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

nebulas Documentation

VersiÃşn 1.0

nebulas

07 de enero de 2020

Category:

Nebulas es una nueva generaciÃşn de blockchain pÃžblico que apunta a mejorar el ecosistema cripto de forma sostenida. BasÃąndose en su mecanismo de valuaciÃşn del blockchain, Nebulas introduce conceptos tales como _incentivos orientados a futuro, _sistemas consensuales y la habilidad de evolucionar su propio cÃşdigo sin necesidad de realizar forks.

La comunidad de Nebulas es abierta; cualquier persona puede colaborar en su crecimiento y ayudarnos a crear un mundo descentralizado.

La wiki de Nebulas es una herramienta para la comunidad, que permite publicar documentos en forma colaborativa. Esto incluye las guÃ■as de uso, las guÃ■as de desarrollo, los recursos de aprendizaje, y otros documentos Ãžtiles.

CHAPTER 1

Use Wiki

1.1 Learn

1.2 Develop

1.3 Use Nebulas

1.4 CÃşmo colaborar

1.4.1 WhatâĂŹs Nebulas

The Future of Collaboration

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

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

There are four technical features:

- Quantifiable: measure the value of Blockchain data

- Self-evolving: low-cost instant upgrade capability
- Incentive: positive ecosystem incentives
- On-chain Governance: improved decentraalized autonomous organization (DAO)

Autonomous Metanet

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

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

ÂąAtenciÃşn! Este artÃ■culo estÃą en traducciÃşn. Es posible que encuentres lagunas de contenido o bien secciones y subsecciones en otro idioma.

Nebulas Rank (NR)

Nebulas Rank es un algoritmo de valoraciÃşn (*ranking*), de cÃşdigo abierto, utilizado para ponderar la influencia de las relaciones entre direcciones, contratos inteligentes y aplicaciones distribuidas (*DApps*).

Sirve para que los usuarios puedan utilizar la creciente cantidad de informaciÃşn disponible en los blockchains; ademÃąs, es de utilidad para que los desarrolladores utilicen el *framework* de bÃžsqueda directamente en sus aplicaciones.

En Nebulas, medimos el valor de acuerdo a estos parÃąmetros:

- Liquidez

Las finanzas son una actividad social que permite optimizar los recursos sociales a travÃľs de la liquidez de capitales y a su vez promover el desarrollo econÃşmico.

Los blockchains son, esencialmente, una red de valores en la que los activos financieros pueden moverse libremente. Los volÃžmenes diarios de criptodivisas como Bitcoin y Ethereum (las mÃąs conocidas actualmente), son de mil millones de dÃşlares o mÃąs. A partir de esta informaciÃşn, podemos ver que, a mayor volumen de transacciones y mayor escala, mayor es la liquidez. Como consecuencia de ello, esa mayor liquidez genera una mayor calidad en las transacciones y aumenta el valor de esos activos. Este concepto, el de liquidez, es la primera dimensiÃşn que *Nebulas Rank* toma en cuenta en su ponderaciÃşn.

- PropagaciÃşn

Las plataformas sociales tales como WeChat y Facebook cuentan ya con mÃąs de tres mil millones de usuarios activos, y su base de usuarios crece mensualmente. Este crecimiento es el resultado del reflejo de las redes sociales existentes, y de un fuerte crecimiento viral. En particular, la llamada *transmisiÃşn viral* (compuesta de vectores como velocidad, alcance, vinculaciÃşn y profundidad de la transmisiÃşn de informaciÃşn), es un indicador clave para monitorear la calidad de las redes sociales y el crecimiento de sus bases de usuarios.

En el mundo blockchain podemos observar este mismo patrÃşn. Una propagaciÃşn viral intensa usualmente es una buena indicaciÃşn del alcance y la profundidad del activo digital; esto puede ayudar a promover la calidad y la escala del activo. De este modo, la transmisiÃşn viral del activo (su alcance y profundidad), son en conjunto la segunda dimensiÃşn que *Nebulas Rank* toma en cuenta en su ponderaciÃşn.

- Interoperabilidad

Durante la infancia de internet sÃşlo existÃ■an sitios web rudimentarios, con informaciÃşn privada. En la actualidad es posible reenviar o copiar la informaciÃşn de distintas plataformas en la red, con lo que los silos de informaciÃşn privada son cada vez mÃąs escasos.

Esta tendencia es el proceso de identificaciÃşn de informaciÃşn de mayor dimensiÃşn. Desde nuestro punto de vista, el mundo de los blockchains deberÃ■a seguir un patrÃşn similar, aunque a una velocidad mucho mayor. La informaciÃşn de los activos de los usuarios, de los contratos inteligentes y de las aplicaciones distribuidas (*DApps*) estarÃą cada vez mÃąs enriquecida, y la interacciÃşn de la informaciÃşn de mayor dimensiÃşn serÃą mÃąs frecuente, haciendo que una mejor interoperabilidad sea cada vez mÃąs necesaria. Este indicador (interoperabilidad) es la tercera dimensiÃşn que *Nebulas Rank* toma en cuenta en su ponderaciÃşn.

BasÃąndonos en las tres dimensiones mencionadas, hemos iniciado la construcciÃşn del sistema *Nebulas Rank*, buscando el enriquecimiento de los datos, construyendo un mejor modelo, desempolvando valores dimensionales mÃąs diversificados y estableciendo un sistema de ponderaciÃşn en el mundo blockchain.

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

New Consensus Incentives

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

New Upgrade Capabilities

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

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

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

Decentralized Collaboration with Smart Assets

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

Visit nextDAO to learn more.

Learning Resources

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

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

- Weekly & Monthly Report
- Announcements
- AMA

Interviews

Interviews with Nebulas Team:

- Interview with the Founder of Nebulas Hitters Xu - Seeing Through The Blockchain Bubble
- The Nebulas That IâĂŹm Looking Forward to [Youtube]
- Why Join Nebulas by Ph.D Samuel Chen [Youtube]
- Nebulers' Thoughts on the Future of Blockchain [Youtube]
- One day in Nebulas [Youtube]
- The Inspiration Behind the Nebulas NOVA Design by Mengggo Liu
- Take the Lead to Set Up Nebulas Research Institute by Xuepeng Fan
- Let Nebulas Fly Higher and Farther! by Congming Chen

- My Heart Belongs to Nebulas, I Hope We Shine Together by Zaiyang Tang
- Exclusive Interview to Nebulas Technical Director Dr. Joel
- My First Job at Nebulas by Dr. Yulong Zeng
- Life Is A Challenge by Dr. Dai

Interviews with Members of the Community:

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

Events

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

1.4.2 Using Nebulas

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

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

1. Use an application built on Nebulas

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

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

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

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

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

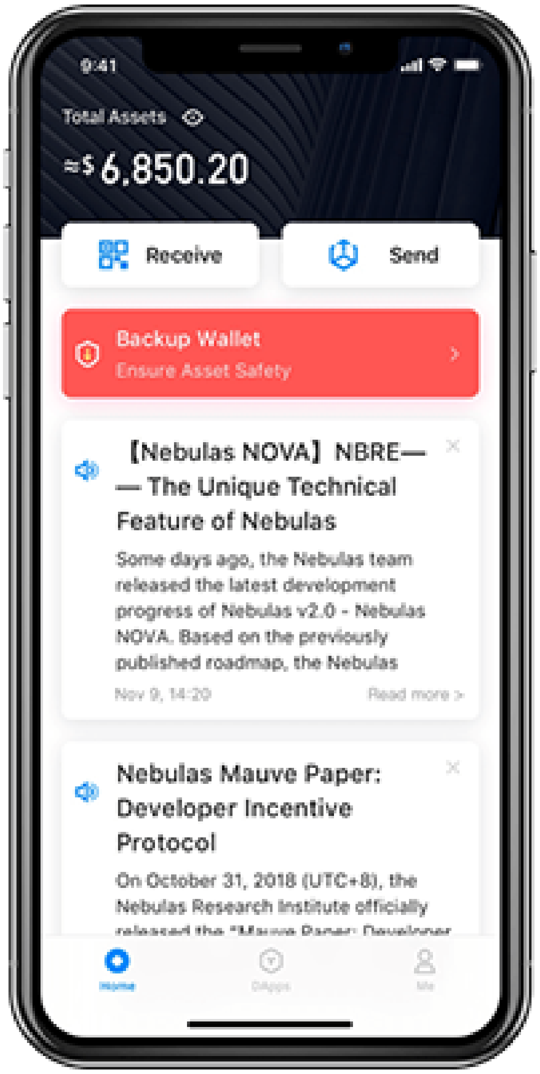
3. What's a wallet and how to hold NAS?

NAS nano pro

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

The NAS nano pro wallet comes with four main features:

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



Nebulas Web Wallet

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

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

Other wallets

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

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

Click here to learn more details about these wallets.

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

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

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

Buy & Sell NAX from Exchanges:

- gate.io
- MXC

You can hold NAX via NAS nano pro and NAS web wallet.

1.4.3 Develop

Getting started

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

For getting started guides and documents, see here:

Tutoriales

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

En esta pÃągina se listan todos los recursos de aprendizaje disponibles en nuestra wiki, en nuestro sitio web oficial, o bien aquellos recursos creados por terceros. Por el momento estos documentos se ofrecen en inglÃľs; prÃşximamente estarÃąn disponibles en espaÃśol.

Recibiremos con agrado todas las contribuciones que surjan de nuestra comunidad. Esta pÃągina se puede editar a travÃľs de nuestro github, por medio de pull requests.

Go-Nebulas

- ÃŽnete a la Testnet
- ÃŽnete a la Mainnet
- Explorador

Tutoriales (Nebulas 101):

01 Compilar e instalar Nebulas

La versiÃşn actual de Nebulas Mainnet es 2.0, que se llama Nebulas Nova.

Nebulas Nova pretende descubrir el valor de los datos de blockchain y tambiÃľn significa el futuro de la colaboraciÃşn.

Consulta nuestra introducciÃşn en YouTube para mÃąs detalles.

Puede descargar el cÃşdigo fuente de Nebulas para compilar la chain privada localmente.

