimated gas consumption for the deployment via `EstimateGas`, and cannot use the default value. And you could also set a larger value. The actual gas consumption is decided by the deployment execution.
- `contract`: the contract information, the parameters passed in when the contract is deployed
  - `function`:the contract method to be called
  - `args`: parameters for the contract initialization method. Use empty string if there is no parameter, and use JSON array if there is a parameter.

For example, execute save() method of the smart contract:

```
> curl -i -H 'Accept: application/json' -X POST http://
↪localhost:8685/v1/admin/transactionWithPassphrase -H 'Content-
↪Type: application/json' -d '{"transaction":{"from":
↪"n1LkDi2gGMqPrjYcczUiweyP4RxTB6Go1qS","to":
↪"n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM", "value":"100","nonce":1,
↪"gasPrice":"20000000000","gasLimit":"2000000","contract":{
↪"function":"save","args":"[0]"}}, "passphrase": "passphrase"}'

{"result":{"txhash":
↪"5337f1051198b8ac57033fec98c7a55e8a001dbd293021ae92564d7528de3f84
↪","contract_address":""}}
```

Verify the execution of the contract method `save` is successful Executing a contract method is actually submitting a transaction on chain as well. We can verify the result through checking the receipt of the transaction via `GetTransactionReceipt`.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getTransactionReceipt -d '{"hash":
↪"5337f1051198b8ac57033fec98c7a55e8a001dbd293021ae92564d7528de3f84
↪"}'

{"result":{"hash":
↪"5337f1051198b8ac57033fec98c7a55e8a001dbd293021ae92564d7528de3f84
↪","chainId":100,"from":"n1LkDi2gGMqPrjYcczUiweyP4RxTB6Go1qS","to":
↪"n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM","value":"100","nonce":"1",
↪"timestamp":"1524712532","type":"call","data":
↪"eyJGdW5jdGlvbiI6InNhdmUiLCJBcmdzIjoiWzBdIn0=","gas_price":
↪"20000000000","gas_limit":"2000000","contract_address":"","status
↪":1,"gas_used":"20361"}}
```

As shown above, the status of the transaction becomes 1. It means the contract method has been executed successfully.

Execute the smart contract takeout() method:

```
> curl -i -H 'Accept: application/json' -X POST http://
↪localhost:8685/v1/admin/transactionWithPassphrase -H 'Content-
↪Type: application/json' -d '{"transaction":{"from":
↪"n1LkDi2gGMqPrjYcczUiweyP4RxTB6Go1qS","to":
↪"n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM", "value":"0","nonce":2,
↪"gasPrice":"20000000000","gasLimit":"2000000","contract":{
↪"function":"takeout","args":"[50]"}}, "passphrase": "passphrase"}'

{"result":{"txhash":
↪"46a307e9beb21f52992a7512f3705fe58ee6c1887122a1b52f5ce5fd5f536a91
↪","contract_address":""}}
```

Verify the execution of the contract method `takeout` is successful In the execution of the above contract method `save`, we save 100 NAS into the smart contract `n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM`. Using the contract method `takeout`, we'll withdrawn 50 NAS from the 100 NAS. The balance of the smart contract should be 50 NAS now.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/accountstate -d '{"address":
↪"n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM"}'

{"result":{"balance":"50","nonce":"0","type":88}}
```

The result is as expected.

Query Smart Contract Data

In a smart contract, the execution of some methods won't change anything on chain. These methods are designed to help us query data in readonly mode from blockchains. In Nebulas, we provide an API `call` for users to execute these readonly methods.

```
// transaction - from, to, value, nonce, gasPrice, gasLimit,␣
↪contract
call(from, to, value, nonce, gasPrice, gasLimit, contract)
```

The parameters of `call` is the same as the parameters of executing a contract method .

Call the smart contract method `balanceOf`:

```
> curl -i -H 'Accept: application/json' -X POST http://
↪localhost:8685/v1/user/call -H 'Content-Type: application/json' -
↪d '{"from":"n1LkDi2gGMqPrjYcczUiweyP4RxTB6Go1qS","to":
↪"n1rVLTRxQEXscTgThmbTnn2NqdWFEKwpYUM","value":"0","nonce":3,
↪"gasPrice":"20000000000","gasLimit":"2000000","contract":{
↪"function":"balanceOf","args":""}}'
```

```
{"result":{"result":"{\"balance\":\"50\",\"expiryHeight\":\"84\"}",
↪"execute_err":"","estimate_gas":"20209"}}
```

Next step: Tutorial 4

Smart Contract Storage

04 Smart Contract Storage

YouTube Tutorial

Earlier we covered how to write smart contracts and how to deploy and invoke smart contracts in the Nebulas.

Now we introduce in detail the storage of the smart contract. Nebulas smart contracts provide on-chain data storage capabilities. Similar to the traditional key-value storage system (eg: redis), smart contracts can be stored on the Nebulas by paying with (gas).

LocalContractStorage

Nebulas‘ Smart Contract environment has built-in storage object `LocalContractStorage`, which can store numbers, strings, and JavaScript objects. The stored data can only be used in smart contracts. Other contracts can not read the stored data.

Basics

The `LocalContractStorage` API includes `set`, `get` and `del`, which allow you to store, read, and delete data. Storage can be numbers, strings, objects

Storing `LocalContractStorage` DataïijŽ

```
// store data. The data will be stored as JSON strings
LocalContractStorage.put(key, value);
// Or
LocalContractStorage.set(key, value);
```

Reading `LocalContractStorage` DataïijŽ

```
// get the value from key
LocalContractStorage.get(key);
```

Deleting `LocalContractStorage` DataïijŽ

```
// delete data, data can not be read after deletion
LocalContractStorage.del(key);
// Or
LocalContractStorage.delete(key);
```

Examples:

```
'use strict';

var SampleContract = function () {
};

SampleContract.prototype = {
    init: function () {
    },
    set: function (name, value) {
        // Storing a string
        LocalContractStorage.set("name",name);
        // Storing a number (value)
        LocalContractStorage.set("value", value);
        // Storing an objects
        LocalContractStorage.set("obj", {name:name, value:value});
    },
    get: function () {
        var name = LocalContractStorage.get("name");
        console.log("name:" + name)
        var value = LocalContractStorage.get("value");
        console.log("value:" + value)
        var obj = LocalContractStorage.get("obj");
        console.log("obj:" + JSON.stringify(obj))
    },
    del: function () {
        var result = LocalContractStorage.del("name");
        console.log("del result:" + result)
    }
};

module.exports = SampleContract;
```

Advanced

In addition to the basic `set`, `get`, and `del` methods, `LocalContractStorage` also provides methods to bind properties of smart contracts. We could read and write binded properties directly without invoking `LocalContractStorage` interfaces to `get` and `set`.

Binding Properties

Object instance, field name and descriptor should be provided to bind properties.

Binding Interface

```
// define a object property named `fieldname` to `obj` with␣
↪descriptor.
// default descriptor is JSON.parse/JSON.stringify descriptor.
// return this.
defineProperty(obj, fieldName, descriptor);

// define object properties to `obj` from `props`.
// default descriptor is JSON.parse/JSON.stringify descriptor.
// return this.
defineProperties(obj, descriptorMap);
```

Here is an example to bind properties in a smart contract.

```
'use strict';

var SampleContract = function () {
    // The SampleContract `size` property is a storage property.␣
↪Reads and writes to` size` will be stored on the chain.
    // The `descriptor` is set to null here, the default JSON.
↪stringify () and JSON.parse () will be used.
    LocalContractStorage.defineMapProperty(this, "size");

    // The SampleContract `value` property is a storage property.␣
↪Reads and writes to` value` will be stored on the chain.
    // Here is a custom `descriptor` implementation, storing as a␣
↪string, and returning Bignumber object during parsing.
    LocalContractStorage.defineMapProperty(this, "value", {
        stringify: function (obj) {
            return obj.toString();
        },
        parse: function (str) {
            return new BigNumber(str);
        }
    });
    // Multiple properties of SampleContract are set as storage␣
↪properties in batches, and the corresponding descriptors use JSON␣
↪serialization by default
    LocalContractStorage.defineProperties(this, {
        name: null,
        count: null
    });
};

module.exports = SampleContract;
```

Then, we can read and write these properties directly as the following example.

```
SampleContract.prototype = {
    // Used when the contract first deploys, can not be used a␣
↪second after the first deploy.
    init: function (name, count, size, value) {
        // Store the data on the chain when deploying the contract
        this.name = name;
        this.count = count;
        this.size = size;
        this.value = value;
    },
    testStorage: function (balance) {
        // value will be read from the storage data on the chain,␣
↪and automatically converted to Bignumber set according to the␣
↪descriptor
        var amount = this.value.plus(new BigNumber(2));
        if (amount.lessThan(new BigNumber(balance))) {
            return 0
        }
    }
};
```

Binding Map Properties

What's more, `LocalContractStorage` also provides methods to bind map properties. Here is an example to bind map properties and use them in a smart contract.

```
'use strict';

var SampleContract = function () {
    // Set `SampleContract`'s property to `userMap`. Map data then␣
↪can be stored onto the chain using `userMap`
    LocalContractStorage.defineMapProperty(this, "userMap");

    // Set `SampleContract`'s property to `userBalanceMap`, and␣
↪custom define the storing and serializtion reading functions.
    LocalContractStorage.defineMapProperty(this, "userBalanceMap", {
        stringify: function (obj) {
            return obj.toString();
        },
        parse: function (str) {
            return new BigNumber(str);
        }
    });

    // Set `SampleContract`'s properties to mulitple map batches
    LocalContractStorage.defineMapProperties(this,{
        key1Map: null,
        key2Map: null
    });
};

SampleContract.prototype = {
    init: function () {
    },
    testStorage: function () {
        // Store the data in userMap and serialize the data onto␣
↪the chain
        this.userMap.set("robin","1");
        // Store the data into userBalanceMap and save the data␣
↪onto the chain using a custom serialization function
        this.userBalanceMap.set("robin",new BigNumber(1));
    },
    testRead: function () {
        //Read and store data
        var balance = this.userBalanceMap.get("robin");
        this.key1Map.set("robin", balance.toString());
        this.key2Map.set("robin", balance.toString());
    }
};

module.exports = SampleContract;
```

Iterate Map

In contract, map does‘t support iterator. if you need to iterate the map, you can use the following way: define two map, arrayMap, dataMap, arrayMap with a strictly increasing counter as key, dataMap with data key as key.

```
"use strict";

var SampleContract = function () {
   LocalContractStorage.defineMapProperty(this, "arrayMap");
   LocalContractStorage.defineMapProperty(this, "dataMap");
   LocalContractStorage.defineProperty(this, "size");
};

SampleContract.prototype = {
    init: function () {
        this.size = 0;
    },
    set: function (key, value) {
        var index = this.size;
        this.arrayMap.set(index, key);
        this.dataMap.set(key, value);
        this.size +=1;
    },

    get: function (key) {
        return this.dataMap.get(key);
    },

    len:function(){
      return this.size;
    },

    iterate: function(limit, offset){
        limit = parseInt(limit);
        offset = parseInt(offset);
        if(offset>this.size){
           throw new Error("offset is not valid");
        }
        var number = offset+limit;
        if(number > this.size){
          number = this.size;
        }
        var result  = "";
        for(var i=offset;i<number;i++){
            var key = this.arrayMap.get(i);
            var object = this.dataMap.get(key);
            result += "index:"+i+" key:"+ key + " value:" +object+"_
↪";
        }
        return result;
    }

};

module.exports = SampleContract;
```

Next step: Tutorial 5

Interacting with Nebulas by RPC API

05 Interacting with Nebulas by RPC API

YouTube Tutorial

Nebulas chain node can be accessed and controlled remotely through RPC. Nebulas chain provides a series of APIs to get node information, account balances, send transactions and deploy calls to smart contracts.

The remote access to the Nebulas chain is implemented by gRPC, and also could be accessed by HTTP via the proxy (grpc-gateway). HTTP access is a interface implemented by RESTful, with the same parameters as the gRPC interface.

API

We‘ve implemented RPC server and HTTP sercer to provide API service in Go-Nebulas.

Modules

All interfaces are divided into two modules: API and Admin.

- API: Provides interfaces that are not related to the user‘s private key.
- Admin: Provides interfaces that are related to the user‘s private key.

It‘s recommended for all Nebulas nodes to open API module for public users and Admin module for authorized users.

Configuration

RPC server and HTTP server can be configured in the configuration file of each Nebulas node.

```
rpc {
    # gRPC API service port
    rpc_listen: ["127.0.0.1:8684"]
    # HTTP API service port
    http_listen: ["127.0.0.1:8685"]
    # Open module that can provide http service to outside
    http_module: ["api", "admin"]
}
```

Example

HTTP

Here is some examples to invoke HTTP interfaces using `curl`.

GetNebState

We can invoke `GetNebState` in API module to fetch the current state of local Nebulas node, including chain identity, tail block, protocl version and so on.

```
> curl -i -H Accept:application/json -X GET http://localhost:8685/
↪v1/user/nebstate

{"result":{"chain_id":100,"tail":
↪"0aa1cceb7801b110fdd5217ba0a4356780c940133924d1c1a4eb60336934dab1
↪","lib":
↪"0000000000000000000000000000000000000000000000000000000000000000
↪","height":"479","protocol_version":"/neb/1.0.0","synchronized
↪":false,"version":"0.7.0"}}
```

UnlockAccount

We can invoke `UnlockAccount` in Admin module to unlock an account in memory. All unlocked accounts can be used to send transactions directly without passphrases.

```
> curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/unlock -d '{"address":
↪"n1NrMKTYESZRCwPFDLFKiKREzZKaN1nhQvz", "passphrase": "passphrase"}
↪'

{"result":{"result":true}}
```

RPC

RPC server is implemented with GRPC. The serialization of GPRC is based on Protocol Buffers. You can find all rpc protobuf files in Nebulas RPC Protobuf Folder.

Here is some examples to invoke rpc interfaces using `golang`.

GetNebState

We can invoke `GetNebState` in API module to fetch the current state of local Nebulas node.

```
import(
    "github.com/nebulasio/go-nebulas/rpc"
    "github.com/nebulasio/go-nebulas/rpc/pb"
)

// GRPC server connection address configuration
addr := fmt.Sprintf("127.0.0.1:%d",uint32(8684))
conn, err := grpc.Dial(addr, grpc.WithInsecure())
if err != nil {
    log.Warn("rpc.Dial() failed:", err)
}
defer conn.Close()

// API interface to access node status information
api := rpcpb.NewAPIServiceClient(conn)
resp, err := ac.GetNebState(context.Background(), & rpcpb.
↪GetNebStateRequest {})
```

```
if err != nil {
    log.Println("GetNebState", "failed", err)
} else {
    log.Println("GetNebState tail", resp)
}
```

LockAccount

Account `n1NrMKTYESZRCwPFDLFKiKREzZKaN1nhQvz` has been unlocked after invoking `v1/admin/account/unlock` via HTTP request above. We can invoke `LockAccount` in Admin module to lock it again.

```
import(
    "github.com/nebulasio/go-nebulas/rpc"
    "github.com/nebulasio/go-nebulas/rpc/pb"
)

// GRPC server connection address configuration
addr := fmt.Sprintf("127.0.0.1:%d",uint32(8684))
conn, err := grpc.Dial(addr, grpc.WithInsecure())
if err != nil {
    log.Warn("rpc.Dial() failed:", err)
}
defer conn.Close()

// Admin interface to access, lock account address
admin := rpcpb.NewAdminServiceClient(conn)
from := "n1NrMKTYESZRCwPFDLFKiKREzZKaN1nhQvz"
resp, err = management.LockAccount(context.Background(), & rpcpb.
↪LockAccountRequest {Address: from})
if err != nil {
    log.Println("LockAccount", from, "failed", err)
} else {
    log.Println("LockAccount", from, "result", resp)
}
```

API List

For more interfaces, please refer to the official documentation:

- API Module
- Admin Module.

Next

Good job! Now let's join the official Testnet and/or Mainnet to experience Nebulas to the fullest!

Join the Testnet & Join the Mainnet

DApp Development

- Smart Contract
- Tutorials [PDF]
- Chrome Extension (Similar to MetaMask)

SDKs

- Javascript SDK
- JAVA SDK
- PHP SDK
- Python SDK
- Web NebPay SDK, How to use NebPay in your Dapp
- Android NebPay SDK
- iOS NebPay SDK

Standard Tokens

- NRC20
- NRC721

Community Tools

- Nebulearn (credit: Tehjr)
- Demo DApp (credit: ChengOrangeJu, yupnano, Kurry)
- Nebulas&React (thanks to Howon)
- Debug Tools (thanks to xiwangzishi)

Official Nebulas Documents

- Nebulas PoD Node Decentralization Strategy - Based on the Proof of Devotion (PoD) Mechanism: [PDF], [GitHub]
- NAX Whitepaper: [PDF], [GitHub]
- Orange Paper - Nebulas Governance: [PDF], [GitHub]

- Nebulas Mauve Paper: Developer Incentive Protocol: [English], [Chinese] [GitHub], Official Interpretation
- Nebulas Rank Yellow Paper: [English], [Chinese], [Korean], [Portuguese], [GitHub], Official Interpretation
- Technical Whitepaper: [English], [Chinese] [GitHub]
- Non-technical Whitepaper: [English], [Chinese]

Dive into Nebulas

- Dive into Nebulas 1 - An Introduction
- Dive into Nebulas 2 - A Quick Start
- Dive into Nebulas 3 - Managing Accounts
- Dive into Nebulas 4 - Transactions

How to build a DApp on Nebulas

- How to build a DApp on Nebulas: [Part 1], [Part 2], [Part 3]
- Details on the Smart Contract Ranking Algorithm: [Part 1], [Part 2]
- New Nebulas Smart Contract feature
- Claim Nebulas Testnet Token Step by Step
- Why Choose Nebulas at a Hackathon?
- How to architect a DApp using Nuxt.js and Nebulas by Honey Thakuria
- Nebulas: JavaScript Meets Smart Contracts âĂŤâĂŤ An Intro to Nebulas for Ethereum Smart Contract Developers by Michal Zalecki

How to use Nebulas Web Wallet

Web Wallet: Github

1. Creating A NAS Wallet Nebulas Wallet
2. Sending NAS from your Wallet Nebulas Wallet
3. Signing a Transaction Offline Nebulas Wallet
4. View Wallet Information Nebulas Wallet
5. Check TX Status Nebulas Wallet
6. Deploy a Smart Contract Nebulas Wallet
7. Call a Smart Contract on Nebulas Nebulas Wallet

Tech AMA

- Tech Reddit AMA
- Nebulas' First Reddit AMA Recap
- Live Reddit AMA with Nebulas Founder Hitters Xu
- Nebulas AMA Series#1 Testnet with Nebulas Co-Founder Robin Zhong
- Nebulas AMA Series#2 Testnet with Nebulas Co-Founder Robin Zhong
- Nebulas AMA Series#3 General Question with Nebulas Co-Founder Robin Zhong
- Answers from AMA with Nebulas developer Roy Shang

Welcome to recommend more resources from the community, and you can edit this page on Github directly. Help others and learn things together. As always, translations and bug reports are always welcome. Learn more about how to contribute.

Data Center API

> The data center provides external price and on-chain data query interfaces, and provides underlying data support for Nebulas ecosystem applications such as browsers and wallets.

Node

Mainnet Node:

API List

- *NRC20 transaction list*
- *Transaction Details*
- *Transaction List*
- *Off-chain (Pending) Transaction List*
- *Query the Transaction List by Addresses*
- *History Transaction Statistics*
- *Current Transaction Statistics*
- *Block List*
- *Block Details*
- *Highest Block*
- *Contract List*

API Details

NRC20 Transaction List

Query by address to obtain the NRC20 transaction list [paged].

Parameters

`address` address

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`txnList` transaction list

`totalPage` total page number

`maxDisplayCnt` maximum number of displays

`currentPage` current page number

`txnCnt` total

Example

```
//Request
curl -X GET https://data.nebulas.io/tx/nrc20?address=
↪"n1zJjyWVKr8HiL1b6dN7JGBtCZFuKQHenLG"&page=1&page_size=2

//Result
{"code":0,"msg":"success","data":{"txnList":[{"hash":
↪"eb3c8d848679ad1786d5986da48bcc5d5732141fd6fccde0a9a3bfd78ae1d970
↪","block":{"hash":
↪"e188cb8f384f010cc5cfd84ed6779a96f74f7cb363383024b26502c0e85e767d
↪","height":4841297},"from":{"hash":
↪"n1QUMEs7qkvCDuimT2zYsbf62z9pRtofQiP"},"to":{"hash":
↪"n1zJjyWVKr8HiL1b6dN7JGBtCZFuKQHenLG"},"status":1,"value":
↪"346000000000000000000","nonce":27,"timestamp":1595146350000,
↪"currentTimestamp":1595149022681,"timeDiff":2672681,"type":"call",
↪"gasPrice":"20000000000","gasLimit":"830000","gasUsed":"63615",
↪"data":"{\"Function\":\"transfer\",\"Args\":\"[\\\
↪"n1zJjyWVKr8HiL1b6dN7JGBtCZFuKQHenLG\\\",\\\
↪"346000000000000000000\\\"]\"}","contractAddress":"","executeError
↪":"","tokenName":"MRH2","decimal":18,"txFee":"1272300000000000"},{
↪"hash":
↪"fc148958467c69d9ad1c0a5038a41b096d3c2302a604e0edecb52c7a6e9e1675
↪","block":{"hash":
↪"f7b9ed2a907c39a75789246ce071213d0903027ee014cece16063b397bde8e1e
↪","height":4841292},"from":{"hash":
↪"n1N9dDUn3AcdrjLhfCSzNxBGzfxHYTgKH1W"},"to":{"hash":
↪"n1zJjyWVKr8HiL1b6dN7JGBtCZFuKQHenLG"},"status":1,"value":
↪"323000000000000000000","nonce":41,"timestamp":1595146275000,
↪"currentTimestamp":1595149022681,"timeDiff":2747681,"type":"call",
↪"gasPrice":"20000000000","gasLimit":"830000","gasUsed":"63615",
↪"data":"{\"Function\":\"transfer\",\"Args\":\"[\\\
↪"n1zJjyWVKr8HiL1b6dN7JGBtCZFuKQHenLG\\\",\\\
↪"323000000000000000000\\\"]\"}","contractAddress":"","executeError
↪":"","tokenName":"MRH2","decimal":18,"txFee":"1272300000000000"}],
↪"totalPage":243,"maxDisplayCnt":500,"currentPage":1,"txnCnt":485}}
```

Transaction Details

Query transaction details by address.

Parameters

`hash` transaction hash

Return

`tx_hash` transaction hash

`address_main` transaction origination address

`address_supporting` transaction acceptance address

`direction` transaction acceptance address

`tx_type` transaction type

`timestamp` timestamp

`block_timestamp` transaction timestamp

`contract_address` contract address

`tx_value` transaction amount

`real_value` amount

`gas_price` transaction fee price

`gas_limit` transaction fee limit

`gas_used` transaction usage

`statue` transaction status: 0 failed, 1 successful, 2 pending

`block_height` transaction block height

`is_nrc20` if it's a NRC20 transaction

Example

```
//Request
curl -X GET https://data.nebulas.io/tx/detail?hash=
↪"eb3c8d848679ad1786d5986da48bcc5d5732141fd6fccde0a9a3bfd78ae1d970"

//Result
{"code":0,"msg":"success","data":{"ext_info":{"data":
↪"eyJGdW5jdGlvbiI6InRyYW5zZmVyIiwiQXJncyI6IltcIm4xekpqeVdWS3I4SGlMMWI2ZE43SkdCd
↪","execute_error":"","execute_result":"\"\""},"tx_hash":
↪"eb3c8d848679ad1786d5986da48bcc5d5732141fd6fccde0a9a3bfd78ae1d970
↪","address_main":"n1QUMEs7qkvCDuimT2zYsbf62z9pRtofQiP","address_
↪supporting":"n1zJjyWVKr8HiL1b6dN7JGBtCZFuKQHenLG","direction":
↪"send","tx_type":"call","timestamp":1595146335,"block_timestamp
↪":1595146350,"contract_address":
↪"n1pxpisjJMsnVLrqHBnS52o8wsuMUsycfDV","tx_value":"0","real_value":
↪"346000000000000000000","gas_price":"20000000000","gas_limit
↪":830000,"gas_used":63615,"status":1,"block_height":4841297,"is_
↪nrc20":true}}
```

Transaction List

Query the transaction list.

