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

nebulas Documentation

ìűlJìŃlJ 1.0

nebulas

11ìŻŤ 21, 2019

Category:

CHAPTER 1

Welcome to the open-source Nebulas wiki!

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. Include usage guides, development guides, learning resources, and other useful documents.

CHAPTER 2

Use Wiki

2.1 Mainnet

2.2 Dapps

2.3 Ecosystem

2.4 Get Involved

2.4.1 Wiki Usage Guide

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.

2.4.2 How to Contribute

Your contribution matters!

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.

Most of our projects and their corresponding bounties can be found here.

1. Code & Documentation

1.1. 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!

Learn more:

- Our github: github.com/nebulasio/go-nebulas
- Our Roadmap: nebulas.io/roadmap.html (Stay tuned!)

1.2. Bug Reporting

We have always valued bug reporting!

If you find a bug, please report it to the Nebulas community. You will be rewarded for it. Check the Nebulas Bounty Program here 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. More TBA.

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.

Mail group list: lists.nebulas.io/cgi-bin/mailman/listinfo

Mainnet bug list: lists.nebulas.io/cgi-bin/mailman/listinfo/mainnet-bugs

Testnet bug list: lists.nebulas.io/cgi-bin/mailman/listinfo/testnet-bugs

1.3. 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 many 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.

1.4. Documentation 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.

1.5. Wiki UI Design

We welcome UI developers to optimize our wiki page and make it more user friendly and easier to read.

Download the design template of the Nebulas wiki >

Download the logo >

If you have any questions or comments, please do not hesitate to post on our GitHub.

2. 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.

Discord: Available to all community members. You can subscribe to Nebulas News, as well as participate in group discussions. Discord is many users‘ first choice.

Mailing lists: Discussion groups for core development and bug reporting. We welcome developers to subscribe.

Forum: Reddit/r/nebulas (for all), Reddit/r/nasdev (for developers)

Communication: Slack (for developers), Telegram (for non-developers)

Community developers are welcome to create an IRC (Internet Relay Chat) channel for better communication among developers.

3. Bounties

We, the Nebulas team, happily introduce several bounties to reward early contributors. You can check them out here.

4. 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.

2.4.3 Bounty Program

Nearly all projects are posted on the Nebulas Project Page 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!

The Nebulas Wiki Bounty Program

Previously users who created or modified content on the Nebulas Wiki were entitled to potentially win a bounty in the form of NAS. Nowadays, the process is quite different.

To qualify for the wiki bounty, go to the aforementioned project page and search for "wiki," or simply click here to see all the available listings.

The Nebulas Bug Bounty Program

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, Nebulas testnet, nebPay, Web wallet, neb.js and 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.

Image 1

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. Join our Nebulas maillist!

The Nebulas NAT Bug Bounty Program

NAT includes about 7 different smart contracts.

For bugs pertaining the NAT smart contracts, you may go here to claim your bounty. Do note that you will still have to fill in the following form detailing your bug, after claiming, in order to become elligible for the bounty.

The smart contracts can be updated at any time. They are listed below:

multisig: n1orrpFGmcQSvGrbKTD7RHweTPe61ut7svw

NAT NRC20: n1mpgNi6KKdSzr7i5Ma7JsG5yPY9knf9He7

distribute: n1uBbtFZK3Acs2T6JUMv6bSAvS6U6nnur6j

pledge_proxy: n1obU14f6Cp4Wv7zANVbtmXKNkpKCqQDgDM

pledge: n1zmbyLPCt2i8biKm1tNRwgAW3mhyKUtEpW

vote: n1pADU7jnrvpPzcWusGkaizZoWgUywMRGMY

NR_DATA: n21KaJxgFw7gTHR9A5VFYHsQrWdL61dCqvK

2.4.4 WhatâĂŹs Nebulas

Nebulas: Next Generation Public Blockchain

Nebulas is aiming to build a continuously improving ecosystem.

Nebulas is a next-generation public blockchain. It introduces Nebulas Rank (NR), a new measure of value for every unit of the blockchain universe, like addresses, DApps and smart contracts. Based on NR, it involves Nebulas Incentive (NI), which motivates developers with Developer Incentive Protocol, and users with the Proof of Devotion consensus algorithm. Moreover, it proposes Nebulas Force (NF), which gives the blockchain and smart contracts within it a self-evolving capacity. In unison, NR, NI, and NF produce a continuously improving and expanding blockchain ecosystem, using the principles contained in the Nebulas Governance article to guide its evolution.

There are three technical features: value ranking, self-evolution, and native incentive.

Facing the opportunity and challenge as above, we aim to create a self-evolving blockchain system based on value incentive.

Principles

The Nebulas blockchain has three major principles:

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.

Nebulas Force (NF)

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.

Nebulas Incentive (NI)

The Nebulas Incentive includes Proof of Devotion (PoD) and the Developer Incentive Protocol (DIP).

Proof-of-Devotion (PoD)

Based on the NebulasâĂŹ NR system, we shall adopt the PoD (Proof-of-Devotion) consensus algorithm. PoD gives an âĂIJinfluentialâĂİ user of the Nebulas blockchain an opportunity to become a bookkeeper and receive NebulasâĂŹ block rewards and transactional fees as revenue, which will in turn encourage them to continuously contribute to the stability and security of Nebulas.

Developer Incentive Protocol (DIP)

On Nebulas, we proposed the concept of DIP (Developer Incentive Protocol) for developers of smart contracts and DApps. DIPâĂŹs core concept: in the interval of pre-specified blocks, for those developers whose smart contracts and DApps were deployed online during the most recent interval, with a NR value higher than a specified threshold, DIP shall reward them the corresponding developer incentives (NAS token), and these incentives shall be recorded on blocks by bookkeepers. With the DIPâĂŹs positive incentive mechanism, more and more developers will get incentives to create valuable smart contracts and DApps, which will help to build a positive feedback ecosystem for the developer community.

Nebulas Community Governance

Orange Paper

The Nebulas Orange Paper released on April 30 of 2019 highlights how Nebulas can use its unique and innovative technology to manage public assets on the chain in order to achieve its vision: âĂIJTo explore a new decentralized collaboration model, implement a decentralized autonomous organization that provides positive incentives and can self-evolve (Decentralized Autonomous Organization, DAO).âĂİ

The tenets upon which Nebulas' governance is based are as follows:

1. Organizational structure and supervision mechanism: Nebulas Community Groups will operate independently but are mutually constrained by one another: Nebulas Council, Nebulas Foundation, Nebulas Technical CommitteeâĂŁâĂŤâĂŁarticulating its basic composition, powers and obligations;
2. On-chain Collaboration: Nebulas community project introduction, âĂIJNAT On-chain votingâĂİ will achieve community collaboration and system upgrade process;
3. Economies and Incentives: The design of the on-chain voting economy and how the economy provides positive incentives to each community member during Nebulas governance.

Learn more about Nebulas' Governance by reading our Orange Paper here.

The NAT Token

NAT is Nebulas‘ governance NRC20 token derived from the Nebulas Rank and powers the Nebulas on-chain voting system. Its total supply is capped at 100 billion.

Initially, 0 NAS voting was used. One address would be created per voting option, and a user would transfer 0 NAS to the corresponding address in order to vote. To count the votes for referenda where a wallet can only vote a single time, the transfers are parsed, duplicate addresses are merged, and then the number of transactions are counted. It is a very limited voting system.

The NAT on-chain voting system has several ways to target the weaknesses of the 0 NAS voting system.

- Introduces the possibility of weighted voting for special types of elections.
- Creates incentives to voting in the form of NAT rewards.
- Individuals‘ voting power is determined by their Nebulas Rank and their involvement with the Nebulas community and ecosystem.
- Protection from bad actors due to vastly higher supply (compared to NAS).

Note: NAT transferred will be burned.

For more in-depth information, read the Nebulas Governance Orange Paper.

How to Obtain NAT

There are a myriad of ways to obtain NAT.

- ***Via airdrops:*** NAT airdrops are conducted every week and are based on one‘s Nebulas Rank. To calculate how many tokens you will receive from the airdrops consider reading this article.
- ***Via pledging:*** pledging NAS will allow you to receive NAT tokens for as long as the NAS tokens are locked in a smart contract. Cancelling your pledge will return your NAS tokens back to your wallet but you will stop receiving NAT.
- ***Via voting or Nebulas Rank:*** if you have a non-zero Nebulas Rank you are elligible to receive voting incentives. Currently, the incentive index is set at 10. Thus, the number of voting incentives you will receive is equal to 10 times your Nebulas Rank that week, or 10 times the number of NAT that was sent out of that address for the week, whichever is lower.

Learn More

Where to vote

Go.Nebulas.io Proposals.

Active nebulas.io Ballots.

Useful Links

nebulas.io NAT‘s Main Page.

Why NAT is a fundamental component of the Nebulas ecosystem.

NAS on-chain voting starts!

Three minutes to take you into the world of NAT.

Participate in Nebulas ecosystem voting & receive NAT incentives!

How to obtain NATâĂŁâĂŤâĂŁPart 1: How to Pledge your NAS.

How to obtain NATâĂŁâĂŤâĂŁPart 2: How to Pledge your NAS via offline mode.

How to obtain NATâĂŁâĂŤâĂŁPart 3: Receiving NAT incentives & how to improve your NR.

How to receive NAT without a Nebulas Rank.

Value ranking

To enable value discovery in blockchain, **Nebulas Rank** measures multidimensional data in the blockchain world and powers the decentralized search framework.

Self-evolution

To avoid the damage caused by forking to the blockchain, **Nebulas Force** enables rapid iteration and upgradability to its blockchain without the need for hard forks.

Native incentives

With forward-looking incentive and consensus mechanisms, the **Nebulas Incentive** rewards developers and users who contribute to the sustainability and growth of the ecosystem.

This is an excerpt of the Nebulas Non-technical Whitepaper.

If you want to know more about Nebulas, please subscribe to the official blog, or visit our website: nebulas.io. Read our Non-technical White Paper (English), Technical White Paper (English).

2.4.5 Go-Nebulas

Nebulas Technical Committee

Nebulas Nova Tech Tradeoffs (11.21.2018)

Summary

1. The process to submit IR (LLVM Intermediate Representation) and who can submit IR (LLVM Intermediate Representation)
2. The time window for NR & DIP
3. How much NAS for DIP & how to distribute NAS for DIP

Detailed minutes:

1. The process to submit IR and who can submit IR

1. Nebulas Nova will use an auth_table to decide whose IR can be executed and the lifetime of each IR.
2. auth_table is a set of tuples, and each tuple includes IR name, submitterâĂŹs address, the valid start and end height for the submitter.
3. Only the auth_adminâĂŹs auth_table can update in Nebulas Nova. The auth_admin account should be created by a cold wallet. Each IR should be managed by different accounts. Nebulas Technical Committee will further discuss the community governance details with the community. Before the we finalized the governance details , the Nebulas team will not recklessly open the IR submission access.The NBRE only executes several

predefined IRs, like checking the auth_table, and the IRs defined in auth_table. Other IRs will not be executed

4. However, each node may change the code. And that could be the auth_admin account, and the auth_table. Consequently, it may change the behaviors in NBRE, and the node shall fail to sync data with the main-net

2. The time window for NR & DIP

1. In the yellow paper introducing Nebulas Rank, we have mentioned that to avoid the affect of loop attack, we will remove the forwarding loop before we calculate the In-and-Out degree for the transaction graph, thus the time-window is important for anti-manipulation.
2. If the time-window is too short, there may be more cheating.
3. For now, we suggest the time window in several days.
4. We should monitor the cheating status, and adjust the time-window if necessary.
5. time window for DIP should be much more larger than the time window of NR, for now, we suggests 7 days

3. How much NAS for DIP & how to distribute NAS for DIP

1. For each month, we suggest around 500 NAS in total for now, and adjust the amount subject to the DIP feedback in the future, the winners shall be relatively stable, so a winner will get reward in several months.
2. We will have a special account for distributing NAS for DIP. The account can only send special transactions for DIP.

About Nebulas Technical Committee

- The Nebulas Technical Committee adheres to the spirit of openness, sharing, and transparency, and is committed to promoting the decentralization, and the community of the research and development of the Nebulas technology. Blockchain technology opens up possibilities for building new and self-motivated open source communities. NebulasâĂŹ technical concepts unclude mechanisms for evaluation, self-evolution, and self-incentives, which provide a guarantee for building a world of decentralized collaboration. The Nebulas Technical Committee will fully promote the realization of the Nebulas vision.*
- Subscribe to nebulas mailing list and learn the latest progress of Nebulas: mailing list*
- For more info, please visit Nebulas official website.*

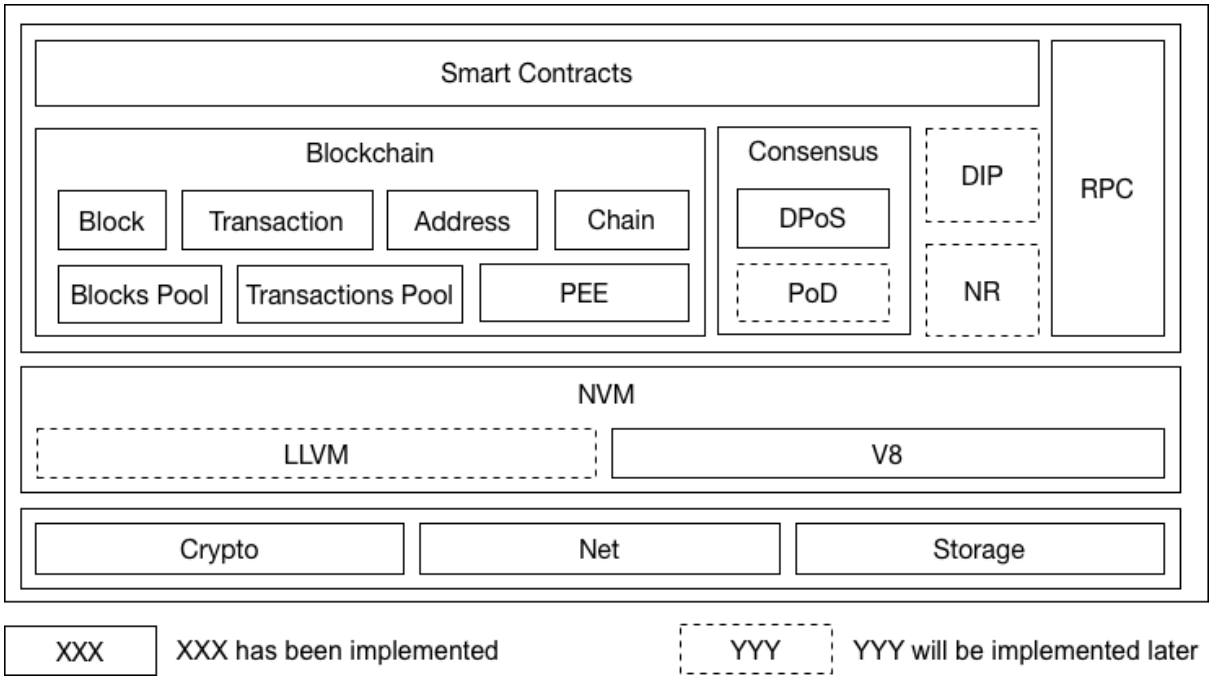
Papers

- The Nebulas Technical White Paper.