- Para saber mÃąs acerca de Nebulas, sÃ■rvase leer el libro blanco no-tÃľcnico.
- Para aprender mÃąs acerca de su tecnologÃ■a, lÃľase el libro blanco tÃľcnico y el cÃşdigo en github.

  Por el momento, Nebulas sÃşlo puede correr en entornos Mac y Linux. Estamos trabajando para lanzar la versiÃşn de Windows.

Entorno Golang

Actualmente, Nebulas estÃą escrito en Golang y C++.

Mac OSX

Se recomienda Homebrew para instalar Golang en entornos Mac:

```
# instalaciÃşn
brew install go

# configuraciÃşn de las variables de entorno
export GOPATH=/path/to/workspace
```

Importante: GOPATH es una variable de entorno que apunta al directorio de trabajo local de golang, y que es personalizable. Luego de configurar GOPATH, es necesario guardar los proyectos GO en ese directorio.

Linux

```
# descarga
wget https://dl.google.com/go/go1.12.linux-amd64.tar.gz

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

# configuraciÃşn de las variables de entorno
export PATH=$PATH:/usr/local/go/bin
export GOPATH=/path/to/workspace
```

Compilar Nebulas

Descarga

Es necesario clonar el cÃşdigo fuente mediante estos comandos de consola:

```
# ingresar al espacio de trabajo
cd /path/to/workspace

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

# ingresar al repositorio
cd go-nebulas

# la rama master es siempre la mÃąs estable
git checkout master
```

ConstrucciÃşn de Neb

- Configurar el entorno de ejecuciÃşn

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

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

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

Una vez que este proceso se completa, habrÃą un nuevo ejecutable, llamado `neb`, en el directorio raÃ■z.

```
shangshu at shangshudeMacBook-Pro in ~/workspace/blockchain/src/github.com/nebulasio/go-nebulas on master
❯ make build                                                                                          16:44
cd cmd/neb; go build -ldflags "-X main.version=1.0.1 -X main.commit=fbcdd8927b3ed5db9ccf889388bf8efb500d3357
 -X main.branch=master -X main.compileAt=`date +%s`" -o ../../neb-fbcdd8927b3ed5db9ccf889388bf8efb500d3357
cd cmd/crashreporter; go build -ldflags "-X main.version=1.0.1 -X main.commit=fbcdd8927b3ed5db9ccf889388bf8e
fb500d3357 -X main.branch=master -X main.compileAt=`date +%s`" -o ../../neb-crashreporter
rm -f neb
ln -s neb-fbcdd8927b3ed5db9ccf889388bf8efb500d3357 neb
```

make build

Iniciar neb

Bloque inicial (Genesis Block)

Antes de crear un nuevo *blockchain* Nebulas, es necesario definir la configuraciÃşn del bloque inicial, o gÃľnesis.

ConfiguraciÃşn del bloque inicial

```
# Esquema definido en core/pb/genesis.proto.

meta {
# Chain identity
chain_id: 100
}

consensus {
dpos {
# DinastÃ■a inicial, incluyendo los mineros iniciales
dynasty: [
[ miner address ],
...
]
}
}

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

Existe un archivo genesis.conf de ejemplo en `conf/default/genesis.conf`.

Nodo

Antes de poder lanzar un nodo neb, es necesario definir su configuraciÃşn.

ConfiguraciÃşn del nodo Neb

```
# El esquema estÃą definido en neblet/pb/config.proto:Config.

# ConfiguraciÃşn de la red
network {
# Para el primer nodo en un blockchain Nebulas, no es necesario el␣
↪parÃąmetro `seed`.
# En otros casos, todo nodo requiere nodos seed que los␣
↪âĂIJpresentenâĂİ en el blockchain de Nebulas.
# seed: ["/ip4/127.0.0.1/tcp/8680/ipfs/
↪QmP7HDFcYmJL12Ez4ZNVCKjKedfE7f48f1LAkUc3Whz4jP"]

# Servicio de alojamiento de la red p2p. Soporta mÃžltiples IP y␣
↪puertos.
listen: ["0.0.0.0:8680"]

# La clave privada se utiliza para generar el ID del nodo. Si no se␣
↪utiliza una clave privada, el nodo generarÃą un ID nuevo.
# private_key: "conf/network/id_ed25519"
}

# ConfiguraciÃşn del blockchain
chain {
# ID de la red del chain (cadena)
chain_id: 100

# UbicaciÃşn del almacenamiento de la base de datos
datadir: "data.db"

# UbicaciÃşn de los archivos keystore de las cuentas
keydir: "keydir"

# ConfiguraciÃşn del bloque inicial
genesis: "conf/default/genesis.conf"

# Algoritmo de firma (signature)
signature_ciphers: ["ECC_SECP256K1"]

# DirecciÃşn del minero
miner: "n1SAQy3ix1pZj8MPzNeVqpAmu1nCVqb5w8c"

# DirecciÃşn coinbase; todas las recompensas por minerÃ■a se␣
↪enviarÃąn a esta direcciÃşn:
coinbase: "n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE"
```

```
# La palabra clave para acceder al archivo keystore del minero
passphrase: "passphrase"
}

# ConfiguraciÃşn de la API
rpc {
# Puerto APIÂăGRPC
rpc_listen: ["127.0.0.1:8684"]

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

# mÃşdulo http
http_module: ["api", "admin"]
}

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

# UbicaciÃşn del registro
log_file: "logs"

# HabilitaciÃşn del registro de errores; `false` para␣
↪deshabilitarlo, `true` para habilitarlo
enable_crash_report: false
}

# ConfiguraciÃşn de las mÃl'tricas
stats {
# HabilitaciÃşn de las mÃl'tricas; `false` para deshabilitarlo,␣
↪`true` para habilitarlo
enable_metrics: false

# ConfiguraciÃşn de InfluxDB
influxdb: {
host: "http://localhost:8086"
db: "nebulas"
user: "admin"
password: "admin"
}
}
```

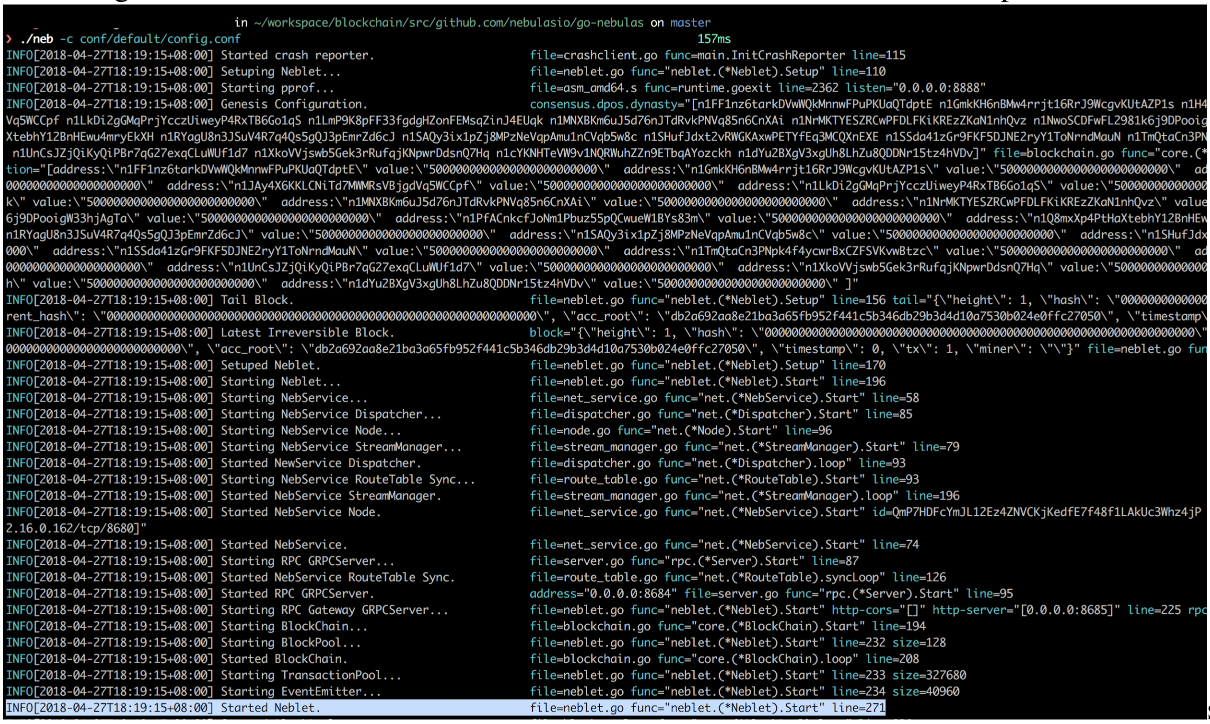
Existen distintos ejemplos que se pueden consultar en la carpeta `$GOPATH/src/github.com/nebulasio/go-nebulas/conf/`

Correr nodos

El blockchain Nebulas que se ha configurado hasta este momento es privado y difiere de las redes oficiales Testnet y Mainnet de Nebulas.

Para crear un nuevo nodo Nebulas, es necesario ejecutar estos comandos:

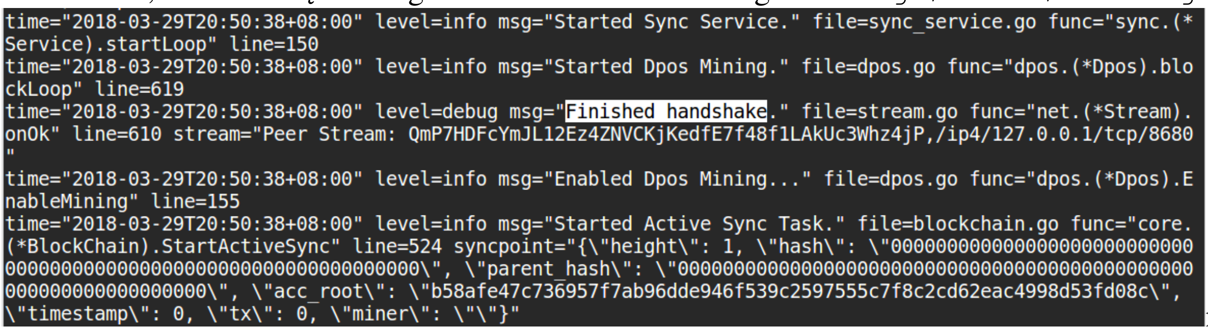
```
cd $GOPATH/src/github.com/nebulasio/go-nebulas
./neb -c conf/default/config.conf
```

Luego de iniciado, se deberÃ■a ver esta salida por terminal: 

seed node start

Por defecto, el nodo utiliza el archivo `conf/default/config.conf` y no minarÃą nuevos bloques. Para iniciar un nodo de minado Nebulas, ejecÃžtese lo siguiente:

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

Luego de iniciado el nodo, y si se conecta de forma satisfactoria con el nodo *seed*, se verÃą la siguiente salida en el registro `logs/miner/neb.log`: 

node start

IMPORTANTE

Es posible iniciar una cantidad arbitraria de nodos de forma local. Es necesario asegurarse de que los puertos especificados en los archivos de configuraciÃşn no entren en conflicto entre sÃ■.

CapÃ■tulo siguiente: parte 2 del tutorial:

Enviar transacciones en Nebulas

02 Sending Transactions on Nebulas

Youtube Tutorial

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

Nebulas provides three methods to send transactionsïijŽ

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

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

Prepare Accounts

In Nebulas, each address represents an unique account.

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

The Sender

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

The Receiver

Create a new wallet to receive the tokens.

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

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

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

Start the Nodes

Start Seed Node

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

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

Start Miner Node

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

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

How long a new block will be minted?

In Nebulas, 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": "67066180000000000000",
        "nonce": "0",
        "type": 87
    }
}
```