Parameters

`block_height` block height, optional parameter

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page request URL

`count` transactiontotal

`total_page` total number of pages

`current_page` current page number

`list` transaction list

`server_timestamp` Timestamp

Example

```
//Request
curl -X GET https://data.nebulas.io/tx/list?page=1&page_size=2


//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/tx/
↪list?page=2&page_size=2","previous":null,"count":13352569,"total_
↪page":6676285,"current_page":1,"list":[{"tx_hash":
↪"b3805aaf308fea5fc975e5b87632dbd27c7795c3b2bcc03b69263f8301bfcc6a
↪","chainId":1,"address_from":"n1Tx1bxFsgyTcjQ82h59kMQ3XbZGXSiq8w2
↪","address_to":"n1KxWR8ycXg7Kb9CPTtNjTTEpvka269PniB","value":
↪"5860031796400000000000","nonce":7181,"timestamp":1595152078,
↪"block_timestamp":1595157150,"block_date":"20200719","tx_type":
↪"binary","gas_price":"20000000000","gas_limit":200000,"gas_used
↪":20000,"contract_address":"","status":1,"execute_error":"",
↪"execute_result":"","block_height":4842017},{"tx_hash":
↪"a920a35b4450d1cb7395b96a6b4f3ae3a616a6f0a129f6586efb6f6b2f71c13f
↪","chainId":1,"address_from":"n1EoNsJNXG1tN3z9rvjwPKoBXbJMqAjmESC
↪","address_to":"n1gtEoDBNnPrAHzRY9tSnpSSg3QppLYXUdR","value":"0",
↪"nonce":4915,"timestamp":2147483647,"block_timestamp":1595157075,
↪"block_date":"20200719","tx_type":"call","gas_price":"20000000000
↪","gas_limit":9000000,"gas_used":989604,"contract_address":"",
↪"status":1,"execute_error":"","execute_result":"{\"hasNext\
↪":false,\"period\":298,\"start\":4836000,\"end\":4842000,\"page\
↪":4}","block_height":4842012}],"server_timestamp":1595157487445}}
```

Off-chain (Pending) Transaction List

Quary the off-chain transaction list.

Parameters

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page requestURL

`count` transactiontotal

`total_page` total number of pages

`current_page` current page number

`list` transaction list

`server_timestamp` Timestamp

Example

```
//Request
curl -X GET https://data.nebulas.io/tx/list/pending?page=1&page_
↪size=2


//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/tx/
↪list/pending?page=2&page_size=2","previous":null,"count":156,
↪"total_page":78,"current_page":1,"list":[{"tx_hash":
↪"4138b43575e419a534a878be45dd8e43f4ddb61ac122ed82a8d5050fde156f7b
↪","chainId":1,"address_from":"n1Gfg8uqtFsvGKZn6XtVmEb116ZtagRHDoe
↪","address_to":"n22CMMXaxkAjjbsWtVXSmFJgEsnVZ3UwUWf","value":"0",
↪"nonce":2818,"timestamp":1595154990,"block_timestamp":null,"block_
↪date":null,"tx_type":"pod","gas_price":"1000000000000","gas_limit
↪":50000000000,"gas_used":null,"contract_address":null,"status":2,
↪"execute_error":"","execute_result":"","block_height":null},{"tx_
↪hash":
↪"9a6a2fa7c02c18a0f61c96cf4b545da6e41bf5e64a8641062e66c81134b924f8
↪","chainId":1,"address_from":"n1bc9szRj57bD4HqM5672v92gnBvPayacJE
↪","address_to":"n22CMMXaxkAjjbsWtVXSmFJgEsnVZ3UwUWf","value":"0",
↪"nonce":2894,"timestamp":1595154990,"block_timestamp":null,"block_
↪date":null,"tx_type":"pod","gas_price":"1000000000000","gas_limit
↪":50000000000,"gas_used":null,"contract_address":null,"status":2,
↪"execute_error":"","execute_result":"","block_height":null}],
↪"server_timestamp":1595157812578}}
```

Query the Transaction List by Addresses

Query the transaction list by addresses

Parameters

`address` address

`is_nrc20` If it‘s a NRC20 transaction

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page requestURL

`count` transactiontotal

`total_page` total number of pages

`current_page` current page number

`list` transaction list

`server_timestamp` Timestamp

Example

```
//Request
curl -X GET https://data.nebulas.io/tx/listByAddress?
↪address=n1Gfg8uqtFsvGKZn6XtVmEb116ZtagRHDoe&page=1&page_size=2


//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/tx/
↪listByAddress?address=n1Gfg8uqtFsvGKZn6XtVmEb116ZtagRHDoe&page=2&
↪page_size=2","previous":null,"count":2829,"total_page":1415,
↪"current_page":1,"list":[{"tx_hash":
↪"9e862c72614aba4138e8e1fcbf44ef0a8f2007520015e899bf75da3d9136049b
↪","chainId":1,"address_from":"n1Gfg8uqtFsvGKZn6XtVmEb116ZtagRHDoe
↪","address_to":"n22CMMXaxkAjjbsWtVXSmFJgEsnVZ3UwUWf","value":"0",
↪"nonce":2818,"timestamp":1595154930,"block_timestamp":1595154990,
↪"block_date":"20200719","tx_type":"pod","gas_price":"1000000000000
↪","gas_limit":50000000000,"gas_used":150885,"contract_address":"",
↪"status":1,"execute_error":"","execute_result":"\"\"","block_
↪height":4841873},{"tx_hash":
↪"4158b45975e419a534a878be45dd8e43f4ddb61ac122ed82a8d5050fde156f7b
↪","chainId":1,"address_from":"n1Gfg8uqtFsvGKZn6XtVmEb116ZtagRHDoe
↪","address_to":"n22CMMXaxkAjjbsWtVXSmFJgEsnVZ3UwUWf","value":"0",
↪"nonce":2818,"timestamp":1595154990,"block_timestamp":null,"block
```

History Transaction Statistics

History transaction statistics

Parameters

`days` statistical days, optional parameter, default is 15

Return

`price` NAS price

`date` date

`transaction_count` transaction amount

Example

```
//Request
curl -X GET https://data.nebulas.io/tx/count/history?days=2

//Result
{"code":0,"msg":"success","data":[{"price":0.4775322101523956,"date
↪":"20200718","transaction_count":2354},{"price":0,"date":"20200717
↪","transaction_count":2231}]}
```

Current Transaction Statics

Current (24 hours) transaction statics

Parameters

None

Return

`count` transactionæŢř

Example

```
//Request
curl -X GET https://data.nebulas.io/tx/count/today

//Result
{"code":0,"msg":"success","data":{"count":1074}}
```

Block List

Block list.

Parameters

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page requestURL

`count` transactiontotal

`total_page` total number of pages

`current_page` current page number

`list` block list

Example

```
//Request
curl -X GET https://data.nebulas.io/block/list?page=1&page_size=2

//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/
↪block/list?page=2&page_size=2","previous":null,"count":4842132,
↪"total_page":2421066,"current_page":1,"list":[{"block_hash":
↪"c7e7862a725acb12c599cb1bbf5dd2e9781a4edffb4d850d2ad70776f0517dcc
↪","parent_hash":
↪"87f27c9e71d6797eb80fcb4f4ee39d87c7ea153f7124b9679851fb672e6631eb
↪","height":4842132,"tx_count":1,"nonce":0,"coinbase":
↪"n1HEitaphPaJG1iJHVU5AVtC7GvhHmKf1YV","timestamp":1595158875,
↪"chain_id":1,"state_root":
↪"5f173a48643475b17564114b761112e2c091f12916606a91f3a9d3dff9037532
↪","txs_root":
↪"70204ae37a62a318de69fe33505edb10a3983bc417394c97186c56d831c9a3ff
↪","events_root":
↪"442586640329dc93208dae2177d924bf0dc58663b12ec85caa7ef87c87012beb
↪","miner":"n1RKTn2SXD67TvBaf55eHcFHa1AKzJcmrh2","randomSeed":
```

Block Details

Query the block details

Parameters

`h` block hash or height

Return

`gas_info` gas infomation

`dynasty` miner address list

`node` miner node information

`block_hash` block hash

`parent_hash` parent block hash

`height` block height

`tx_count` transaction amount

`nonce` nonce

`coinbase` coinbase address

`timestamp` Timestamp

`chain_id` Chain ID

`miner` miner address

`is_finality` If it‘s finality

Example

```
//Request
curl -X GET https://data.nebulas.io/block/detail?h=4820298

//Result
{"code":0,"msg":"success","data":{"gas_info":{"gas_limit":"0","gas_
↪reward":"0","avg_gas_price":"0"},"dynasty":[
↪"n1FfapZbhjFb2Lt9vxuSW7odjEPh8DeYsFG",
↪"n1FoHjHE3rMGsYEEPgadvrHeuiVmvkL44p3",
↪"n1HpAFoBMnhMRSJ4jXECWbhAVbx1fzh1Pcj",
↪"n1Jkdiq1H1HSXYJXtvDDkYm84Tmapo4hhMv",
↪"n1NSK8TiFQRGXzxcc5jmTCuCLbnCejUcw8s",
↪"n1Qm4ZaRf7HnQNVhg13BxjprVW42vAi8dYY",
↪"n1QyuEwxQED86Wq1ADMCTFjCay4BUnJDBGX",
↪"n1S7CvsNWbPtaxH8pMqSby8hW6EkHExM59C",
↪"n1Sg1yFp4fGFLtvmxHXBrcATSBWwNJfUyBc",
```

Highest Block

Query the highest block.

Parameters

None

Return

`height` height

`server_timestamp` Timestamp

Example

```
//Request
curl -X GET https://data.nebulas.io/block/max

//Result
{"code":0,"msg":"success","data":{"height":4842503.0,"server_
↪timestamp":1595164447597}}
```

Contract List

Query the contract list.

Parameters

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page requestURL

`count` total

`total_page` total number of pages

`current_page` current page number

`list` contract list

Example

```
//Request
curl -X GET https://data.nebulas.io/contracts?page=1&page_size=2

//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/
↪contracts?page=2&page_size=2","previous":null,"count":15423,
↪"total_page":7712,"current_page":1,"list":[{"block_timestamp
↪":1593176850000,"type":"NORMAL","address":
↪"n1orbrZ5r7wa6C4dBmCo46ZkPtMiMaS5GcP","creator_address":
↪"n1SD9Bp3pEDy1g8acG7qmVJu5DPWreLeqsm","deploy_tx_hash":
↪"bff2e9c73c1c3a85ac5911614fef1892f9f35cbd253aa6eef79e905ebca2b150
↪","block_height":4710023},{"block_timestamp":1591266855000,"type":
↪"NORMAL","address":"n1nvqH9aE57USRPfGm33sTRPac9kRjsXiKu","creator_
↪address":"n1QaTZhe1TABmigiGsutCJR2kdrhHNXqNeZ","deploy_tx_hash":
↪"ce06c9e6a5b5bd0ca985df3b837f39a6bd475e45247b392629badeb9b64d3450
↪","block_height":4582759}]}}
```

token list

QUery the token list.

Parameters

None

Return list

`token_name` token name

`description` token description

`contract` contractaddress

`total` total

`token_decimals` token decimals

Example

```
//Request
curl -X GET https://data.nebulas.io/token/list

//Result
{"code":0,"msg":"success","data":[{"token_name":"ATP","description":
↪"ATP-DESC","contract":"n1zUNqeBPvsyrw5zxp9mKcDdLTjuaEL7s39","total
↪":"1000000000000000000000000000","token_decimals":18},{"token_
↪name":"WITI","description":"WITI-DESC","contract":
↪"n1hiWG7Ce8HhTaJGzSJoAaJ9w1CJd7Do2rm","total":"2000000000000000000
↪","token_decimals":8},{"token_name":"NAT","description":"NAT",
↪"contract":"n1mpgNi6KKdSzr7i5Ma7JsG5yPY9knf9He7","total":
↪"1000000000000000000000000000000","token_decimals":18},{"token_name
↪":"WICM","description":"WITTICISM TOKEN","contract":
↪"n1jJHWrXMysNm7odsUed3RGddiEH1jbJXtt","total":
↪"2000000000000000000000000000","token_decimals":18},{"token_name":
↪"NAX","description":"NAX","contract":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","total":
↪"10000000000000000000","token_decimals":9},{"token_name":"MRH2",
↪"description":"MRH2","contract":
↪"n1pxpisjJMsnVLrqHBnS52o8wsuMUsycfDV","total":
↪"1000000000000000000000000000","token_decimals":18}]}
```

Token Details

Query the token details

Parameters

`token` token name

`contract` select one between contract address and token

Return

`token_name` token name

`description` token description

`contract` contract address

`total` total

`token_decimals` token decimals

Example

```
//Request
curl -X GET https://data.nebulas.io/token/detail?token=NAX

//Result
{"code":0,"msg":"success","data":{"token_name":"NAX","description":
↪"NAX","contract":"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","total":
↪"10000000000000000000","token_decimals":9}}
```

Token Transaction List

Query token dettails

Parameters

`token` token name

`contract` select one between the contract address and token

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page request URL

`count` total

`total_page` total number of pages

`current_page` current page number

`list` transaction list

`token` token information

Example

```
//Request
curl -X GET https://data.nebulas.io/token/tx/list?token=NAX&page=1&
↪page_size=2

//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/
↪token/tx/list?page=2&page_size=2&token=NAX","previous":null,"count
↪":8318,"total_page":4159,"current_page":1,"list":[{"token":{
↪"token_name":"NAX","description":"NAX","contract":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","total":
↪"10000000000000000000","token_decimals":9},"tx_hash":
↪"63d2395551bc1ad95138d2efeee941573a6eb7154c8db532988f64663e1ea2b3
↪","address_main":"n1dTbtBu98qsxLNmmZKRWmh5qMS6L2pSzKy","address_
↪supporting":"n1aM6Bag362YnAVtjLkRWzFVjQdrZpsNmwk","direction":
↪"send","tx_type":"call","timestamp":1595164824,"block_timestamp
↪":1595164845,"contract_address":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","tx_value":"0","real_value":
↪"2486000000000","gas_price":"20000000000","gas_limit":810000,"gas_
↪used":21105,"status":1,"block_height":4842530,"is_nrc20":true},{
↪"token":{"token_name":"NAX","description":"NAX","contract":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","total":
↪"10000000000000000000","token_decimals":9},"tx_hash":
↪"7063f2234170f43747149711bf119a03cdd8b51405212114c7c4ae9021295100
↪","address_main":"n1dTbtBu98qsxLNmmZKRWmh5qMS6L2pSzKy","address_
↪supporting":"n1aM6Bag362YnAVtjLkRWzFVjQdrZpsNmwk","direction":
↪"send","tx_type":"call","timestamp":1595164477,"block_timestamp
↪":1595164500,"contract_address":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","tx_value":"0","real_value":
↪"3707000000000","gas_price":"20000000000","gas_limit":810000,"gas_
↪used":20673,"status":0,"block_height":4842507,"is_nrc20":true}],
↪"server_timestamp":1595165282662,"token":{"token_name":"NAX",
↪"description":"NAX","contract":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","total":
↪"10000000000000000000","token_decimals":9}}}
```

Token Holders List

Query token holders list.

Parameters

`token` token name

`contract` contract addressïijŇand token (select one)

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

next next page requestURL

count total

total_page total number of pages

current_page current page number

list address list

Example

```
//Request
curl -X GET https://data.nebulas.io/token/holders?token=NAX&page=1&
↪page_size=2

//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/
↪token/holders?page=2&page_size=2&token=NAX","previous":null,"count
↪":1399,"total_page":700,"current_page":1,"list":[{"address":
↪"n1QbEBKhsGxJUahRQXqV88T9oi4d9ywK453","contract":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","balance":
↪"390084128964550818"},{"address":
↪"n214bLrE3nREcpRewHXF7qRDWCcaxRSiUdw","contract":
↪"n1etmdwczuAUCnMMvpGasfi8kwUbb2ddvRJ","balance":
↪"100395915000000000"}]}}
```

NAS Holders List

Query NAS holders listãĂĆ

Parameters

page page number; optional parameter, default 1

page_size page size; optional parameter, default 20

Return

next next page requestURL

count total

total_page total number of pages

`current_page` current page number

`list` address list

Example

```
//Request
curl -X GET https://data.nebulas.io/holders?page=1&page_size=2

//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/
↪holders?page=2&page_size=2","previous":null,"count":339598,"total_
↪page":400,"current_page":1,"list":[{"tx_count":12,"percentage":59.
↪493851,"address":"n1gczhpkT54RaT4PB55CNoYbqmEQcfo4hqq","nonce":0,
↪"type":88,"balance":"35901357730000000000000000","date_of_found":
↪"20191009","rank":1.0},{"tx_count":88344,"percentage":20.707855,
↪"address":"n1KxWR8ycXg7Kb9CPTtNjTTEpvka269PniB","nonce":49290,
↪"type":87,"balance":"12496083258556280663196250","date_of_found":
↪"20180424","rank":2.0}]}}
```

NAX Distribution History

Query NAX distribution history.

Parameters

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page requestURL

`count` total

`total_page` total number of pages

`current_page` current page number

`list` history list

Example

```
//Request
curl -X GET https://data.nebulas.io/nax/history?page=1&page_size=2

//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/
↪nax/history?page=2&page_size=2","previous":null,"count":299,
↪"total_page":150,"current_page":1,"list":[{"nas_price":0.
↪4775322101523956,"nax_price":0.002492307692307692,"start_timestamp
↪":"2020-07-18T10:08:00","end_timestamp":"2020-07-19T11:08:15",
↪"total_distributed_nax":"1233673083506519749","total_destroyed_nax
↪":"1351842110126118095","total_vote_nax":"514052192214285714.2857
↪","avg_reward_rate":0.950916,"created_at":"2020-07-18T10:12:16",
↪"stage":298,"start":4836000,"end":4842000,"estimate_nax":
↪"7421906713080442","distributed_nax":"3425198001737814",
↪"destroyed_nax":"3996708711342628","total_supplied_nax":
↪"1233673083506411630","pledged_nas":"27848503662499186630500000",
↪"total_supplied_nas":"60343663688079320000000000","status":1},{
↪"nas_price":0.4541347259763038,"nax_price":0.002418367088607595,
↪"start_timestamp":"2020-07-17T09:07:00","end_timestamp":"2020-07-
↪18T10:08:00","total_distributed_nax":"1230247885504781935","total_
↪destroyed_nax":"1347845401414775467","total_vote_nax":
↪"513406552357142857.1429","avg_reward_rate":0.932828,"created_at":
↪"2020-07-17T09:12:05","stage":297,"start":4830000,"end":4836000,
↪"estimate_nax":"7429336049129572","distributed_nax":
↪"3429332827592696","destroyed_nax":"4000003221536876","total_
↪supplied_nax":"1230247885504674237","pledged_nas":
↪"27850287662499186630500000","total_supplied_nas":
↪"58103599263060000000000000","status":1}]}}
```

NAX Statistics

Query NAX statistics.

Parameters

None

Return

`last_distributed_nax` last distributed NAX

`total_distributed_nax` total distributed NAX

`current_pledged_nas` current dstaking nas `current_total_nas` current total NAS

`list` distribution history list

Example

```
//Request
curl -X GET https://data.nebulas.io/nax/summary

//Result
{"code":0,"msg":"success","data":{"last_distributed_nax":
↪"3425198001737814","total_distributed_nax":"1233673083506519749",
↪"current_pledged_nas":"27856286662499186630500000","current_total_
↪nas":"71910268469860000000000000","estimate_nax":"7414484806367362
↪","end_height":4848000,"list":[{"nas_price":0.4775322101523956,
↪"nax_price":0.002492307692307692,"start_timestamp":"2020-07-
↪18T10:08:00","end_timestamp":"2020-07-19T11:08:15","total_
↪distributed_nax":"1233673083506519749","total_destroyed_nax":
↪"1351842110126118095","total_vote_nax":"514052192214285714.2857",
↪"avg_reward_rate":0.950916,"created_at":"2020-07-18T10:12:16",
↪"stage":298,"start":4836000,"end":4842000,"estimate_nax":
↪"7421906713080442","distributed_nax":"3425198001737814",
↪"destroyed_nax":"3996708711342628","total_supplied_nax":
↪"1233673083506411630","pledged_nas":"27848503662499186630500000",
↪"total_supplied_nas":"60343663688079320000000000","status":1},{
↪"nas_price":0.4541347259763038,"nax_price":0.002418367088607595,
↪"start_timestamp":"2020-07-17T09:07:00","end_timestamp":"2020-07-
↪18T10:08:00","total_distributed_nax":"1230247885504781935","total_
↪destroyed_nax":"1347845401414775467","total_vote_nax":
↪"513406552357142857.1429","avg_reward_rate":0.932828,"created_at":
↪"2020-07-17T09:12:05","stage":297,"start":4830000,"end":4836000,
↪"estimate_nax":"7429336049129572","distributed_nax":
↪"3429332827592696","destroyed_nax":"4000003221536876","total_
↪supplied_nax":"1230247885504674237","pledged_nas":
↪"27850287662499186630500000","total_supplied_nas":
↪"58103599263060000000000000","status":1},{"nas_price":0.
↪46206315262654346,"nax_price":0.0023882391304347827,"start_
↪timestamp":"2020-07-16T08:06:30","end_timestamp":"2020-07-
↪17T09:07:00","total_distributed_nax":"1226818552677189239","total_
↪destroyed_nax":"1343845398193238591","total_vote_nax":
↪"511429083862068965.5172","avg_reward_rate":0.999776,"created_at":
↪"2020-07-16T08:09:31","stage":296,"start":4824000,"end":4830000,
↪"estimate_nax":"7436772821951523","distributed_nax":
↪"3435182528484704","destroyed_nax":"4001590293466819","total_
↪supplied_nax":"1226818552677081958","pledged_nas":
↪"27865941662499186630500000","total_supplied_nas":
↪"60326540408079320000000000","status":1},{"nas_price":0.
↪49049006922562666,"nax_price":0.0026480666666666665,"start_
↪timestamp":"2020-07-15T07:06:30","end_timestamp":"2020-07-
↪16T08:06:30","total_distributed_nax":"1223383370148704535","total_
↪destroyed_nax":"1339843807899771772","total_vote_nax":
↪"507907569000000000.0000","avg_reward_rate":0.930706,"created_at":
↪"2020-07-15T07:10:30","stage":295,"start":4818000,"end":4824000,
↪"estimate_nax":"7444217038990514","distributed_nax":
↪"3475204652507346","destroyed_nax":"3969012386483168","total_
↪supplied_nax":"1223383370148597666","pledged_nas":
```

NAX Address Distribution History

Query NAX address distribution history

Parameters

`address` address

`page` page number; optional parameter, default 1

`page_size` page size; optional parameter, default 20

Return

`next` next page requestURL

`count` total

`total_page` total number of pages

`current_page` current page number

`list` data list

Example

```
//Request
curl -X GET https://data.nebulas.io/nax/profits?
↪address=n1d4wXxTVmK4rzyiN1vxhMuzxT441bxr8Fg&page=1&page_size=2

//Result
{"code":0,"msg":"success","data":{"next":"http://data.nebulas.io/
↪nax/profits?address=n1d4wXxTVmK4rzyiN1vxhMuzxT441bxr8Fg&page=2&
↪page_size=2","previous":null,"count":10,"total_page":5,"current_
↪page":1,"list":[{"address":"n1d4wXxTVmK4rzyiN1vxhMuzxT441bxr8Fg",
↪"tx_hash":
↪"ed36d5bc52f48c55a92551044b14fd1e35b9e9b3a57226f9e910b8e9709ea9c0
↪","block_timestamp":1593710040,"profit":"-1000000000","stage":-1,
↪"source":1,"block_height":4745566},{"address":
↪"n1d4wXxTVmK4rzyiN1vxhMuzxT441bxr8Fg","tx_hash":
↪"c27185fc66fd346fb53dde96cfa73fec957fc40e2e38c19cf5f051fd42bcc516
↪","block_timestamp":1593710115,"profit":"-1000000000","stage":-1,
↪"source":1,"block_height":4745571}]}}
```

Address Information

Query address information

Parameters

`address` address

Return

`address` address

`nonce` transaction nonce

`type` type: 87 normal address, 88 contract address

`balance` balance

`date_of_found` created date

Example

```
//Request
curl -X GET https://data.nebulas.io/address/info?
↪address=n1d4wXxTVmK4rzyiN1vxhMuzxT441bxr8Fg

//Result
{"code":0,"msg":"success","data":{"contract":{},"address":
↪"n1d4wXxTVmK4rzyiN1vxhMuzxT441bxr8Fg","nonce":124,"type":87,
↪"balance":"223860601614356302521OO","date_of_found":"20200130"}}
```

Address Statistics

Query address statistics

Parameters

`days` days, optional parameter, default is 30

Return

`created_at` created date

`updated_at` updated date

`date` date

`all_address_count` address count

`contract_count` contract count

Example

```
//Request
curl -X GET https://data.nebulas.io/address/count/history?days=5

//Result
{"code":0,"msg":"success","data":[{"created_at":"2020-07-19T00:00:00
↪","updated_at":"2020-07-19T00:00:00","date":"20200718","all_
↪address_count":339596,"contract_count":15922},{"created_at":"2020-
↪07-18T00:00:00","updated_at":"2020-07-18T00:00:00","date":
↪"20200717","all_address_count":339587,"contract_count":15922},{
↪"created_at":"2020-07-17T00:00:00","updated_at":"2020-07-
↪17T00:00:00","date":"20200716","all_address_count":339581,
↪"contract_count":15922},{"created_at":"2020-07-17T00:00:00",
↪"updated_at":"2020-07-17T00:00:00","date":"20200716","all_address_
↪count":339581,"contract_count":15922},{"created_at":"2020-07-
↪16T00:00:00","updated_at":"2020-07-16T00:00:00","date":"20200715",
↪"all_address_count":339577,"contract_count":15922}]}
```

Public Chain Statistics Informations

Query public chain statistics information.