- The Nebulas Non-technical White Paper.
- The Nebulas Yellow Paper âĂŤ the Nebulas Rank. You can access the repository here.
- The Nebulas Mauve Paper âĂŤ the Developer Incentive Protocol. You can access the repository here.
- The Nebulas Orange Paper âĂŤ Nebulas Governance. You can access the repository here.

As always, translations and bug reports are always welcome. Learn more about how to contribute.

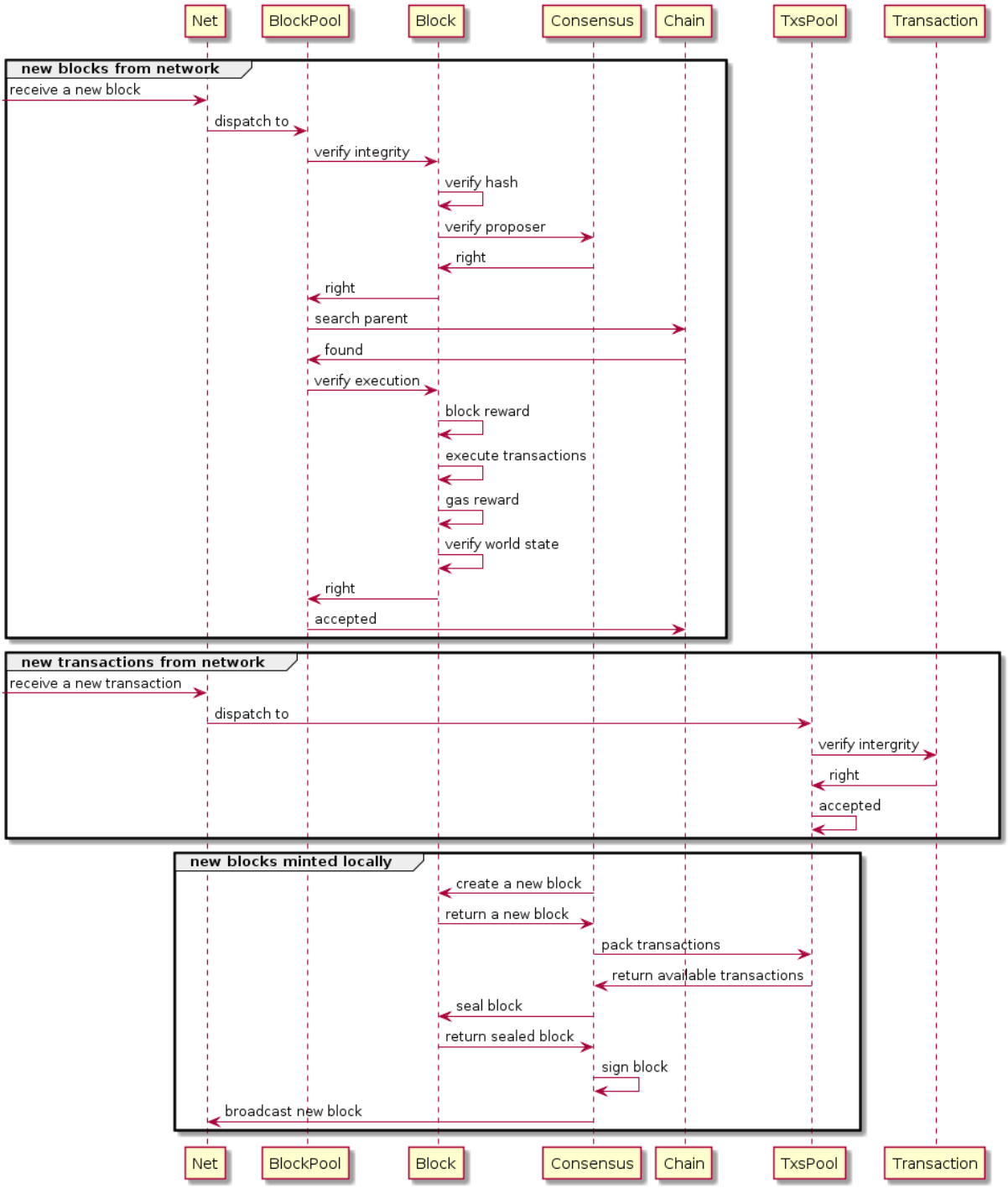
Design Overview



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.

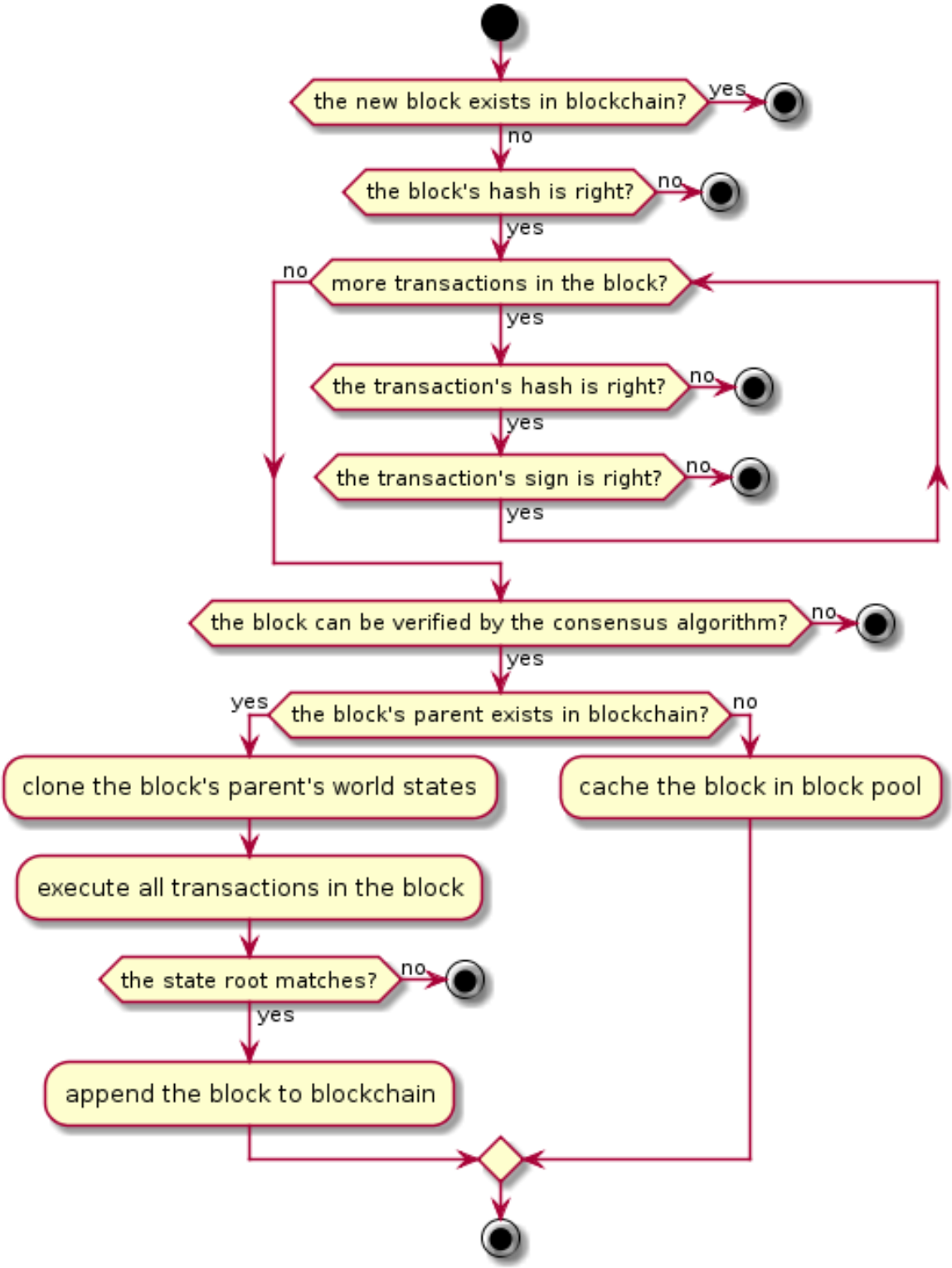
A new block from network

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.



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.