> **Note** Type is used to check if this account is a smart contract account. `88` represents smart contract account and `87` means a non-contract account.

As we see, the receiver has been rewarded 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
    }
}
```

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":"1000000","gasLimit":
↪"2000000"}, "passphrase":"passphrase"}'

{"result":{"data":"CiAbjMP5dyVsTWILfXL1MbwZ8Q6xOgX/
↪JKinks1dpToSdxIaGVcH+WT/
↪SVMkY18ix7SG4F1+Z8evXJoA35caGhlXbip8PupTNxwV4SRM87r798jXWADXpWngIhAAAAAAAAAAAAA
↪"}}
```

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+Z8evXJoA35caGhlXbip8PupTNxwV4SRM87r798jXWADXpWngIhAAAAAAAAAAAAA
↪"}'

{"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":"1000000","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":"1000000","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":"1000000","gas_limit":"2000000","contract_
↪address":"","status":1,"gas_used":"20000"}}âŔŐ
```

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

- **0: Failed.** It means the transaction has been submitted on chain but its execution failed.
- **1: Successful.** It means the transaction has been submitted on chain and its execution successeed.
- **2: Pending.** It means the transaction hasn‘t been packed into a block.

Double Check

Let‘s double check the receiver‘s balance.

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

{"result":{"balance":"3000000000000000000","nonce":"0","type":87}}
```

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

Next step: Tutorial 3

Write and run a smart contract with JavaScript

03 Write and run a smart contract

YouTube Tutorial

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

Preparation

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

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

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

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

Write a smart contract

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

Write a brief specification of a smart contract:

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

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

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

Smart contract example:

```
'use strict';

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

```
    this.balance = new BigNumber(0);
    this.expiryHeight = new BigNumber(0);
  }
};

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- save(): The user can save money to the bank safe by calling the save() method;
- takeout(): Users can withdraw money from bank safe by calling takeout() method;
- balanceOf(): The user can check the balance with the bank vault by calling the 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: `"1000000"`;
- `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":"1000000","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 asynchronous process, which need to be packaged by the miner. Just as the previous transfer transaction, the transfer does not arrive in real time, it depends on the speed of the miner packing. Therefore we need to wait for a while (about 1 minute), then you can verify whether the contract is deployed successfully by querying the contract address or calling this smart contract.

Verify the deployment of the contract is successful

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

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

{"result":{"hash":
↪"aaebb86d15ca30b86834efb600f82cbcaf2d7aaffbe4f2c8e70de53cbed17889
↪","chainId":100,"from":
↪"n1H4MYms9F55ehcvygwWE71J8tJC4CRr2so","to":
↪"n1H4MYms9F55ehcvygwWE71J8tJC4CRr2so","value":"0","nonce":
↪"1","timestamp":"1524711841","type":"deploy","data":
↪"eyJTb3VyY2VUeXBlIjoianMiLCJTb3VyY2UiOiJcInVzZSBzdHJpY3RcIjt2YXIgRGVwb3Npc
↪ZmFsc2U6dHJ1ZX07fX07bW9kdWxlLmV4cG9ydHM9QmFua1ZhdWx0Q29udHJhY3Q7IiwiQXJncy
↪","gas_price":"1000000","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 `"1000000"`;

- `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":"1000000","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
↪":"1000000","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":"1000000","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":"1000000","gasLimit":"2000000","contract":{"function":
↪"balanceOf","args":""}}'

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

Next step: Tutorial 4

Smart Contract Storage

04 Smart Contract Storage

YouTube Tutorial

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

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

LocalContractStorage

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

Basics

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

Storing `LocalContractStorage` DataïijŽ

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

Reading `LocalContractStorage` DataïijŽ

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

Deleting `LocalContractStorage` DataïijŽ

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

Examples:

```
'use strict';

var SampleContract = function () {
};

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

module.exports = SampleContract;
```

Advanced

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

Binding Properties

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

Binding Interface

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

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

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

```
'use strict';

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

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

module.exports = SampleContract;
```

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

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

Binding Map Properties

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

```
'use strict';

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

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

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

```
    });
};

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

module.exports = SampleContract;
```

Iterate Map

In contract, map does't support iterator. if you need to iterate the map, you can use the following way: define two map, arrayMap, dataMap, arrayMap with a strictly increasing counter as key, dataMap with data key as key.

```
"use strict";

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

SampleContract.prototype = {
    init: function () {
        this.size = 0;
    },

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

    get: function (key) {
```

```
        return this.dataMap.get(key);
    },

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

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

};

module.exports = SampleContract;
```

Next step: Tutorial 5

Interacting with Nebulas by RPC API

05 Interacting with Nebulas by RPC API

YouTube Tutorial

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

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

API

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

Modules

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

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

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

Configuration

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

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

Example

HTTP

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

GetNebState

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

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

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

UnlockAccount

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

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

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

RPC

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

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

GetNebState

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

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

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

// API interface to access node status information
api := rpcpb.NewAPIServiceClient(conn)
resp, err := ac.GetNebState(context.Background(), & rpcpb.
↪GetNebStateRequest {})
if err != nil {
    log.Println("GetNebState", "failed", err)
} else {
    log.Println("GetNebState tail", resp)
}
```

LockAccount

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

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

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

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

API List

For more interfaces, please refer to the official documentation:

- API Module
- Admin Module.

Next

Nice job! Let's join official Testnet or Mainnet to enjoy Nebulas now!

Join the Testnet Join the Mainnet

DApp Development

- Tutoriales [PDF]
- ExtensiÃşn para el navegador Chrome (Similar as MetaMask)
- Contratos inteligentes

SDK

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

Standard Token

- NRC20
- NRC721

Community Tools

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

Documentos oficiales de Nebulas

- El Papel Naranja de Nebulas âĂŤ Nebulas Governance. Puedes acceder al repositorio aquÃ■.
- El Papel Malva de Nebulas âĂŤ Protocolo de Incentivo para Desarrolladores: [InglÃl's], [Chino], Es, Puedes acceder al repositorio aquÃ■, Acerca de Mauve Paper y el Protocolo de Incentivo para Desarrolladores
- El Papel Amarillo de Nebulas âĂŤ el Nebulas Rank: [InglÃl's], [Chino], [Coreano], [PortuguÃl's], Es, Puedes acceder al repositorio aquÃ■, InterpretaciÃşn oficial de âĂIJNebulas Rank: Yellow PaperâĂİ
- El Papel Blanco TÃl'cnico de Nebulas: [InglÃl's], [Chino], Es.
- El Papel Blanco No TÃl'cnico de Nebulas.

Como siempre, las traducciones y los informes de errores son siempre bienvenidos. Aprende mÃąs sobre cÃşmo contribuir.

IntroducciÃşn a Nebulas

- IntroducciÃşn.
- Primeros pasos.
- AdministraciÃşn de cuentas.
- Transacciones.

Aplicaciones descentralizadas

- CÃşmo crear una DApp: Primera parte, Segunda parte, Tercera parte.
- Detalles del algoritmo de valoraciÃşn de contratos inteligentes: Primera parte, Segunda parte.
- Otros recursos Nueva caracterÃ■stica en los contratos inteligentes de la red Nebulas. CÃşmo obtener tokens testnet, guÃ■a paso a paso. Â£Por quÃľ elegir Nebulas en una Hackathon?. CÃşmo construir una DApp usando Nuxt.js y Nebulas, escrito por Honey Thakuria. JavaScript y Smart Contracts: una introducciÃşn a Nebulas para desarrolladores de contratos inteligentes de Ethereum, escrito por Michal Zalecki.

CÃşmo usar la cartera de Nebulas

Web Wallet: https://github.com/nebulasio/web-wallet

- CÃşmo crear una cartera Nebulas.
- CÃşmo enviar NAS desde tu cartera Nebulas.
- CÃşmo firmar una transacciÃşn sin conexiÃşn.
- Visualizar informaciÃşn de la cartera.
- Verificar el estado de una transacciÃşn.
- Implementar un contrato inteligente.
- Realizar una llamada a contrato inteligente.
- CÃşmo usar NebPay en una Dapp.