Parameters

None

Return

`block_count` block count

`tx_count` transaction count

`contract_count` contract count

`normal_address_count` address count

Example

```
//Request
curl -X GET https://data.nebulas.io/chain/summary

//Result
{"code":0,"msg":"success","data":{"block_count":4842814,"tx_count
↪":13075893,"contract_count":15922,"normal_address_count":323676}}
```

DApp Development

Smart Contracts

You may see working examples of simple smart contracts:

Smart Contracts for Rookies

This tutorial is intended for beginners and will help you to understand the basics of smart contracts under Nebulas: you'll be assisted to download the core and run a node instance, to set up a development environment, and to write some basic examples of smart contracts. In addition, you will find a quick reference guide for more seasoned developers, and a FAQ containing all the common doubts about the subject.

On [Github] by Arielsbecker

1. What is a smart contract?
2. Installing the core
3. Setting up a development environment
4. Hello World, a barebones smart contract
5. AddressMetadata, a smart contract with storage capabilities
6. PiggyBank, a smart contract that moves money around
7. Reference guide
8. FAQ

Languages

In Nebulas, there are two supported languages for writing smart contracts:

- JavaScript
- TypeScript

They are supported by the integration of Chrome V8, a widely used JavaScript engine developed by The Chromium Project for Google Chrome and Chromium web browsers.

Execution Model

The diagram below is the Execution Model of the Smart Contract:

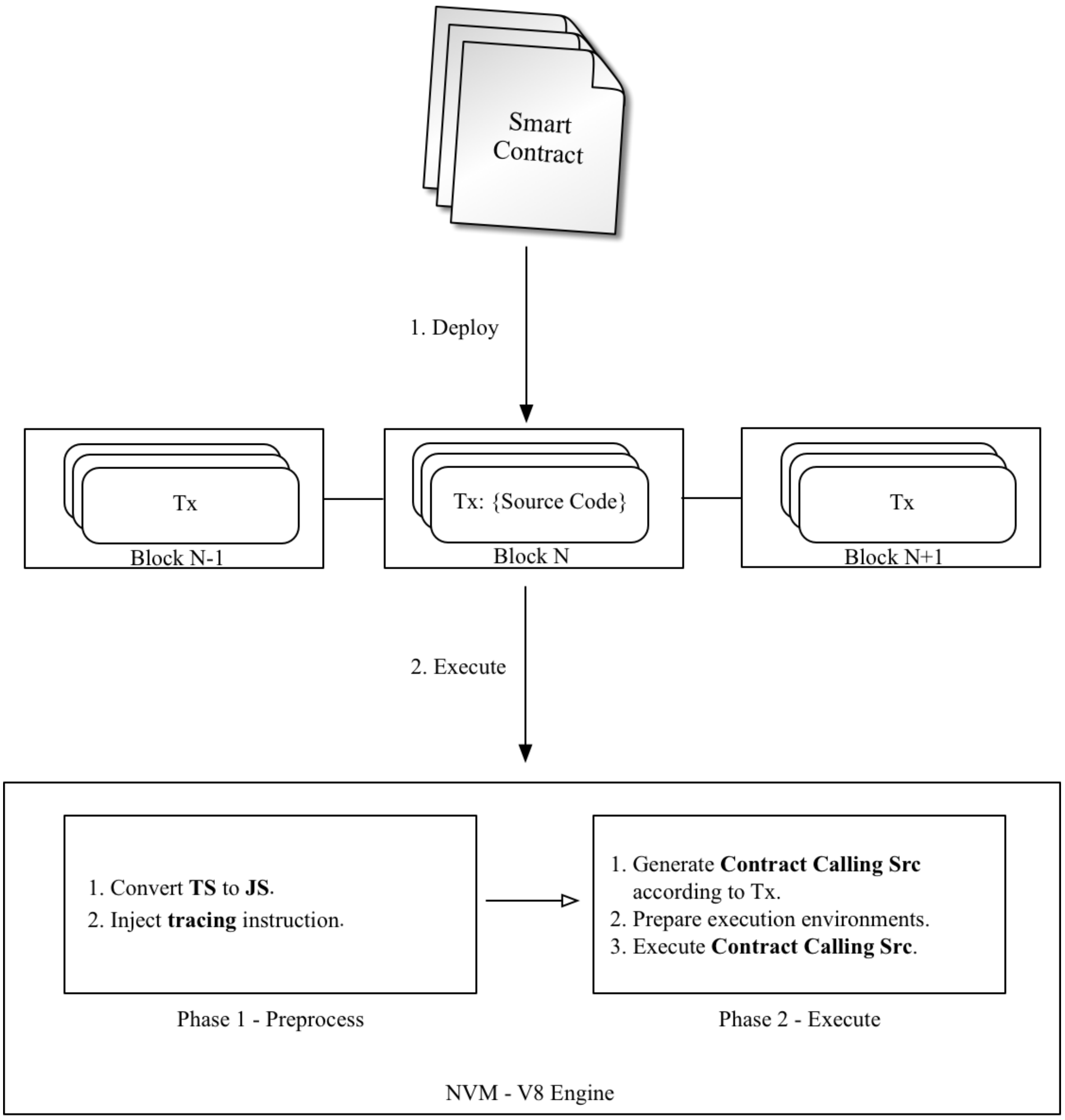

Smart Contract
1. Deploy
Tx
Block N-1
Tx: {Source Code}
Block N
Tx
Block N+1
2. Execute
1. Convert **TS** to **JS**.
2. Inject **tracing** instruction.
Phase 1 - Preprocess
1. Generate **Contract Calling Src** according to Tx.
2. Prepare execution environments.
3. Execute **Contract Calling Src**.
Phase 2 - Execute
NVM - V8 Engine

Fig. 1.1: Smart Contract Execution Model

1. The whole src of the Smart Contract and its arguments are packaged in the Transaction and deployed on Nebulas.
2. The execution of Smart Contract is divided in two phases:
3. Preprocess: inject tracing instruction, etc.
4. Execute: generate executable src and execute it.

Contracts

Contracts in Nebulas are similar to classes in object-oriented languages. They contain persistent data in state variables and functions that can modify these variables.

Writing Contract

A contract must be a Prototype Object or Class in JavaScript or TypeScript.

A Contract must include an `init` function, it will be executed only once when deploying. Functions whose names start with _ are `private` and can't be executed in a Transaction. The others are all `public` and can be executed in a Transaction.

Since the Contract is executed on Chrome V8, all instance variables are in memory, it's not wise to save all of them to state trie in Nebulas. In Nebulas, we provide `LocalContractStorage` and `GlobalContractStorage` objects to help developers define fields needing to be saved to state trie. And those fields should be defined in `constructor` of the Contract, before other functions.

The following is a sample contract:

```
class Rectangle {
    constructor() {
        // define fields stored to state trie.
        LocalContractStorage.defineProperties(this, {
            height: null,
            width: null,
        });
    }

    // init function.
    init(height, width) {
        this.height = height;
        this.width = width;
    }

    // calc area function.
    calcArea() {
        return this.height * this.width;
    }
    // verify function.
    verify(expected) {
        let area = this.calcArea();
        if (expected != area) {
            throw new Error("Error: expected " + expected + ",␣
↪actual is " + area + ".");
        }
    }
}
```

Visibility

In JavaScript, there is no function visibility, all functions defined in prototype object are public.

In Nebulas, we define two kinds of visibility `public` and `private`:

- `public` All functions whose name matches the regexp `^[a-zA-Z$][A-Za-z0-9_$]*$` are public, except `init`. Public functions can be called via Transaction.
- `private` All functions whose name starts with _ are private. A private function can only be called by public functions.

Global Objects

console

The `console` module provides a simple debugging console that is similar to the JavaScript console mechanism provided by web browsers.

The global console can be used without calling `require('console')`.

console.info([...args])

- `...args <any>`

The console.info() function is an alias for `console.log()`.

console.log([...args])

- `...args <any>`

Print `args` to Nebulas Logger at level `info`.

console.debug([...args])

- `...args <any>`

Print `args` to Nebulas Logger at level `debug`.

console.warn([...args])

- `...args <any>`

Print `args` to Nebulas Logger at level `warn`.

console.error([...args])

- `...args <any>`

Print `args` to Nebulas Logger at level `error`.

LocalContractStorage

The `LocalContractStorage` module provides a state trie based storage capability. It accepts string only key value pairs. And all data is stored to a private state trie associated with the current contract address. Only the contract can access it.

BigNumber

The `BigNumber` module uses the bignumber.js, a JavaScript library for arbitrary-precision decimal and non-decimal arithmetic operations. The contract can use `BigNumber` directly to handle the value of the transaction and other value transfers.

```
var value = new BigNumber(0);
value.plus(1);
...
```

Blockchain

The `Blockchain` module provides an object for contracts to obtain transactions and blocks executed by the current contract. Also, the NAS can be transferred from the contract and the address check is provided.

Blockchain API:

```
// current block
Blockchain.block;
```

```
// current transaction, transaction's value/gasPrice/gasLimit auto␣
↪change to BigNumber object
Blockchain.transaction;

// transfer NAS from contract to address
Blockchain.transfer(address, value);

// verify address
Blockchain.verifyAddress(address);
```

properties:

- `block`: current block for contract execution
- `timestamp`: block timestamp
- `seed`: random seed
- `height`: block height
- `transaction`: current transaction for contract execution
- `hash`: transaction hash
- `from`: sender address of the transaction
- `to`: recipient address of the transaction
- `value`: transaction value, a BigNumber object for contract use
- `nonce`: transaction nonce
- `timestamp`: transaction timestamp
- `gasPrice`: transaction gasPrice, a BigNumber object for contract use
- `gasLimit`: transaction gasLimit, a BigNumber object for contract use
- `transfer(address, value)`: transfer NAS from contract to address
- params:
  - `address`: nebulas address to receive NAS
  - `value`: transfer value, a BigNumber object
- return:
  - `0`: transfer success
  - `1`: transfer failed
- `verifyAddress(address)`: verify address
- params:
  - `address`: address need to check
- return:

– 1: address is valid
– 0: address is invalid

Example to use:

```
'use strict';

var SampleContract = function () {
    LocalContractStorage.defineProperties(this, {
        name: null,
        count: null
    });
    LocalContractStorage.defineMapProperty(this, "allocation");
};

SampleContract.prototype = {
    init: function (name, count, allocation) {
        this.name = name;
        this.count = count;
        allocation.forEach(function (item) {
            this.allocation.put(item.name, item.count);
        }, this);
        console.log('init: Blockchain.block.coinbase = ' +␣
↪Blockchain.block.coinbase);
        console.log('init: Blockchain.block.hash = ' + Blockchain.
↪block.hash);
        console.log('init: Blockchain.block.height = ' + Blockchain.
↪block.height);
        console.log('init: Blockchain.transaction.from = ' +␣
↪Blockchain.transaction.from);
        console.log('init: Blockchain.transaction.to = ' +␣
↪Blockchain.transaction.to);
        console.log('init: Blockchain.transaction.value = ' +␣
↪Blockchain.transaction.value);
        console.log('init: Blockchain.transaction.nonce = ' +␣
↪Blockchain.transaction.nonce);
        console.log('init: Blockchain.transaction.hash = ' +␣
↪Blockchain.transaction.hash);
    },
    transfer: function (address, value) {
        var result = Blockchain.transfer(address, value);
        console.log("transfer result:", result);
        Event.Trigger("transfer", {
            Transfer: {
                from: Blockchain.transaction.to,
                to: address,
                value: value
            }
        });
    },
    verifyAddress: function (address) {
        var result = Blockchain.verifyAddress(address);
        console.log("verifyAddress result:", result);
    }
};

module.exports = SampleContract;
```

Event

The `Event` module records execution events in the contract. The recorded events are stored in the event trie on the chain, which can be fetched by `FetchEvents` method in block with the execution transaction hash. All contract event topics have a `chain.contract.` prefix before the topic they set in contract.

```
Event.Trigger(topic, obj);
```

- `topic`: user-defined topic
- `obj`: JSON object

You can see the example in `SampleContract` above.

Math.random

- `Math.random()` returns a floating-point, pseudo-random number in the range from 0 inclusive, up to, but not including 1. The typical usage is:

```
"use strict";

var BankVaultContract = function () {};

BankVaultContract.prototype = {

    init: function () {},

    game: function(subscript){

        var arr =[1,2,3,4,5,6,7,8,9,10,11,12,13];

        for(var i = 0;i < arr.length; i++){
            var rand = parseInt(Math.random()*arr.length);
            var t = arr[rand];
            arr[rand] =arr[i];
            arr[i] = t;
        }

        return arr[parseInt(subscript)];
    },
};
module.exports = BankVaultContract;
```

- `Math.random.seed(myseed)` if needed, you can use this method to reset the random seed. The argument `myseed` must be a **string**.

```
"use strict";

var BankVaultContract = function \(\) {};

BankVaultContract.prototype = {

  init: function () {},

  game:function(subscript, myseed){

      var arr =[1,2,3,4,5,6,7,8,9,10,11,12,13];

      console.log(Math.random());

      for(var i = 0;i < arr.length; i++){

          if (i == 8) {
              // reset random seed with `myseed`
              Math.random.seed(myseed);
          }

          var rand = parseInt(Math.random()*arr.length);
          var t = arr[rand];
          arr[rand] =arr[i];
          arr[i] = t;
      }
      return arr[parseInt(subscript)];
  },

};

module.exports = BankVaultContract;
```

Date

```
"use strict";

var BankVaultContract = function () {};

BankVaultContract.prototype = {
    init: function () {},
    test: function(){
        var d = new Date();
        return d.toString();
    }
};

module.exports = BankVaultContract;
```

Tips:

- Unsupported methodsïijŽ`toDateString()`, `toTimeString()`, `getTimezoneOffset()`,`toLocaleXXX()`.
- `new Date()`/`Date.now()` returns the timestamp of current block in milliseconds.
- `getXXX` returns the result of `getUTCXXX`.

accept

this method aims to make it possible to send a binary transfer to a contract account. As `to` is a smart contact address, which has declared the function `accept()` and it excuted correctly, the transfer will succeed. If the Tx is a non-binary Tx, it will be treated as a normal function.

```
"use strict";
var DepositeContent = function (text) {
    if(text){
            var o = JSON.parse(text);
            this.balance = new BigNumber(o.balance);//ä¡Źéćİä£ąæĄŕ
            this.address = o.address;
    }else{
            this.balance = new BigNumber(0);
            this.address = "";
        }
};

DepositeContent.prototype = {
    toString: function () {
          return JSON.stringify(this);
    }
};

var BankVaultContract = function () {
    LocalContractStorage.defineMapProperty(this, "bankVault", {
         parse: function (text) {
              return new DepositeContent(text);
         },
         stringify: function (o) {
              return o.toString();
         }
    });
};

BankVaultContract.prototype = {
    init: function () {},

    save: function () {
          var from = Blockchain.transaction.from;
          var value = Blockchain.transaction.value;
          value = new BigNumber(value);
          var orig_deposit = this.bankVault.get(from);
          if (orig_deposit) {
                value = value.plus(orig_deposit.balance);
          }

          var deposit = new DepositeContent();
          deposit.balance = new BigNumber(value);
          deposit.address = from;
          this.bankVault.put(from, deposit);
    },

    accept:function(){
        this.save();
        Event.Trigger("transfer", {
            Transfer: {
                from: Blockchain.transaction.from,
                to: Blockchain.transaction.to,
                value: Blockchain.transaction.value,
            }
           });
    }

};
module.exports = BankVaultContract;
```

NRC20

Abstract

The following standard allows for the implementation of a standard API for tokens within smart contracts. This standard provides basic functionality to transfer tokens, as well as allows tokens to be approved so they can be spent by another on-chain third party.

Motivation

A standard interface allows that a new token can be created by any application easily : from wallets to decentralized exchanges.

Methods

name

Returns the name of the token - e.g. `"MyToken"`.

```
// returns string, the name of the token.
function name()
```

symbol

Returns the symbol of the token. E.g. “TK“.

```
// returns string, the symbol of the token
function symbol()
```

decimals

Returns the number of decimals the token uses - e.g. `8`, means to divide the token amount by `100000000` to get its user representation.

```
// returns number, the number of decimals the token uses
function decimals()
```

totalSupply

Returns the total token supply.

```
// returns string, the total token supply, the decimal value is␣
↪decimals* total.
function totalSupply()
```

balanceOf

Returns the account balance of a address.

```
// returns string, the account balance of another account with␣
↪address
function balanceOf(address)
```

transfer

Transfers `value` amount of tokens to `address`, and MUST fire the `Transfer` event. The function SHOULD `throw` if the `from` account balance does not have enough tokens to spend.

Note Transfers of 0 values MUST be treated as normal transfers and fire the `Transfer` event.

```
// returns `true`, if transfer success, else throw error
function transfer(address, value)
```

transferFrom

Transfers `value` amount of tokens from address `from` to address `to`, and MUST fire the `Transfer` event.

The `transferFrom` method is used for a withdraw workflow, allowing contracts to transfer tokens on your behalf. This can be used for example to allow a contract to transfer tokens on your behalf and/or to charge fees in sub-currencies. The function SHOULD `throw` unless the `from` account has deliberately authorized the sender of the message via some mechanism.

Note Transfers of 0 values MUST be treated as normal transfers and fire the `Transfer` event.

```
// returns `true`, if transfer success, else throw error
function transferFrom(from, to, value)
```

approve

Allows `spender` to withdraw from your account multiple times, up the `currentValue` to the `value` amount. If this function is called again it overwrites the current allowance with `value`.

NOTE: To prevent attack vectors, the user needs to give a previous approve value, and the default value that is not approve is 0.

```
// returns `true`, if approve success, else throw error
function approve(spender, currentValue, value)
```

allowance

Returns the amount which `spender` is still allowed to withdraw from `owner`.

```
// returns string, the value allowed to withdraw from `owner`.
function allowance(owner, spender)
```

Events

transferEvent

MUST trigger when tokens are transferred, including zero value transfers.

A token contract which creates new tokens SHOULD trigger a Transfer event with the `from` address set to `totalSupply` when tokens are created.

```
function transferEvent: function(status, from, to, value)
```

approveEvent

MUST trigger on any call to `approve(spender, currentValue, value)`.

```
function approveEvent: function(status, from, spender, value)
```

Implementation

Example implementations are available at

- NRC20.js

```
'use strict';

var Allowed = function (obj) {
    this.allowed = {};
    this.parse(obj);
}

Allowed.prototype = {
    toString: function () {
        return JSON.stringify(this.allowed);
    },

    parse: function (obj) {
        if (typeof obj != "undefined") {
            var data = JSON.parse(obj);
            for (var key in data) {
                this.allowed[key] = new BigNumber(data[key]);
            }
        }
    },

    get: function (key) {
        return this.allowed[key];
    },
    set: function (key, value) {
        this.allowed[key] = new BigNumber(value);
    }
}

var StandardToken = function () {
    LocalContractStorage.defineProperties(this, {
        _name: null,
        _symbol: null,
        _decimals: null,
        _totalSupply: {
            parse: function (value) {
                return new BigNumber(value);
            },
            stringify: function (o) {
                return o.toString(10);
            }
        }
    });

    LocalContractStorage.defineMapProperties(this, {
        "balances": {
            parse: function (value) {
                return new BigNumber(value);
            },
            stringify: function (o) {
                return o.toString(10);
            }
        },
        "allowed": {
            parse: function (value) {
                return new Allowed(value);
            },
            stringify: function (o) {
                return o.toString();
            }
        }
    });
};

StandardToken.prototype = {
    init: function (name, symbol, decimals, totalSupply) {
        this._name = name;
        this._symbol = symbol;
        this._decimals = decimals || 0;
        this._totalSupply = new BigNumber(totalSupply).mul(new␣
↪BigNumber(10).pow(decimals));

        var from = Blockchain.transaction.from;
        this.balances.set(from, this._totalSupply);
        this.transferEvent(true, from, from, this._totalSupply);
    },

    // Returns the name of the token
    name: function () {
        return this._name;
    },

    // Returns the symbol of the token
    symbol: function () {
        return this._symbol;
    },

    // Returns the number of decimals the token uses
    decimals: function () {
        return this._decimals;
    },

    totalSupply: function () {
        return this._totalSupply.toString(10);
    },

    balanceOf: function (owner) {
        var balance = this.balances.get(owner);

        if (balance instanceof BigNumber) {
            return balance.toString(10);
        } else {
            return "0";
        }
    },

    transfer: function (to, value) {
        value = new BigNumber(value);
        if (value.lt(0)) {
            throw new Error("invalid value.");
        }

        var from = Blockchain.transaction.from;
        var balance = this.balances.get(from) || new BigNumber(0);

        if (balance.lt(value)) {
            throw new Error("transfer failed.");
        }

        this.balances.set(from, balance.sub(value));
        var toBalance = this.balances.get(to) || new BigNumber(0);
        this.balances.set(to, toBalance.add(value));
        this.transferEvent(true, from, to, value);
    },

    transferFrom: function (from, to, value) {
        var spender = Blockchain.transaction.from;
        var balance = this.balances.get(from) || new BigNumber(0);

        var allowed = this.allowed.get(from) || new Allowed();
        var allowedValue = allowed.get(spender) || new BigNumber(0);
        value = new BigNumber(value);

        if (value.gte(0) && balance.gte(value) && allowedValue.
↪gte(value)) {

            this.balances.set(from, balance.sub(value));

            // update allowed value
            allowed.set(spender, allowedValue.sub(value));
            this.allowed.set(from, allowed);

            var toBalance = this.balances.get(to) || new␣
↪BigNumber(0);
            this.balances.set(to, toBalance.add(value));

            this.transferEvent(true, from, to, value);
        } else {
            throw new Error("transfer failed.");
        }
    },

    transferEvent: function (status, from, to, value) {
        Event.Trigger(this.name(), {
            Status: status,
            Transfer: {
                from: from,
                to: to,
                value: value
            }
        });
    },

    approve: function (spender, currentValue, value) {
        var from = Blockchain.transaction.from;

        var oldValue = this.allowance(from, spender);
        if (oldValue != currentValue.toString()) {
            throw new Error("current approve value mistake.");
        }

        var balance = new BigNumber(this.balanceOf(from));
        var value = new BigNumber(value);

        if (value.lt(0) || balance.lt(value)) {
            throw new Error("invalid value.");
        }

        var owned = this.allowed.get(from) || new Allowed();
        owned.set(spender, value);

        this.allowed.set(from, owned);

        this.approveEvent(true, from, spender, value);
    },

    approveEvent: function (status, from, spender, value) {
        Event.Trigger(this.name(), {
            Status: status,
            Approve: {
                owner: from,
                spender: spender,
                value: value
            }
        });
    },

    allowance: function (owner, spender) {
        var owned = this.allowed.get(owner);

        if (owned instanceof Allowed) {
            var spender = owned.get(spender);
            if (typeof spender != "undefined") {
                return spender.toString(10);
            }
        }
        return "0";
    }
};

module.exports = StandardToken;
```

NRC721

Abstract

A class of unique tokens. NRC721 is a free, open standard that describes how to build unique tokens on the Nebulas blockchain. While all tokens are fungible (every token is the same as every other token) in NRC20, NRC721 tokens are all unique.