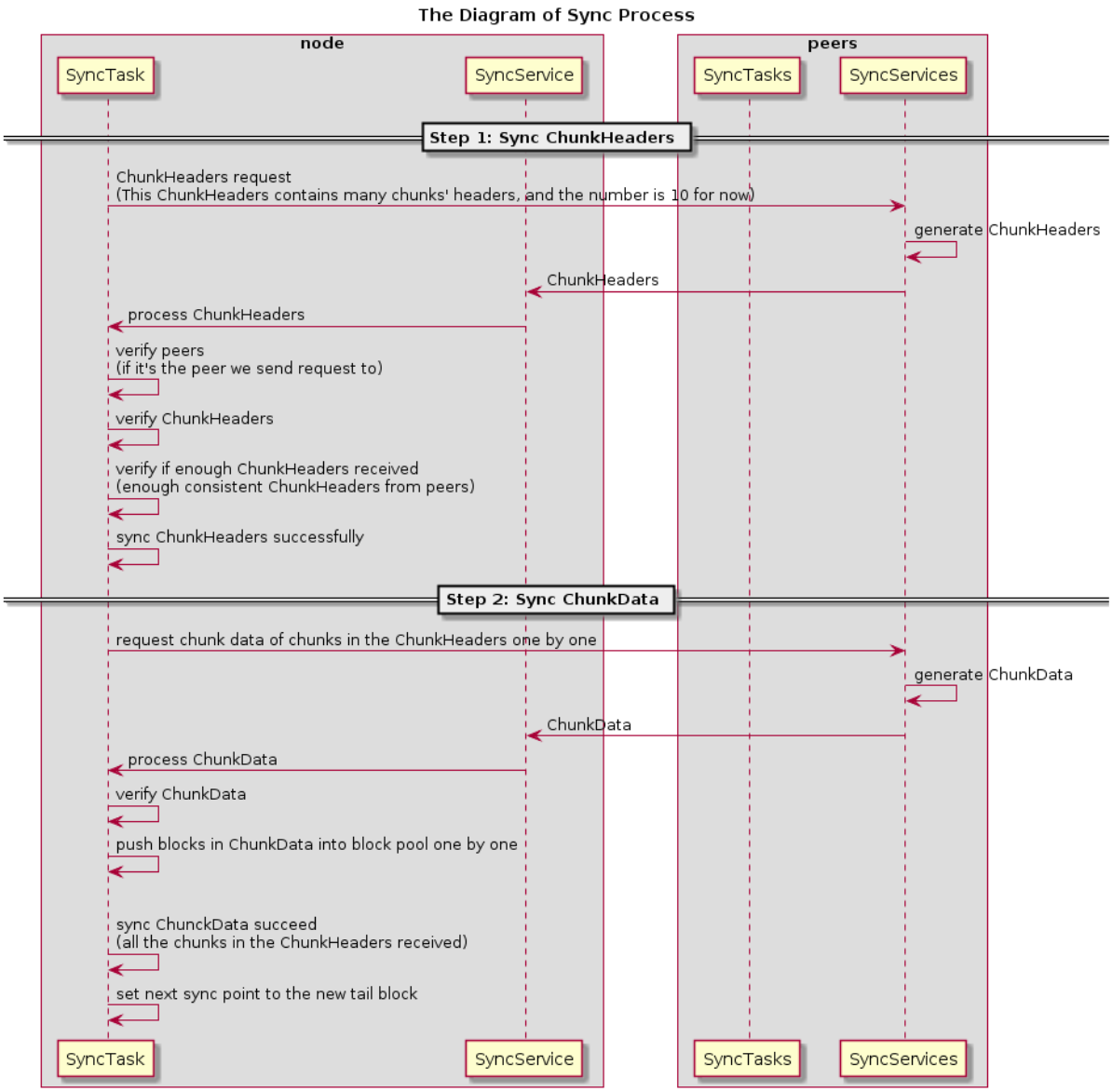
The procedure is as following,

```
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:



Block Downloader

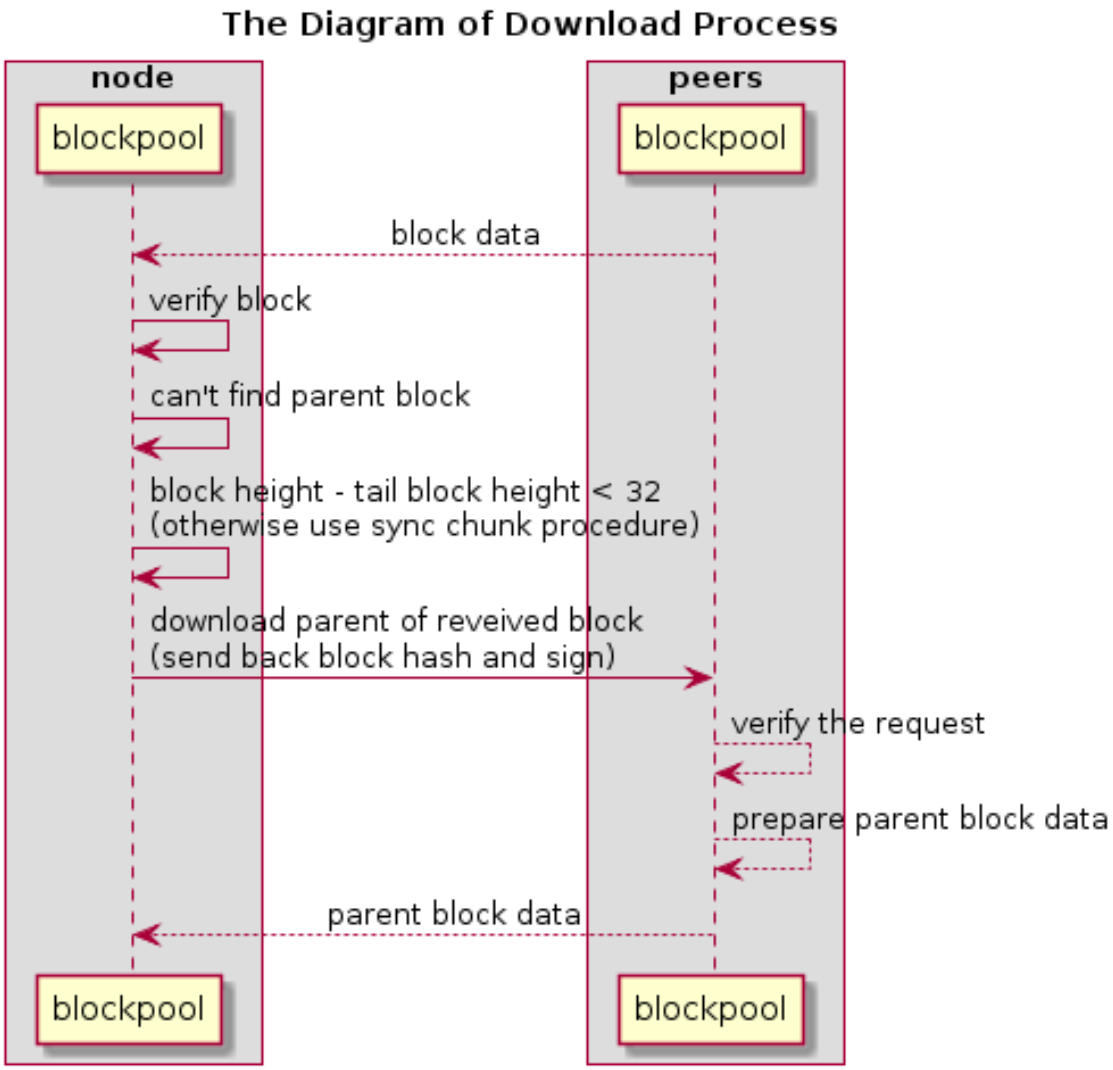
When the length gap between our local chain with the canonical chain is smaller than 32, we'll use Block downloader to download the missing blocks one by one.

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:



Merkle Patricia Tree

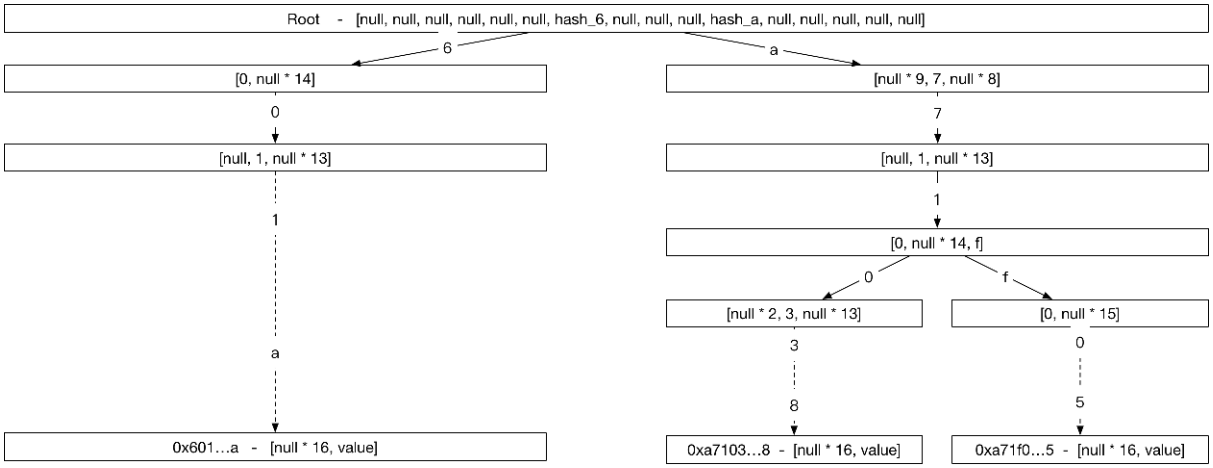
Basic: Radix Tree

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



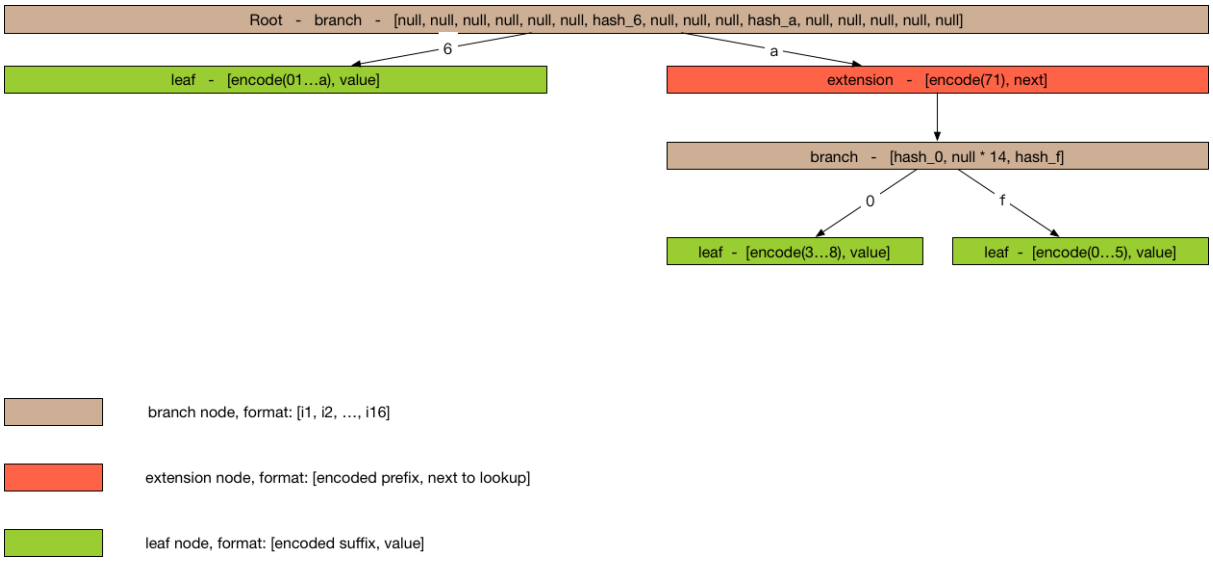
Problems: much space for a single entry 40 steps for each lookup

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



How to store Merkle Patricia Tree

Key/Value Storage

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

key = hash(value)

How to update Merkle Patricia Tree

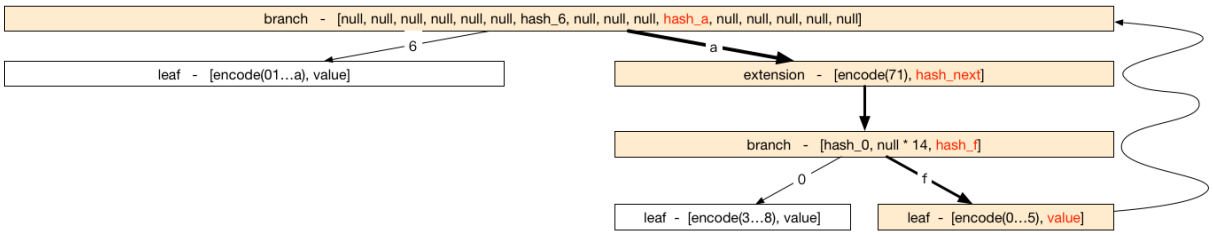
Query

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



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.

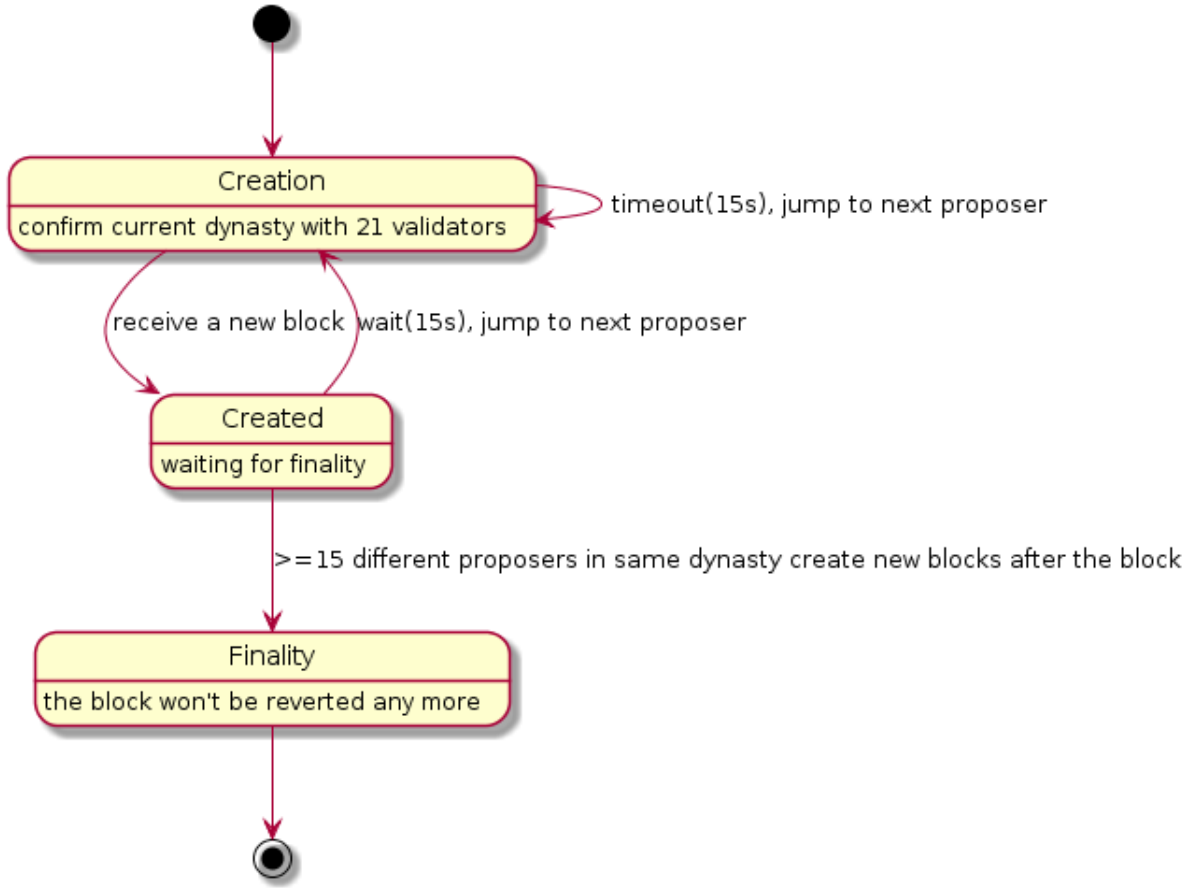
DPoS(Delegate Proof-of-Stake)

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



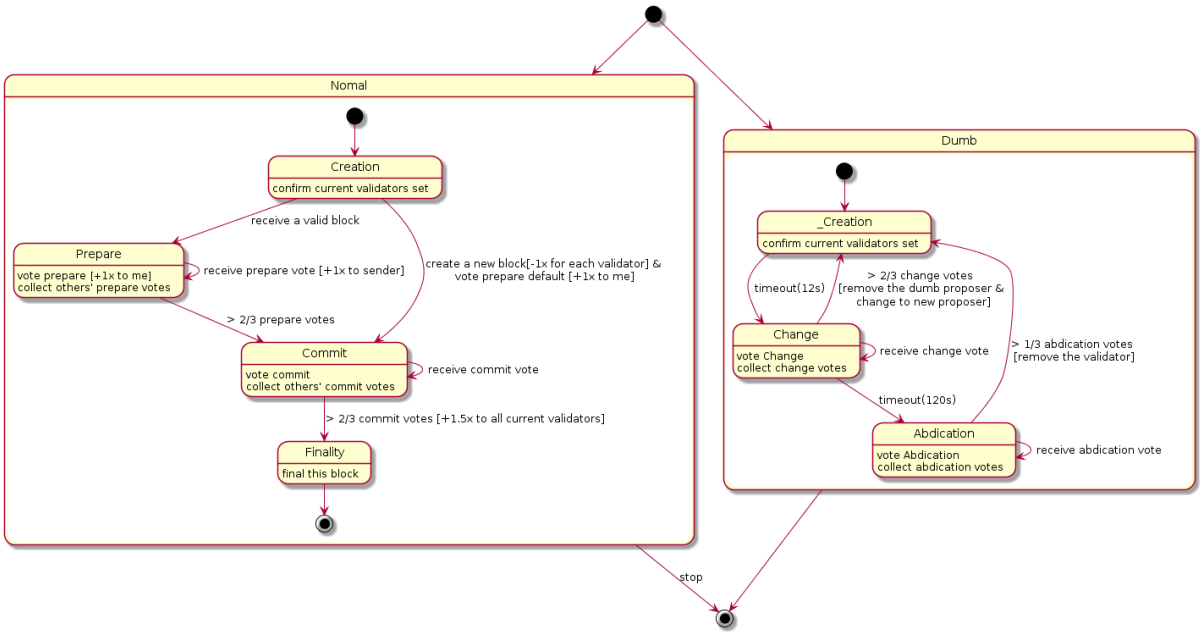
Fork Choice Rules

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)

How to Join the Nebulas Mainnet

Introduction

The Nebulas Mainnet 2.0 (Nebulas Nova) has been released. This tutorial will teach you how to join and work with the Nebulas Mainnet.

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

Build

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

- **NBRE:** The Nebulas Blockchain Runtime Environment is the platform for running Nebulas Protocol Representation, such as the DIP, the NR, etcetera.

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

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:["/ip4/52.2.205.12/tcp/8680/ipfs/
↪QmQK7W8wrByJ6So7rf84sZzKBxMYmc1i4a7JZsne93ysz5","/ip4/52.56.55.
↪238/tcp/8680/ipfs/QmVy9AHxBpd1iTvECDR7fvdZnqXeDhnxkZJrKsyuHNYKAh",
↪"/ip4/13.251.33.39/tcp/8680/ipfs/
↪QmVm5CECJdPAHmzJWN2X7tP335L5LguGb9QLQ78riA9gw3"]
```

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:

- **NBRE:** The Nebulas Blockchain Runtime Environment is the platform for running Nebulas Protocol Representation, such as the DIP, the NR, etcetera.
- **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"]
```

API List

Test Endpoint:

- GetNebState: returns nebulas client info.
- GetAccountState: returns the account balance and nonce.
- LatestIrreversibleBlock: returns the latest irreversible block.
- Call: execute smart contract locally. The tx won‘t be submitted on chain.
- SendRawTransaction: submit signed transaction. The transaction must be signed before sending.
- GetTransactionReceipt: get transaction receipt info from the transaction 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.

Config

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
}
```

How to Develop

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.

Contributor License Agreement

TBD.

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 https://github.com/golang/dep

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 https://github.com/golang/lint.

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 Slack.

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.