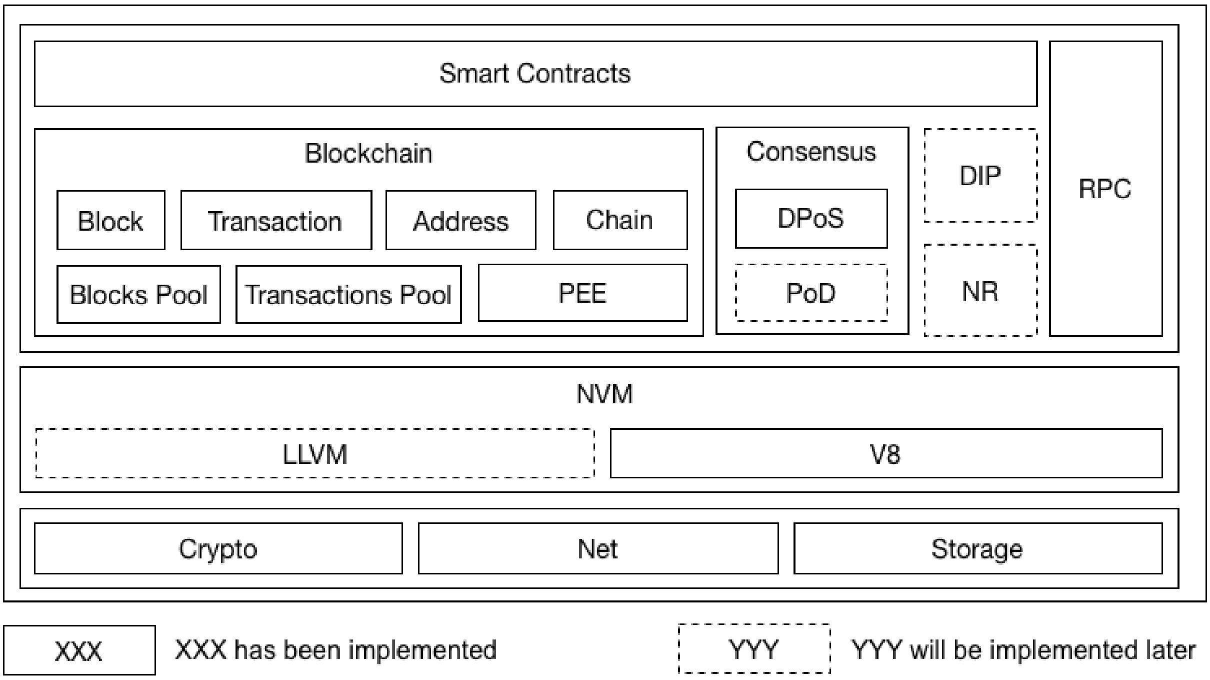
Preguntas y respuestas en Reddit

- 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

Nos alegra inmensamente que estÃl's considerando escribir tutoriales o documentos sobre Nebulas. Si ya has escrito algo, por favor avÃ■sanos por medio de un issue, y aÃśadiremos tu nombre y tu enlace a esta pÃągina tan pronto como nos sea posible.

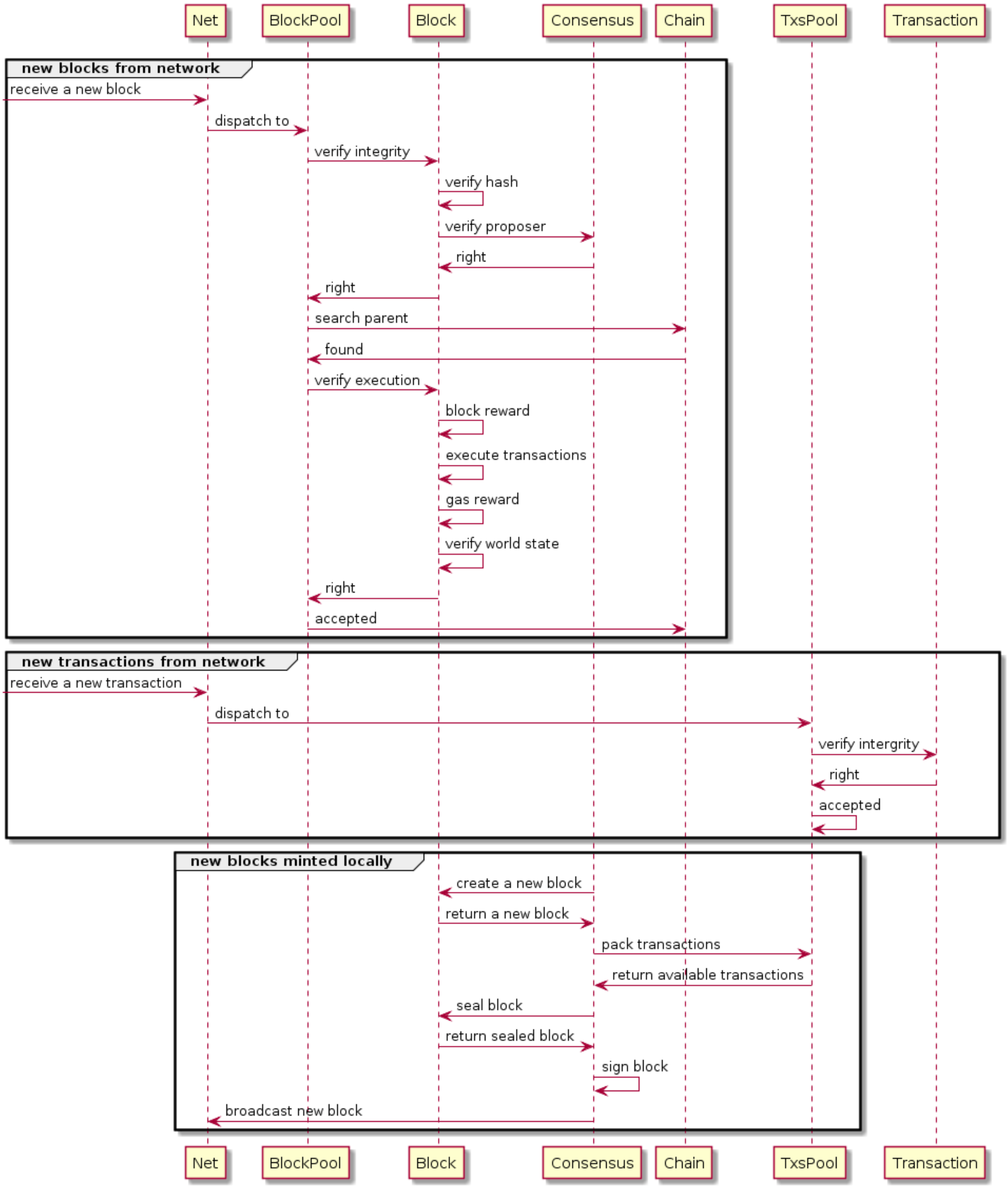
Design Overview



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

Modelo

Nebulas utiliza un modelo de cuentas en vez del modelo UTXO (*Unspent Transaction Output*). La ejecuciÃşn de transacciones requiere el consumo de *gas*.

Estructura de datos

Estructura de los bloques

```
+---------------+----------------+--------------+
|  blockHeader  |  transacciones | dependencias |
+---------------+----------------+--------------+
```

- blockHeader: informaciÃşn de encabezado
- transactions: matriz de transacciones
- dependency: relaciÃşn de dependencias entre transacciones

Estructura del encabezado de los bloques

```
+-----------+--------+--------------+------------+-------------+----
↪---+--------+
|  chainid  |  hash  |  parentHash  |  coinbase  |  timestamp  | ␣
↪alg  |  sign  |
+-----------+--------+--------------+------------+-------------+----
↪---+--------+
+-------------+-----------+--------------+-----------------+
|  stateRoot  |  txsRoot  |  eventsRoot  |  consensusRoot  |
+-------------+-----------+--------------+-----------------+
```

- chainid: identificador de la cadena a la que pertenece el bloque
- hash: hash del bloque
- parentHash: hash del bloque padre
- coinbase: cuenta que recibirÃą la *recompensa de acuÃśaciÃşn* (minting reward)
- timestamp: el nÃžmero de nanosegundos transcurridos desde el 1 de enero de 1970, UTC
- alg: algoritmo de firma (*signature*) utilizado
- sign: firma del hash del bloque
- stateRoot: hash raÃ■z del estado de cuenta
- txsRoot: hash raÃ■z del estado de la transacciÃşn
- eventsRoot: hash raÃ■z del estado de los eventos
- consensusRoot: estado del consenso, incluyendo proponente y dinastÃ■a de validadores

Estructura de las transacciones

```
+-----------+--------+--------+------+---------+---------+----------
↪---+
|  chainid  |  hash  |  from  |  to  |  value  |  nonce  | ␣
↪timestamp  |
+-----------+--------+--------+------+---------+---------+----------
↪---+
+--------+-----------+-----------+
|  data  |  gasPrice  |  gasLimit  |
+--------+-----------+-----------+
```

- chainid: identificador de la cadena a la que pertenece el bloque
- hash: hash de la transacciÃşn
- from: direcciÃşn de la cartera del emisor
- to: direcciÃşn de la cartera del receptor
- value: monto transferido
- nonce: *nonce* de la transacciÃşn
- timestamp: el nÃžmero de nanosegundos transcurridos desde el 1 de enero de 1970, UTC
- alg: algoritmo de firma (*signature*) utilizado
- sign: firma del hash del bloque
- data: datos de la transacciÃşn, incluyendo el tipo de transacciÃşn (binaria, implementaciÃşn de contrato inteligente, llamada a contrato inteligente) y su *payload*.
- gasPrice: el precio unitario del gas consumido por la transacciÃşn
- gasLimit: la cantidad mÃąxima permitida de gas a consumir en la transacciÃşn

ActualizaciÃşn del blockchain

En nuestra opiniÃşn, un blockchain deberÃ■a ocuparse Ãžnicamente de procesar nuevos bloques para lograr un crecimiento seguro y eficiente. Aun mÃąs: un blockchain sÃşlo puede obtener nuevos bloques a travÃľs de dos canales, que detallamos a continuaciÃşn.