Motivation

NRC721 defines a minimum interface a smart contract must implement to allow unique tokens to be managed, owned, and traded. It does not mandate a standard for token metadata or restrict adding supplemental functions.

Methods

name

Returns the name of the token - e.g. `"MyToken"`.

```
// returns string, the name of the token.
function name()
```

balanceOf

Returns the number of tokens owned by `owner`.

```
// returns The number of NFTs owned by `owner`, possibly zero
function balanceOf(owner)
```

ownerOf

Returns the address of the owner of the tokens.

```
// returns the address of the owner of the tokens
function ownerOf(tokenId)
```

transferFrom

Transfers the ownership of an token from one address to another address. The caller is responsible to confirm that `to` is capable of receiving token or else they may be permanently lost.

Transfers `tokenId` tokenId from address `from` to address `to`, and MUST fire the `Transfer` event.

The function SHOULD `throws` unless the transaction from is the current owner, an authorized operator, or the approved address for this token. `throws` if `from` is not the current owner. `throws` if `to` is the contract address. `throws` if `tokenId` is not a valid token.

```
// if transfer fail, throw error
function transferFrom(from, to, tokenId)
```

approve

Set or reaffirm the approved address for an token.

The function SHOULD `throws` unless transcation from is the current token owner, or an authorized operator of the current owner.

```
function approve(to, tokenId)
```

setApprovalForAll

Enable or disable approval for a third party (`operator`) to manage all of transaction from's assets.

`operator` Address to add to the set of authorized operators. `approved` True if the operators is approved, false to revoke approval

```
function setApprovalForAll(operator, approved)
```

getApproved

Get the approved address for a single token.

```
// return the approved address for this token, or "" if there is␣
↪none
function getApproved(tokenId)
```

isApprovedForAll

Query if an address is an authorized operator for another address.

```
// return true if `operator` is an approved operator for `owner`,␣
↪false otherwise
function isApprovedForAll(owner, operator)
```

Events

_transferEvent

This emits when ownership of any token changes by any mechanism.

```
function _transferEvent: function(status, from, to, value)
```

_approveEvent

This emits when the approved address for an token is changed or reaffirmed.

When a Transfer event emits, this also indicates that the approved address for that token (if any) is reset to none

```
function _approveEvent: function(status, from, spender, value)
```

Implementation

Example implementations are available at

- NRC721BasicToken.js

```
'use strict';

var Operator = function (obj) {
    this.operator = {};
    this.parse(obj);
};

Operator.prototype = {
    toString: function () {
        return JSON.stringify(this.operator);
    },

    parse: function (obj) {
        if (typeof obj != "undefined") {
            var data = JSON.parse(obj);
            for (var key in data) {
                this.operator[key] = data[key];
            }
        }
    },

    get: function (key) {
        return this.operator[key];
    },

    set: function (key, value) {
        this.operator[key] = value;
    }
};

var StandardToken = function () {
    LocalContractStorage.defineProperties(this, {
        _name: null,
    });
    LocalContractStorage.defineMapProperties(this, {
        "tokenOwner": null,
        "ownedTokensCount": {
            parse: function (value) {
                return new BigNumber(value);
            },
            stringify: function (o) {
                return o.toString(10);
            }
        },
        "tokenApprovals": null,
        "operatorApprovals": {
            parse: function (value) {
                return new Operator(value);
            },
            stringify: function (o) {
                return o.toString();
            }
        },

    });
};

StandardToken.prototype = {
    init: function (name) {
        this._name = name;
    },

    name: function () {
        return this._name;
    },

    // Returns the number of tokens owned by owner.
    balanceOf: function (owner) {
        var balance = this.ownedTokensCount.get(owner);
        if (balance instanceof BigNumber) {
            return balance.toString(10);
        } else {
            return "0";
        }
    },

    //Returns the address of the owner of the tokenID.
    ownerOf: function (tokenID) {
        return this.tokenOwner.get(tokenID);
    },

    /**
     * Set or reaffirm the approved address for an token.
     * The function SHOULD throws unless transcation from is the␣
↪current token owner, or an authorized operator of the current␣
↪owner.
     */
    approve: function (to, tokenId) {
        var from = Blockchain.transaction.from;

        var owner = this.ownerOf(tokenId);
        if (to == owner) {
            throw new Error("invalid address in approve.");
        }
        if (owner == from || this.isApprovedForAll(owner, from)) {
            this.tokenApprovals.set(tokenId, to);
            this._approveEvent(true, owner, to, tokenId);
        } else {
            throw new Error("permission denied in approve.");
        }
    },

    // Returns the approved address for a single token.
    getApproved: function (tokenId) {
        return this.tokenApprovals.get(tokenId);
    },

    /**
     * Enable or disable approval for a third party (operator) to␣
↪manage all of transaction from's assets.
     * operator Address to add to the set of authorized operators.
     * @param approved True if the operators is approved, false to␣
↪revoke approval
     */
    setApprovalForAll: function(to, approved) {
        var from = Blockchain.transaction.from;
        if (from == to) {
            throw new Error("invalid address in setApprovalForAll.
↪");
        }
        var operator = this.operatorApprovals.get(from) || new␣
↪Operator();
        operator.set(to, approved);
        this.operatorApprovals.set(from, operator);
    },

    /**
     * @dev Tells whether an operator is approved by a given owner
     * @param owner owner address which you want to query the␣
↪approval of
     * @param operator operator address which you want to query the␣
↪approval of
     * @return bool whether the given operator is approved by the␣
↪given owner
```

```
     */
    isApprovedForAll: function(owner, operator) {
        var operator = this.operatorApprovals.get(owner);
        if (operator != null) {
            if (operator.get(operator) === "true") {
                return true;
            } else {
                return false;
            }
        }
    },


    /**
     * @dev Returns whether the given spender can transfer a given␣
↪token ID
     * @param spender address of the spender to query
     * @param tokenId uint256 ID of the token to be transferred
     * @return bool whether the msg.sender is approved for the␣
↪given token ID,
     *  is an operator of the owner, or is the owner of the token
     */
    _isApprovedOrOwner: function(spender, tokenId) {
        var owner = this.ownerOf(tokenId);
        return spender == owner || this.getApproved(tokenId) ==␣
↪spender || this.isApprovedForAll(owner, spender);
    },

    /**
     * Transfers the ownership of an token from one address to␣
↪another address.
     * The caller is responsible to confirm that to is capable of␣
↪receiving token or else they may be permanently lost.
     * Transfers tokenId from address from to address to, and MUST␣
↪fire the Transfer event.
     * The function SHOULD throws unless the transaction from is␣
↪the current owner, an authorized operator, or the approved␣
↪address for this token.
     * Throws if from is not the current owner.
     * Throws if to is the contract address.
     * Throws if tokenId is not a valid token.
     */
    transferFrom: function (from, to, tokenId) {
        var sender = Blockchain.transaction.from;
        var contractAddress = Blockchain.transaction.to;
        if (contractAddress == to) {
            throw new Error("Forbidden to transfer money to a smart␣
↪contract address");
        }
        if (this._isApprovedOrOwner(sender, tokenId)) {
            this._clearApproval(from, tokenId);
            this._removeTokenFrom(from, tokenId);
            this._addTokenTo(to, tokenId);
            this._transferEvent(true, from, to, tokenId);
        } else {
            throw new Error("permission denied in transferFrom.");
        }

    },



     /**
     * Internal function to clear current approval of a given token␣
↪ID
     * Throws if the given address is not indeed the owner of the␣
↪token
     * @param sender owner of the token
     * @param tokenId uint256 ID of the token to be transferred
     */
    _clearApproval: function (sender, tokenId) {
        var owner = this.ownerOf(tokenId);
        if (sender != owner) {
            throw new Error("permission denied in clearApproval.");
        }
        this.tokenApprovals.del(tokenId);
    },

    /**
     * Internal function to remove a token ID from the list of a␣
↪given address
     * @param from address representing the previous owner of the␣
↪given token ID
     * @param tokenId uint256 ID of the token to be removed from␣
↪the tokens list of the given address
     */
    _removeTokenFrom: function(from, tokenId) {
        if (from != this.ownerOf(tokenId)) {
            throw new Error("permission denied in removeTokenFrom.
↪");
        }
        var tokenCount = this.ownedTokensCount.get(from);
        if (tokenCount.lt(1)) {
            throw new Error("Insufficient account balance in␣
↪removeTokenFrom.");
        }
        this.ownedTokensCount.set(from, tokenCount.sub(1));
    },

    /**
     * Internal function to add a token ID to the list of a given␣
↪address
```

```
     * @param to address representing the new owner of the given␣
↪token ID
     * @param tokenId uint256 ID of the token to be added to the␣
↪tokens list of the given address
     */
    _addTokenTo: function(to, tokenId) {
        this.tokenOwner.set(tokenId, to);
        var tokenCount = this.ownedTokensCount.get(to) || new␣
↪BigNumber(0);
        this.ownedTokensCount.set(to, tokenCount.add(1));
    },

    /**
     * Internal function to mint a new token
     * @param to The address that will own the minted token
     * @param tokenId uint256 ID of the token to be minted by the␣
↪msg.sender
     */
    _mint: function(to, tokenId) {
        this._addTokenTo(to, tokenId);
        this._transferEvent(true, "", to, tokenId);
    },

    /**
     * Internal function to burn a specific token
     * @param tokenId uint256 ID of the token being burned by the␣
↪msg.sender
     */
    _burn: function(owner, tokenId) {
        this._clearApproval(owner, tokenId);
        this._removeTokenFrom(owner, tokenId);
        this._transferEvent(true, owner, "", tokenId);
    },

    _transferEvent: function (status, from, to, tokenId) {
        Event.Trigger(this.name(), {
            Status: status,
            Transfer: {
                from: from,
                to: to,
                tokenId: tokenId
            }
        });
    },

    _approveEvent: function (status, owner, spender, tokenId) {
        Event.Trigger(this.name(), {
            Status: status,
            Approve: {
                owner: owner,
                spender: spender,
                tokenId: tokenId
            }
        });
    }

};

module.exports = StandardToken;
```

Tools

All the development tools: official dev tools and tools from the community. We welcome you to join us and build the Nebulas ecosystem together. You can recommend more tools and edit this page on Github directly.

- **Cross-platform Nebulas Smart Contract IDE**

  Full functions: web

  Local NVM: Mac OS, Windows, Linux

- **nebPay**

  Nebulas payment JavaScript API. Users can use it in browser on both PC and mobile. Users can make NAS payments through the Chrome extension and the iOS/Android wallet.

- **Development Environment for Nebulas**

JavaScript Development Tools

- **VS Code**
- **sublime**

DApp Development Framework

- **Nasa.js** The acclaimed Nebulas DApp client development framework, lightweight and easy to use.
- **Nebulas DApp Local Development Debugging Tool**

Contract Development Tools

- **Smart Contract Integrated Development Environment**
- **Nebulas smart contract IDE**

Contract Deployment Tools

- **Web-wallet**
- **WebExtensionWallet**

Nebpay

- **JavaScript SDK**
- **iOS SDK**
- **Android SDK**

Nebulas API

- **Go**
- **Python**
- **Java**
- **JavaScript**
- **PHP**
- **ruby**
- **NET**
- **unity3d**
- **swift**

Static Scanning Tools

- **Nebulas Smart Contract Code Checker**
- **NebulasÂăSmart Contract Lint Tool**
- **Nebulas javascript/typescript smart contract static check tool**

Command Line Tools

- **A CLI Tool forÂăNebulas**

Testing Tools

- **NebTest will automate unit testing ofÂănebulasÂăsmart contracts**

Others

NebulasDB is a nebulas-based, decentralized, non-relational database, and provides a JS-SDK client

- **The console is easy to use to develop for data operations**
- **Nebulas-Utils is an utiliy package for Nebulas Chain Development**
- **Based onÂăNebulasÂăJS API; putsÂănebulas.js and nebpay.js in one package**

RPC

Remote Procedure Calls (RPCs) provide a useful abstraction for building distributed applications and services.

Nebulas provides both gRPC and RESTful API for users to interact with Nebulas.

grpc provides a concrete implementation of the gRPC protocol, layered over HTTP/2. These libraries enable communication between clients and servers using any combination of the supported languages.

grpc-gateway is a plugin of protoc. It reads gRPC service definition, and generates a reverse-proxy server which translates a RESTful JSON API into gRPC. We use it to map gRPC to HTTP.

Endpoint

Default endpoints:

API	URL	Protocol
gRPC	http://localhost:8684	Protobuf
RESTful	http://localhost:8685	HTTP

gRPC API

We can run the gRPC example testing client code:

```
go run main.go
```

The testing client gets account state from sender address, makes a transaction from sender to receiver, and also checks the account state of receiver address.

We can see client log output like:

```
GetAccountState n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5 nonce 4 value␣
↪3142831039999999999992
SendTransaction n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5 ->␣
↪n1Zn6iyyQRhqthmCfqGBzWfip1Wx8wEvtrJ value 2 txhash:
↪"2c2f5404a2e2edb651dff44a2d114a198c00614b20801e58d5b00899c8f512ae"
GetAccountState n1Zn6iyyQRhqthmCfqGBzWfip1Wx8wEvtrJ nonce 0 value 10
```

HTTP

We have also provided HTTP to access the RPC API. The file that ends with **gw.go** is the mapping file. Now we can access the rpc API directly from our browser, you can update the **rpc_listen** and **http_listen** in **conf/default/config.conf** to change the RPC/HTTP ports, respectively.

Example:

```
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/user/nebstate
```

if successful, response will be returned like this

```
{
    "result":{
        "chain_id":100,
        "tail":
↪"b10c1203d5ae6d4d069d5f520eb060f2f5fb74e942f391e7cadbc2b5148dfbcb
↪",
        "lib":
↪"da30b4ed14affb62b3719fb5e6952d3733e84e53fe6e955f8e46da503300c985
↪",
        "height":"365",
        "protocol_version":"/neb/1.0.0",
        "synchronized":false,
        "version":"0.7.0"
    }
}
```

Or, there is an error from gRPC, and the reponse will carry the error message.

```
{
    "error":"message..."
}
```

RPC methods

- *GetNebState*
- *GetAccountState*

RPC API Reference

GetNebState

Return the state of the neb.

Protocol	Method	API
gRpc		GetNebState
HTTP	GET	/v1/user/nebstate

Parameters

none

Returns

`chain_id` Block chain id: * `1`: mainnet.

- `1001`: testnet.

`tail` current neb tail hash.

`lib` current neb lib hash.

`height` current neb tail block height.

`protocol_version` current neb protocol version.

`synchronized` peer sync status.

`version` neb version.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/user/nebstate

// Result
{
    "result":{
        "chain_id":100,
        "tail":
↪"b10c1203d5ae6d4d069d5f520eb060f2f5fb74e942f391e7cadbc2b5148dfbcb
↪",
        "lib":
↪"da30b4ed14affb62b3719fb5e6952d3733e84e53fe6e955f8e46da503300c985
↪",
        "height":"365",
        "protocol_version":"/neb/1.0.0",
        "synchronized":false,
        "version":"0.7.0"
    }
}
```

GetAccountState

Return the state of the account. Balance and nonce of the given address will be returned.

Protocol	Method	API
gRpc		GetAccountState
HTTP	POST	/v1/user/accountstate

Parameters

`address` Hex string of the account addresss.

`height` block account state with height. If not specified, use 0 as tail height.

Returns

`balance` Current balance in unit of 1/(10^18) nas.

`nonce` Current transaction count.

`type` The type of address, 87 stands for normal address and 88 stands for contract address.

`height` Current height of blockchain.

`pending` pending transactions of address in Tx pool.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/accountstate -d '{"address":
↪"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3"}'
```

```
// Result
{
    result {
        "balance":"9489999998980000000000"
        "nonce":51
        "type":87
        "height":"100",
        "pending":"0"
    }
}
```

LatestIrreversibleBlock

Return the latest irreversible block.

Protocol	Method	API
gRpc		LatestIrreversibleBlock
HTTP	GET	/v1/user/lib

Parameters

none

Returns

`hash` Hex string of block hash.

`parent_hash` Hex string of block parent hash.

`height` block height.

`nonce` block nonce.

`coinbase` Hex string of coinbase address.

`timestamp` block timestamp.

`chain_id` block chain id.

`state_root` Hex string of state root.

`txs_root` Hex string of txs root.

`events_root` Hex string of event root.

`consensus_root`

- `Timestamp` time of consensus state.
- `Proposer` proposer of current consensus state.
- `DynastyRoot` Hex string of dynasty root.

  `miner` the miner of this block.

`is_finality` block is finality.

`transactions` block transactions slice.

- `transaction` *GetTransactionReceipt* response info.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/user/lib

// Result
{
    "result":{
        "hash":
↪"c4a51d6241db372c1b8720e62c04426bd587e1f31054b7d04a3509f48ee58e9f
↪",
        "parent_hash":
↪"8f9f29028356d2fb2cf1291dcee85785e1c20a2145318f36c136978edb6097ce
↪",
        "height":"407",
        "nonce":"0",
        "coinbase":"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5",
        "timestamp":"1521963660",
        "chain_id":100,
        "state_root":
↪"a77bbcd911e7ee9488b623ce4ccb8a38d9a83fc29eb5ad43009f3517f1d3e19a
↪",
        "txs_root":
↪"664671e2fda200bd93b00aaec4ab12db718212acd51b4624e8d4937003a2ab22
↪",
        "events_root":
↪"2607e32c166a3513f9effbd1dc7caa7869df5989398d0124987fa0e4d183bcaf
↪",
        "consensus_root":{
            "timestamp":"1521963660",
            "proposer":"GVeOQnYf20Ppxa2cqTrPHdpr6QH4SKs4ZKs=",
            "dynasty_root":
↪"IfTgx0o271Gg4N3cVKHe7dw3NREnlYCN8aIl8VvRXDY="
        },
        "miner": "n1WwqBXVMuYC3mFCEEuFFtAXad6yxqj4as4"
        "is_finality":false,
        "transactions":[]
    }
}
```

Call

Call a smart contract function. The smart contract must have been submited. Method calls are run only on the current node, not broadcast.

Protocol	Method	API
gRpc		Call
HTTP	POST	/v1/user/call

Parameters

The parameters of the `call` method are the same as the SendTransaction parameters. Special attention:

`to` Hex string of the receiver account addresss. **The value of ‘‘to‘‘ is a contract address.**

`contract` transaction contract object for call smart contract.

- Sub properties(**‘‘source‘‘ and ‘‘sourceType‘‘ are not need**):
- `function` the contract call function for call contract function.
- `args` the params of contract. The args content is JSON string of parameters array.

Returns

`result` result of smart contract method call.

`execute_err` execution error.

`estimate_gas` estimate gas used.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/call -d '{"from":
↪"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3","to":
↪"n1mL2WCZyRi1oELEugfCZoNAW3dt8QpHtJw","value":"0","nonce":3,
↪"gasPrice":"20000000000","gasLimit":"2000000","contract":{
↪"function":"transferValue","args":"[500]"}}'

// Result
{
   "result": {
       "result": "0",
       "execute_err": "insufficient balance",
       "estimate_gas": "22208"
   }
}
```

SendRawTransaction

Submit the signed transaction. The transaction signed value should be return by SignTransactionWithPassphrase.

Protocol	Method	API
gRpc		SendRawTransaction
HTTP	POST	/v1/user/rawtransaction

Parameters

`data` Signed data of transaction

Returns

`txhash` Hex string of transaction hash.

`contract_address` returns only for deployed contract transaction.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/rawtransaction -d '{"data":"CiCrHtxyyIJks2/
↪RErvBBA862D6iwAaGQ9OK1NisSGAuTBIYGiY1R9Fnx0z0uPkWbPokTeBIHFFKRaosGhgzPLPtjEF5c
↪i9wAiEAAAAAAAAAAADeC2s6dkAAAoAjDd/
↪5jSBToICgZiaW5hcnlAZEoQAAAAAAAAAAAAAAAAAA9CQFIQAAAAAAAAAAAAAAAAAABOIFgBYkGLnnv
↪"}'

// Result
{
    "result":{
        "txhash":
↪"f37acdf93004f7a3d72f1b7f6e56e70a066182d85c186777a2ad3746b01c3b52"
    }
}
```

Deploy Contract Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/rawtransaction -d '{"data":"CiDam3G9Sy5fV6/
↪ZcjasYPwSF39ZJDIHNB0Us94vn6p6ohIaGVfLzJ83pom1DO1gD307f1JdTVdDLzbMXO4aGhlXy8yfN
↪CEbThvI0iKcjHhgBZUB"}'

// Result
{
    "result":{
        "txhash":
↪"f37acdf93004f7a3d72f1b7f6e56e70a066182d85c186777a2ad3746b01c3b52
↪",
        "contract_address":
↪"4702b597eebb7a368ac4adbb388e5084b508af582dadde47"
```

```
    }
}
```

GetBlockByHash

Get block header info by the block hash.

Protocol	Method	API
gRpc		GetBlockByHash
HTTP	POST	/v1/user/getBlockByHash

Parameters

`hash` Hex string of block hash.

`full_fill_transaction` If true it returns the full transaction objects, if false only the hashes of the transactions.

Returns

See *LatestIrreversibleBlock* response.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getBlockByHash -d '{"hash":
↪"c4a51d6241db372c1b8720e62c04426bd587e1f31054b7d04a3509f48ee58e9f
↪", "full_fill_transaction":true}'

// Result
{
    "result":{
        "hash":
↪"c4a51d6241db372c1b8720e62c04426bd587e1f31054b7d04a3509f48ee58e9f
↪",
        "parent_hash":
↪"8f9f29028356d2fb2cf1291dcee85785e1c20a2145318f36c136978edb6097ce
↪",
        "height":"407",
        "nonce":"0",
        "coinbase":"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5",
        "timestamp":"1521963660",
        "chain_id":100,
        "state_root":
↪"a77bbcd911e7ee9488b623ce4ccb8a38d9a83fc29eb5ad43009f3517f1d3e19a
↪",
        "txs_root":
↪"664671e2fda200bd93b00aaec4ab12db718212acd51b4624e8d4937003a2ab22
↪",
        "events_root":
↪"2607e32c166a3513f9effbd1dc7caa7869df5989398d0124987fa0e4d183bcaf
↪",
        "consensus_root":{
            "timestamp":"1521963660",
            "proposer":"GVeOQnYf20Ppxa2cqTrPHdpr6QH4SKs4ZKs=",
            "dynasty_root":
↪"IfTgx0o271Gg4N3cVKHe7dw3NREnlYCN8aIl8VvRXDY="
        },
        "miner": "n1WwqBXVMuYC3mFCEEuFFtAXad6yxqj4as4"
        "is_finality":false,
        "transactions":[{
            "hash":
↪"1e96493de6b5ebe686e461822ec22e73fcbfb41a6358aa58c375b935802e4145
↪",
            "chainId":100,
            "from":"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
            "to":"n1orSeSMj7nn8KHHN4JcQEw3r52TVExu63r",
            "value":"10000000000000000000","nonce":"34",
            "timestamp":"1522220087",
            "type":"binary",
            "data":null,
            "gas_price":"1000000",
            "gas_limit":"2000000",
            "contract_address":"",
            "status":1,
            "gas_used":"20000"
        }]
    }
}
```

GetBlockByHeight

Get block header info by the block height.

Protocol	Method	API
gRpc		GetBlockByHeight
HTTP	POST	/v1/user/getBlockByHeight

Parameters

`height` Height of transaction hash.

`full_fill_transaction` If true it returns the full transaction objects, if false only the hashes of the transactions.

Returns

See *LatestIrreversibleBlock* response.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getBlockByHeight -d '{"height": 256, "full_
↪fill_transaction": true}'

// Result
{
    "result":{
        "hash":
↪"c4a51d6241db372c1b8720e62c04426bd587e1f31054b7d04a3509f48ee58e9f
↪",
        "parent_hash":
↪"8f9f29028356d2fb2cf1291dcee85785e1c20a2145318f36c136978edb6097ce
↪",
        "height":"407",
        "nonce":"0",
        "coinbase":"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5",
        "timestamp":"1521963660",
        "chain_id":100,
        "state_root":
↪"a77bbcd911e7ee9488b623ce4ccb8a38d9a83fc29eb5ad43009f3517f1d3e19a
↪",
        "txs_root":
↪"664671e2fda200bd93b00aaec4ab12db718212acd51b4624e8d4937003a2ab22
↪",
        "events_root":
↪"2607e32c166a3513f9effbd1dc7caa7869df5989398d0124987fa0e4d183bcaf
↪",
        "consensus_root":{
            "timestamp":"1521963660",
            "proposer":"GVeOQnYf20Ppxa2cqTrPHdpr6QH4SKs4ZKs=",
            "dynasty_root":
↪"IfTgx0o271Gg4N3cVKHe7dw3NREnlYCN8aIl8VvRXDY="
        },
        "miner": "n1WwqBXVMuYC3mFCEEuFFtAXad6yxqj4as4"
        "is_finality":false,
        "transactions":[{
            "hash":
↪"1e96493de6b5ebe686e461822ec22e73fcbfb41a6358aa58c375b935802e4145
↪",
            "chainId":100,
            "from":"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
            "to":"n1orSeSMj7nn8KHHN4JcQEw3r52TVExu63r",
            "value":"10000000000000000000","nonce":"34",
            "timestamp":"1522220087",
            "type":"binary",
            "data":null,
            "gas_price":"1000000",
            "gas_limit":"2000000",
            "contract_address":"",
            "status":1,
            "gas_used":"20000"
        }]
    }
}
```

GetTransactionReceipt

Get transactionReceipt info by transaction hash. If the transaction is not submitted or only submitted but is not packaged on chain, it will return "not found" error.