How to debug Go-Nebulas project

ä¡IJèĂĚïijŽWenbo Liu aries.lwb@gmail.com, July 17, 2017

Go-NebulaséąźçŽőåIJřåİĂïijŽhttps://github.com/nebulasio/go-nebulas.git

çőĂäżŃ

è£ŹçŕĞç§■æŰĞå§žäžŐMac OSX åŠŇ Ubuntuçşżçż§ïijŇçőĂå■ŢäżŃçż■åęĆä¡ŢèřČèŕŢGo-NebulaséąźçŽőïijŇäÿżèęĄäżŃçż■äÿL'çğ■æŰźæşŢèřČèŕŢïijŽdlvåŚ¡äžd'èąŇèřČèŕŢïijŇGogland IDEèřČèŕŢïijŇäžěåŔŁVisual Studio CodeèřČèŕŢãĂĆ

èřČèŕŢåŹÍDelveåőL'èčĚ

åIJÍ Mac OSX äÿŁåőL'èčĚDelve

GoogleåőŸæŰźäÿžgolangçŽĎèřČèŕŢä¿Ńå■ŘçŤÍgdbïijŇä¡ĘæŸŕdelveæŸŕæŻt'åŘĹéĂĆçŽĎèřČèŕŢåŹÍïij NebulasãĂĆæŹőéĂŽçŽĎgoéąźçŽőæŸŕåŔŕäžěçŽĎïijŇåĚůä¡Şä¡ŞçŐřåřśæŸŕèřČèŕŢGo-NebulaséąźçŽőæŮűïijŇæŰ■çĆźæŮăæşŢåĄIJä¡ŔïijŇäijŽæřÿè£IJhangä¡ŔãĂĆæĹŚäžňå£ĚéążäżőgithubäÿŁä binaryïijŇæ■ěéłd'åęĆäÿŃïijŽ

åĚĹçŤÍHomebrewåőL'èčĚæIJL'bugçŽĎDelveïijŽ

```
brew install go-delve/delve/delve
rm /usr/local/bin/dlv
```

åőL'èčĚæ■d'æIJL'éŮőécŸçŽĎDelveïijŇåĚűåðđåřśæŸŕäÿžäžĘèől'åőČåÿőæĹŚäżňåIJÍMacæIJžåŹÍäÿŁç■ certèŕĄäźęãĂĆåęĆæđIJä¡äèĞłåůśæĎ£æĎŘçźĄçŘŘçŽĎæL'ŃåŁĺåĹŻåžžèŕĄäźęïijŇäź§åŔŕäžěäÿ■çŤÍåőL'èčĚ a self-signed certificateãĂŚãĂĆ çňňäžŇæİąrmåŚ¡äžd'æŸŕäÿžäžĘåĹäéŹd'è£ŹäÿłæIJL'éŮőécŸçŽĎdlv binaryïijŇæĹŚäžňéIJĂèęĄäžŐæžŘçăĄçijŰèŕŚåĞžäÿĂäÿłæ■čçąőçŽĎçL'ĹæIJñïijŇåźűäÿŤåĹl'çŤÍHomebewä äÿŃè¡¡æžŘäžčçăĄ

```
mkdir -p /Users/xxx/go-delve/src/github.com/derekparker
cd /Users/xxx/go-delve/src/github.com/derekparker
git clone https://github.com/derekparker/delve.git
```

åĹŻåžžäÿĂäÿłäÿt'æŮűæŰĞäžűåd'źïijŇäžŐgithubäÿŃè¡¡äžčçăĄąãĂĆæşĺæĎŔæŰĞäžűåd'źäÿ■æăĞæşĺçžč not foundãĂĆåĚűåőČéČĺåĹĘèŕůæăźæ■őèĞłåůśæIJžåŹÍçŐŕåćČèő¿ç¡őãĂĆ

çijŰèŕŚ

```
export GOPATH=/Users/xxx/go-delve
cd /Users/xxx/go-delve/src/github.com/derekparker/delve
make install
```

åžŤèŕěäijŽåĞžçŐřåęĆäÿŃæŔŘçd'žïijŇèąĺæŸŐçijŰèŕŚæĹŘåŁ§ïijŽ

```
scripts/gencert.sh || (echo "An error occurred when generating and␣
↪installing a new certicate"; exit 1)
go install -ldflags="-s" github.com/derekparker/delve/cmd/dlv
codesign -s "dlv-cert"  /Users/xxx/go-delve/bin/dlv
```

çĎűåŘŐcp /Users/liuwb/go-delve/bin/dlv/usr/local/bin/ïijŇæŁŁçijŰèŕŚåě¡çŽĎdlvæŃůèt'Ìè£Ż/usr/local/
debuggerãĂĆè¿ŞåĚěåŚ¡äžd'dlv versionïijŇåęĆæđIJèČ¡æ■čåÿÿè£ŘèąŇïijŇæŸ¿çd'žçL'ĹæIJňåŘůïijŇèŕt'æŸŐ

åIJÍ Ubuntu äÿŁåőL'èčĚDelve

åŕźäžŐUbuntuçşžçž§ïijŇåŔŕäžěçŻt'æŐěä¡£çŤĺäÿŃéİćçŽĎæŇĞäžd'åőL'èčĚDelveïijŽ

```
go get -u github.com/derekparker/delve/cmd/dlv
```

äÿŃè¡¡Go-NebulasåůěḉŃäžčçăĄ

```
mkdir /Users/xxx/workspace/blockchain/src/github.com/nebulasio/
cd /Users/xxx/workspace/blockchain/src/github.com/nebulasio/
git clone https://github.com/nebulasio/go-nebulas.git
```

åĹŻåžžäÿĂäÿłäÿt'æŮűæŰĞäžűåd'źïijŇäžŐgithubäÿŃè¡¡äžčçăĄąĂĆæşĺæĎŔæŰĞäžűåd'źäÿ■æăĞæşĺçžč

DelveåŚ¡äžd'èąŇèřČèŕŢ åęĆæđIJä¡ăäžěåL'■çŤĺgdbèřČèŕŢè£ĞCçĺŃåžŔïijŇåŕźdlvåŚ¡äžd'èąŇèřČèŕŢçŽ
è£ŹéĞŇåŔłäžŃçż■debugéČĺåĹĘãĂĆ

è¿ŞåĚěåęĆäÿŃåŚ¡äžd'è£ŻåĚèèřČèŕŢ

```
export GOPATH=/Users/xxx/workspace/blockchain/
cd /Users/xxx/workspace/blockchain/
dlv debug github.com/nebulasio/go-nebulas/cmd/neb -- --config /
↪Users/xxx/workspace/blockchain/src/github.com/nebulasio/go-
↪nebulas/conf/default/config.conf
```

è£ŘèąŇæŮăèŕŕçŽĎèŕİïijŇäijŽè£ŻåĚědebug sessionïijŽ

```
Type 'help' for list of commands.
(dlv)
```

æĹŚäžňæL'ŞçőŮåIJÍnebçŽĎåĞ¡æŢřåĚěåŔčèő¿ç¡őæŰ■çĆźïijŇè¿ŞåĚěåŚ¡äžd'

```
(dlv) break main.neb
Breakpoint 1 set at 0x4ba6798 for main.neb() ./src/github.com/
↪nebulasio/go-nebulas/cmd/neb/main.go:80
(dlv)
```

dlvèřČèŕŢåŹĺæŔŘçd'žäžčçăĄåřĘåIJÍcmd/neb/main.goçŽĎèąŇåŘů80èąŇåĄIJä¡ŔïijŇæşĺæĎŔè£ŹæŮűne

```
(dlv) continue
> main.neb() ./src/github.com/nebulasio/go-nebulas/cmd/neb/main.
↪go:80 (hits goroutine(1):1 total:1) (PC: 0x4ba6798)
    75:         sort.Sort(cli.CommandsByName(app.Commands))
    76:
    77:         app.Run(os.Args)
    78:     }
    79:
=>  80:     func neb(ctx *cli.Context) error {
    81:         n, err := makeNeb(ctx)
    82:         if err != nil {
    83:             return err
    84:         }
    85:
```

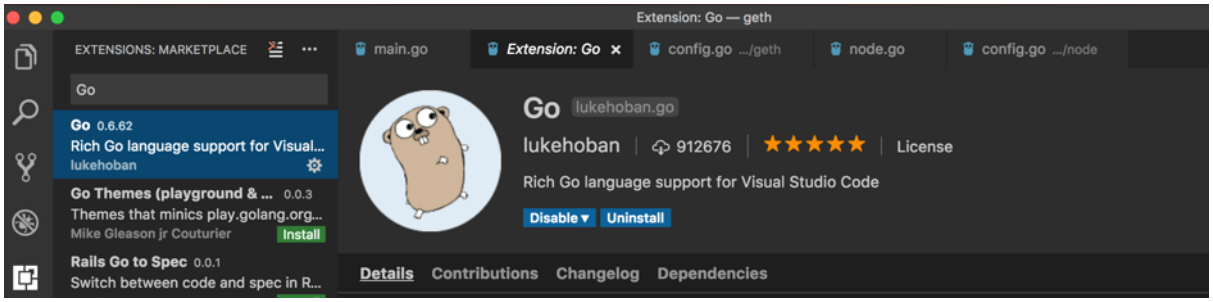
æ§ěçIJŃåŔŸéĞŕïijŇåŔŕçŤĺprintåŚ¡äżď’ïijŽ

```
(dlv) print ctx
*github.com/nebulasio/go-nebulas/vendor/github.com/urfave/cli.
↪Context {
    App: *github.com/nebulasio/go-nebulas/vendor/github.com/urfave/
↪cli.App {
        Name: "neb",
        HelpName: "debug",
        Usage: "the go-nebulas command line interface",
        UsageText: "",
        ArgsUsage: "",
        Version: ", branch , commit ",
        Description: "",
        Commands: []github.com/nebulasio/go-nebulas/vendor/github.
↪com/urfave/cli.Command len: 11, cap: 18, [
            (*github.com/nebulasio/go-nebulas/vendor/github.com/
↪urfave/cli.Command)(0xc4201f4000),
            (*github.com/nebulasio/go-nebulas/vendor/github.com/
↪urfave/cli.Command)(0xc4201f4128),
            (*github.com/nebulasio/go-nebulas/vendor/github.com/
↪urfave/cli.Command)(0xc4201f4250),
            (*github.com/nebulasio/go-nebulas/vendor/github.com/
↪urfave/cli.Command)(0xc4201f4378),
            (*github.com/nebulasio/go-nebulas/vendor/github.com/
↪urfave/cli.Command)(0xc4201f44a0),
```

æŻt’åď’ŽæŁĂæIJŕèţĎæŰŹïijŇèŕůåŔĆèĂČ https://github.com/derekparker/delve/tree/master/Documentation/cli https://blog.gopheracademy.com/advent-2015/debugging-with-delve/ http://hustcat.github.io/getting-started-with-delve/

Visual Studio CodeèřČèŕŢ

Visual Studio CodeæŸŕå¿őè¡råĚňåŔŷåŔŚåŷČçŽĎèůĺåźşåŔřäžčçăAçijŰè¿ŚåůěåĚůïijŇäŷŃè¡¡åIJřåİĂïijŽ //code.visualstudio.com/Download VS CodeéIJĂèęĄåőL'èčĚGoæŔŠäžű



æL'ŞåijĂæŰĞäžűåď'ź/Users/xxx/workspace/blockchain/src/github.com/nebulasio/go-nebulas/ïijŇåIJÍ.vscodeæŰĞäžűåď'źäŷŃåĹŻåžžäŷď'äŷłæŰĞäžűsettings.jsonåŠŇlaunch.jsonãĂĆ settings.jsonæŰĞäžűåĘĚåőźïijŽ

```
// Place your settings in this file to overwrite default and user␣
↪settings.
{
    "go.gopath": "/Users/xxx/workspace/blockchain/",
    "go.formatOnSave": true,
    "go.gocodeAutoBuild": false,
    "go.toolsGopath": "/Users/xxx/workspace/gotools",
    "explorer.openEditors.visible": 0,
}
```

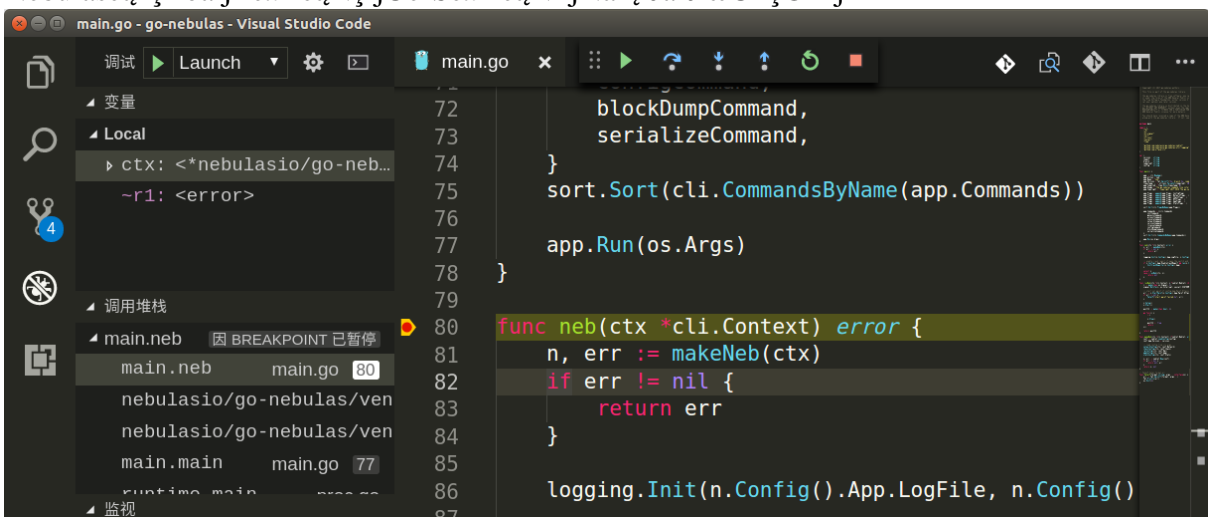
go.toolsGopathæŸŕanalysis toolsåőL'èčĚçŽĎåIJřåİĂïijŇåŔŕäžěæŇĞåőŽäŷžäžżä¡ŢçŽőå¡ŢïijŇè£ŹäžŻan toolsåŔŕäžěä¿ŻåĚűåőČworkspaceåĚśäžńãĂĆ

launch.jsonæŰĞäžűåĘĚåőźïijŽ

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Launch",
            "type": "go",
            "request": "launch",
            "mode": "debug",
            "program": "${workspaceRoot}/cmd/neb",
            "env": {
                "GOPATH": "/Users/xxx/workspace/blockchain/"
            },
            "args": [
                "--config",
                "/Users/xxx/workspace/blockchain/src/github.com/
↪nebulasio/go-nebulas/conf/default/config.conf"
            ],
            "showLog": true
```

```
            }
        ]
}
```

åIJÍcmd/neb/main.goïijŇnebåĞ¡æŢřäÿ■èő¿ç¡őæŰ■çĆźïijŇF5è£ŘèąŇïijŇGo-NebulaséąźçŻőäijŽè£ŻèąŇçijŰèŕŚè£ŘèąŇïijŇåĄIJåIJÍæŰ■çĆźïijŽ



çĎűåŘŐïijŇåřśåŔŕäżěåijĂå£ČçŽĎåŘŕåŁÍNebulasäżčçăĄèřČèŕŢäźŃæŮĚïijĄ

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

Tutorials

Nebulas 101 - 01 Compile and Install Nebulas

The current version of Nebulas Mainnet is 2.0, 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

# environment variables
export GOPATH=/path/to/workspace
```

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/go1.12.linux-amd64.tar.gz

# extract
tar -C /usr/local -xzf go1.12.linux-amd64.tar.gz

# environment variables
export PATH=$PATH:/usr/local/go/bin
export GOPATH=/path/to/workspace
```

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

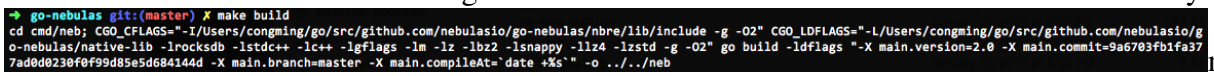
- Set up runtime environment

```
cd /path/to/workspace
source setup.sh
```

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