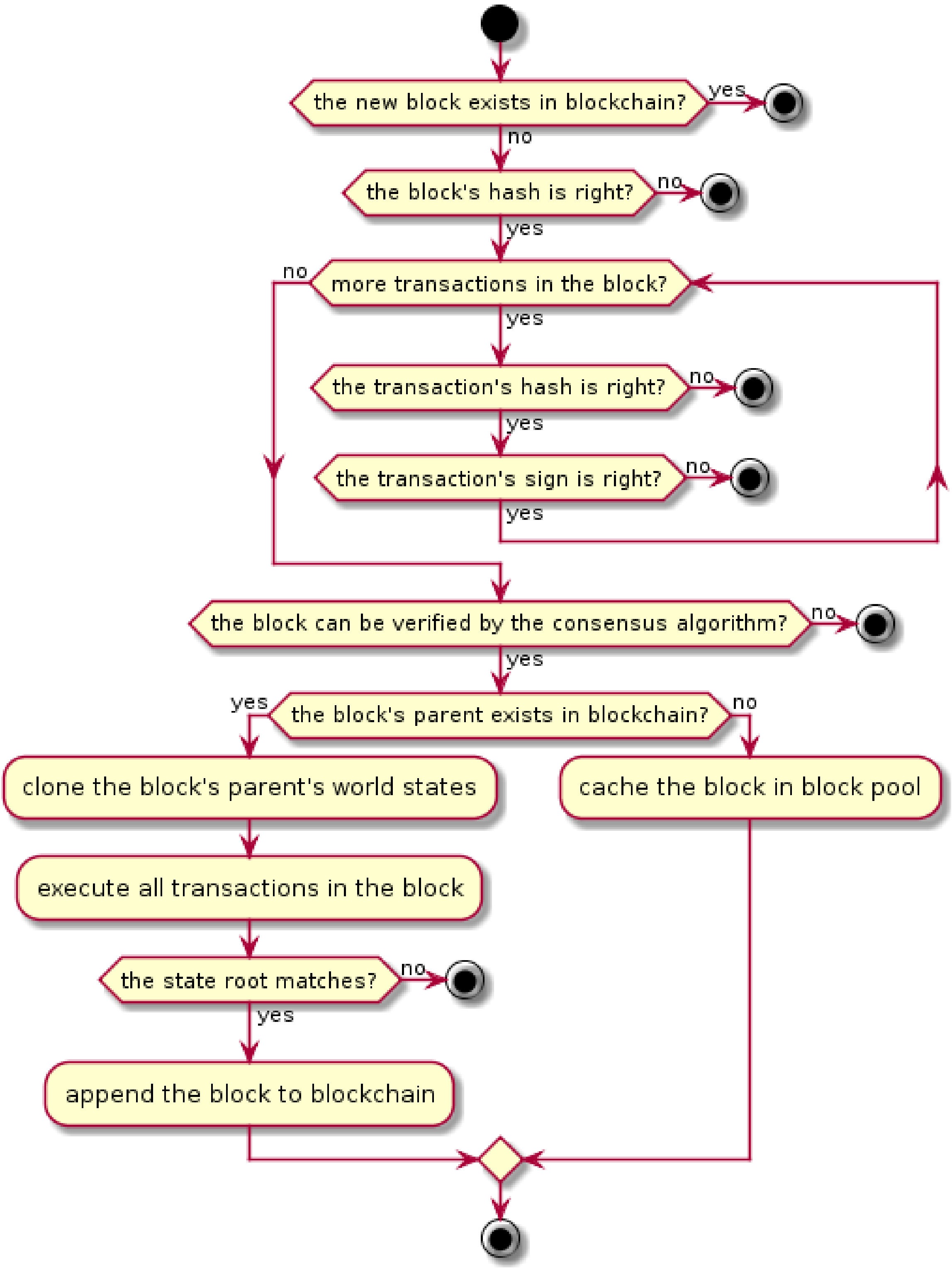
Desde la red

A causa de la inestabilidad en la latencia de la red, es imposible asegurar directamente que los nuevos bloques recibidos se puedan vincular a la cadena actual. AsÃ■, se hace necesaria la implementaciÃşn de *pozos de bloque* (*blocks pool*) para almacenar los nuevos bloques.

Desde un minero local

En principio, necesitamos que el **pozo de transacciones** almacene las transacciones recibidas desde la red. A partir de ello, es necesario esperar a que se genere un nuevo bloque a travÃľs de un componente local de consenso, tal como *DPoS*.

Sin importar la procedencia del nuevo bloque, Nebulas utiliza los mismos pasos para procesarlo; veamos esos pasos en detalle.



Estado global

Cada bloque contiene el estado global, que se compone del estado de cuatro componentes que detallamos abajo. Todos ellos son mantenidos mediante Merkle Trees.

Estado de cuentas

Todas las cuentas del bloque actual se almacenan en Estados de Cuentas.

Las cuentas se dividen en dos tipos: normales, y contratos inteligentes.

Normales

Incluye:

- **Direcciones de carteras**
- **Balance**
- ***nonce***: el *nonce* de cada cuenta, cuyo valor se incrementa de 1 en 1.

Contratos Inteligentes

Incluye:

- **Direcciones de contratos inteligentes**
- **Balance**
- **Lugar de nacimiento**: el *hash* de la transacciÃşn en la que tuvo lugar la implementaciÃşn del contrato.
- **Variables**: Contiene todos los valores de las variables del contrato.

Estados de transacciones

Todas las transacciones enviadas al blockchain se almacenan en Estados de Transacciones.

Estados de eventos

Mientras se ejecutan las transacciones, es posible que se disparen mÃžltiples eventos. Todos los eventos disparados por transacciones en el blockchain se almacenan en Estados de Eventos.

Estado del Consenso

El contexto del algoritmo de consenso se almacena en Estado de Consenso.

Con respecto a DPoS, el estado del consenso incluye:

- **timestamp**: el timestamp actual.
- **proposer**: propositor actual.
- **dynasty**: dinastÃ■a de validadores actual.

SerializaciÃşn

Elegimos utilizar Protocol Buffers para la serializaciÃşn general, en vista de los siguientes beneficios que otorga:

- Solidez comprobada a gran escala.
- Eficiencia. Omite *key literals* y en su lugar utiliza codificaciÃşn *varint*.
- Brinda soporte a multi-tipos.
- Brinda soporte a clientes multilenguaje.
- API fÃącil de utilizar.
- Su *schema* es ideal para las comunicaciones.
- Su *schema* es ideal para hacer versionado y extensiones; por ejemplo, para aÃśadir nuevos campos de mensaje o para dejar otros en desuso.

En especial y para mejorar la legibilidad del cÃşdigo de los contratos inteligentes, utilizamos JSON en vez de *protobuf* para su codificaciÃşn.

SincronizaciÃşn

En ocasiones recibiremos un bloque cuya altura es mucho mayor que la del Ãžltimo bloque de la cadena. Cuando esto ocurre, necesitamos sincronizar los bloques desde los nodos de pares para estar a tono con ellos.

Nebulas provee dos mÃl'todos para la sincronizaciÃşn de bloques con nodos de pares: *Chunks Downloader* y *Block Downloader*. Si la diferencia es mayor a 32 bloques, debemos elegir Chunk Downloader para descargar gran cantidad de bloques en trozos. Si no, elegiremos *Block Downloader* para descargar los bloques uno a la vez.

Chunks Downloader

El *chunk* es una colecciÃşn de 32 bloques sucesivos. *Chunks Downloader* nos permite descargar, cada vez, un mÃąximo de 10 chunks que suceden a nuestro Ãžltimo bloque. Este mecanismo nos ayuda a minimizar el nÃžmero de paquetes de red y a lograr una mayor seguridad.

Procedimiento

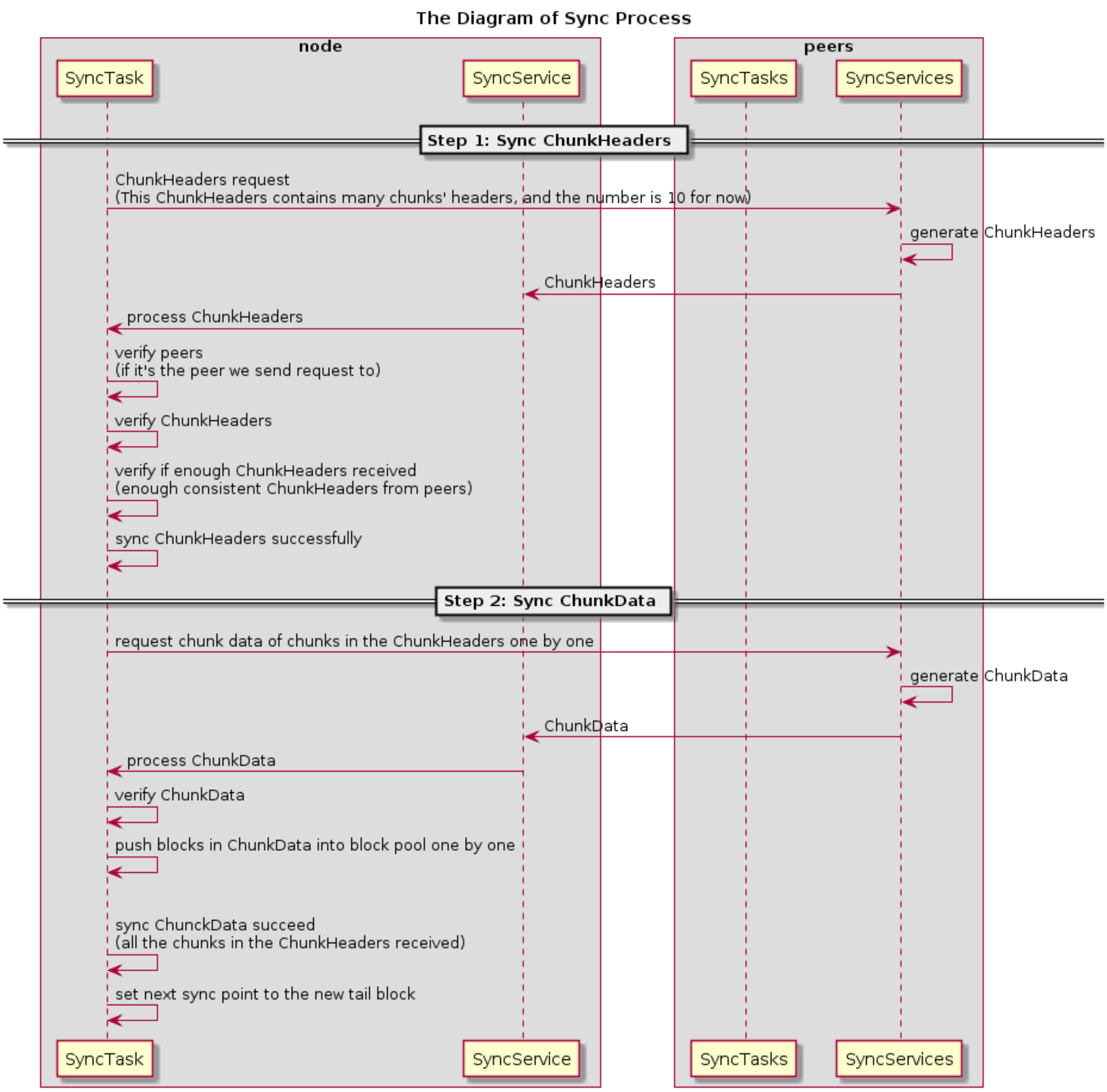
```
1. **A** envÃ■a su Ãžltimo bloque a una cantidad _n_ de pares␣
↪remotos.
2. Los pares remotos localizan el _chunk_ **C** que contiene el␣
↪Ãžltimo bloque de **A**.
3. Los pares remotos le envÃ■an a **A** el encabezado de **C**, los␣
↪encabezados de los siguientes 9 _chunks_ despuÃl's de **C**, y␣
↪los hashes de todos esos encabezados.
4. Si **A** recibe mÃąs de la mitad de los mismos encabezados **H**,
↪ **A** intentarÃą sincronizar los _chunks_ representados por␣
↪**H**.
5. Si **A** recibiÃş todos los chunks representados por **H** y los␣
↪enlazÃş correctamente a su cadena, se repite todo el proceso␣
↪desde el punto 1.
```