Protocol	Method	API
gRpc		GetTransactionReceipt
HTTP	POST	/v1/user/getTransactionReceipt

Parameters

`hash` Hex string of transaction hash.

Returns

`hash` Hex string of tx hash.

`chainId` Transaction chain id.

`from` Hex string of the sender account addresss.

`to` Hex string of the receiver account addresss.

`value` Value of transaction.

`nonce` Transaction nonce.

`timestamp` Transaction timestamp.

`type` Transaction type.

`data` Transaction data, return the payload data.

`gas_price` Transaction gas price.

`gas_limit` Transaction gas limit.

`contract_address` Transaction contract address.

`status` Transaction status, 0 - failed, 1 - success, 2 - pending.

`gas_used` transaction gas used

`execute_error` the execution error of this transaction

`execute_result` return value of the smart-contract function

Note: the data length of `execute_result` is limited to 255 Bytes, if you want to receive a large return value from you smart-contract, please use api `call` instead.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getTransactionReceipt -d '{"hash":
↪"cda54445ffccf4ea17f043e86e54be11b002053f9edbe30ae1fbc0437c2b6a73
↪"}'

// Result
{
    "result":{
        "hash":
↪"cda54445ffccf4ea17f043e86e54be11b002053f9edbe30ae1fbc0437c2b6a73
↪",
        "chainId":100,
        "from":"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
        "to":"n1PxKRaJ5jZHXwTfgM9WqkZJJVXBxRcggEE",
        "value":"10000000000000000000",
        "nonce":"53",
        "timestamp":"1521964742",
        "type":"binary",
        "data":null,
        "gas_price":"1000000",
        "gas_limit":"20000",
        "contract_address":"",
        "status":1,
        "gas_used":"20000",
        "execute_error":"",
        "execute_result":"\"\""
    }
}
```

GetTransactionByContract

Get transactionReceipt info by contract address. If contract does not exist or is not packaged on chain, a "not found" error will be returned.

Protocol	Method	API
gRpc		GetTransactionByContract
HTTP	POST	/v1/user/getTransactionByContract

Parameters

`address` Hex string of contract account address.

Returns

The result is the same as that of GetTransactionReceipt

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getTransactionByContract -d '{"address":
↪"n1sqDHGjYtX6rMqFoq5Tow3s3LqF4ZxBvE3"}'

// Result
{
    "result":{
        "hash":
↪"c5a45a789278f5cce9e95e8f31c1962567f58844456fed7a6eb9afcb764ca6a3
↪",
        "chainId":100,
        "from":"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
        "to":"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
        "value":"0",
        "nonce":"1",
        "timestamp":"1521964742",
        "type":"deploy",
        "data":
↪"eyJTb3VyY2VUeXBlIjoianMiLCJTb3VyY2UiOiJcInVzZSBzdHJpY3RcIjtcblxudmFyIENvbnRyY
↪.....
↪UmFuZG9tMlwiOiByMTIsXG4gImRlZmF1bHRTZWVkUmFuZG9tM1wiOiByMTMsXG4gICAgICAgICAgIC
↪",
        "gas_price":"1000000",
        "gas_limit":"20000",
        "contract_address":"n1sqDHGjYtX6rMqFoq5Tow3s3LqF4ZxBvE3",
        "status":1,
        "gas_used":"20000",
        "execute_error":"",
        "execute_result":"\"\""
    }
}
```

Subscribe

Return the subscribed events of transaction & block. The request is a keep-alive connection.

Note that `subscribe` doesn't guarantee all new events will be received successfully, it depends on the network condition. Please run a local node to use `subscribe` api.

Protocol	Method	API
gRpc		Subscribe
HTTP	POST	/v1/user/subscribe

Parameters

`topics` repeated event topic name, string array.

The topic name list:

- `chain.pendingTransaction` The topic of pending a transaction in transaction_pool.
- `chain.latestIrreversibleBlock` The topic of updating latest irreversible block.
- `chain.transactionResult` The topic of executing & submitting tx.
- `chain.newTailBlock` The topic of setting new tail block.
- `chain.revertBlock` The topic of reverting block.

Returns

`topic` subscribed event topic name.

`data` subscribed event data.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/subscribe -d '{"topics":["chain.linkBlock",
↪ "chain.pendingTransaction"]}'

// Result
{
    "result":{
        "topic":"chain.pendingTransaction",
        "data":"{
                \"chainID\":100,
                 \"hash\":\
↪"b466c7a9b667db8d15f74863a4bc60bc989566b6c3766948b2cacb45a4fbda42\
↪",
                 \"from\":\"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3\",
                 \"to\":\"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3\",
                 \"nonce\":6,
                 \"value\":\"0\",
                 \"timestamp\":1522215320,
                 \"gasprice\": \"20000000000\",
                 \"gaslimit\":\"20000000\",
                 \"type\":\"deploy\"}"
    }
    "result":{
        "topic":"chain.pendingTransaction",
        "data": "..."
    }
    ...
}
```

GetGasPrice

Return current gasPrice.

Protocol	Method	API
gRpc		GetGasPrice
HTTP	GET	/v1/user/getGasPrice

Parameters

none

Returns

`gas_price` gas price. The unit is 10^-18 NAS.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/user/getGasPrice

// Result
{
    "result":{
        "gas_price":"20000000000"
    }
}
```

EstimateGas

Return the estimate gas of transaction.

Protocol	Method	API
gRpc		EstimateGas
HTTP	POST	/v1/user/estimateGas

Parameters

The parameters of the `EstimateGas` method are the same as the SendTransaction parameters.

Returns

`gas` the estimate gas.

`err` error message of the transaction being executed.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/estimateGas -d '{"from":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5","to":
↪"n1SAeQRVn33bamxN4ehWUT7JGdxipwn8b17", "value":
↪"1000000000000000000","nonce":1,"gasPrice":"20000000000","gasLim
↪":"2000000"}'
```

```
// Result
{
    "result": {
        "gas":"20000",
        "err":""
    }
}
```

GetEventsByHash

Return the events list of transaction.

Protocol	Method	API
gRpc		GetEventsByHash
HTTP	POST	/v1/user/getEventsByHash

Parameters

`hash` Hex string of transaction hash.

Returns

`events` the events list. - `topic` event topic; - `data` event data.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getEventsByHash -d '{"hash":
↪"ec239d532249f84f158ef8ec9262e1d3d439709ebf4dd5f7c1036b26c6fe8073
↪"}'

// Result
{
    "result":{
        "events":[{
            "topic":"chain.transactionResult",
            "data":"{
                \"hash\":\
↪"d7977f96294cd232781d9c17f0f3212b48312d5ef0f556551c5cf48622759785\
↪",
                \"status\":1,
                \"gas_used\":\"22208\",
                \"error\":\"\"
            }"
        }]
    }
}
```

GetDynasty

GetDynasty get dpos dynasty.

Protocol	Method	API
gRpc		GetDynasty
HTTP	POST	/v1/user/dynasty

Parameters

`height` block height

Returns

`miners` repeated string of miner address.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/dynasty -d '{"height": 1}'

// Result
{
    {
        "result":{
            "miners":[
                "n1FkntVUMPAsESuCAAPK711omQk19JotBjM",
                "n1JNHZJEUvfBYfjDRD14Q73FX62nJAzXkMR",
                "n1Kjom3J4KPsHKKzZ2xtt8Lc9W5pRDjeLcW",
                "n1TV3sU6jyzR4rJ1D7jCAmtVGSntJagXZHC",
                "n1WwqBXVMuYC3mFCEEuFFtAXad6yxqj4as4",
                "n1Zn6iyyQRhqthmCfqGBzWfip1Wx8wEvtrJ"
            ]
        }
    }
}
```

Next Step:

RPC Management

Beside the NEB API RPC interface nebulas provides additional management APIs. Neb console supports both API and management interfaces. Management RPC uses the same gRPC and HTTP port, which also binds NEB API RPC interfaces.

Nebulas provide both gRPC and RESTful management APIs for users to interact with Nebulas. Our admin proto file defines all admin APIs. **We recommend using the console access admin interfaces, or restricting the admin RPC to local access.**

Default management RPC Endpoint:

RPC Management methods

- *NodeInfo*
- *Accounts*
- *NewAccount*
- *UnLockAccount*
- *LockAccount*
- *SignTransactionWithPassphrase*
- *SendTransactionWithPassphrase*
- *SendTransaction*
- *SignHash*
- *StartPprof*
- *GetConfig*

RPC Management API Reference

NodeInfo

Return the p2p node info.

Parameters

none

Returns

`id` the node ID.

`chain_id` the block chainID.

`coninbase` coinbase

`peer_count` Number of peers currently connected.

`synchronized` the node synchronization status.

`bucket_size` the node route table bucket size.

`protocol_version` the network protocol version.

`RouteTable*[] route_table` the network routeTable

```
message RouteTable {
        string id = 1;
        repeated string address = 2;
}
```

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/admin/nodeinfo

// Result
{
    "result":{
        "id":"QmP7HDFcYmJL12Ez4ZNVCKjKedfE7f48f1LAkUc3Whz4jP",
        "chain_id":100,
        "coinbase":"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5",
        "peer_count":4,
        "synchronized":false,
        "bucket_size":64,
        "protocol_version":"/neb/1.0.0",
        "route_table":[
            {
                "id":"QmP7HDFcYmJL12Ez4ZNVCKjKedfE7f48f1LAkUc3Whz4jP
↪",
                "address":[
                    "/ip4/127.0.0.1/tcp/8680",
                    "/ip4/192.168.1.206/tcp/8680"
                ]
            },
            {
                "id":"QmUxw4PZ8kMEnHD8WaSVE92dtvdnwgufM6m5DrWemdk2M7
↪",
                "address":[
                    "/ip4/192.168.1.206/tcp/10003","/ip4/127.0.0.1/
↪tcp/10003"
                ]
            }
        ]
    }
}
```

Accounts

Return account list.

Parameters

none

Returns

addresses account list

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/admin/accounts

// Result
{
    "result":{
        "addresses":[
            "n1FkntVUMPAsESuCAAPK711omQk19JotBjM",
            "n1JNHZJEUvfBYfjDRD14Q73FX62nJAzXkMR",
            "n1Kjom3J4KPsHKKzZ2xtt8Lc9W5pRDjeLcW",
            "n1NHcbEus81PJxybnyg4aJgHAaSLDx9Vtf8",
            "n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5",
            "n1TV3sU6jyzR4rJ1D7jCAmtVGSntJagXZHC",
            "n1WwqBXVMuYC3mFCEEuFFtAXad6yxqj4as4",
            "n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
            "n1Zn6iyyQRhqthmCfqGBzWfip1Wx8wEvtrJ"
        ]
    }
}
```

NewAccount

NewAccount create a new account with passphrase.

Parameters

passphrase New account passphrase.

Returns

address New Account address.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/new -d '{"passphrase":"passphrase
↪"}'

// Result
```

```
{
    "result":{
        "address":"n1czGUvbQQton6KUWga4wKDLLKYDEn39mEk"
    }
}
```

UnLockAccount

UnlockAccount unlock account with passphrase. After the default unlock time, the account will be locked.

Parameters

`address` UnLock account address.

`passphrase` Unlock account passphrase.

`duration` Unlock account duration. The unit is ns (10e-9 s).

Returns

`result` UnLock account result, unit is ns.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/unlock -d '{"address":
↪"n1czGUvbQQton6KUWga4wKDLLKYDEn39mEk","passphrase":"passphrase",
↪"duration":"1000000000"}'

// Result
{
    "result":{
        "result":true
    }
}
```

LockAccount

LockAccount lock account.

Parameters

`address` Lock account address.

Returns

`result` Lock account result.

HTTP Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/lock -d '{"address":
↪"n1czGUvbQQton6KUWga4wKDLLKYDEn39mEk"}'

// Result
{
    "result":{
        "result":true
    }
}
```

SignTransactionWithPassphrase

SignTransactionWithPassphrase sign transaction. The transaction's `from` address must be unlocked before the 'sign' call.

Parameters

`transaction` this is the same as the *SendTransaction* parameters.

`passphrase` from account passphrase

Returns

`data` Signed transaction data.

sign normal transaction Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/sign -d '{"transaction":{"from":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5","to":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5", "value":
↪"1000000000000000000","nonce":1,"gasPrice":"1000000","gasLimit":
↪"2000000"}, "passphrase":"passphrase"}'

// Result
{
    "result":{
        "data":
↪"CiBOW15yoZ+XqQbMNr4bQdJCXrBTehJKukwjcfW5eASgtBIaGVduKnw+6lM3HBXhJEzzuvv3yNdYA
↪BwhwhqUkp/
↪gEJtE4kndoc7NdSgqD26IQqa0Hjbtg1JaszAvHZiW+XH7C+Ky9XTKRJKuTOc446646d/
↪Sbz/nxQE="
    }
}
```

SendTransactionWithPassphrase

SendTransactionWithPassphrase send transaction with passphrase.

Parameters

`transaction` transaction parameters, which are the same as the *SendTransaction* parameters.

`passphrase` `from` address passphrase.

Returns

`txhash` transaction hash.

`contract_address` returns only for deployed contract transaction.

Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/transactionWithPassphrase -d '{
↪"transaction":{"from":"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5","to":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5", "value":
↪"1000000000000000000","nonce":1,"gasPrice":"1000000","gasLimit":
↪"2000000"},"passphrase":"passphrase"}'

// Result
{
    "result":{
        "hash":
↪"143eac221da8079f017bd6fd6b6a08ea0623114c93c638b94334d16aae109666
↪",
        "contract_address":""
    }
}
```

SendTransaction

Send the transaction. Parameters `from`, `to`, `value`, `nonce`, `gasPrice` and `gasLimit` are required. If the transaction is to send contract, you must specify the `contract`.

Parameters

`from` Hex string of the sender account addresss.

`to` Hex string of the receiver account addresss.

`value` Amount of value sending with this transaction. The unit is Wei (10^-18 NAS).

`nonce` Transaction nonce.

`gas_price` gasPrice sending with this transaction.

`gas_limit` gasLimit sending with this transaction.

`type` transaction payload type. If the type is specified, the transaction type is determined and the corresponding parameter needs to be passed in, otherwise the transaction type is determined according to the contract and binary data. [optional]

- type enum:
  - `binary`: normal transaction with binary
  - `deploy`: deploy smart contract
  - `call`: call smart contract function

`contract` transaction contract object for deployed/calling smart contract. [optional]

- Sub properties:
  - `source` contract source code for deployed contract.
  - `sourceType` contract source type for deployed contract. Currently support `js` and `ts`
    * `js` the contract source written with javascript.
    * `ts` the contract source written with typescript.
  - `function` the contract call function for call contarct function.
  - `args` the params of contract. The args content is JSON string of parameters array.

`binary` any binary data with a length limit = 64bytes. [optional]

Notice:

- `from = to` when deploying a contract, the `to` address must be equal to the `from` address.
- `nonce` the value is **plus one**(+1) on the nonce value of the current from address. Current nonce can be obtained from GetAccountState.
- `gasPrice` and `gasLimit` needed for every transaction. We recommend using GetGasPrice and EstimateGas.
- `contract` parameter only needed for smart contract deployment and calling. When a smart contract is deployed, the `source` and `sourceType` must be specified, the `args` are optional and passed when the initialization function takes a parameter. The `function` field is used to call the contract method.

Returns

`txhash` transaction hash.

`contract_address` returns only for deploying contract transaction.

Normal Transaction Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/transaction -d '{"from":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5","to":
↪"n1SAeQRVn33bamxN4ehWUT7JGdxipwn8b17", "value":
↪"1000000000000000000","nonce":1000,"gasPrice":"1000000","gasLimit
↪":"2000000"}'

// Result
{
    "result":{
      "txhash":
↪"fb5204e106168549465ea38c040df0eacaa7cbd461454621867eb5abba92b4a5
↪",
      "contract_address":""
    }
}
```

Smart Contract Deployment Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/transaction -d '{"from":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5","to":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5", "value":"0","nonce":2,
↪"gasPrice":"1000000","gasLimit":"2000000","contract":{
"source":"\"use strict\";var BankVaultContract=function()
↪{LocalContractStorage.defineMapProperty(this,\"bankVault\")};
↪BankVaultContract.prototype={init:function(){},
↪save:function(height){var deposit=this.bankVault.get(Blockchain.
↪transaction.from);var value=new BigNumber(Blockchain.transaction.
↪value);if(deposit!=null&&deposit.balance.length>0){var␣
↪balance=new BigNumber(deposit.balance);value=value.plus(balance)}
↪var content={balance:value.toString(),height:Blockchain.block.
↪height+height};this.bankVault.put(Blockchain.transaction.from,
↪content)},takeout:function(amount){var deposit=this.bankVault.
↪get(Blockchain.transaction.from);if(deposit==null){return 0}
↪if(Blockchain.block.height<deposit.height){return 0}var␣
↪balance=new BigNumber(deposit.balance);var value=new␣
↪BigNumber(amount);if(balance.lessThan(value)){return 0}var␣
↪result=Blockchain.transfer(Blockchain.transaction.from,value);
↪if(result>0){deposit.balance=balance.dividedBy(value).toString();
↪this.bankVault.put(Blockchain.transaction.from,deposit)}return␣
↪result}};module.exports=BankVaultContract;","sourceType":"js",
↪"args":""}}'

// Result
{
    "result":{
        "txhash":
↪"3a69e23903a74a3a56dfc2bfbae1ed51f69debd487e2a8dea58ae9506f572f73
↪",
        "contract_address":"n21Y7arNbUfLGL59xgnA4ouinNxyvz773NW"
    }
}
```

SignHash

SignHash sign the hash of a message.

Parameters `address` Sign address

`hash` A sha3256 hash of the message, base64 encoded.

`alg` Sign algorithm

- `1` SECP256K1

Returns

`data` Signed transaction data.

sign normal transaction Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/sign/hash -d '{"address":
↪"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5","hash":"W+rOKNqs/
↪tlvz02ez77yIYMCOr2EubpuNh5LvmwceI0=","alg":1}'

// Result
{
    "result":{
        "data":
↪"a7HHsLRvKTNazD1QEogY+Fre8KmBIyK+lNa4zv0Z72puFVkY9uZD6nGixGx/
↪6s1x6Baq7etGwlDNxVvHsoGWbAA="
    }
}
```

StartPprof

StartPprof starts pprof

Parameters

`listen` the address to listen

Returns

`result` start pprof result

Example

```
// Request
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/pprof -d '{"listen":"0.0.0.0:1234"}'

// Result
{
    "result":{
        "result":true
    }
}
```

GetConfig

GetConfig return the config current neb is using

Parameters

none

Returns

`config` neb config

Example

```
// Request
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/admin/getConfig

// Result
{
    "result":{
        "config":{
            "network":{
                "seed":[],
                "listen":["0.0.0.0:8680"],
                "private_key":"conf/network/ed25519key",
                "network_id":1
            },
            "chain":{
                "chain_id":100,
                "genesis":"conf/default/genesis.conf",
                "datadir":"data.db",
                "keydir":"keydir",
                "start_mine":true,
                "coinbase":"n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5",
                "miner":"n1Zn6iyyQRhqthmCfqGBzWfip1Wx8wEvtrJ",
                "passphrase":"",
                "enable_remote_sign_server":false,
                "remote_sign_server":"",
                "gas_price":"",
                "gas_limit":"",
                "signature_ciphers":["ECC_SECP256K1"]
            },
            "rpc":{
                "rpc_listen":["127.0.0.1:8684"],
                "http_listen":["127.0.0.1:8685"],
                "http_module":["api","admin"],
                "connection_limits":0,
                "http_limits":0,
                "http_cors":[]
            },
            "stats":{
                "enable_metrics":false,
                "reporting_module":[],
                "influxdb":{
                    "host":"http://localhost:8086",
                    "port":0,
                    "db":"nebulas",
                    "user":"admin",
                    "password":"admin"
                },
                "metrics_tags":[]
            },
            "misc":null,
            "app":{
                "log_level":"debug",
                "log_file":"logs",
                "log_age":0,
                "enable_crash_report":true,
                "crash_report_url":"https://crashreport.nebulas.io",
                "pprof":{
                    "http_listen":"0.0.0.0:8888",
                    "cpuprofile":"",
                    "memprofile":""
                },
                "version":"0.7.0"
            }
        }
    }
}
```

REPL console

Nebulas provide an interactive javascript console, which can invoke all API and management RPC methods. The console is connected to the local node by default without specifying host.

start console

Start console using the command:

```
./neb console
```

In the case of not specifying the configuration file, the terminal's startup defaults to the configuration of `conf/default/config.conf`. If the local configuration file is not available or you want to specify the configuration file, the terminal starts like this:

```
./neb -c <config file> console
```

console interaction

The console can use the `admin.setHost` interface to specify the nodes that are connected. When the console is started or the host is not specified, the terminal is interacting with the local node. **Therefore, you need to start a local node before starting the console.**

```
> admin.setHost("https://testnet.nebulas.io")
```

Tips: The Testnet only starts the RPC interface of the API, so only the api scheme is available.

console usage

We have API and admin two schemes to access the console cmds. Users can quickly enter instructions using the `TAB` key.