```
cd /path/to/workspace
make build
```

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



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: 100
}

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/`

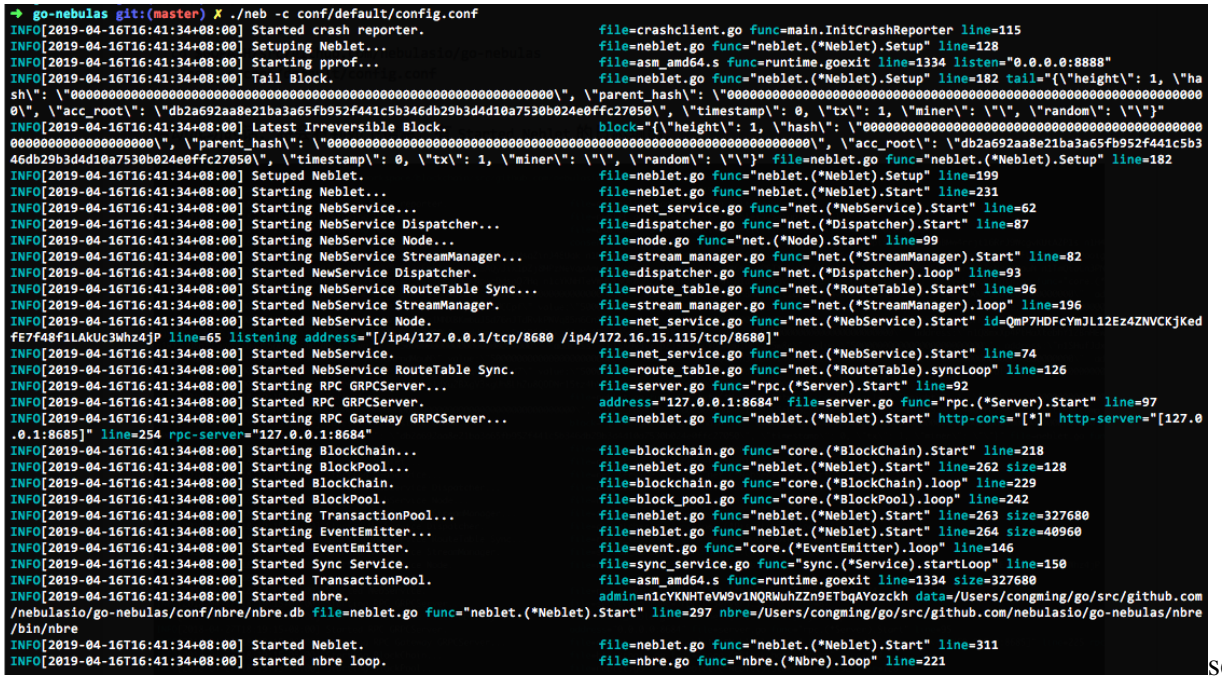
Run Nodes

> 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:

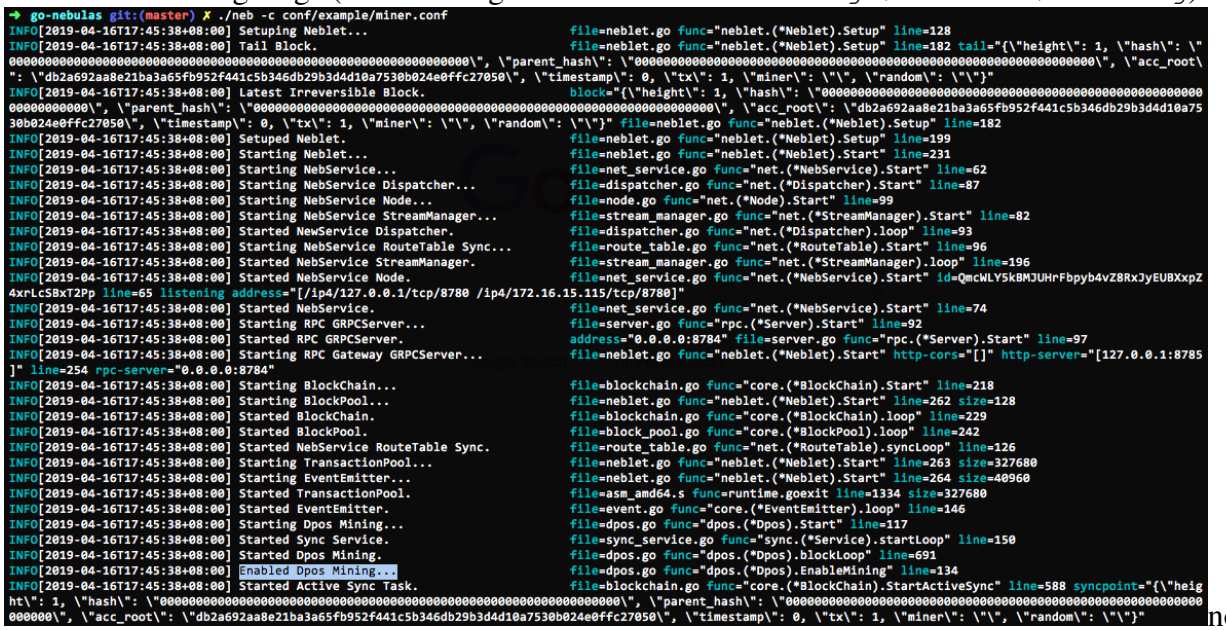


seed node start

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`):



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

Nebulas 101 - 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, DPoS is chosen as the temporary consensus algorithm before Proof-of-Devotion(PoD, described in Technical White Paper) is ready. In this 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": "500000000000000000000000",
        "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+Z8evXJoA35caGhlXbip8PupTNxwV4SRM87r798jXWADXpWngIhAAAAAAAAAAAA
 ↪"}}
```

> **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":""}}âŔŐ
```

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 succeseed.
- **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":"30000000000000000000","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

Nebulas 101 - 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 bal-

anceOf() 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_
 ↪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){var result=Blockchain.
 ↪verifyAddress(address);return{valid:result==0?false:true};}};
 ↪module.exports=BankVaultContract;","sourceType":"js", "args":""}},
 ↪ "passphrase": "passphrase"}'

{"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 asyn-

chronous 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":
↪"eyJTb3VyY2VUeXBlIjoianMiLCJTb3VyY2UiOiJcInVzZSBzdHJpY3RcIjt2YXIgRGVwb3Npc
↪ZmFsc2U6dHJ1ZX07fX07bW9kdWxlLmV4cG9ydHM9QmFua1ZhdWx0Q29udHJhY3Q7IiwiQXJncy
↪","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

Nebulas 101 - 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

Nebulas 101 - 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

Todo List

Go-Nebulas

- åÿőåŁľæţŃèŕŢNebulasïijŇèőľNebulasæŻťåŁååĄěåčőïijŇæŻťåď˙ŽèŕęæČĚ

- åÿőåŁľNebulasåijĂåŔŚåd'Žèŕ■èÍĂçŽĎsdkåžŞïijŇåęĆNode, Ruby, Python, Php, Javaç■Ľ'ç■Ľ'ïijŇæŐěåŔčæŰĞæąč
- åÿőåŁľNebulasåijĂåŔŚåŔŕäžd'äžŠæŰĞæąčïijŇæŐěåŔčæŰĞæąč

Research

- åÿőåŁľäijŸåŇŰNebulas RankçőŮæşŢïijŇæŻť'åd'Žèŕęæ ČĚ
- åÿőåŁľNebulasåĹľçŤĺå¡ćåijŔåŇŰèŕĄæŸŐçŽĎæŰźåijŔèŕĄæŸŐåĚśèŕĘçőŮæşŢPoDçŽĎåőL'åĚĺæĂğï

Wiki

- åÿőåŁľNebulasæŁŁwikiç£żèŕŚæĹŘä¡ăçŽĎæŕ■èŕ■ïijŇèőľ'æŻť'åd'ŽäžžäžĘèğčåŇžåİŮéŞ¿ïijŇäžĘèğčNe
- åÿőåŁľNebulasåĹűä¡IJçğŚæŹőæŢŹçĺŃïijŇåÿőåŁľæŻť'åd'ŽæšąæIJL'çijŰçĺŃèČŇæŹŕçŽĎäžžçŘĘèğčåź

Explorer

- åÿőåŁľNebulasåőŇåŰĎåŇžåİŮæţŔèğĹåŹĺïijŇæŻť'åd'Žèŕęæ ČĚ

Wallet

- åÿőåŁľNebulasåőŇåŰĎç¡Śéąţç L'ĹéŠśåŇĚïijŇæŻť'åd'Žèŕęæ ČĚ
- åÿőåŁľNebulasåőŇåŰĎæąŇéİćçL'ĹéŠśåŇĚïijŇæŻť'åd'Žèŕęæ ČĚ

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.

Infrastructure

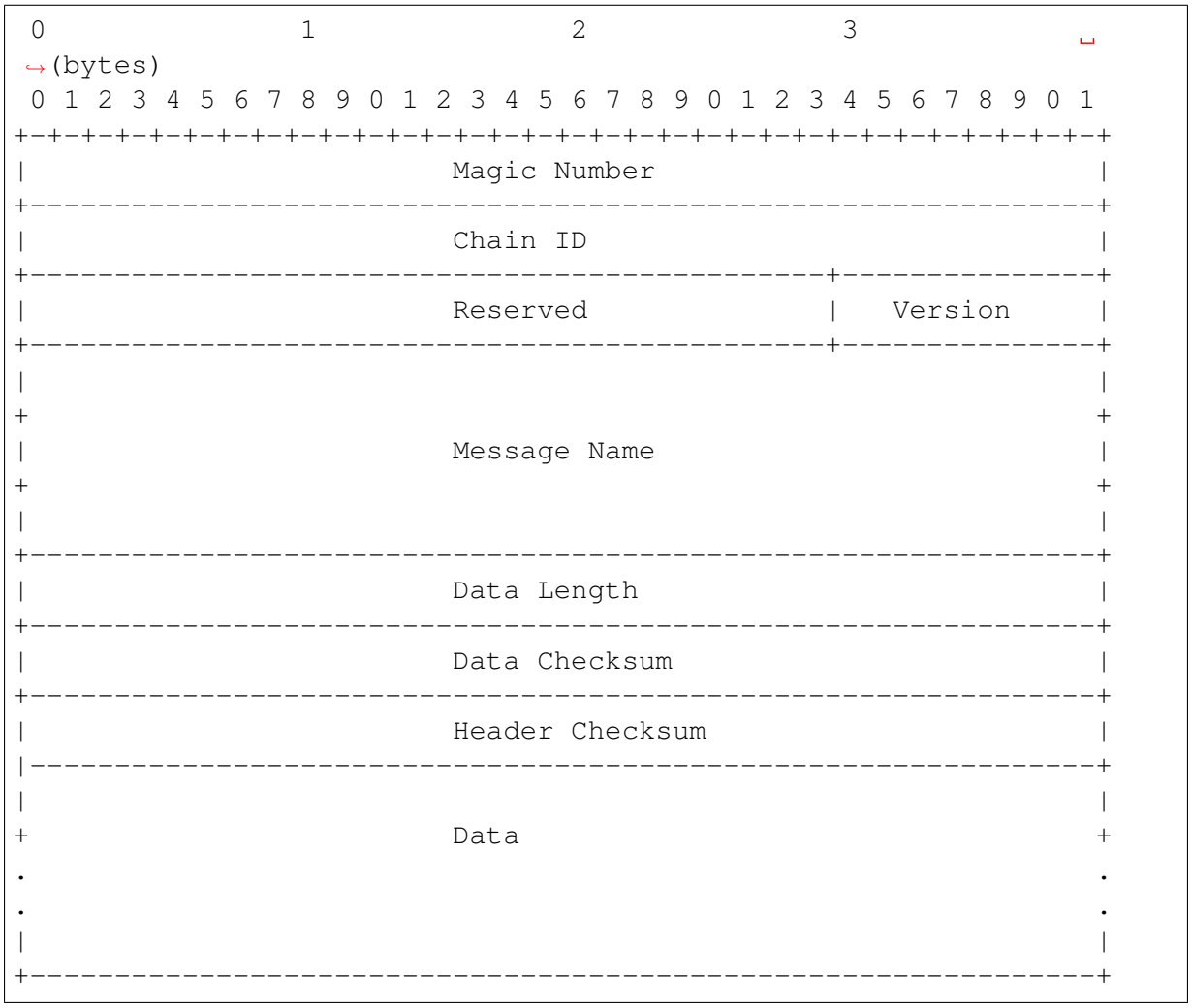
Network Protocol

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:



- 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.

Nebulas V8 Engine

Nebulas V8 Engine is

LLVM Engine

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);
     })
```

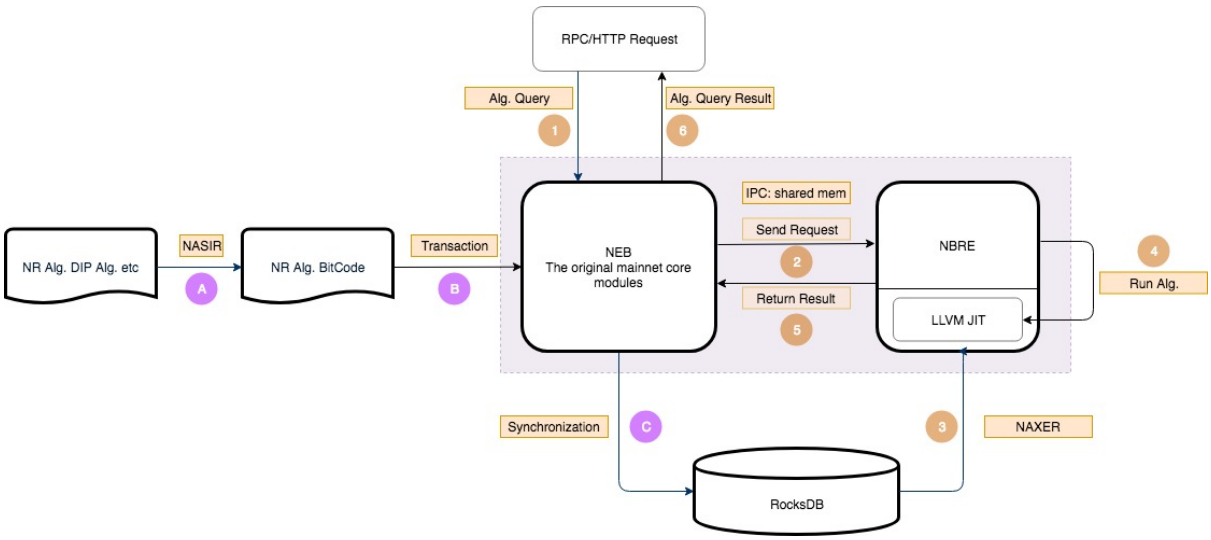
Tips:

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

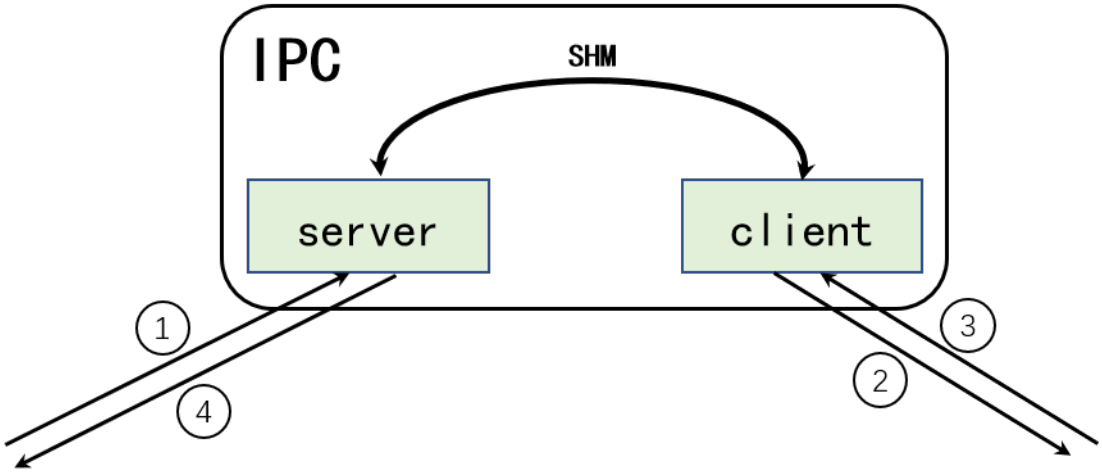
IPC is the messenger for NEB and NBRE interaction.