En los pasos 1 a 3, utilizamos la decisiÃşn mayoritaria para confirmar los *chunks* en la cadena canÃşnica. Luego, descargamos los bloques en los *chunks* del paso 4.

Nota

- `ChunkHeader` contiene una matriz de 32 bloques hash mÃąs el hash de la matriz.
- `ChunkHeaders` contiene una matriz de 10 `ChunkHeaders` mÃąs el hash de la matriz.

AquÃ■ podemos observar un diagrama de este procedimiento de sincronizaciÃşn:



Block Downloader

Cuando la diferencia de longitud entre nuestra cadena local y la cadena canÃşnica es menor a 32, utilizaremos *Block Downloader* para descargar los bloques faltantes, uno a la vez.

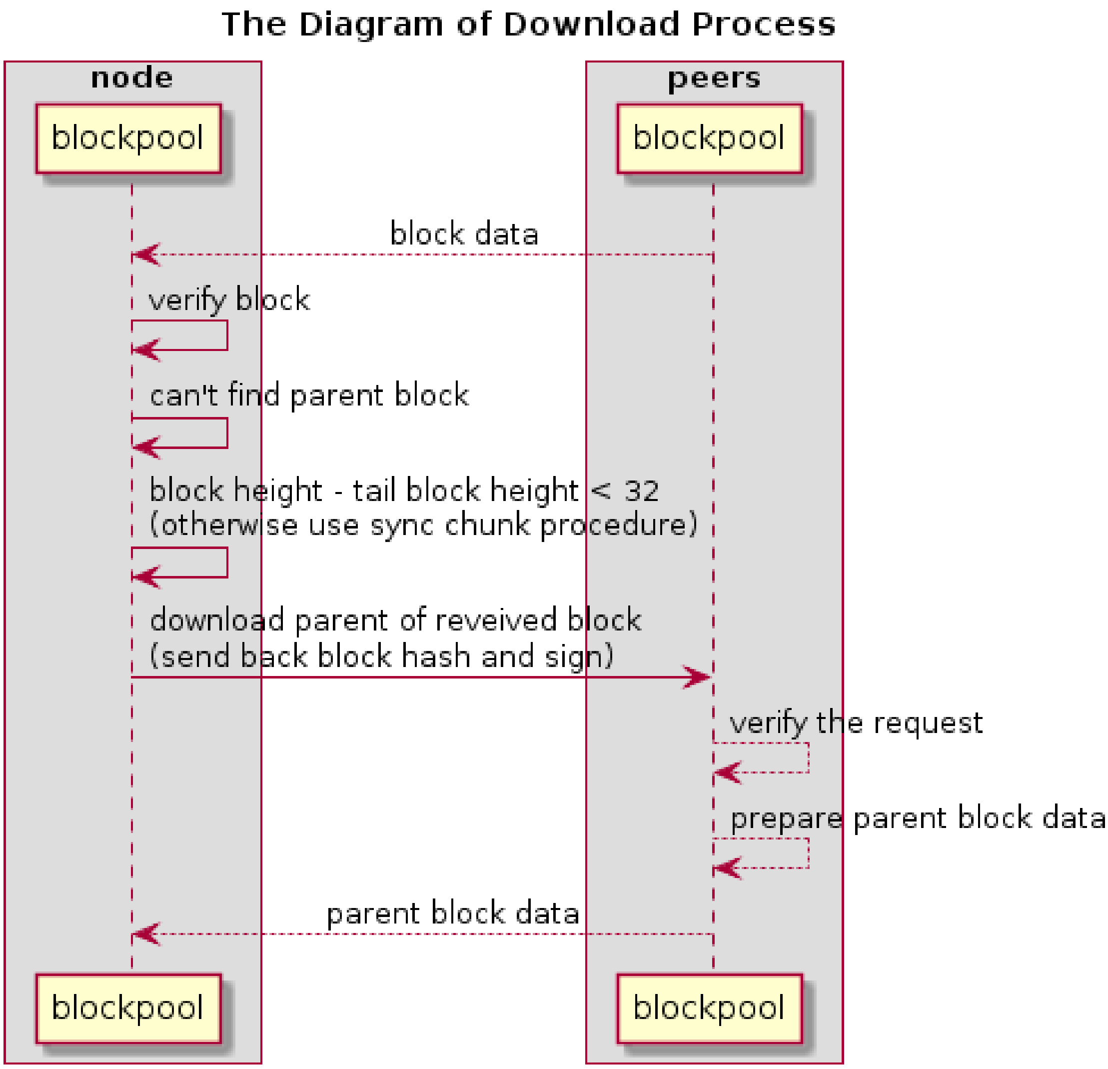
Procedimiento

```
1. **C** transfiere el nuevo bloque **B** a **A** y **A** encuentra␣
↪que la altura de **B** es mayor que la del Ãžltimo bloque actual.
2. **A** reenvÃ■a el hash del bloque **B** a **C** para descargar el␣
↪bloque padre de **B**.
3. Si **A** recibe el bloque padre de **B**, **A** intentarÃa␣
↪enlazar el bloque **B** con el Ãžltimo bloque de la cadena de␣
↪**A**.
```

```
4. Si falla, **A** volverÃą al paso 2 y continuarÃą descargando el␣
↪bloque padre de **B**. Si no falla, el proceso termina.
```

El procedimiento se repite hasta que **A** queda totalmente sincronizado con la cadena canÃşnica.

AquÃ■ podemos ver un diagrama de este procedimiento:



Merkle Patricia Tree

IntroducciÃşn: *Radix Tree*

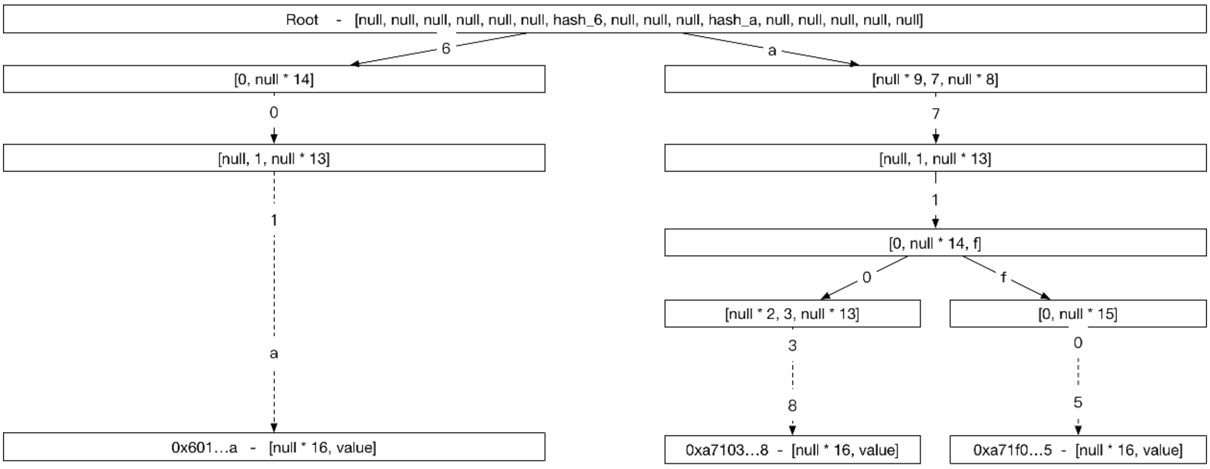
Referencia

Un Radix Tree que utiliza una direcciÃşn a modo de clave tendrÃą las siguientes caracterÃ■sticas:

- Las direcciones se representan mediante caracteres hexadecimales.

- Cada nodo del Ãąrbol es una matriz de 16 elementos (0123...def).
- Nodos *leaf*: su valor puede ser cualquier dato binario.
- non-leaf node: su valor es un hash calculado en base a los datos de sus nodos _*children*.

Para una direcciÃşn de 160-bits, la altura mÃąxima del Ãąrbol es 40.