```
> api.
api.call                    api.getBlockByHash          api.
↪getNebState              api.subscribe
api.estimateGas             api.getBlockByHeight        api.
↪getTransactionReceipt
api.gasPrice                api.getDynasty              api.
↪latestIrreversibleBlock
api.getAccountState         api.getEventsByHash         api.
↪sendRawTransaction
```

```
> admin.
admin.accounts                      admin.nodeInfo                     ␣
↪     admin.signHash
admin.getConfig                     admin.sendTransaction              ␣
↪     admin.signTransactionWithPassphrase
admin.lockAccount                   admin.
↪sendTransactionWithPassphrase admin.startPprof
admin.newAccount                    admin.setHost                      ␣
↪     admin.unlockAccount
```

Some management methods may require passphrase. The user can pass in the password when the interface is called, or the console prompts the user for input when the password is not entered. **We recommend using a console prompt to enter your password because it is not visible.**

Enter the password directly:

```
> admin.unlockAccount("n1UWZa8yuvRgePRPgp8a2jX4J9UwGXfHp6i",
↪"passphrase")
{
    "result": {
        "result": true
    }
}
```

Use terminal prompt:

```
> admin.unlockAccount("n1UWZa8yuvRgePRPgp8a2jX4J9UwGXfHp6i")
Unlock account n1UWZa8yuvRgePRPgp8a2jX4J9UwGXfHp6i
Passphrase:
{
    "result": {
        "result": true
    }
}
```

The interfaces with passphrase prompt:

```
admin.newAccount
admin.unlockAccount
admin.signHash
admin.signTransactionWithPassphrase
admin.sendTransactionWithPassphrase
```

The command parameters of the command line are consistent with the parameters of the RPC interface. NEB RPC and NEB RPC_Admin.

console exit

The console can exit with the `ctrl-C` or `exit` command.

Develop Blog

These are some individual topics about developing on Nebulas.

debuging-with-gdb

OverView

Last week we found a lot of âĂIJFailed to update latest irreversible block.âĂİ in neb log with Leon. The reference code (nebulasio/go-nebulas/core/blockchain.go updateLatestIrreversibleBlock) ïijŇ in the code we found the cur variable is not equal to the tail variable , why? to find the cause, we try to use tool to dynamically display variable information and facilitate single-step debugging.

Goroutines

In c++ program we often use gbd to debug, so we think why not to use gdb to debug golang program . First we try to look up the BlockChain loop goroutine state and print the variables .

In c++ we all use `info threads` and thread x to show thread info but in the golang program ïijŇwe should use `info goroutines` and `goroutine xx bt` to displays the current list of running goroutines.

(gdb) `info goroutines` Undefined info command: "goroutines". Try "help info". (gdb) `source` /usr/local/go/src/runtime/runtime-gdb.py Loading Go Runtime support. (gdb) `info goroutines`

```
1 waiting  runtime.gopark
2 waiting  runtime.gopark
3 waiting  runtime.gopark
4 waiting  runtime.gopark
5 syscall  runtime.notetsleepg
6 syscall  runtime.notetsleepg
7 waiting  runtime.gopark
... ...
```

(gdb) goroutine 84 bt

```
 #0 runtime.gopark (unlockf={void (struct runtime.g , void , bool␣
↪*)} 0xc420c57c80, lock=0x0, reason="select", traceEv=24 '\030',␣
↪traceskip=1) at /data/packages/go/src/runtime/proc.go:288
 #1 0x0000000000440fd9 in runtime.selectgo (sel=0xc420c57f48, ~
↪r1=842353656960) at /data/packages/go/src/runtime/select.go:395
 #2 0x0000000000ad2d73 in github.com/nebulasio/go-nebulas/core.
↪(*BlockChain).loop (bc=0xc4202c6320)at /neb/golang/src/github.com/
↪nebulasio/go-nebulas/core/blockchain.go:184
 #3 0x0000000000460421 in runtime.goexit () at /data/packages/go/
↪src/runtime/asm_amd64.s:2337
 #4 .....
```

But neb has too many goroutines, we donâĂŹt kown which one , we give up

BreakPoints

Second we try to set break point to debug (gdb) `b blockchain.go:381`

Breakpoint 2 at 0xad4373: file /neb/golang/src/github.com/nebulasio/go-nebulas/core/blockchain.go, line 381.

(gdb) `b core/blockchain.go:390`

Breakpoint 3 at 0xad44c6: file /neb/golang/src/github.com/nebulasio/go-nebulas/core/blockchain.go, line 390.

(gdb) `info breakpoints` // show all breakpoints

(gdb) `d 2` //delete No 2 breakpoint

Now let the neb continue its execution until the next breakpoint, enter the c command: (gdb) `c` Continuing

```
Thread 6 "neb" hit Breakpoint 2, github.com/nebulasio/go-nebulas/
↪core.(*BlockChain).updateLatestIrreversibleBlock (bc=0xc4202c6320,
↪ tail=0xc4244198c0)
at /neb/golang/src/github.com/nebulasio/go-nebulas/core/blockchain.
↪go:382
382         miners := make(map[string
```

now we can use p(print) to print variables value

```
(gdb) `p cur`
$2 = (struct github.com/nebulasio/go-nebulas/core.Block *)␣
↪0xc420716f90
(gdb) `p cur.height`
$3 = 0
(gdb) `p bc`
$4 = (struct github.com/nebulasio/go-nebulas/core.BlockChain *)␣
↪0xc4202c6320
(gdb) `p bc.latestIrreversibleBlock`
$5 = (struct github.com/nebulasio/go-nebulas/core.Block *)␣
↪0xc4240bbb00
(gdb) `p bc.latestIrreversibleBlock.height`
$6 = 51743
(gdb) `p tail`
$7 = (struct github.com/nebulasio/go-nebulas/core.Block *)␣
↪0xc4244198c0
(gdb) `p tail.height`
$8 = 51749
```

now we can use `info goroutines` again, to find current goroutine. info goroutines with the * indicating the current execution, so we find the current goroutine nunmber quickly.

the next breakpoint we can use `c` command , so we found the cur and lib is not equal, because of length of the miners is less than ConsensusSizeïijŇ In the loop the cur change to the parent block .

Other

When compiling Go programs, the following points require particular attention:

- Using -ldflags "-s" will prevent the standard debugging information from being printed
- Using -gcflags "-N-l" will prevent Go from performing some of its automated optimizations -optimizations of aggregate variables, functions, etc. These optimizations can make it very difficult for GDB to do its job, so it's best to disable them at compile time using these flags.

References

- Debugging with GDB
- GDBèřČèŕŢGOçÍŃåžŔ

neb-dont-generate-coredump-file

OverView

During Testing, neb may be crash, and we want to get the coredump file which could help us to find the reason. However, neb don't generate coredump file by default. We can find the crash log in /var/log/apport.log when a crash occurred:

```
"called for pid 10110, signal 11, core limit 0, dump mode 1 "
```

The coredump file is very very important, it can serve as useful debugging aids in several situations, and help us to debug quickly. Therefore we should make neb to generate coredump file.

Set the core file size

We can use `ulimit -a` command to show core file size. If it's size is zero, which means coredump file is disabled, then we should set a value for core file size. for temporarily change we can use `ulimit -c unlimited` , and for permanently change we can edit `/etc/security/limits.conf` file, it will take effect after reboot or command `sysctl -p`.

```
<domain>     <type>     <item>         <value>
 * soft     core              unlimited
```

But these ways are't work, neb still can't generate coredump file and `cat /proc/$pid/limits` always "Max core file size 0"

Why? Why? Why? It doesn‘t Work

1. If the setting is wrong? Just try a c++ programe build, run it and we can find that it can generate coredump.
2. Neb is started by supervisord, is it caused by supervisordïij§
3. Try to start neb without supervisord, then the neb coredump is generated!
4. Yes, the reason is supervisord, then we can google "supervisord+coredump" to solve it.

Solution

Supervisord only set RLIMIT_NOFILE, RLIMIT_NOPROC by set_rlimits , others are seted default 0 1. modify supervisord code options.py in 1293 line

```
vim /usr/lib/python2.6/site-packages/supervisor/options.py

soft, hard = resource.getrlimit(resource.RLIMIT_CORE)
resource.setrlimit(resource.RLIMIT_CORE, (-1, hard))
```

1. restart supervisord and it works .

Other seetings

You can also change the name and path of coredump file by changing file `/proc/sys/kernel/core_pattern`:

```
echo "/neb/app/core-%e-%p-%t" > /proc/sys/kernel/core_pattern

%p: pid
%: '%' is dropped
%%: output one '%'
%u: uid
%g: gid
%s: signal number
%t: UNIX time of dump
%h: hostname
%e: executable filename
%: both are dropped
```

References

- supervisord coredump
- core_pattern

Crash Reporter in Nebulas

In this doc, we introduce the crash reporter in Nebulas, which is used to collect crash reports in Nebulas and send it back to Nebulas Team, so the whole community can help improving the quality of Nebulas.

Overview

We, the Nebulas Team and the Nebulas community, always try our best to ensure the stability of Nebulas, since people put their faith and properties on it. That means critical bugs are unacceptable, and we are aware of that. However, we can‘t blindly think Nebulas is stable enough or there won‘t be any bugs. Thus, we have plan B, the crash reporter, to collect crash report and send it back to Nebulas community. We hope the whole community can leverage the crash reports and keep improving Nebulas.

Using crash reporter is a very common practice. For example, Microsoft Windows includes a crash reporting service called Windows Error Reporting that prompts users to send crash reports to Microsoft for online analysis. The information goes to a central database run by Microsoft. Apple also involves a standard crash reporter in macOS, named Crash Reporter. The Crash Reporter can send the crash logs to Apple Inc, for their engineers to review. Open-source community also have their own crash reporter, like Bug Buddy for Gnome, Crashpad for Chrome, Talkback for Mozilla, and etc.

In Nebulas, the crash reporter just works like the other crash reporters. It‘s aware of the crash, collects necessary information about the crash, and sends it back the Nebulas server. The server is hosted by Nebulas, and accessible for the whole community.

As a opensource, decentralized platform, we are aware of that the crash reporter may violate some users‘ privacy concern. Thus, we remove all private information in the crash report, like the user name, user id, user‘s home path and IP address. Furthermore, the crash reporter is optional and users may choose close it if users still have some concerns.

How to use it

To enable or disable the crash reporter, you need to look into the configuration file, `config.conf`, and change `enable_crash_reporter` to `true` to enable it, while `false` to disable it.

How it works

In this section, we would like to share some tech details. If you are not interested in the details, you can ignore this section.

The crash reporter is actually a daemon process, which is started by `neb`. When starting the crash reporter, `neb` will tell it the process id (pid) of `neb` process, and the crash file path. For the crash reporter, it will periodically check if the `neb` process and the crash file exists.

At the time it finds the crash file, it will eliminate the private information and send it back to Nebulas.

Currently, the crash report is generated by the `stderr` output from `neb`. We'd like the work with the whole community to collect detailed information in the future.

Contribution Guideline

The go-nebulas project welcomes all contributors. The process of contributing to the Go project may be different than many projects you are used to. This document is intended as a guide to help you through the contribution process. This guide assumes you have a basic understanding of Git and Go.

Becoming a contributor

Before you can contribute to the go-nebulas project you need to setup a few prerequisites.

Preparing a Development Environment for Contributing

Setting up dependent tools

1. Go dependency management tool

dep is an (not-yet) official dependency management tool for Go. go-nebulas project use it to management all dependencies.

For more information, please visit here

2. Linter for Go source code

Golint is official linter for Go source code. Every Go source file in go-nebulas must be satisfied the style guideline. The mechanically checkable items in style guideline are listed in Effective Go and the CodeReviewComments wiki page.

For more information about Golint, please visit here.

3. XUnit output for Go Test

Go2xunit could convert go test output to XUnit compatible XML output used in Jenkins/Hudson.

Making a Contribution

Discuss your design

The project welcomes submissions but please let everyone know what you‘re working on if you want to change or add to the go-nebulas project.

Before undertaking to write something new for the go-nebulas, please file an issue (or claim an existing issue). Significant changes must go through the change proposal process before they can be accepted.

This process gives everyone a chance to validate the design, helps prevent duplication of effort, and ensures that the idea fits inside the goals for the language and tools. It also checks that the design is sound before code is written; the code review tool is not the place for high-level discussions.

Besides that, you can have an instant discussion with core developers in **developers** channel of Nebulas.IO on Community.nebulas.io.

Making a change

Getting Go Source

First you need to fork and have a local copy of the source checked out from the forked repository.

You should checkout the go-nebulas source repo inside your $GOPATH. Go to $GOPATH run the following command in a terminal.

```
$ mkdir -p src/github.com/nebulasio
$ cd src/github.com/nebulasio
$ git clone git@github.com:{your_github_id}/go-nebulas.git
$ cd go-nebulas
```

Contributing to the main repo

Most Go installations project use a release branch, but new changes should only be made based on the **develop** branch. (They may be applied later to a release branch as part of the release process, but most contributors won‘t do this themselves.) Before making a change, make sure you start on the **develop** branch:

```
$ git checkout develop
$ git pull
```

Make your changes

The entire checked-out tree is editable. Make your changes as you see fit ensuring that you create appropriate tests along with your changes. Test your changes as you go.

Copyright

Files in the go-nebulas repository don‘t list author names, both to avoid clutter and to avoid having to keep the lists up to date. Instead, your name will appear in the change log and in the CONTRIBUTORS file and perhaps the AUTHORS file. These files are automatically generated from the commit logs perodically. The AUTHORS file defines who âĂIJThe go-nebulas AuthorsâĂİâĂŤthe copyright holdersâĂŤare.

New files that you contribute should use the standard copyright header:

```
// Copyright (C) 2017 go-nebulas authors
//
// This file is part of the go-nebulas library.
//
// the go-nebulas library is free software: you can redistribute it␣
↪and/or modify
// it under the terms of the GNU General Public License as␣
↪published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// the go-nebulas library is distributed in the hope that it will␣
↪be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with the go-nebulas library.  If not, see <http://www.gnu.
↪org/licenses/>.
//
```

Files in the repository are copyright the year they are added. Do not update the copyright year on files that you change.

Goimports, Golint and Govet

Every Go source file in go-nebulas must pass Goimports, Golint and Govet check. Golint check the style mistakes, we should fix all style mistakes, including comments/docs. Govet reports suspicious constructs, we should fix all issues as well.

Run following command to check your code:

```
$ make fmt lint vet
```

lint.report text file is the Golint report, **vet.report** text file is the Govet report.

Testing

You‘ve written test code, tested your code before sending code out for review, run all the tests for the whole tree to make sure the changes don‘t break other packages or programs:

```
$ make test
```

test.report text file or **test.report.xml** XML file is the testing report.

Commit your changes

The most importance of committing changes is the commit message. Git will open an editor for a commit message. The file will look like:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch foo
# Changes not staged for commit:
#    modified:   editedfile.go
#
```

At the beginning of this file is a blank line; replace it with a thorough description of your change. The first line of the change description is conventionally a one-line summary of the change, prefixed by the primary affected package, and is used as the subject for code review email. It should complete the sentence “This change modifies Go to _.“ The rest of the description elaborates and should provide context for the change and explain what it does. Write in complete sentences with correct punctuation, just like for your comments in Go. If there is a helpful reference, mention it here. If you‘ve fixed an issue, reference it by number with a # before it.

After editing, the template might now read:

```
math: improve Sin, Cos and Tan precision for very large arguments

The existing implementation has poor numerical properties for
large arguments, so use the McGillicutty algorithm to improve
accuracy above 1e10.

The algorithm is described at http://wikipedia.org/wiki/
↪McGillicutty_Algorithm

Fixes #159

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch foo
# Changes not staged for commit:
#    modified:   editedfile.go
#
```

The commented section of the file lists all the modified files in your client. It is best to keep unrelated changes in different commits, so if you see a file listed that should not be included, abort the command and move that file to a different branch.

The special notation "Fixes #159" associates the change with issue 159 in the go-nebulas issue tracker. When this change is eventually applied, the issue tracker will automatically mark the issue as fixed. (There are several such conventions, described in detail in the GitHub Issue Tracker documentation.)

Creating a Pull Request

For more information about creating a pull request, please refer to the Create a Pull Request in Github page.

Downloads

Bleeding edge code can be cloned from the branch of their git repositories:

- Mainnet
- Explorer
- Web Wallet
- neb.js

Mainnet

Nebulas mainnet Eeagle Nebulas launched on Mar 30, 2018. It's a basic public chain. There are two features:

- Supports javascript development
- Over 2000 TPS.

Nebulas NOVA launched in the end of 2018. There are three features:

- Nebulas Rank: measure the value of on-chain data
- Nebulas Blockchain Runtime Environment: instant upgrade the core protocols immediately
- Developer Incentive Protocol: provide native on-chain incentive for developers

Click here to learn about Nebulas NOVA . Some articles:

- Nebulas NOVA, To Discover Data Value In the Blockchain World, [Youtube]
- 6 Minutes Learning Nebulas NOVA with 92k Lines of Code by Joel Wang [Youtube]

The third important version will be launch in 2020 with PoD consensus mechanism. Click here to learn about the PoD Node Strategy .

Click here to learn how to join the mainnet.

Testnet

A functional equivalent Nebulas Testnet is available now, allowing developers to interact with Nebulas freely. View: How to join the testnet.

Roadmap

Nebulas releases are here and roadmap are here.

1.4.4 Node Strategy

Nebulas PoD Node Decentralization Strategy - Based on the Proof of Devotion (PoD) Mechanism

V1.0.2 by Nebulas Foundation, PDF version

Nebulas began it journey with the Vision of "**Let everyone get values from decentralized collaboration fairly.**" With the continued evolution of the âĂIJ**Autonomous Metanet**âĂİ[1], Nebulas is proceeding to itâĂŹs ultimate goal.

At the core of NebulasâĂŹ PoD Node Decentralization Strategy is the **Proof of Devotion (PoD)**[2] Mechanism. This idea behind Proof of Devotion is to provide a measurable value of all users based on the size of their contribution to the ecosystem which includes pledging, consensus and governance mechanisms. With PoD, we plan to not just decentralize NebulasâĂŹ blockchain nodes but to also decentralize community governance via the formation of a representative system and government committees.

Nebulas is building a new **Decentralized Autonomous Organization (DAO)**[3] for complex data networks that will fully embrace community, decentralization and autonomy on a contribution measured basis.

[1] Autonomous Metanet: An open collaboration system based on blockchain technology, which is oriented around complex data and interaction.

[2] Proof of Devotion (PoD): A consensus mechanism built on the basis of the size of community contributions. This includes both consensus and governance mechanisms. The establishment of consensus committees through community contributors to achieve nebulasâĂŹ blockchain nodes decentralization; Participation in community governance through the representation of governance committees.

[3] Decentralized Autonomous Organization (DAO): An organization that is represented by public and transparent computer code. Financial transaction records and procedural rules of a distributed autonomous organization are stored within the blockchain.

Learn more about the Node Strategy and PoD mechanism:

1. PoD Overview

Proof of Devotion (PoD) Mechanism Overview

1.1 Design Objectives

In order to build a sustainable and beneficial public chain, it is necessary to take into account both the speed and irreversibility of the consensus mechanism as well as the fairness of governance.

At present, we face new application scenarios including simple data interactions to complex, multi-level, on-chain functions. This diverse environment is spawning the creation of new user roles as well as significantly increasing the complexity of the system. Communication scenarios have evolved from in person collaboration to collaboration that ecompasess the world. The goal of collaboration has also changed with the end results going from the physical to the virtual world. This results in time spans for collaborative projects becoming longer and more flexible.*[1]*

To ensure a fair governance system within these new scenarios, a new approach to collaboration is required. Traditional centralized governance cannot cope with these new and complex scenarios that we face daily in our technologically evolving world. In this new world filled with complex data interaction patterns and expanding user roles, centralized single evaluation options are difficult to be adaptable and comprehensive leading to considerable limitations.

Existing decentralized collaboration methods do not take into account the new distribution of benefits caused by the existence of expanded user roles. As a result, there is an uneven distribution of benefits leading to slow development and eventually, an unsustainable ecosystem.

We must protect the interests of all community members so that value comes from the depth of Nebulas‘ ecosystem which in turn follows our core beliefs. Under the premise of ensuring efficiency and irreversibility first, we have designed PoD to pursue fairness from the perspective of contribution and to protect the interests of the community.

1.2 Composition

NebulasâĂŹ Proof of Devotion (PoD) can provide a simple overview of mechanisms built on the basis and magnitude of community contributions which include both consensus mechanisms and governance mechanisms. See Figure 1.1.

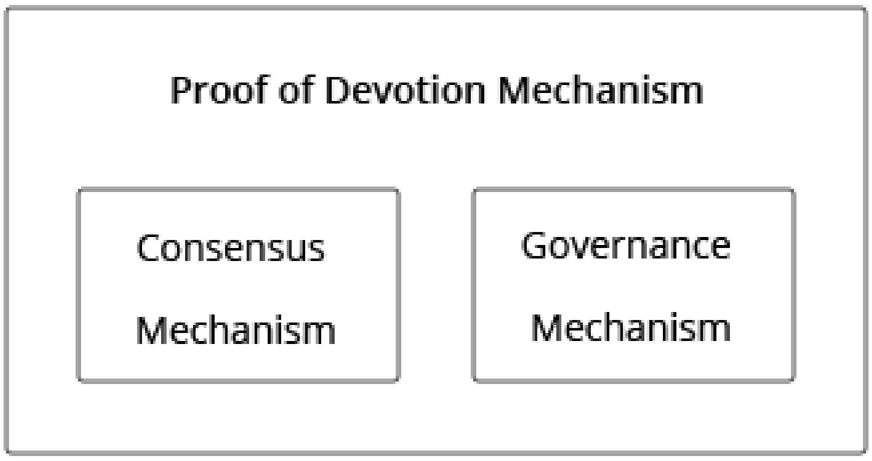


Figure 1.1 PoD Composition

The composition of the PoD mechanism will involve two executive committees split into consensus and governance.

- The consensus mechanism shall be implemented by the Consensus Committee. The consensus committee is selected from all the available nodes via a comprehensive ranking algorithm.
- The governance mechanism shall be implemented by the Governance Committee. The governance committee is composed of the most dedicated contributors of the Consensus Committee.

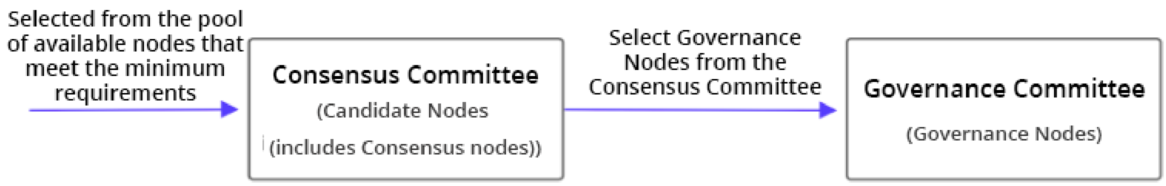

Selected from the pool of available nodes that meet the minimum requirements
Consensus Committee
(Candidate Nodes
(includes Consensus nodes))
Select Governance Nodes from the Consensus Committee
Governance Committee
(Governance Nodes)

Figure 1.2 PoD Executive Committee

1.3 Incentive Allocation

Since the launch of the Nebulas mainnet on March 31, 2018, DPoS*[2]* has been used as the interim consensus mechanism until the release of PoD. The DPoS consensus mechanism generates 8,219.1744 NAS in revenue per day; generating 2,999,941 NAS per year.

This collected revenue will be used exclusively for the Nebulas PoD Node Decentralization Strategy.

The incentive ratio of the two PoD Executive Committee (consensus and governance) will be about 5:1. Which equates to:

- The total incentive amount for the consensus mechanism per year will be: 2,499,951 NAS
- The total incentive amount for the governance mechanism per year will be: 499,999 NAS

The incentive for the consensus mechanism will be evenly divided by the consensus nodes who have generated blocks during the active polling cycle (selection) of the consensus committee. Any candidate nodes that have not been selected during the polling cycle (selection) will not receive any incentive during this period.

The incentive for the governance mechanism will be evenly divided by the governance nodes who has participated in all voting proposals during the governance cycle. Any governance nodes that do not participate in **ALL** voting proposals during the governance cycle will not receive any governance incentive during this period.

1.4 Contribution Measurement: NAX

The measurement for the weight of contribution to the Nebulas ecosystem is the NAX*[3]* Smart Asset. As a smart asset, NAX can only be obtained by **decentralized staking (dStaking)***[4]* the NAS asset. As per the ***Nebulas NAX White Paper*** (Github, PDF), NAX adopts a dynamic distribution model where the daily total issuance quantity is related to the pledge rate of the entire Nebulas ecosystem; the number of NAX obtained by an address is related to the quantity of NAS pledged and the age/duration of the pledge (the longer, the better), which can be considered a measure of the contribution of that address to the community and ecosystem. Therefore, NAX can be considered effective proof of those who contribute to the Nebulas ecosystem.

The Go Nebulas community collaboration platform will also utilize NAX as an ecosystem contribution incentive to encourage community members to continue to build communities.

---

[1] Orange Paper: Nebulas Governance

[2] **Delegated Proof of Stake Consensus (DPoS):** Delegates are chosen by stakeholder votes, and delegates then decide on the issue of consensus in a democratic way. This includes but is not limited to: All network parameters, cost estimates, block intervals, transaction size, etc.

[3] **NAX:** This smart asset is generated by decentralized pledging and is the first token on nextDAO. Users on the Nebulas blockchain can obtain NAX by pledging NAS. NAX adopts a dynamic distribution strategy where the actual issuance quantity is related to the global pledge rate, the amount of NAS pledged individually and the age of the pledge.