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

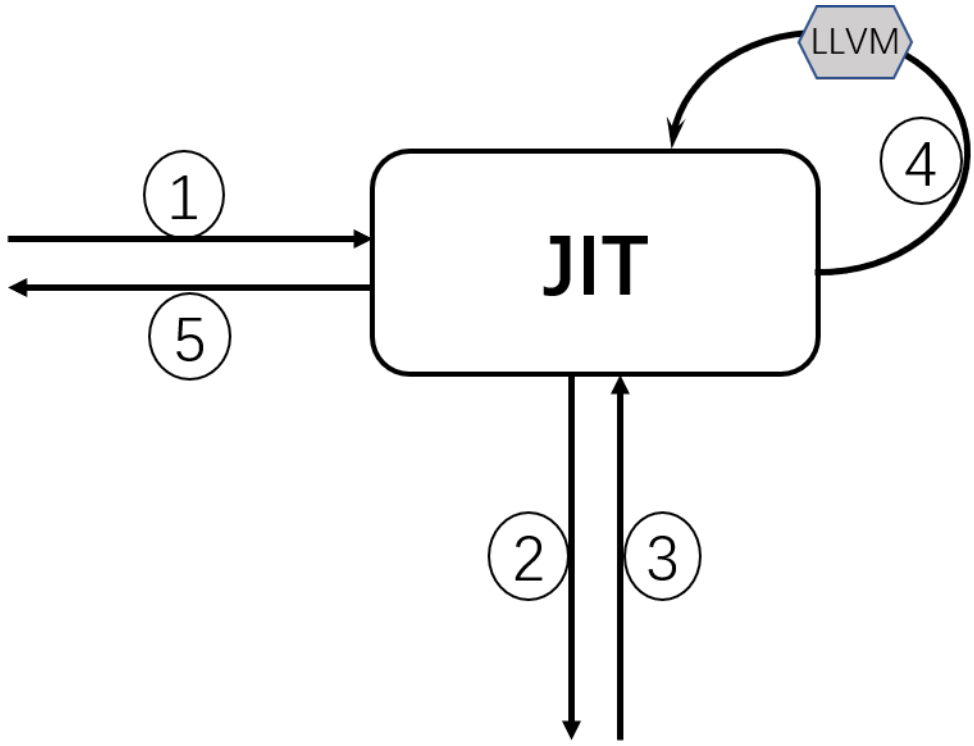
NBRE's updates are performed by adding algorithms and interface programs to the database. When a new function is updated or called, the corresponding program will be loaded into the JIT in the database.

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.

2.4.6 Roadmap

Please visit our new Roadmap here.

Milestones

- In 2017 December, Nebulas test-net will be online.
- In 2018 Q1, Nebulas v1.0 will be released and main-net will be online (ahead of the original schedules).

v1.0 (2018 Q1)

- Fully functional blockchain, with JavaScript and TypeScript as the languages of Smart Contract.
- A user-friendly Nebulas Wallet for both desktop and mobile device to manage their own assets on Nebulas.
- A web-based Nebulas Block Explorer to let developers and users search and view all the data on Nebulas.

v2.0 (2018 Q4)

- Add Nebulas Rank (NR) to each addresses on Nebulas, help users and developers finding more values inside.
- Implement Developer Incentive Protocol (DIP) to encourage developers build more valuable decentralized applications on Nebulas.

v3.0 (2019 Q4)

- Fully functional Nebulas Force and PoD implementation.

Long term goals

- Scalability for large transaction volume.
- Subchain support.
- Zero-knowledge Proof integration.

Versions

v0.1.0 [done]

Goals

- Implement a nebulas kernel.
- In-memory blockchain with PoW consensus.
- Fully P2P network support.

Download here.

v0.2.0 [done]

Goals

- Provide (RPC) API to submit/query transaction externally.
- Implement Sync Protocol to bootstrap any nodes that join into nebulas network at any time, from any tail.

Core

- Implement transaction pool.
- Prevent record-replay attack of transaction.
- Integrate Protocol Buffer for serialization.

Net

- Refactor the design of network.
- Implement Sync Protocol.
- Implement Broadcast and Relay function.

API

- Add Balance API.
- Add Transaction API.
- Add some debugging API, eg âĂIJDump ChainâĂİ, âĂIJDump BlockâĂİ.

Crypto

- Support Ethereum-keystore file.
- Support multi key files management in KeyStore.

Download here.

v0.3.0 [done]

Goals

- Support disk storage for all blockchain data.
- Add smart contract execution engine, based on Chrome V8.

Core

- Add disk storage with a middleware of storage.
- Implement smart contract transaction.

NVM

- Integrate Chrome V8 as Smart Contract execution engine.

Download here.

v0.4.0 [done]

Goals

- Implement Gas calculating in Smart Contract Execution Engine.
- Support more API.
- Add repl in neb application.
- Add metrics and reporting capability.

Core

- Add Gas related fields in Transaction.
- Implemented Gas calculation mechanism.

NVM

- Add execution limits to V8 Engine.
- Add Gas calculation mechanism.

CMD

- Add repl in neb application

Misc

- Add more API.
- Add metrics and reporting capability.

Download here.

v0.5.0 [done]

Goals

- Prepare for test-net releasing, improve stability.

Core

- Improve stability and missing functions if we miss anything.

Consensus

- Implement DPoS consensus algorithm and keep developing PoD algorithm.

NVM

- Finalize the Gas Cost Matrix.
- Support Event liked pubsub functionality.

Misc

- Add more metrics to monitor the stability of neb applications.

Download here.

v0.6.0 [done]

Goals

- Stability improvement, performance optimization.
- Reconstruct P2P network.
- Redesign block sync logic.

Testnet

- Fix bugs & improv the performance.

Network

- Add *Stream* for single connection management.
- Add *StreamManager* for connections management.
- Implement priority message *chan*.
- Add route table persistence strategy.
- Improve strategy to process TCP packet splicing.

Log

- Add console log(CLog), printing log to both console & log files, to inform developers whatâĂŹs happening in Neb.
- Add verbose log(VLog), printing log to log files, to inform devs how Neb works in details.
- Log adjustment.

Sync

- Use chunk header hash to boost the sync performance.
- Adjust the synchronous retry logic and timeout configuration.
- Fix bugs in synchronization and add more metrics statistics.

Download here.

v0.6.1 [done]

Goals

- Improve test net compatibility.

Core

- Upgrade the storage structure of the block

Download here.

v0.8.0 [done]

Goals

- New Nebulas Block Explorer.
- New Nebulas Wallet.
- New web-based Playground tools to interactive with Nebulas.

v1.0.0 [done]

Goals

- Ready for main-net.
- Support JavaScript and TypeScript as Smart Contract Language.
- Stable and high performance blockchain system.
- Release new Nebulas Block Explorer.
- Release new Nebulas Wallet for both desktop and mobile device.
- A web-based playground tools for developer.

Download explorer.

Download wallet.

Download neb.js.

2.4.7 DApp Development

Smart Contract

Languages

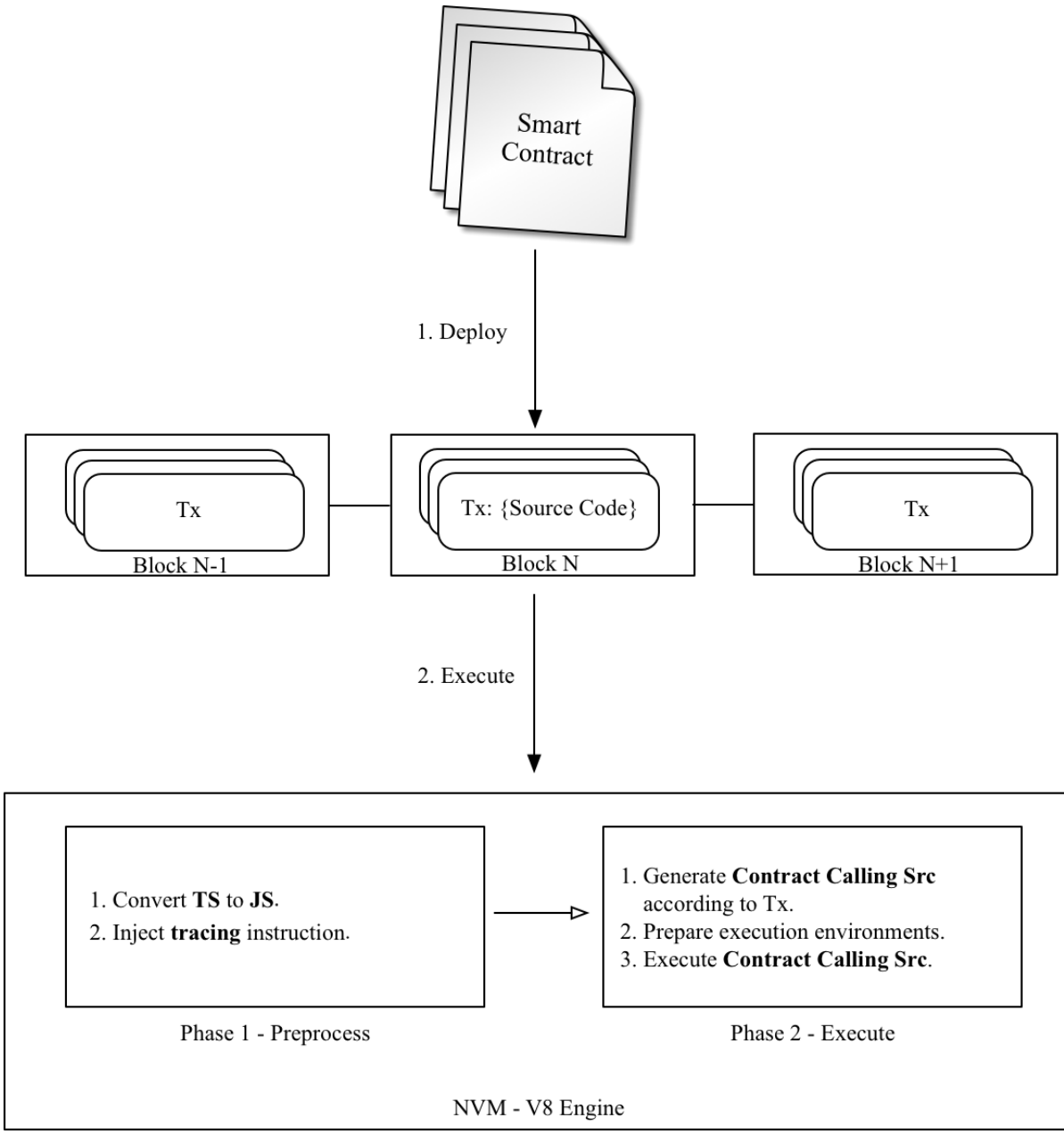
In Nebulas, there are two supported languages for writing smart contracts:

- 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 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:
   (a) Preprocess: inject tracing instruction, etc.
   (b) 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.

```
interface Descriptor {
    // serialize value to string;
    stringify?(value: any): string;

    // deserialize value from string;
    parse?(value: string): any;
}

interface DescriptorMap {
    [fieldName: string]: Descriptor;
}

interface ContractStorage {
    // get and return value by key from Native Storage.
    rawGet(key: string): string;
    // set key and value pair to Native Storage,
    // return 0 for success, otherwise failure.
    rawSet(key: string, value: string): number;

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

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

    // define a StorageMap property named `fieldname` to `obj` with␣
↪descriptor.
    // default descriptor is JSON.parse/JSON.stringify descriptor.
    // return this.
    defineMapProperty(obj: any, fieldName: string, descriptor?:␣
↪Descriptor): any;
```

```
    // define StorageMap properties to `obj` from `props`.
    // default descriptor is JSON.parse/JSON.stringify descriptor.
    // return this.
    defineMapProperties(obj: any, props: DescriptorMap): any;

    // delete key from Native Storage.
    // return 0 for success, otherwise failure.
    del(key: string): number;

    // get value by key from Native Storage,
    // deserialize value by calling `descriptor.parse` and return.
    get(key: string): any;

    // set key and value pair to Native Storage,
    // the value will be serialized to string by calling␣
↪`descriptor.stringify`.
    // return 0 for success, otherwise failure.
    set(key: string, value: any): number;
}

interface StorageMap {
    // delete key from Native Storage, return 0 for success,␣
↪otherwise failure.
    del(key: string): number;

    // get value by key from Native Storage,
    // deserialize value by calling `descriptor.parse` and return.
    get(key: string): any;

    // set key and value pair to Native Storage,
    // the value will be serialized to string by calling␣
↪`descriptor.stringify`.
    // return 0 for success, otherwise failure.
    set(key: string, value: any): number;
}
```

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**.

  ‘‘‘js

  “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
```js
"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¡ŹéćÌä£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;
```

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**

DApps Design Guide

TBA

You can download PDF here.

Learning Resources

All learning resources. Videos and documents. Welcome to recommend more resources from the community, and you can edit this page on Github directly. Help others and learn things together.

Official Nebulas Documents

- Nebulas Mauve Paper: Developer Incentive Protocol: [English], [Chinese]
- About the Nebulas Mauve Paper: Developer Incentive Protocol
- Nebulas Rank Yellow Paper: [English], [Chinese], [Korean], [Portuguese]
- Official Interpretation of the âĂIJNebulas Rank Yellow PaperâĂİ
- Technical Whitepaper: [English], [Chinese]
- 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 Wallet

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

- How to use NebPay in your Dapp

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

RPC Overview

Overview

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:

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␣
↪314283103999999999992
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*
- *LatestIrreversibleBlock*
- *Call*
- *SendRawTransaction*
- *GetBlockByHash*
- *GetBlockByHeight*
- *GetTransactionReceipt*
- *GetTransactionByContract*
- *GetGasPrice*
- *EstimateGas*
- *GetEventsByHash*
- *Subscribe*
- *GetDynasty*

RPC API Reference

GetNebState

Return the state of the neb.

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.

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":"94899999989800000000"
        "nonce":51
        "type":87
            "height":"100",
        "pending":"0"
    }
}
```

LatestIrreversibleBlock

Return the latest irreversible block.

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.

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.

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/
↪5jSBToICgZiaW5hcnlAZEoQAAAAAAAAAAAAAAAAA9CQFIQAAAAAAAAAAAAAAAAABOIFgBYkGLnnv
↪"}'

// 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.

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.

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.

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.

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.

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.

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.

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","gasLimit
↪":"2000000"}'

// Result
{
    "result": {
        "gas":"20000",
        "err":""
    }
}
```

GetEventsByHash

Return the events list of transaction.

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.