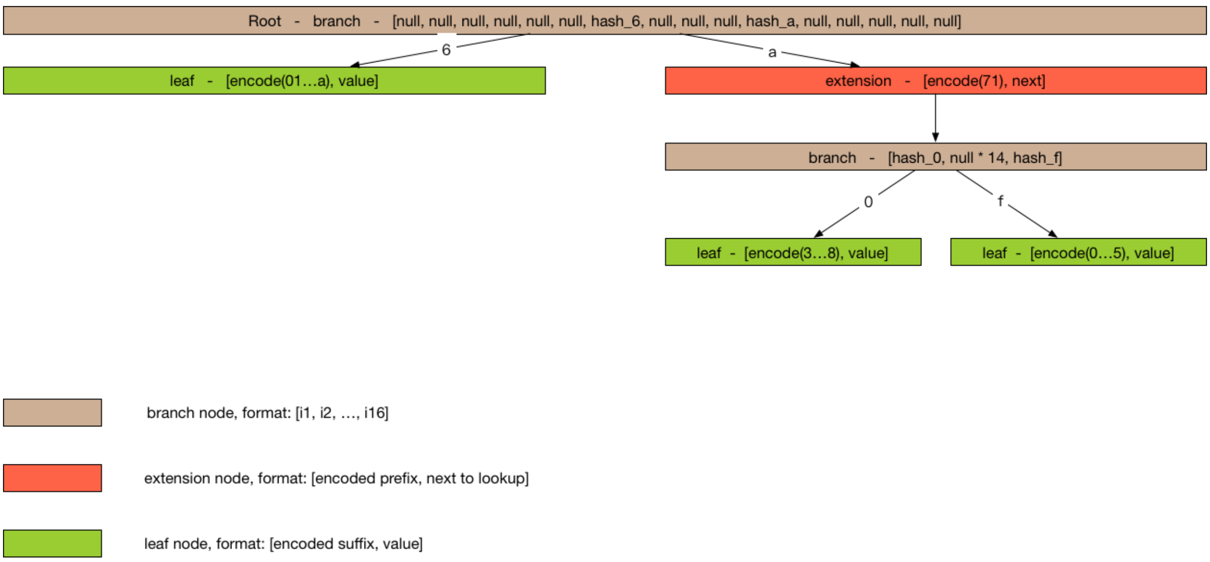
Problemas: mucho espacio para una entrada sencilla; 40 pasos para cada *lookup*

Avanzado: *Merkle Patricia Tree*

Referencia, http://gavwood.com/Paper.pdf

Para reducir el espacio de almacenamiento de Radix Tree, los nodos en *Merkle Patricia Tree* se dividen en tres tipos:

- nodo de extensiÃşn: comprime los nodos utilizando prefijos comunes.
- nodo hoja: comprime los nodos utilizando prefijos Ãžnicos
- nodo rama: Ã■dem a los nodos en el Radix Tree.



CÃşmo almacenar *Merkle Patricia Tree*

Almacenamiento de los pares clave/valor

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

key = hash(value)

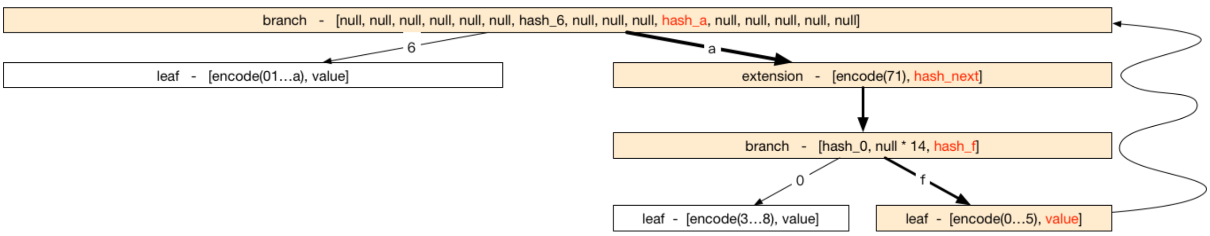
CÃşmo actualizar *Merkle Patricia Tree*

Consulta

DFS de arriba hacia abajo

Actualizar, Eliminar o AÃśadir

1. Realizar consultas en el nodo, desde arriba hacia abajo.
2. Actualizar el hash y el path desde abajo hacia arriba.



Performance Cada operaciÃşn cuesta O (log (n)).

CÃşmo realizar verificaciones usando *Merkle Patricia Tree*

Teoremas

1. Los *Merkle Trees* iguales deben tener el mismo *root hash*.
2. Los *Merkle Trees* distintos deben tener distintos *root hash*.

Usando los teoremas se puede verificar el resultado de la ejecuciÃşn de las transacciones.

VerificaciÃşn rÃąpida

Un cliente *lightweight*, sin necesidad de sincronizar enormes bloques de transacciones, puede determinar inmediatamente el estado y el balance exacto de cualquier cuenta simplemente consultando la red para buscar una ruta desde la raÃ■z hasta la cuenta nodo.

Consenso

Cada algoritmo de consenso puede ser descripto como una combinaciÃşn de *State Machine* y *Fork Choice Rules*.

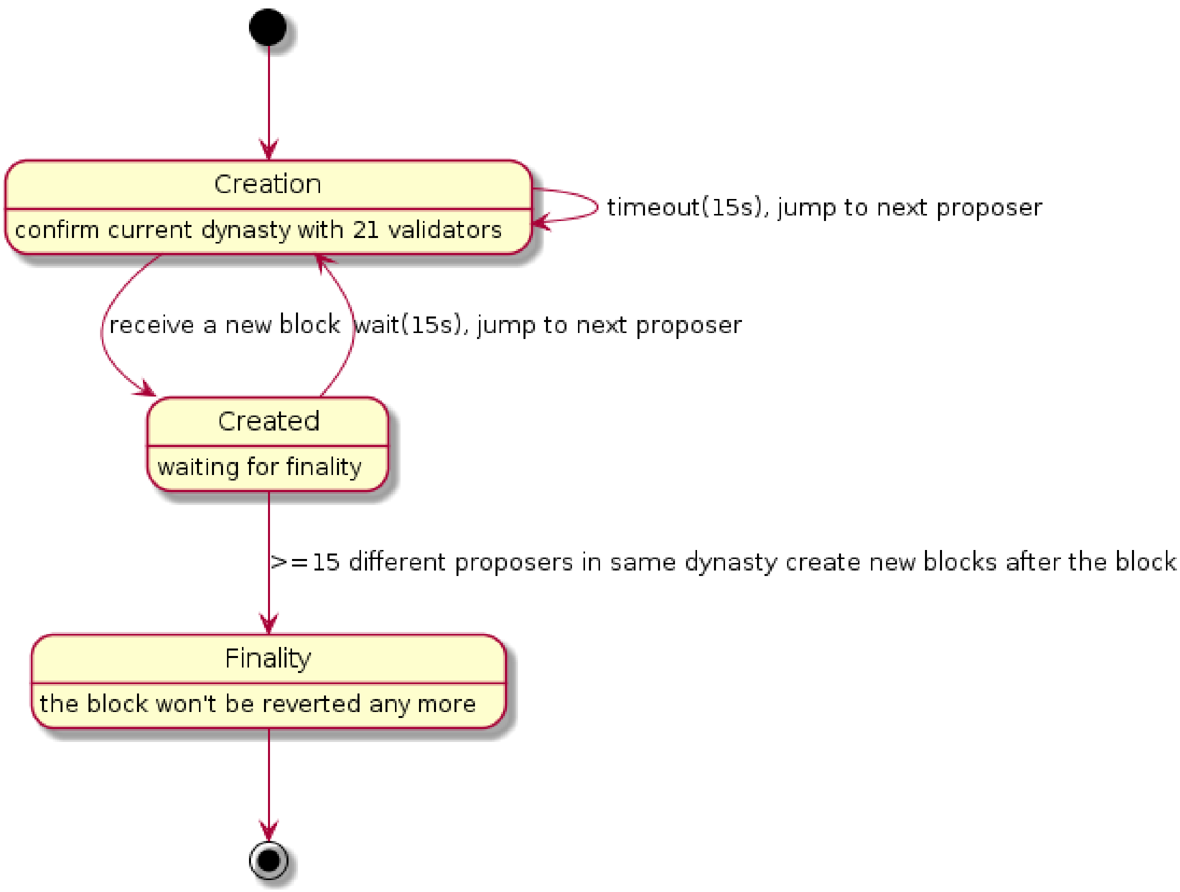
DPoS (Delegate Proof-of-Stake)

DPoS puede ser descrita como una *state machine* o mÃąquina de estado finito.

Advertencia

El consenso en Nebulas serÃą *PoD*; la elecciÃşn del algoritmo DPoS es sÃşlo una soluciÃşn temporaria. Luego de la verificaciÃşn formal del algoritmo *PoD*, haremos la transiciÃşn de la *mainnet* a ese algoritmo definitivo. Todos los *testigos* (contables y mineros) de DPoS son, por ahora, cuantas mantenidas oficialmente por Nebulas; nos encargaremos de realizar una transiciÃşn suave de DPoS a PoD y crearemos un nuevo fondo para administrar todas las recompensas para los contables y para incentivar el crecimiento de nuestro ecosistema.

State Machine



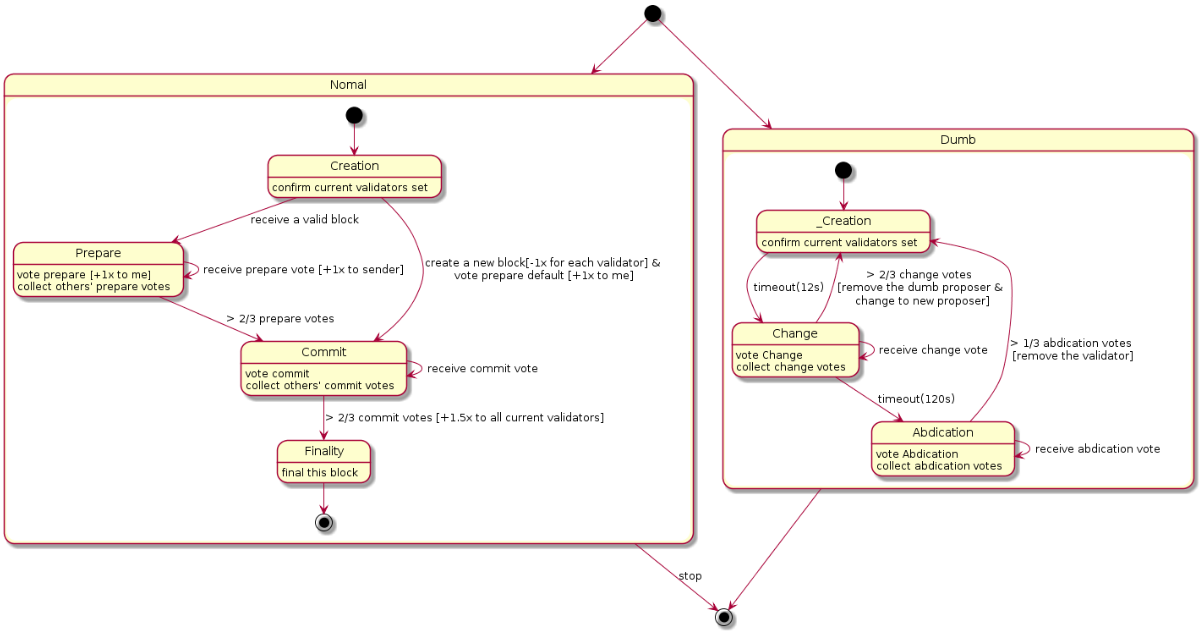
Fork Choice Rules

1. Se debe elegir la ruta mÃąs larga como la ruta canÃşnica.
2. Si las rutas A y B tienen la misma longitud, se debe elegir aquella con el menor *hash*.