[4] **dStaking Decentralized Pledge:** Unlike traditional pledges (staking) that requires the transfer of assets to smart contracts, decentralized pledges record the user‘s pledge while the assets remain at the user‘s personal address.

2. Consensus

This chapter will introduce the Consensus Mechanism of PoD mechanism, follow here:

- *2.1 Minimum Requirements for Node Selection*
- *2.2 Node Selection Rules*

The consensus mechanism utilizes smart contract management which is primarily comprised of node selection rules and the consensus algorithm. This smart contract jointly completes the block generation and ensures the normal operation of the mainnet.

The average block time on the Nebulas mainnet is 15 seconds. During each polling period, the 21 selected consensus nodes take turns generating 10 blocks each. As a result, one polling cycle is 210 blocks which takes about 52.5 minutes. The consensus mechanism execution process during each polling cycle is shown in the following figure:

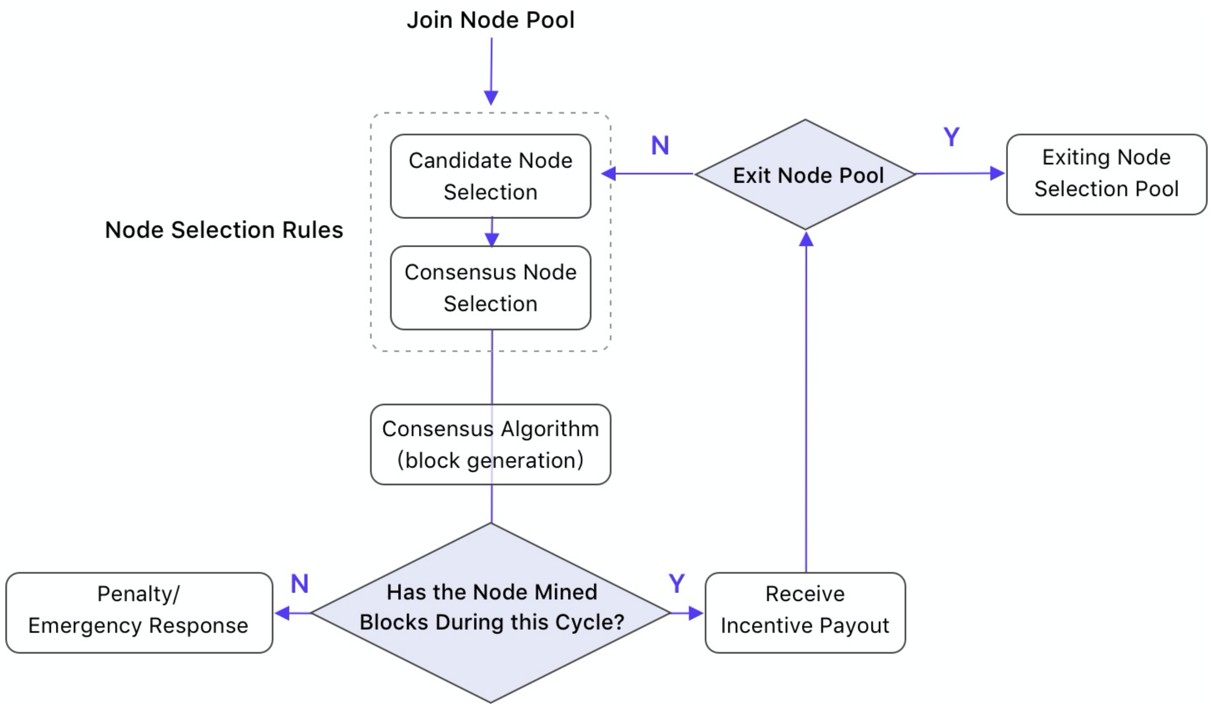

Join Node Pool
Node Selection Rules
Candidate Node Selection
Consensus Node Selection
N
Exit Node Pool
Y
Exiting Node Selection Pool
Consensus Algorithm (block generation)
Penalty/ Emergency Response
N
Has the Node Mined Blocks During this Cycle?
Y
Receive Incentive Payout

Figure 2.1 The consensus mechanism execution process for each polling cycle

2.1 Minimum Requirements for Node Selection

Any individual or organization can apply to become a consensus node and must meet all of the following eligibility requirements to participate in the candidate selection process:

- The server meets the minimum requirements (see Appendix A - recommended hardware configuration);
- The server is guaranteed to be in operation;
- The node pledge (vote) is not less than 100,000 NAX;
- Pledge of 20,000 NAS as deposit;
- No record of severe level abuse or manipulation on the network (see *2.5.1 penalties*)

2.2 Node Selection Rules

The node selection rule consists of two steps:

1. **Candidate node selection:** During each polling cycle, among all nodes that meet the minimum selection requirements, a total of 51 nodes are selected according to the comprehensive candidate node ranking algorithm via smart contract;
2. **Consensus node selection:** During each polling cycle, the algorithm selects 21 consensus nodes which are selected in a consistent method and best represents the user‘s rights and interests in a group of candidate nodes which is based on the consensus node selection algorithm via smart contract. The consensus nodes are responsible for block

generation and can obtain consensus incentives as long as they participate in the process of block generation (online, creating blocks, not manipulating the network, etcâĂę).

The candidate node and the consensus node together constitute the consensus committee. The selection process is shown in the following figure:

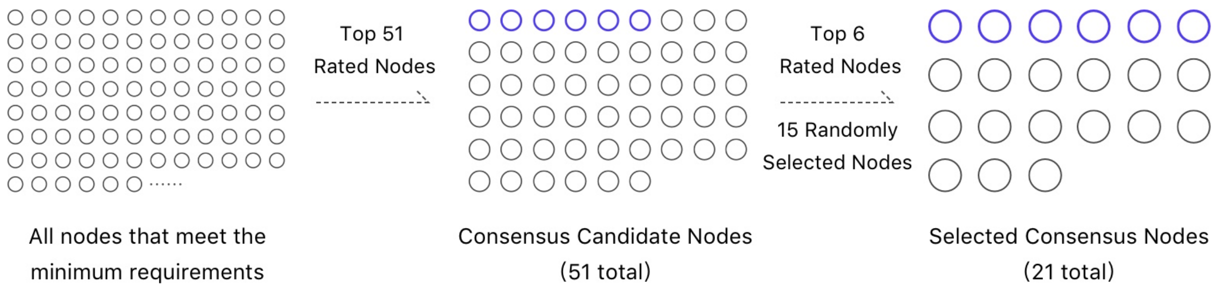


Figure 2.2 Node Selection Process

2.2.1 Candidate Node Comprehensive Ranking Algorithm

Under the premise of meeting the minimum requirements of becoming a candidate node (*2.1 Minimum requirements for node selection*), all nodes are ranked by the comprehensive candidate node ranking algorithm; the top 51 nodes selected via the ranking algorithm will be selected as candidate nodes.

The candidate node ranking algorithm references two primary factors: NAX poll number V(i) and block stabilization index S(i). The final candidate node ranking index âĂIJR(i)âĂİ is:

R(i) = V(i) ÃŮ S(i)

Assuming that S(i) is the same for multiple nodes and the NAX vote V(i) is also the same for multiple nodes, the first node to reach the required NAX vote V(i) is selected.

2.2.1.1 NAX Votes

Number of NAX votes V(i): All community members and node operators can pledge NAX to further support the activation of a node which helps the node improve their overall ranking.

2.2.1.2 Block Generation Stability Index

Block generation stability index S(i): This rating is determined by the ratio of successful block generation when itâĂŹs chosen as a consensus node. If this node has not yet had the chance to be a consensus node, the initial value of S(i) is 0.8. During each polling cycle, a candidate node has three possible values:

1. Not participating in block generation;
2. Successful/accepted block generation;
3. Invalid block generation.

A. Not participating in block generation

The S(i+1) of the following cycle of candidate nodes that have not participated in block generation is:

S(i+1) = S(i) + 0.01

B. Successful block generation

Consensus nodes need to generate 10 blocks per polling cycle (polling cycles consist of 210 blocks). If a node generates 10 blocks successfully during the cycle, the S(i+1) of the next cycle is:

S(i+1) = S(i) + 0.1 (S<=1)

S(i) is gradually increased to the maximum level of 1. In general, functioning nodes with stable block generation will reach the maximum level of S(i)=1.

C. Invalid block generation

If the node generates an invalid block, the S(i+1) of the next cycle is:

S(i+1) = S(i) ÃŮ (10 - C) / 10

Where C is the number of invalid blocks. The larger C, the lower S(i) value. If S(i) falls to the K threshold (K initial value is 0.5), the consensus node cannot be selected as a candidate node for the next 20 polling cycles, as detailed in *2.5.1 Penalties*.

2.2.2 Consensus Node Selection Algorithm

Consensus nodes are selected from the 51 candidate nodes that are initially selected during the polling cycle. The selection method is as follows:

1. The top 6 candidate nodes are selected automatically as per their score (detailed above).
2. 14 consensus nodes are selected from the candidate pool containing the remaining 45 candidate nodes according to the following formula:

   RConsensus = (R(i) / Sum(R)) ÃŮ Random()

The formula explanation is as follows:

RConsensusïijŽConsensus node ranking index.

R(i)ïijŽCandidate nodes ranking index. R(i) is the derived score of two primary factors: NAX poll number V(i) and block stabilization index S(i); as a result, R(i) is treated as the community support rate of the node and its contribution of historical block generation.

Sum(R): The sum score of 51 candidate nodes ranking index; as a result, R(i)/Sum(R) can be treated as an individual node contribution ratio within the 51 candidate nodes.

Random()ïijŽA random probability.

3.The last consensus node, lucky node, will be selected randomly from the 31 candidate nodes that were not selected.

2.3 Consensus Algorithm

The consensus algorithm is based on the well understood and mature DPoS consensus mechanism where the block generation of the following polling cycle is scheduled to be produced by the nodes within the consensus committee; the selected 21 consensus nodes take turns to generate blocks. After the polling cycle is complete, the next selected 21 consensus nodes take turns to generate blocks in the following cycle.

Byzantine fault tolerant BFT*[1]* operation is used to ensure the consistency and stability of the blockchain and the PoD mechanism.

2.3.1 Block Generation Order

The order of block generation of the 21 consensus nodes is randomly selected via a Verifiable Random Function (VRF*[2]*) in one polling cycle. The consensus nodes and the order responsible for the block generation remain unchanged during each polling cycle.

2.3.2 Packaging of Generated Blocks

Consensus nodes package transactions that are contained within the transaction cache pool when it is time to generate a new block. The specific methods is as follows:

1. Consensus nodes package blocks strictly according to the predefined order and duration of the polling cycle.
2. Package as many transactions as possible within packing time-frame;
3. Transactions with a higher overall GasPrice (when compared to other pending transactions) will take priority;
4. A non-verifiable transaction is disregarded when it‘s execution fails.

2.3.3 On-chain Confirmation

The on-chain confirmation for consensus nodes guarantees the consistency and security of the chain as well as penalizing any nodes that may harm the integrity of the blockchain. The Nebulas blockchain utilizes the following rules:

1. The longest subchain is chosen as the optimal chain.
2. The optimal chain is selected according to hash order of previous blocks if the subchains are with the same length.
3. The use of BFT operation across the network for irreversible transactions requires the confirmation from âĚŤ+1 of consensus nodes within the network;
4. The penalty mechanism should be adopted for attacks such as generating blocks when unexpected and attempting double-spends (see *2.5.1 Penalties*).

2.4 Exit Mechanism

Voting for PoD nodes is a fair and free service. All members of the community can withdraw support via NAX for a node or apply to exit the node pool at any time.

2.4.1 Withdrawal of NAX Support for a Node (votes)

All members or organizations within the community may at any time withdraw their support for a community operated node. When support/votes are withdrawn, the node operators‘ NAX support level is immediately reduced (*2.2.1.1 NAX votes*, V(i)) and will affect their ranking in following rounds of node selection. As stated in the minimum requirements (*2.1 Minimum requirements for node selection*), if the total amount of NAX support for a node drops under 100,000 NAX, they cannot be selected as a consensus node.

Quantity of votes to withdraw: The voters may choose how much NAX to withdraw for their support. Community members or organizations can only apply to revoke their own NAX.

NAX return time-frame: Once a withdrawal request has been issued, NAX is returned to the voterâĂŹs original address after 120 polling cycles (approximately 5 days) has passed.

2.4.2 Exiting the Node Pool

All nodes can exit the pool at any time. Once the exit request has been issued, the node will immediately lose its candidacy for following cycles.

Return of NAS security deposit: All NAS deposit required for candidacy is returned in one sum (partial refund is not an option).

NAS security deposit return time-frame: Once the exit request has been issued, the security deposit is returned after 820 polling cycles (approximately 1 month) and will be returned to the original address.

Return of NAX deposit: Any NAX that has been voted/issued for a node which is exiting the pool will be returned to the corresponding address after 120 polling cycles (approximately 5 days).

2.5 Penalties and Emergency Response

2.5.1 Penalties

2.5.1.1 Block Generation Penalties

In order to maintain the security of the PoD system, the corresponding Penalties are carried out according to the situation; the more malicious act of a node, the higher the punishment.

The three block generation penalties for the consensus node are as follows:

Table 2.1: Consensus Mechanism Safety Rating Table

Medium and Severe punishment process:

1. Once a medium and severe penalty occurs, restrictions are automatically executed, and the node will be observed to see if it generates a minimum of one block within 200 polling cycles (approximately 7 days) after the incident.
    (a) If the node proceeds to generate at least one block, the penalty will be disregarded.
    (b) If there is still no block, it is deemed that the problem has not been resolved, the node will have about 1,000 NAS (5% of NAS deposit) frozen.
    (c) If after a penalty occurs, the node can also exit the node strategy. Afterward, NAX will be returned to the original address after 120 polling cycles (approximately 5 days) after the successful submission of the node exit application.
2. During the voting phase of the next governance cycle, the governance committee votes to determine whether the node punishment is justified.
    (a) If the governance committee votes that the punishment is justified, the NAS that has been frozen will be donated to the Go Nebulas Community Collaboration Fund (See 3.2.2 Community Assets).
    (b) If the governance committee votes that the node did not cause intentional harm to the network, the block generation stability index S(i) of the node will be restored to the level prior to the punishment and the NAS will be unfrozen.

See the 3.2.3 Penalties for consensus mechanism and 3.3.3 Processing of voting results of 3.2 Governance scope.

2.5.1.2 Governance Penalties

In addition to the block generation penalties (listed above), when a consensus node is selected as a governance node, the governance node must complete all governance tasks (taking part in votes). If the governance node does not take part in the governance process for two consecutive governance cycles, it cannot be selected for the next 820 polling cycles (approximately one month). See 3.4.1 Individual governance node penalties.

2.5.2 Emergency Response

In the event of an attack on the Nebulas mainnet from a hacker or other unforeseen threats/emergencies and in order to ensure that the network can quickly respond to these attacks and reduce the harm of them, the Nebulas Foundation has reserved emergency smart contract management methods. The Nebulas Foundation can immediately blacklist the address in question and prohibit transfers from blacklisted addresses.

The entire process is open and transparent. The Nebulas Foundation will thoroughly review the incident and openly accept the supervision from the community.

---

[1] **BFT (Byzantine Fault Tolerance):** It is a fault-tolerant technique in the field of distributed computing. Byzantine fault-tolerant comes from the Byzantine Fault problem. The

Byzantine Fault problem models the real world, where computers and networks can behave unpredictably due to hardware errors, network congestion or disruption, and malevolence. Byzantine fault-tolerant techniques are designed to handle real-world abnormal behavior and meet the specification requirements of the problems to be addressed.

[2] **VRF (Verifiable Random Function):** Verifiable random functions: It is an encryption scheme that maps the input to a verifiable pseudo-random output. The program was proposed by Micali (the founder of Algorand), Rabin and Vadhan in 1999. To date, VRF has been widely used in various encryption scenarios, protocols and systems.

3. Governance

This chapter will introduce the Governance Mechanism of PoD mechanism, follow here:

Nebulas‘ focus on the contribution of different roles to the diverse ecosystem via decentralized collaboration and the utilized governance mechanism is an important portion of PoD mechanism.

The governance mechanisms are a range of tools for community self-governmenance via the organization of community collaboration and management of community assets by the governance committee.

3.1 Governance Committee

The implementation of governance mechanisms is managed by the Governance Committee and is made up of governance nodes.

Governance cycle: One governance cycle will occur every 820 consensus node polling cycles (approximately 1 month).

Governance node selection: Governance nodes are selected from the consensus committee and the selected 51 consensus nodes where the largest number of block generators for the past 820 consensus node polling cycles are eligible to become governance nodes in the governance cycle. If there are equally qualified nodes available to become governance nodes and not enough spaces are left, the node(s) which have achieved the number of generated blocks first will be selected.

3.2 Governance Scope

3.2.1 Community Collaboration

The proposal operation of the Nebulas community is an important part of the continuation of the Autonomous Metanet. All proposals and projects of the Nebulas community are public information which are displayed and managed via the Go Nebulas collaboration platform (go.nebulas.io). All community members can put forward their own ideas, opinions and suggestions on the future development of the Nebulas via this platform. Ideas and suggestions include but are not limited to *[1]*:

1. Research and development of the Nebulas mainnet;
2. Community collaboration process optimization, governance recommendations, etcâĂę;
3. Improvement suggestions and bug reports for existing Nebulas community products;
4. Development and maintenance of community eco-products;
5. Community operations and market expansion.

For a proposal to go from idea to implementation, it will go through multiple steps including:

- Proposal establishment;
- Project execution;
- Project acceptance.

Each step needs to be voted for and approved by the Governance Committee. The Governance Committee has three types of voting tasks in each governance cycle:

1. **Proposal establishment voting:** Vote on the proposals submitted from the Nebulas community and decide whether to approve the project and its budget.
2. **Project acceptance voting:** Review and vote on projects that have been established, completed and issue funding.

The Governance Committee process is as follows:

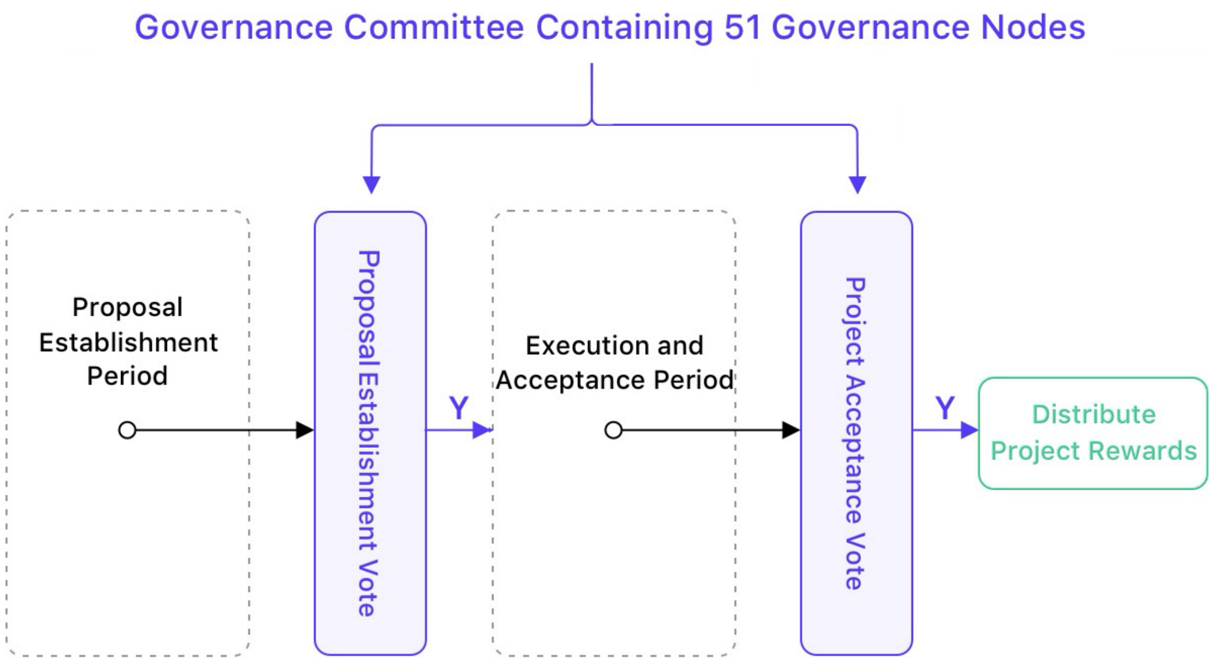

Governance Committee Containing 51 Governance Nodes
Proposal Establishment Period
Proposal Establishment Vote
Y
Execution and Acceptance Period
Project Acceptance Vote
Y
Distribute Project Rewards

Figure 3.1 Governance Committee Voting Process

Proposal establishment period: All community members are welcome to create and share proposals on Go Nebulas (go.nebulas.io). For projects that are approved, the Nebulas Technical Committee will help to establish the project. All projects are separated into two categories:

1. **No budget required for this project:** For example, proposals that include the discussion of improving existing Nebulas projects. These include suggestions on adjusting the structure of the governance organization and the adjustment of the mainnet parameters. After a proposal is voted on and approved, the relevant person in charge may accelerate the implementation of this proposal.
2. **Proposals that require a budget:** This process is facilitated by the Nebulas Technical Committee, which handles project budgets and fund release. Project creators can submit budgets, project objectives, execution steps and expected duration. Creators can also apply to be the project owner or elect a community member to operate the project. Projects should be submitted in accordance with the standard template available on Go Nebulas.

Project execution and acceptance period: The execution and acceptance period is an internal operational process of Go Nebulas; the governance Committee does not directly participate in this process. The process is divided into three stages:

1. **Set budget period:** The project creator or the Go Nebulas operation team can be set as the project creator. Project creators set the reward for successful completion of the project and members of the community are welcome to participate in projects;
2. **Execution period:** The project creator confirms the project owner/manager. At this point, the project owner begins to execute the project and once progressing and upon completion, submits the project results;
3. **Project Review Period:** Once a project is marked as completed, the project creator and the Go Nebulas Operations Team will review the project and its results. Afterwards, a recommendation on whether the project has been successfully completed or not will be

given to the Governance Committee. The committee then decides what further action, if any is required. If none is required, the project will receive its funding as decided in the budget period.

3.2.2 Community Assets

The Governance Committee is responsible for managing the use of the public community assets. Public community assets include:

1. **Use and distribution of the Go Nebulas Community Collaboration Fund:** The primary source of this fund is the DPoS revenue generated from Nebulas since the launch of the Nebulas mainnet on March 30, 2018. Some of these assets have been used for programs such as the Nebulas Incentive Program. The remaining assets will be used for the Go Nebulas Community Collaboration Fund after node decentralization. Since the maximum amount of NAS issued per governance cycle is capped at no more than $30,000 USDT, the actual use of the governance mechanism within six months of its launch is $180,000 USDT equivalent NAS.

2. **Incentive allocation of the Nebulas PoD Node decentralization strategy:** The incentive for Nebulas PoD Node Decentralization Strategy includes two parts: consensus incentive and governance incentive. The source is 8,219.1744 NAS revenue generated daily via DPoS. For the specific allocation method, review section 1.3 Incentive allocation.

The use of public assets, changes to the allocation program, etc... will require the use of the proposal process and can be implemented only after the adoption of the resolution by the Governance Committee.

3.2.3 Penalties for Consensus Mechanism

During the voting phase of the governance cycle, the governance committee will also need to vote on the results of medium and severe security violations.

1. If the governance committee votes that the punishment is justified, the NAS that has been frozen will be donated to the Go Nebulas Community Collaboration Fund.

2. If the governance committee votes that the node did not cause intentional harm to the network, the block generation stability index S(i) of the node will be restored to the level prior to the punishment and the NAS will be unfrozen.

3.3 Governance Method: Vote

3.3.1 Voting Cycle

Governance nodes must vote within 120 polling cycles (about 5 days) after the end of the previous governance cycle. Not participating in voting is considered an Abstain vote.

3.3.2 Voting Methods

The voting of governance nodes is conducted on the public chain with the results viewable to all. All governance nodes are expected to participate in **all** governance periods. All votable items will have the following options (must choose one):

- Support
- Oppose
- Abstain

Each proposal can only be voted for once by each governance node by utilizing 1 NAX per item being voted. NAX used for voting is destroyed and will not be returned.