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"
            ]
```

```
            }
        }
}
```

Management RPC

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:

Management RPC methods

- *NodeInfo*
- *Accounts*
- *NewAccount*
- *UnLockAccount*
- *LockAccount*
- *SignTransactionWithPassphrase*
- *SendTransactionWithPassphrase*
- *SendTransaction*
- *SignHash*
- *StartPprof*
- *GetConfig*

Management RPC 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.

2.4.8 Tutorials

Here is all you need to dive into Nebulas.

NebulasIO

- Website: https://nebulas.io
- Non-Tech Whitepaper: [PDF]
- Tech Whitepaper: [PDF], [GitHub]
- Yellow Paper - Nebulas Rank: [PDF], [GitHub]
- Mauve Paper - Developer Incentive Protocol: [PDF], [GitHub]
- Orange Paper - Nebulas Governance: [PDF], [GitHub]

Go-Nebulas

- Wiki: https://github.com/nebulasio/wiki
- Join the Testnet: https://github.com/nebulasio/wiki/blob/master/testnet.md
- Join the Mainnet: https://github.com/nebulasio/wiki/blob/master/mainnet.md
- Explorer: https://explorer.nebulas.io
- Tutorials:
  - Nebulas 101

        * Installation (credit: Victor)
        * Sending a Transaction (credit: Victor)
        * Writing Smart Contract in JavaScript (credit: otto)
        * Introducing Smart Contract Storage (credit: Victor)
        * Interacting with Nebulas by RPC API (credit: Victor)

Wallet

- Web Wallet: https://github.com/nebulasio/web-wallet
  - English
    * Creating A NAS Wallet
    * Sending NAS from your Wallet
    * Signing a Transaction Offline
    * View Wallet Information
    * Check TX Status
    * Deploy a Smart Contract

DApp Development

- Web SDK: https://github.com/nebulasio/neb.js
- Smart Contract: https://github.com/nebulasio/wiki/blob/master/smart_contract.md
- Standard Protocol:
  - NRC20: https://github.com/nebulasio/wiki/blob/master/NRC20.md

Community Tools

- Nebulearn: https://nebulearn.com/official-docs/go-nebulas (credit: Tehjr)
- Demo DApp: https://github.com/15010159959/super-dictionary (credit: ChengOrangeJu, yupnano, Kurry)
- Chrome Extension: https://github.com/ChengOrangeJu/WebExtensionWallet (credit: ChengOrangeJu, yupnano)

Contribution

You are welcome to contribute to the Nebulas ecosystem in any way that you can. If you write something, please let us know by submitting an issue, and we will add your name and url to this page as soon as possible.

2.4.9 Community

Dynamics

- (01/31/2019) - Nebulas Had A Fruitful Trip to Korea!
- (01/29/2019) - Week 1 Winners of Nebulas NOVA Testnet Developer Incentive Program
- (01/23/2019) - Game of Chains 2019: An Interview with Dr. Chen of Nebulas
- (01/23/2019) - Winners of Nebulas NOVA Developer Incentive Program AMA
- (01/22/2019) - Nebulas NOVA Testnet Developer Incentive Program Launches Today
- (01/17/2019) - The First Winners of Nebulas Wiki Bounty Program
- (01/11/2019) - Understanding Nebulas NOVA (Part 2)
- (01/11/2019) - Nebulas New Explorer Goes Live
- (01/10/2019) - AMA on Nebulas NOVA Developer Incentive Program
- (01/09/2019) - Nebulas Testnet Developer Incentive Program Event Guide
- (01/05/2019) - Nebulas 2018; the year in review!
- (01/04/2019) - How well do you know Nebulas NOVA?
- (01/03/2019) - Understanding Nebulas NOVA (Part 1)
- (12/31/2018) - Nebulas NOVA Testnet Released, Public Beta Testing Begins!
- (12/29/2018) - The Nebulas That IâĂŹm Looking Forward to
- (12/27/2018) - NAS nano has been upgraded to NAS nano pro
- (12/22/2018) - The Inspiration Behind the Nebulas NOVA Design
- (12/21/2018) - Why Join Nebulas
- (12/20/2018) - LetâĂŹs Check Your Core Nebulas Rank!
- (12/11/2018) - NebulersâĂŹ Thoughts on the Future of Blockchain
- (12/06/2018) - Behold: The Age of Nebulas NOVA is Upon Us!
- (11/28/2018) - Nebulas Collaborates with Key Universities at Home and Abroad - Sharing the Nebulas Wisdom
- (11/22/2018) - Public Chain TechnologyâĂŁâĂŤâĂŁthe future of blockchain?
- (11/21/2018) - Exploring the Public Chain Technology AllianceâĂŁâĂŤâĂŁBlockchainâĂŹs bridge from concept to creation!
- (11/19/2018) - To Introduce 100 Million Incremental Users to the Blockchain World
- (11/14/2018) - Nebulers are all over the world!
- (11/14/2018) - Embrace An Open and Mutually Beneficial Blockchain Ecosystem

- (11/12/2018) - DApp Development and Architecture DesignâĂŁâĂŤâĂŁInterview with Honey Thakuria
- (11/05/2018) - Let #NebulasNOVA Be a Hot Trend on TwitterïijĄ
- (11/01/2018) - Join Our Mauve Paper Reading Activity!
- (10/25/2018) - Nebulas Joining Public Chain Technology Alliance (PCTA) to Empower Developers Community
- (10/12/2018) - Nebulas Partners with UDAP to Tokenize Everything
- (09/28/2018) - Nebulas Partners with WeOne to Accelerate Global Esports Growth on the Blockchain
- (09/27/2018) - NAS nano Receives Security Audit from Knownsec
- (09/20/2018) - Liberal Radicalism: Can Quadratic Voting Be the Perfect Voting System?
- (09/04/2018) - Hello Beijing and Nebulas Team
- (08/29/2018) - Nebulas Achieving Cooperation with Knownsec, Multiple Protection and Big Data Technologies Supporting Nebulas Ecosystem Security
- (08/24/2018) - Nebulas Community Meetup ReportâĂŁâĂŤâĂŁAmbassadors Visit Beijing
- (08/09/2018) - Seeing Through The Blockchain Bubble: Sitting Down For An Interview With Nebulas.io Founder Hitters Xu
- (08/07/2018) - Blockchain Pioneers Initiate âĂIJBitsclub Vision ProgramâĂİ to Create Seamless Connection of Classical Industry and Blockchain
- (08/01/2018) - Nebulas Featured on ChinaâĂŹs Official State Website
- (08/01/2018) - Why I Love NebulasâĂŁâĂŤâĂŁPart 1: JavaScript!
- (07/31/2018) - Nebulas Partners with JOYSO to Deploy Cutting-Edge Decentralized Exchange
- (07/30/2018) - NAS Nano v2.0 is Officially Released
- (07/28/2018) - GO ! Nebulers ð§ŸĘ
- (07/28/2018) - Nebulas Incentive Program Recap
- (07/27/2018) - Nebulas Melbourne Meet-up, July 23, 2018
- (07/26/2018) - An open letter to the Nebulas community.
- (07/24/2018) - Nebulas Partners with Cocos
- (07/20/2018) - Nebulas in the Top Three of MIITâĂŹs Public Blockchain Evaluation list
- (07/17/2018) - Nebulas Partners with KingSoft Cloud (KSYUN) to Explore Blockchain Games
- (07/13/2018) - What Nebulas Research Team Says About Nebulas Rank Yellow Paper
- (07/12/2018) - Nebulas and Egretia Reach Strategic Cooperation

- (07/08/2018) - Why Choose Nebulas at a Hackathon?
- (07/05/2018) - Official Interpretation of âĂIJNebulas Rank Yellow PaperâĂİ
- (07/03/2018) - Nebulas Attended The Silicon Valley Blockchain Week
- (06/30/2018) - âĂIJThe Nebulas Rank Yellow PaperâĂİ is now public, providing the blockchain world with a more complete value measurement system.
- (06/24/2018) - Nebulas and Udacity partner to create the Global Blockchain Talent Scholarship
- (06/24/2018) - Nebulas mainnet transaction volume exceeds Ethereum, reaching nearly 700,000 for the first time
- (06/23/2018) - Cell Evolution raises 5 million RMB at more than $4.5 million USD valuation
- (06/14/2018) - Nebulas selected by ChinaâĂŹs MIIT in Global Public Chain Evaluation
- (06/14/2018) - SPIKING and NEBULAS Partner to Develop Financial Signals Search and Processing Technology for All Blockchains
- (06/09/2018) - Nebulas welcomes DeepCloud AI
- (06/01/2018) - Experienced Game Developer Bids Adieu to Ethereum and Embraces Nebulas
- (05/31/2018) - Nebulas launches DApp Store in NAS Nano update
- (05/29/2018) - Nebulas CTO Robin Zhong: âĂIJSuper nodes lead to split communitiesâĂİ, and the three key criteria for evaluating âĂŸblockchain 3.0âĂŹ
- (05/29/2018) - Latest Nebulas update paves the way for blockchain games and more!
- (05/08/2018) - Nebulas and Tencent GIS Meetup Promotes Blockchain Innovation
- (05/01/2018) - Nebulas Labs and Atlas Protocol will join Silicon Valley Entrepreneurs Festival
- (04/25/2018) - Next month: Nebulas to host its first workshops and hackathons in the US, and attend Consensus 2018 in New York City
- (04/10/2018) - LLVM x Blockchain
- (03/17/2018) - NAS Center Grand Opening & Nebulas Mainnet Launch Celebration
- (03/02/2018) - Thinking In Blockchain, a Nebulas meetup @Berkeley
- (03/02/2018) - Nebulas Enters Strategic Partnership with Dolphin Browser to Integrate the Nebulas Blockchain within its 200m User Ecosystem
- (02/16/2018) - Decentralization is the Essence of Blockchain
- (01/29/2018) - Crypto bubble 2018: Things we can do before it bursts
- (01/18/2018) - Nebulas Partners with GIFTO to Organize Blockchain Virtual Gifts for 30 Million Users

Events

Since June 2017, the Nebulas meetups and hackathons have been held in 17 cities, 8 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.

Events History >

You are welcome to organize local meetups and participate in the history of Nebulas!

Nebulas Community

Announcement

- (04/08/2019) - Nebulas Community Group (community consultation draft)
- (04/03/2019) - The Declaration of Nebulas Independence: Nebulas Governance is Approaching
- (03/25/2019) - Go Nebulas: the future of collaboration!
- (01/29/2019) - Nebulas NOVA Testnet Developer Incentive Program Launches Today
- (12/27/2018) - NAS nano has been upgraded to NAS nano pro
- (12/07/2018) - Nebulas Wiki Bounty Program
- (12/05/2018) - Announcement: Nebulas Testnet Operation Maintenance
- (11/27/2018) - Nebulas Bug Bounty Improvement
- (10/17/2018) - ATP Smartdrop Applying Process Begins
- (10/15/2018) - Nebulas Mainnet Snapshot of ATP Smartdrop Ended
- (10/10/2018) - Announcement on Token Swap Ends via NAS nano
- (10/08/2018) - Claim Instructions of ATP Smartdrop
- (09/25/2018) - Upgrading NAS nano to Version 2.2.0
- (09/25/2018) - Announcing the NAS Token Swap via NAS nano v2.2.0
- (09/21/2018) - Nebulas Nova: âĂIJDiscover Value in an Organized Blockchain WorldâĂİ
- (09/13/2018) - Announcing the Nebulas Technical Committee
- (09/01/2018) - Announcing Unreleased NAS Locking Addresses
- (08/30/2018) - Increasing Bug Bounty Rewards for Inter-contract Call Functions Testing
- (08/22/2018) - Nebulas Testnet Inter-contract Call Function Public Beta Bounty
- (08/21/2018) - Nebulas Mainnet Security Upgrade Notice
- (08/08/2018) - Announcing the Adjustment of Reserved NAS Distribution to the Nebulas Team

- (06/29/2018) - Nebulas Mainnet Token Swap Announcement
- (06/28/2018) - Nebulas Rank Yellow PaperâĂŁâĂŤâĂŁBlockchain 3.0 Primer
- (06/20/2018) - Nebulas Mainnet Upgrade
- (05/11/2018) - Supplementary explanation for NAS mainnet coinswap
- (04/30/2018) - Announcement on NAS mainnet coin swap starts in exchanges
- (04/25/2018) - Statement on Asset Security from the Nebulas Tech Team
- (04/13/2018) - Important Announcement on the Nebulas Lock Up Bonus Program
- (03/29/2018) - Nebulas Foundation Notice
- (03/27/2018) - Nebulas Testnet Upgrade Announcement
- (12/13/2017) - Disclaimer Regarding Private Pools for NAS Pre-Sale
- (12/13/2017) - New Pricing Rules for NAS Pre-Sale Early Bird Participants
- (11/24/2017) - Initiation of âĂIJNAS Token Bonus ProgramâĂİ

Weekly Report

You can take a quick look at all the Nebulas weekly reports here.

- Weekly Report #75 (04/08/2019) - Nebulas Bi-Weekly Development Commits #76
- Weekly Report #75 (04/01/2019) - Nebulas Weekly Report #75 Go Nebulas has Liftoff!
- Weekly Report #74 (03/25/2019) - Nebulas Bi-Weekly Development Commits #74
- Weekly Report #73 (03/18/2019) - Nebulas Weekly Report #73: PCTA meetup and AMA sessions
- Weekly Report #72 (03/11/2019) - Nebulas Bi-Weekly Development Commits #72
- Weekly Report #71 (03/04/2019) - Nebulas Weekly Report #71: Nebulas community development roadmap officially released
- Weekly Report #70 (02/25/2019) - Nebulas Bi-Weekly Development Commits #70
- Weekly Report #69 (02/19/2019) - Nebula Weekly Report #69: Nebulas Actively Explore Japan & Hong Kong Market in The New Year
- Weekly Report #68 (02/12/2019) - Nebulas Bi-Weekly Development Commits #68
- Weekly Report #67 (02/04/2019) - Nebulas Bi-Weekly Community Dynamics#67
- Weekly Report #66 (01/29/2019) - Development: Nebulas Bi-Weekly Development Commits #66
- Weekly Report #65 (01/22/2019) - Community: Nebulas Bi-Weekly Community Dynamics#65
- Weekly Report #64 (01/15/2019) -Development : We are currently working on new API features and more test cases via the testnet

- Weekly Report #63 (01/07/2019) -Community : Nebulas Bi-Weekly Community Dynamics#63
- Weekly Report #62 (12/31/2018) -Development : We finished all developments of Nebulas Nova features
- Weekly Report #61 (12/24/2018) -Community : Interview with Nebulas Team Series
- Weekly Report #60 (12/17/2018) -Development : The implementation and integration testing of the on-chain NR algorithm have been completed
- Weekly Report #59 (12/10/2018) -Community ïijŽThe Roadmap of Autonomous Metanet was Officially Released
- Weekly Report #58 (12/03/2018) -Development ïijŽCommunity can now submit their NRC 20 project to NAS nano
- Weekly Report #57 (11/26/2018) -Community : Nebulas Technical Committee Meeting Minutes(2018.11.21)
- Weekly Report #56 (11/19/2018) -Development : NBRE Has New Development
- Weekly Report #55 (11/12/2018) -Community : PCTA Launch Press Conference Successfully Held
- Weekly Report #54 (11/05/2018) -Development : Adding ATP Transfer Support
- Weekly Report #53 (10/29/2018) -Community : Nebulas Joined PCTA As One of Its First Partners
- Weekly Report #52 (10/22/2018) -Development : Improving Some Functionalities of NBRE
- Weekly Report #51 (10/15/2018) -Community : NAS Token Swap via NAS nano (v2.2.0) has Ended
- Weekly Report #50 (10/08/2018) -Development : Compeleting the Basic Implementation of NBRE
- Weekly Report #49 (10/01/2018) -Community : Nebulas Nova Development Roadmap was Announced
- Weekly Report #48 (09/24/2018) -Development : Finishing Functional Verification of NBRE
- Weekly Report #47 (09/17/2018) -Community : Nebulas Team Establishes of Nebulas Technical Committee
- Weekly Report #46 (09/10/2018) -Development : Nebulas Rank has been Realized and Open Sourced
- Weekly Report #45 (09/03/2018) -Community : Nebulas Team Announced Unreleased Locking NAS Addresses
- Weekly Report #44 (08/27/2018) -Development : NAS nano is Back to Apple Store Again

- Weekly Report #43 (08/20/2018) -Community : Nebulas Inter-contract Call Function Starts Open Beta
- Weekly Report #42 (08/13/2018) -Announcing the Adjustment of Reserved NAS Distribution to the Nebulas Team
- Weekly Report #41 (08/06/2018) -Nebulas Founders Initiate Bitsclub Vision Program
- Weekly Report #40 (07/31/2018) -Nebulas Incentive Program Season 1 Recap
- Weekly Report #39 (07/24/2018) -Nebulas in the Top Three of MIITâĂŹs Public Blockchain Evaluation list
- Weekly Report #38 (07/17/2018) -The 50 Monthly Super Contributors Were Announced
- Weekly Report #37 (07/10/2018) -Official Interpretation of the Nebulas Rank Yellow Paper
- Weekly Report #36 (07/03/2018) -The âĂIJNebulas RankâĂİ Yellow Paper is now public
- Weekly Report #35 (06/26/2018) -Nebulas participates in the Silicon Valley Blockchain Week hackathon
- Weekly Report #34 (06/19/2018) -Dapp built on Nebulas wins Beijing hackathon
- Weekly Report #33 (06/12/2018) -NIP gets upgraded with Super Contributors
- Weekly Report #32 (06/05/2018) -NAS Nano update features a built-in Dapp Store
- Weekly Report #31 (05/29/2018) -NAS Nano, the official Nebulas mobile wallet, launches on Android and iOS
- Weekly Report #30 (05/22/2018) -the winners for Week 1 of NIP and awarded nearly $350,000 USD in NAS
- Weekly Report #29 (05/15/2018) -Nebulas Attends Blockchain Technology Forum at Google NY
- Weekly Report #28 (05/08/2018) -Our three co-founders attended the Nebulas Community Meeting in Shanghai
- Weekly Report #27 (05/01/2018) -The Nebulas Incentive Program Is About to Kick Off
- Weekly Report #26 (04/24/2018) -NVM new feature design
- Weekly Report #25 (04/17/2018) -Important Announcement on Nebulas Lock Up Bonus Programm
- Weekly Report #24 (04/10/2018) -Dive into NebulasâĂŁâĂŤâĂŁour new Tech column on Medium
- Weekly Report #23 (04/03/2018) -Nebulas Wins Best Performance Award in Innovative District
- Weekly Report #22 (03/27/2018) -Nebulas held an online Tech Reddit AMA
- Weekly Report #21 (03/20/2018) -Nebulas set the launch date for its Mainnet 1.0
- Weekly Report #20 (03/13/2018) -Nebulas held a NYAI Meetup

- Weekly Report #19 (03/06/2018) -Nebulas Global Tour officially kicked off
- Weekly Report #18 (02/27/2018) -Nebulas First Reddit AMA Comes to a Successful Conclusion
- Weekly Report #17 (02/20/2018) -Nebulas Writing Contest Rounded Off
- Weekly Report #16 (02/13/2018) -Nebulas is Holding an Online Reddit AMA
- Weekly Report #15 (02/06/2018) -NebulasâĂŹ Silicon Valley Meetup
- Weekly Report #14 (01/30/2018) -NebulasâĂŹ Trip to Silicon Valley
- Weekly Report #13 (01/23/2018) -Nebulas Davos and Silicon Valley Trips
- Weekly Report #12 (01/16/2018) -Hitters Present at the Blockchain Meetup of The Economist China Readers Club
- Weekly Report #11 (01/09/2018) -Nebulas Testnet Upgraded
- Weekly Report #10 (12/26/2017) -Nebulas CTO Robin Zhong Present at CIE Seminar
- Weekly Report #09 (12/19/2017) -Tsinghua University Talks Well-received
- Weekly Report #08 (12/12/2017) -NAS Token Exchange With Bonus Program a Complete Success
- Weekly Report #07 (12/05/2017) -Nebulas Token Exchange Program with Bonus is ending soon!
- Weekly Report #06 (11/27/2017) -Initiation of âĂIJNAS Token Bonus ProgramâĂİ
- Weekly Report #05 (11/20/2017) -Singapore FinTech Festival
- Weekly Report #04 (11/13/2017) -Columbia University, New York / Nebulas Meetup
- Weekly Report #03 (11/6/2017) -Developing v0.3.0 and improving the Go-nebulas
- Weekly Report #02 (10/30/2017) -Singapore Fintech Festival
- Weekly Report #01 (10/16/2017) -Welcome to the #1 of Nebulas Weekly Report

Ask Me Anything

Nebulas Reddit AMA

- Reddit Questions and Answers - Reddit Weekly Question Recap! (10.29–11.4)
- Reddit Questions and Answers - Reddit Weekly Discussion Recap!ïijĹ10.21–10.26ïijL’
- Nebulas Reddit AMA Recap - With Nebulas Founder Hitters Xu and Co-founder Aero Wang
- Reddit Questions and Answers - Nebulas Weekly AMA & Constructive SuggestionsâĂŁâĂŤâĂŁAugust 6 to August 19
- Reddit Questions and Answers - Nebulas Weekly AMA & Constructive SuggestionsâĂŁâĂŤâĂŁJuly 30 to August 6 2018

- Reddit Questions and Answers - Nebulas Weekly AMA & Constructive Suggestion-sâĂŁâĂŤâĂŁJuly 20 to July 29
- Nebulas First Live Reddit AMA - With Nebulas Founder Hitters Xu
- NebulasâĂŹ First Reddit AMA Recap - Answers and Viewpoints of Nebulas Founder Hitters Xu
- Tech Reddit AMA - With Nebulas CTO Robin Zhong
- Nebulas AMA Series#1 - Testnet with Nebulas Co-Founder and CTO Robin Zhong
- Nebulas AMA Series#2 - Testnet with Nebulas Co-Founder and CTO Robin Zhong
- Nebulas AMA Series#3 - General Question with Nebulas Co-Founder and CTO Robin Zhong
- Answers from AMA - With Nebulas lead core developer Roy Shang

PCTA Reddit AMA Series

- PCTA Reddit AMA Series 1 Recap - Nebulas & XMAX Reddit AMA Recap#Part 1
- PCTA Reddit AMA Series 1 Recap - Nebulas & XMAX Reddit AMA Recap#Part 2
- PCTA Reddit AMA Series 2 Recap - BCH Hard Fork, Beneficial or Harmful
- PCTA Reddit AMA Series 3 Recap - What can we learn from the recent market crash?

Nebulas Interviews

Interviews with Nebulas Team

- Interview with Nebulas Team Series 3 - The Nebulas That IâĂŹm Looking Forward to
- Interview with Nebulas Team Series 2 - Why Join Nebulas
- Interview with Nebulas Team Series 1 - Nebulers‘ Thoughts on the Future of Blockchain

6 Minutes Learning Nebulas NOVA with 92k Lines of Code

Nebulas team just released the Nebulas NOVA on testnet. We invited the Nebulas Chief Architect, PhD. Chen, to introduce Nebulas NOVA Github code for community. You can go to Nebulas Github to check out the code and find bugs, you have chance to get NAS reward. And also welcome to participate in Nebulas Bounty program.

Nebulas NOVA, To Discover Data Value In the Blockchain World

- Nebulas NOVA, To Discover Data Value In the Blockchain World [(https://www.youtube.com/watch?v=jLIYkG35Ljo)

Interviews with Members of Nebulas Research Institute

- Interview with the leader of Nebulas Research Institute Dr. Xuepeng Fan - Take the Lead to Set Up Nebulas Research Institute
- Interview with Nebulas Mainnet Development Lead Dr. Congming Chen - Let Nebulas Fly Higher and Farther!
- Interview with Nebulas Senior Researcher Dr. Zaiyang Tang - My Heart Belongs to Nebulas, I Hope We Shine Together
- Interview with Nebulas Technical Director Dr. Joel - Exclusive Interview to Nebulas Technical Director Dr. Joel
- Interview with the Senior Researcher of Nebulas Research Institute Dr. Yulong Zeng - My First Job at Nebulas
- Interview with Nebulas Research Institute Intern Dr. Dai - Life Is A Challenge

Interviews with Members of the Nebulas Technical Committee

- Interview with the CEO and Founder of Nebulas Hitters Xu - Seeing Through The Blockchain Bubble: Sitting Down For An Interview With Nebulas.io Founder Hitters Xu

Interviews with Members of the Community

- Interview with Pluto and Xuxue (Nebulas Incentive Program Week 1 Champions) - Nebulas Incentive ProgramâĂŁâĂŤâĂŁInterview with the Champion of Week 1
- Interview with Jason Mansfield (Multi-time winner of Nebulas Incentive Program Season 1) - Interview with a Nebulas DApp Developer: Jason Mansfield
- Interview with Honey Thakuria (Accenture Hackathon Winner using Nebulas) - DApp Development and Architecture DesignâĂŁâĂŤâĂŁInterview with Honey Thakuria

2.4.10 Ecosystem

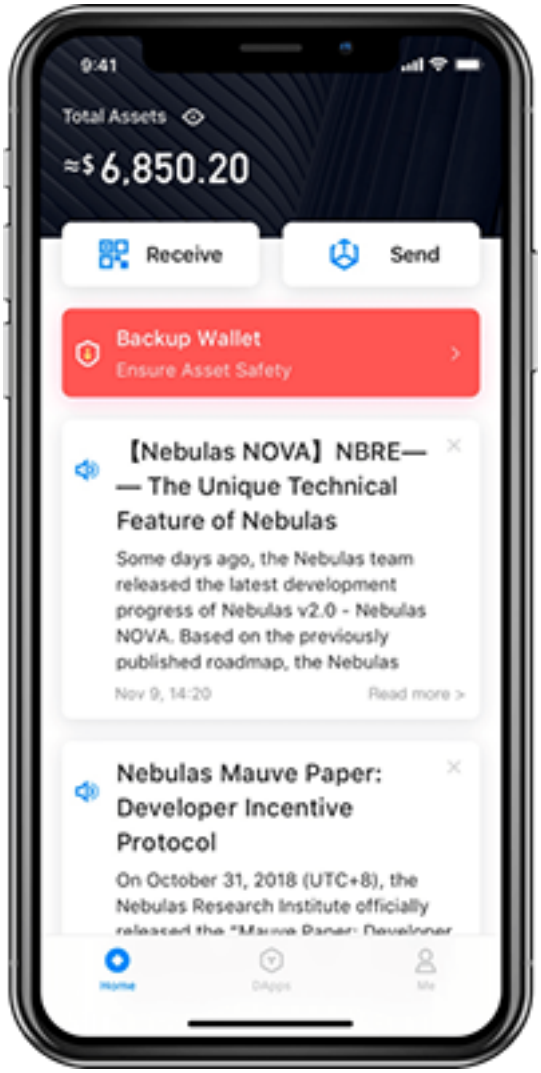
NAS nano

NAS nano 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 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 NAT and ATP.



Nebulas Web Wallet Tutorial

- 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

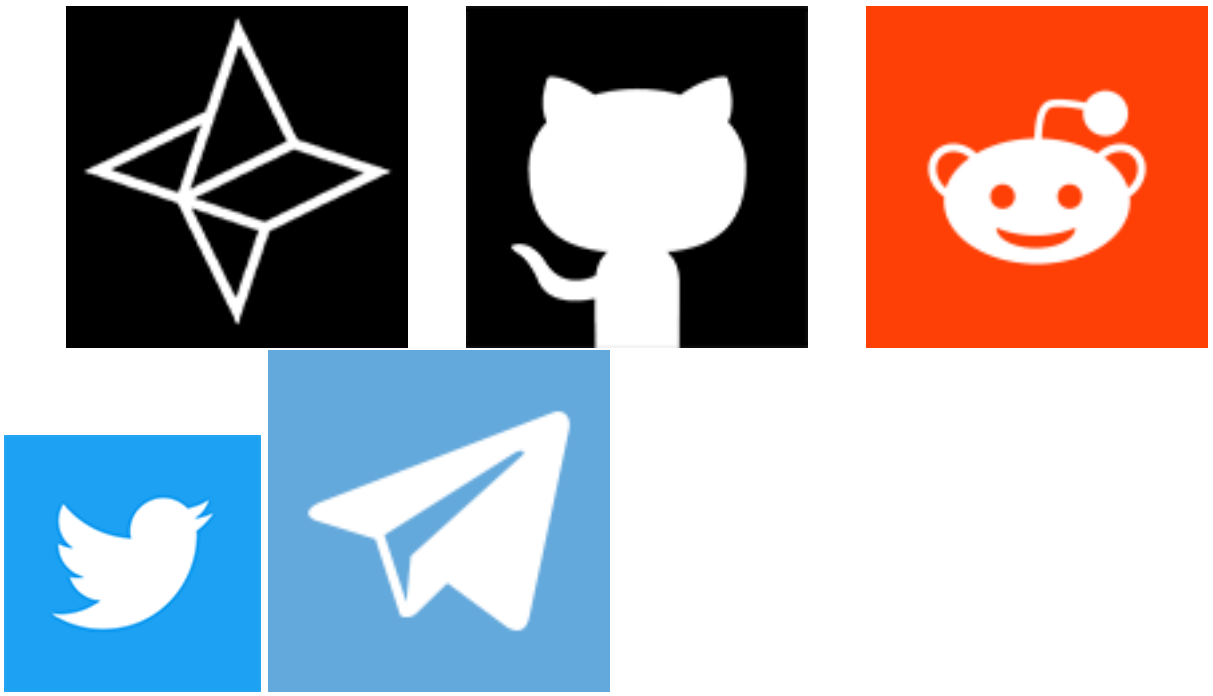
Ecosystem DApps List

You can find recommended DApps and the monthly/weekly champions of the First Season of the Nebulas Incentive Program here: Summary

You are welcome to recommend more DApps!

The Nebulas DApps Store by the community.

2.4.11 Useful Links



2.4.12 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

(a) Nebulas Name Service (NNS)

(b) Lightning Network

(c) Nebulas Token (NAS)

(d) Smart Contracts

i. Language Support

ii. Ethereum Compatibility

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.

2.4.13 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.

Contributors

For a complete list of everyone who contributed to the wiki, click here.

- genindex

- modindex
- search