3.3.3 Processing of Voting Results

The adoption of a proposal or item requires the following conditions:

Table 3.1: Processing of Voting Results Table

* If a single project budget will exceed the maximum dollar value, it is suggested to split the proposed project into a multi-phase project.

** If the total amount of all approved projects during the governance cycle exceeds the maximum budget, projects are ranked by their support rate. Any proposal that is approved but funding is not available for the current governance cycle is deferred to the next governance cycle.

*** Approval rate = Support votes / (Support votes + Oppose votes)

3.4 Penalty Mechanism

3.4.1 Individual Governance Node Penalties

If a consensus node becomes a governance node for two consecutive governance cycles without taking part in governance voting, the node will not be able to be selected as a governance node for 820 consensus polling cycles (approximately one month).

3.4.2 Governance Failure

1. If there are fewer than 26 (of the 51 selected) governance nodes participating in the voting during a governance cycle, the cycle will be declared invalid; no decision made will be executed and all governance incentives will be donated to the Go Nebulas Community Collaboration Fund (See *3.2.2 Community Assets*).
2. If there is no proposal or project in a governance cycle, i.e. there is nothing to vote on, the cycle is declared invalid and all governance incentives will be donated to the Go Nebulas Community Collaboration Fund.

---

[1] Go.nebulas.io help Documentation

Appendix

Appendix A. Recommended Hardware Configuration for Node Operation

Monthly recommended configuration server expenditure is approximately: $150 USDT/month

- CPUïijŽ>=4-Core minimum (Recommended 8-Core)
- RAMïijŽ>=16G
- Disk: >= 600G SSD
- NTP: NTP service is required on the server to ensure correct time synchronization for all operational nodes.

Node Installation Tutorial - review the *Nebulas Technical Documentation: Nebulas 101 - 01 Compile Installation*.

ItâĂŹs recommended to build and deploy nodes via docker:

- Install docker and docker-compose
- Execute the following docker command via root

```
sudo docker-compose build

sudo docker-compose up -d
```

Appendix B. Node Multi-User Participation

Nodes can be operated by an individual, business entity or even a group of individuals acting as a single entity. **The distribution of node incentives is determined by the primary node operator.**

Supporting a node participant is the autonomous ideology of the community members and those who choose to support node operators should only make this decision after fully examining the operation of the node. PoD can only guarantee the pledging and withdrawal of any pledged NAX to a node and is not responsible for the commitment of the node operator to their supporters.

In order to facilitate the participation of community users, the Nebulas Foundation will form a demonstration multi-user participation node. This node will be operated and maintained by the Nebulas Foundation. All community members can support this node by pledging NAX to its existence and the benefits (minus the basic cost of server operation) of the node will be equally distributed to those who are involved in its co-construction based on NAX pledge quantity.

Appendix C. Earnings Simulation

Assuming that a node is among the 51 candidate nodes every day for a month, the maximum consensus incentive for the month is approximately 9,920 NAS.

There are over 700 polling cycles per month and considering the existence of random factors in the selection algorithm, the average revenue per node is expected to be about 3,307 NAS. This however can vary greatly depending on multiple factors as detailed in this paper.

Assuming that all 51 governance nodes selected each month participate, the nodes incentive is estimated to be 816 NAS per month per node.

Appendix D. Parameter Table

D.1 Basic Parameters

- Average block time: 15 seconds
- Polling cycle: 210 block height, approx. 52.5 minutes
- Governance cycle: 820 polling cycles (approximately 1 month)
- Consensus nodes: 21
- Candidate nodes: 51 (with 21 consensus nodes)
- Governance nodes: 51 (consensus nodes who generated the largest number of valid blocks per governance cycle)
- Deposit: 20,000 NAS
- Candidate node minimum pledge (vote): 100,000 NAX
- NAS Pledge return time (Once the exit request has been issued): 820 polling cycles (approximately 1 month)
- NAX return time (Once the withdrawal request or the exit request has been issued): 120 polling cycles (approximately 5 days)

D.2 Parameters Related to the Consensus Mechanism

- Number of blocks generated per node within each polling cycle: 10
- Initial Value of Block Generation Stability Index S(i): 0.8
- Max Value of Block Generation Stability Index S(i): 1
- Trigger threshold for penalty mechanism via Block Generation Stability Index S(i): 0.5
- Penalty (Medium): 5% of NAS deposit
- Penalty (Severe): All NAS deposit plus all NAX pledged to that node
- Consensus mechanism penalty duration (low and medium security level): Candidate node cannot be selected for 20 polling cycles (approximately 1 day)
- Consensus mechanism penalty duration (severe security level): Permanent

D.3 Governance Mechanism Parameters

- Governance node voting time: 120 polling cycles (approximately 5 days)
- Governance node minimum participation: 26
- Required proposal approval rate: Greater than 50%
- Required approval rate for the project establishment voting, project acceptance voting, and penalties for consensus mechanism voting: Greater than 67%
- Governance penalty trigger: Not participating in ALL voting proposals (at minimum level) for two governance cycles constantly
- Governance penalty duration: 820 polling cycles (approximately 1 month)

D.4 Incentive Allocation Parameters

- Daily bookkeeping Income (entire network): 8,219.1744 NAS
- Annual bookkeeping income (entire network): 2,999,941 NAS
- Total annual consensus mechanism incentives (entire network): 2,499,951 NAS
- Total annual incentives for governance mechanisms (entire network): 499,999 NAS
- Single project budget: Cannot be greater than $15,000 USDT
- Maximum amount of funds released per governance cycle: Cannot greater than $30,000 USDT

Appendix E. Addresses

- **Sign-in address:** This address is the only one that you can sign in to the node platform with. Please use the Nebulas Chrome Extension to sign in and manage your node on this node platform.
- **Minner address:** Only used for creating the block, signature, polling check. The keystore is on your server.
- **Incentive address:** Your consensus incentive will be sent to this address. We recommend a cold storage wallet for security. The incentive address can be modified via the server configuration.
- **Governance address:** If your node is selected as a governance node, your vote for proposals and projects will be via this address. In addition, your governance incentive will be sent to this address. To partake in governance and to vote, we recommend using a hot wallet such as NAS nano Pro.

The default governance address is the same as the sign-in address. For security reasons, it is recommended to use a different address and allocate hot and cold wallets according to our recommendations.

Appendix F. Changelog

- Nov 20, 2019 - v1.0
- June 2, 2020 - v1.0.1 - PoD Penalty Rule Adjustment (NP289).
- June 26, 2020 - v1.0.2 - Modification to the Stability Index for Nebulas Nodes (NP294), Addition of a lucky node in the consensus node selection process(NP296), Node Governance: Remove “abstain“ from calculations (NP295).

Another post: The launch of NebulasâĂŹ Proof of Devotion consensus protocol has begun on Ambcrypto.

How to Join

The new version mainnet Nebulas Voyager with PoD will be launched on Mar 30, 2020.

In order to best complete the decentralized transition of the mainnet nodes, the Nebulas PoD Node Decentralization Strategy will gradually open the node applications to all. We invite active project parties, partners and community members to deploy nodes and explore the governance processes.

- Apply a node on Nebulas Node Platform: node.nebulas.io
- Prepare your hardware, Node environment.
- Update your code: How to join the Nebulas mainnet

Nebulas Community Collaboration Platform: go.nebulas.io

1.4.5 How to Contribute

Nebulas aims for a continuously improving ecosystem, which means we need help from the community. We need your contributions! It is not limited exclusively to programming, but also bug reports and translations, spreading the tenets of Nebulas, answering questions, and so on.

1. Community Collabration Platform: Go.nebulas.io

Most of our projects and their corresponding bounties can be found on Nebulas Community Collabration Platform: Go.nebulas.

2. Code

2.1 Mainnet Development

Besides programming, mainnet development is still ongoing and needs the help of the community to tackle challenging problems in the blockchain industry. For instance, we need to design manipulation-resistant mechanisms for blockchain, formally verify the new consensus algorithm, improve security of the Nebulas mainnet, apply new crypto algorithms to Nebulas, etcetera.

We are excited to devote ourselves to blockchain and to see how blockchain technology can improve people's lives. We want to share this exciting experience with the whole community. Thus, we call upon all developers.

We are very glad that you are considering to help Nebulas Team or go-nebulas project, including but not limited to source code, documents or others. Read our *guideline* to learn more about development & contribution details.

Our github: github.com/nebulasio/go-nebulas

2.2 Bug Reporting

We have always valued bug reporting!

If you find a bug, please send your bug report via this Bug Report Form. You will be rewarded for it. Check the Nebulas Bounty Program for more details.

Bugs may be found on the Nebulas testnet, mainnet, nebPay, neb.js, web wallet, as well as other tools and documentation. We will follow the OWASP Risk Assessment System to calculate the corresponding bounty/reward based on the risk degree of the bug.

If you have suggestions on how to fix bugs, or improve upon an affiliated project, please do not hesitate to let us know. You can also participate in the development and directly protect the onchain assets. Together, letâĂŹs make Nebulas even more safe, secure, and robust.

To submit bugs and related information, please post the information in the related Nebulas mail groups. When submitting reports, please be careful and pay attention to the mail group in order to prevent bugs from being exploited or create duplicates. We welcome you to follow the mail group and join the discussion.

3. Documentation

3.1 Wiki & Translation

Translating is important to spread Nebulas to the whole world.

We welcome community members from around the world to participate in the translation of Nebulas documentation. You can translate everything from the wiki, including mainnet technical documents, the DApp FAQ, official documents such as the Nebulas academic papers, the Nebulas design principles introduction document, and more. Your contribution will significantly help numerous Nebulas developers and community members. Please note that some documents will require an academic background in Math, Computer Science, Cryptography, and/or other specialties.

wiki.nebulas.io (Github) is the platform to collect all these important documents both non-technical and technical.

How to edit a Wiki page

A full tutorial on how to edit Wiki pages can be found here.

Editing Software

For users who are familiar with git and would like to edit the Wiki locally, reST should be used to edit .rst files, and Pandoc Markdown for .md files.

Click here to learn about the differences between Pandoc Markdown and reST.

Below are some of the learning resources that can be used to further your knowledge of Markdown:

- How to use Markdown by John Gruber
- Markdown Guide by iA Writer

The aftermath

When you edit pages on Github, you should always click on "Preview changes" to view the result of your labor. After your contribution has been merged, you can check the building process here.

3.2 Writing

Developers in the Nebulas community require documentation to help them understand and use the various functions of Nebulas. The community is welcome and encouraged to write technical introductions and FAQs. In addition, Nebulas' community members will also benefit from easy-to-understand introductory guides and user guides on various ecosystem tools.

Your contribution will be of use to all community developers and members, and may also be translated into multiple languages to benefit an even larger amount of members.

4. User Groups

Communication is key for building a vibrant community. People need to talk with each other in order to share their ideas and thoughts on Nebulas.

Nebulas uses several platforms to connect with its global community. Please refer to the âĂIJCommunityâĂİ page on the official website for more information.

Forum: community.nebulas.io (for developers and non-developers)

Reddit: Reddit/r/nebulas (for all), Reddit/r/nasdev (for developers)

Telegram: English (for non-developers)

Community developers are welcome to create an IRC (Internet Relay Chat) channel for better communication among developers.

5. Donations

Donations to the Nebulas Foundation to further the development of Nebulas are greatly appreciated. Both NAS and ETH are accepted. We also welcome community members to support us in material terms. For instance, the donation of meetup locations/venues, local guides, photography, etcetera. We can also make your contribution known to the community if you would like. If you are an enthusiastic community member and are willing to contribute to our community, email contact@nebulas.io for more details.

1.4.6 Bounty Program

Nearly all projects are posted on the **Nebulas Community Collabration Platform: Go.nebulas.io** along with their corresponding bounties, and users are expected to apply in order to claim a project or parts of it. This process applies to the wiki and to the NAT Bug Bounty Program. For now, the Nebulas Bug Bounty Program only requires you to submit a form with the relevant information.

Below you will find in-depth information about all the Bounty Programs so you can get started on contributing to the flourishing Nebulas ecossystem and get rewarded for it!

Bug Bounty

The Nebulas Bug Bounty aims to improve the security of Nebulas Ecosystem, ensuring the establishment of a benign Nebulas ecosystem. The Nebulas Bug Bounty Program provides bounties for the discovered vulnerabilities. This bounty program was initiated and implemented by the Nebulas Technical Committee (NTC), in conjunction with the Nebulas technical team, and community members. NTC encourages the community to disclose security vulnerabilities via the process described below, and play a role in building the Nebulas ecosystem, thereby receiving bounties, and partaking in the evolution of the Nebulas ecosystem.

Bug Category

The Bug Bounty Program divides the bug bounties into 2 categories, common bug bounty and special bug bounty. The common bugs include vulnerabilities discovered in:

- Nebulas mainnet

- NAS nano pro
- nebPay
- Web wallet
- neb.js
- Bug Bounty on Testnet
- others

While the special bugs include vulnerabilities found in the inter-contract call function, etcetera.

Eligibility

The Nebulas Technical Committee will evaluate reward sizes according to the severity calculated by OWASP Risk Rating Method based on **Impact** and **Likelihood**. However, final rewards are determined at the sole discretion of the committee.

Overall Risk Severity

Impact:

- High: Bugs affecting asset security.
- Medium: Bugs affecting system stability.
- Low: Other bugs that do not affect asset security and do not affect system stability.

Likelihood:

- High: The bug can be discovered by anyone who performs an operation, regardless of whether or not the bug has been found.
- Medium: Only certain people can discover it (such as a bug that only developers encounter, ordinary users are not affected.)
- Low: Covers less than 1% specific population, such as certain rare Android models; or any other exceptional cases.

Amount:

To ensure the bug reporter obtains a stable expected reward, the amount in US dollars will be issued in equivalent NAS. The reward amount is divided into 5 categories:

- Critical: US$1,000 or more (No upper limit)
- High: US$500 or more
- Medium: US$250 or more
- Low: US$100 or more
- Improvement: US$30 or more

Note: The Nebulas testnet special vulnerability reward (such as one for testnet inter-contract call function) has been increased accordingly, and the equivalent US dollars are issued in NAS.

Report A Bug

Please send your bug report via this link.

Things to keep in mind:

1. Please ensure the accuracy and clarity of the content, because the reward evaluation will be based on the content submitted in this form.
2. If many people discover the same bug, then their report submissions in chronological order will determine their reward. Community users are welcome to discuss the issues of bugs, but the discussion itself is not considered a report, therefore a report form must still be submitted.

Additional notes:

1. The Nebulas Bug Bounty Program is long-standing. The Nebulas Technical Committee reserves the right to final interpretation of this program, and the rights to adjust or cancel the reward scope, eligibility, and amount.
2. The Nebulas Technical Committee will confirm and evaluate the bug report after its submission. The evaluation time will depend on the severity of the problem and the difficulty of its resolution. The result of the evaluation will be sent to its reporter by email as soon as possible.
3. To avoid the exploitation of bugs, reporters are required to submit the bug bounty application using the proper forms.
4. Reporters shall keep the bugs non-public and confidential until 30 days after the bug submission to Nebulas, and shall not disclose the bugs to any third party. Such confidentiality time limit can be extended by Nebulas unilaterally. If reporters disclose the bugs to any third party and cause any harm to Nebulas or NebulasâĂŹ users, reporters shall be responsible for the compensation for all the losses and damage.
5. The Nebulas Technical Committee encourages community members to converse with the Nebulas technical team and other community members in the Nebulas public discussion group. We also encourage our community members to join us in fixing these bugs.

1.4.7 Frequently Asked Questions

This document will focus on the technology behind the Nebulas platform. For broader questions, please view the Reddit FAQ.

For a better understanding of the Nebulas platform it‘s highly recommended to read the Nebulas Technical Whitepaper.

Table of Contents

Nebulas Rank (NR)

Measures value by considering liquidity and propagation of the address. Nebulas Ranking tries to establish a trustful, computable and deterministic measurement approach. With the value ranking system, we will see more and more outstanding applications surfacing on the Nebulas platform.

When will Nebulas Rank (NR) be ready?

The Nebulas Rank was released in December of 2018. At the time of writing this, June 28th of 2019, the NR Query Server is not online since the NR algorithm was updated, as it needs to be refactored. You are welcome to claim this project here.

Will dApps with more transactions naturally be ranked higher?

Not necessarily, as transaction count would only increase the in-and-out degree over a period of time, up to a certain point. The way the Nebulas Rank is calculated uses, among many other variables, one‘s median account stake. The median account stake is the median of the account balance over a period of time.

How does the Nebulas Rank (NR) separate quality dApps from highly transacted dApps?

By utilizing the Median Account Stake in its calculations, the NR ensures fairness and resists manipulation to a reasonable degree, ensuring the likelihood of high quality dApps floating to the top of the hierarchy.

Is the Nebulas Ranking algorithm open-source?

Yes.

Who can contribute to the algorithm?

At this time the Nebulas core team is responsible for the development of the algorithm. However, anyone is free to make suggestions, bug reports, and contribute with code. The SDK's repository can be accessed here, and the Nebulas Rank Offline Service can be accessed here.

Can the Nebulas Rank (NR) algorithm be cheated?

Nothing is impervious to manipulation, but our goal is to make manipulation of the algorithm as expensive and difficult as possible.

Nebulas Force (NF)

Supports upgrading core protocols and smart contracts on the chains. It provides self-evolving capabilities to Nebulas system and its applications. With Nebulas Force, developers can build rich applications in fast iterations, and the applications can dynamically adapt to community or market changes.

When will Nebulas Force (NF) be ready?

As per the roadmap, Nebulas Force is poised to be released at the end of 2019.

Can smart contracts be upgraded?

Yes, [short summary explaining how it works]

How is Nebulas Force (NF) smart contract upgrading better than other solutions that are currently or soon-to-be available?

answer here

Can the Nebulas blockchain protocol code be upgraded without forking?

Yes, [short summary explaining how it works]

Can the Nebulas Virtual Machine (NVM) be upgraded?

Yes, [short summary explaining how it works]

Developer Incentive Protocol (DIP)

Designed to build the blockchain ecosystem in a better way. The Nebulas token incentives will help top developers to add more value to the Nebulas blockchain.

When will the Developer Incentive Protocol (DIP) be ready?

The Developer Incentive Protocol was deployed on the Nebulas Testnet in January of 2019. It was formally deployed on the Mainnet in May of 2019.

Will there be a limit as to how many rewards one dApp can receive?

answer here

Will developers still be able to do their own ICOs?

answer here

Will only the top Nebulas Rank (NR) dApps receive rewards?

answer here

How often will rewards be given?

answer here

How will you stop cheaters?

The way the DIP is is designed makes it very hard for cheaters to be successful. Since smart contracts can only be called passively, it would be highly cost ineffective for a user to try to cheat the system. More about this topic can be read in the Technical Whitepaper.

Proof of Devotion (PoD) Consensus Algorithm

To build a healthy ecosystem, Nebulas proposes three key points for consensus algorithm: speediness, irreversibility and fairness. By adopting the advantages of PoS and PoI, and leveraging NR, PoD will take the lead in consensus algorithms.

When will the Proof of Devotion (PoD) Consensus Algorithm be ready?

answer here

What consensus algorithm will be used until PoD is ready?

answer here

How are bookkeepers chosen?

The PoD consensus algorithm uses the Nebulas Rank (NR) to qualify nodes to be eligible. One node from the set is randomly chosen to propose the new block and the rest will become the validators.

Do bookkeepers still have to stake?

Yes, once chosen to be a validator for a new block, the validator will need to place a deposit to continue.

How many validators will there be in each set?

answer here

What anti-cheating mechanisms are there?

answer here

Nebulas Search Engine

Nebulas constructs a search engine for decentralized applications based on Nebulas value ranking. Using this engine, users can easily find desired decentralized applications from the massive market.

When will the Nebulas Search Engine be ready?

answer here

Will you be able to search dApps not on the Nebulas platform?

answer here

Will the Nebulas Search Engine also be decentralized?

answer here

Will the Nebulas Rank (NR) control the search results ranking?

answer here

What data will you be able to search?

We plan on developing many different ways to be able to search the blockchain:

- crawl relevant webpages and establish a map between them and the smart contracts
- analyze the code of open-source smart contracts
- establish contract standards that enable easier searching

Fundamentals

Nebulas Name Service (NNS)

By using smart contracts, the Nebulas development team will implement a DNS-like domain system named Nebulas Name Service (NNS) on the chain while ensuring that it is unrestricted, free and open. Any third-party developers can implement their own domain name resolution services independently or based on NNS.

When will the Nebulas Name Service be ready?

answer here

When a name is bid on, how long do others have to place their bid?

answer here

How do others get notified that a name is being bid on?

answer here

When a name is reserved who gets the bid amount?

answer here

If I want to renew my name after one year will I need to deposit more NAS?

answer here

Will we be able to reserve names prior to the launch of NNS?

answer here

Lightning Network

Nebulas implements the lightning network as the infrastructure of blockchains and offers flexible design. Any third-party developers can use the basic service of lightning network to develop applications for frequent transaction scenarios on Nebulas. In addition, Nebulas will launch the worldâĂŹs first wallet app that supports the lightning network.

When will lightning network be supported?

answer here

The Nebulas Token (NAS)

The Nebulas network has its own built-in token, NAS. NAS plays two roles in the network. First, as the original money in the network, NAS provides asset liquidity among users, and functions as the incentive token for PoD bookkeepers and DIP. Second, NAS will be charged as the calculation fee for running smart contracts. The minimum unit of NAS is 10âĹŠ18 NAS.

What will happen to the Nebulas ERC20 tokens when NAS is launched?

The ERC20 tokens were swapped by its owners and exchanges that held them at a 1 to 1 rate.

Will dApps on the Nebulas platform be able to issue their owns ICOs and tokens?

answer here

Smart Contracts

What languages will be supported when Main-net launches?

answer here

Will Ethereum Smart Contracts (Solidity) be fully supported?

answer here

What other language support will follow (and when)?

answer here

binary storage

What is recommended way to store binary data in Nebulas blockchain? Is it possible at all? Do you encourage such use of blockchain? Also, i couldn‘t find information regarding GlobalContractStorage mentioned in docs, what is it?

Currently binary data can be stored on chain by binary transaction. The limit size of binary is 128k. But we donâĂŹt encourage storing data on the chain because the user might store some illegal data.

`GlobalContractStorage`not currently implemented. It provides support for multiple contract sharing data for the same developer.

ChainID & connect

Can you tell us what the chainID of Mainnet and Testnet is? I have compiled the source code of our nebulas, but not even our test network?

chainID of Nebulas:

- Mainnet: 1
- Testnet: 1001
- private: default 100, users can customize the values.

The network connection:

- Mainnet:
  - source code:master
  - wiki:Mainnet
- Testnet:
  - source code:testnet
  - wiki:Testnet

Smart Contract Deployment

Our smart contract deployment, I think is to submit all contract code directly, is the deployment method like this?

> Yeah, We can deploy the contract code directly, just as it is to release code to the NPM repository, which is very simple and convenient.

smart contract IDE

We don‘t have any other smart contract ides, like solidity‘s “Remix“? Or is there documentation detailing which contract parameters can be obtained? (because I need to implement the random number and realize the logic, I calculate the final random number according to the parameters of the network, so I may need some additional network parameters that will not be manipulated.)

> You can use web-wallet to deploy the contract, it has test function to check the parameters and contract execution result.

1.4.8 Licenses

Nebulas Open Source Project License

The preferred license for the Nebulas Open Source Project is the GNU Lesser General Public License Version 3.0 (âĂIJLGPL v3âĂİ), which is commercial friendly, and encourage developers or companies modify and publish their changes.

However, we also aware that big corporations is favoured by other licenses, for example, Apache Software License 2.0 (âĂIJApache v2.0âĂİ), which is more commercial friendly. For the Nebulas Team, we are very glad to see the source code and protocol of Nebulas is widely used both in open source applications and non-open source applications.

In this way, we are still considering the license choice, which kind of license is the best for nebulas ecosystem. We expect to select one of the LGPL v3, the Apache v2.0 or the MIT license. If the latter is chosen, it will come with an amendment allowing it to be used more widely.

Contributor License Agreement

All contributions to Nebulas wikis are licensed under the Creative Commons License SA 4.0.

For a complete list of everyone who contributed to the wiki, click here.

- genindex
- modindex
- search