PoD (Proof-of-Devotion), o Prueba de DevociÃşn

Actualmente este algoritmo se encuentra en desarrollo; puedes encontrar una versiÃşn aquÃ■.

State Machine



Fork Choice Rules

1. Se debe elegir la cadena (chain) con la mayor cantidad de votos.
2. Si los *chains* A y B tienen la misma cantidad de votos, se debe elegir aquel con el menor *hash*.

Diagrama de proceso de transacciones

Cuando se envÃ■a una transacciÃşn es necesario chequear el chain correspondiente. Las transacciones enviadas de forma externa o aquellas que se han empaquetado en el bloque tienen algunas diferencias a la hora de realizar su validaciÃşn.

Procedimiento para realizar nuevas transacciones

VÃa RPC u otro nodo de *broadcasting*

- 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.

Funcionalidad de los eventos

La funcionalidad `Event` se utiliza para que los desarrolladores y los usuarios puedan suscribirse a eventos de su interÃl's. Estos eventos se generan durante la ejecuciÃşn del *blockchain*, y almacenan los pasos de la ejecuciÃşn y sus resultados en el *chain*. Para consultar y verificar los resultados de la ejecuciÃşn de transacciones y contratos inteligentes, se almacenan esos dos tipos de eventos en un *trie* en el *chain*.

Estructura **evento**:

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

Al generarse un evento, se debe procesar con eventEmitter. Los usuarios pueden suscribirse al emisor del evento.

Si no hay suscripciÃşn al evento, este se descartarÃą. Para todo nueva suscripciÃşn, si el canal no se bloquea a tiempo, se descartarÃą el evento debido al mecanismo *non-blocking*.

Lista de eventos:

- TopicNewTailBlock
- TopicRevertBlock
- TopicLibBlock
- TopicPendingTransaction
- TopicTransactionExecutionResult

- EventNameSpaceContract

Referencia

TopicNewTailBlock

Este evento se dispara cuando el Ãžltimo bloque de la cadena se actualiza.

- TÃşpico:`chain.newTailBlock`
- Datos:
  - `height`: altura del bloque
  - `hash`: hash del bloque
  - `parent_hash`: hash del bloque padre
  - `acc_root`: hash raÃ■z del estado de la cuenta
  - `timestamp`: timestamp del bloque
  - `tx`: hash raÃ■z del estado de la transacciÃşn
  - `miner`: minero del bloque

TopicRevertBlock

Este evento ocurre cuando se revierte un bloque en la cadena.

- TÃşpico:`chain.revertBlock`
- Datos: El contenido del tÃşpico es similar a los datos de TopicNewTailBlock.

TopicLibBlock

Este evento se dispara cuando el Ãžltimo bloque irreversible sufre un cambio.

- TÃşpico:`chain.latestIrreversibleBlock`
- Dato: El contenido del tÃşpico es similar a los datos de TopicNewTailBlock.

TopicPendingTransaction

Este evento se dispara cuando se introduce una transacciÃşn en el pozo de transacciones (*transaction pool*).

- TÃşpico:`chain.pendingTransaction`
- Datos:
  - `chainID`: id de cadena

- `hash`: hash de la transacciÃşn
- `from`: cadena que indica la direcciÃşn origen de la transacciÃşn
- `to`: cadena que indica la direcciÃşn destino de la transacciÃşn
- `nonce`: nonce de la transacciÃşn
- `value`: valor de la transacciÃşn
- `timestamp`: timestamp de la transacciÃşn
- `gasprice`: precio del gas para la transacciÃşn
- `gaslimit`: lÃ■mite de gas para la transacciÃşn
- `type`: tipo de transacciÃşn

TopicTransactionExecutionResult

Este evento ocurre cuando se ejecuta el final de una transacciÃşn. This event will be recorded on the chain, and users can query with RPC interface GetEventsByHash.

Este evento registra los resultados de la ejecuciÃşn de la transacciÃşn, y es sumamente importante.

- TÃşpico:`chain.transactionResult`
- Datos:
  - `hash`: hash de la transacciÃşn
  - `status`: estado de la transacciÃşn; 0: fallÃş, 1: sin errores, 2: pendiente
  - `gasUsed`: gas usado para la transacciÃşn
  - `error`: error de ejecuciÃşn de la transacciÃşn. Si no hubo errores, este campo estarÃą vacÃ■o.

EventNameSpaceContract

Este evento ocurre cuando se ejecuta un contrato inteligente. Si la ejecuciÃşn ocurre exitosamente, los eventos quedarÃąn registrados en el chain y pueden ser suscritos; por el contrario, si ocurre un error en la ejecuciÃşn, el evento no se registrarÃą.

Este evento tambiÃľn se registrarÃą en el chain, donde los usuarios lo pueden consultar mediante la interfaz RPCGetEventsByHash.

- TÃşpico: `chain.contract.[topic]` El tÃşpico del evento del contrato tiene el prefijo `chain.contract.`, El contenido es definido por quien escribe el contrato.
- Datos: El contenido es definido por quien escribe el contrato.

SuscripciÃşn

Es posible suscribirse a todos los eventos, y el *cloud chain* expone una interfaz RPC de suscripciÃşn llamada Subscribe. Es importante notar que la suscripciÃşn a los eventos es un mecanismo ajeno al blockchain. Los nuevos eventos se descartarÃąn si la interfaz RPC no se maneja a tiempo.

Consultas

SÃşlo es posible consultar los eventos registrados en el chain, mediante el uso de la interfaz RPC GetEventsByHash.

Al momento, es posible consultar los siguientes eventos:

- TopicTransactionExecutionResult
- EventNameSpaceContract

Gas para transacciones (*Transaction Gas*)

En Nebulas, tanto una transacciÃşn normal en la que se transfiere balance, o un contrato inteligente, utilizan *gas*, cuyo costo se deduce del balance de la direcciÃşn que origina la transacciÃşn o que hace uso del contrato.

Una transacciÃşn contiene dos parÃąmetros: `gasPrice` y `gasLimit`.

- `gasPrice`: el valor de la unidad de *gas*.
- `gasLimit`: el lÃ■mite mÃąximo de uso de *gas*.

El valor del consumo de *gas* en una transacciÃşn estÃą dado por la fÃşrmula:

`gasPrice` * `gasUsed`

que serÃą la recompensa otorgada a los mineros. El valor `gasUsed` debe ser igual o menor a `gasLimit`. El posible valor de `gasUsed` se p uede estimar a travÃľs de la interfaz RPC *estimategas*; el mismo se almacenarÃą en el evento de resultado de la ejecuciÃşn de la transacciÃşn.

Razones para este diseÃśo

Tal como Bitcoin y Ethereum, el *gas* en Nebulas se utiliza para pagar el costo de las transacciones. Tiene dos propÃşsitos fundamentales:

- Como recompensa para mineros, para incentivarlos a empaquetar transacciones. El empaquetamiento de esas transacciones conlleva un costo computacional âĂŤen especial la ejecuciÃşn de contratos inteligentesâĂŤ por lo que los usuarios deben pagar por ese servicio.

- Como deterrente costoso para los atacantes. El ataque *DDoS* (Distributed Denial of Service) es bastante econÃşmico en internet, y cualquier hacker inescrupuloso podrÃ■a enviar volÃžmenes muy elevados de transacciones al servidor objeto de sus ataques. En las redes Bitcoin y Ethereum, cada transacciÃşn tiene un costo econÃşmico, lo que incrementa significativamente el costo de un ataque de este tipo, haciÃľndolos imprÃącticos.

ConstituciÃşn del *gas*

Cuando un usuario emite una transacciÃşn, el *gas* se utilizarÃą del siguiente modo:

- `EmisiÃşn de la transacciÃşn`
- `Almacenamiento de los datos de la transacciÃşn`
- `InserciÃşn del _payload_`
- `EjecuciÃşn del _payload_ (contrato inteligente)`

En todos estos puntos se consumen recursos de la red Nebulas, y por ende es necesario pagar por ellos.

EnvÃ■o de transacciones

Al enviar una transacciÃşn esta se aÃśadirÃą al final del bloque al que fue asignada. Los mineros se encargarÃąn de realizar el proceso, por el cual deben recibir una recompensa; para ello, se usarÃą una cantidad determinada de *gas*, que se calcula del siguiente modo:

```
// TransactionGas: gas consumido en una transacciÃşn normal
TransactionGas = 20000
```

Si la verificaciÃşn de la transacciÃşn falla, el gas se devolverÃą al emisor.

Almacenamiento de datos de transacciÃşn

Al implementar un contrato âĂŤo una llamada al mismoâĂŤ, los datos de su ejecuciÃşn se almacenan en el blockchain, lo que tambiÃľn tiene un costo. La fÃşrmula del costo asociado de *gas* es la siguiente:

```
TransactionDataGas = 1

len(data) * TransactionDataGas
```8

El campo `TransactionDataGas` es un nÃžmero fijo definido en el␣
↪cÃşdigo.

Para cada tipo de _payload_ en una transacciÃşn existen diferentes␣
↪consumos de gas al momento de su ejecuciÃşn. Los tipos de␣
↪transacciÃşn soportados por Nebulas en este momento son los␣
↪siguientes:
```
```

```
* `binary`: permite a los usuarios adjuntar datos binarios durante␣
↪la ejecuciÃşn del contrato inteligente. Estos datos no se␣
↪procesan durante la ejecuciÃşn. En este caso `TransactionDataGas`␣
↪es 0.
* `deploy` y `call`: Los tipos `deploy` y `call` les permiten a los␣
↪usuarios implementar contratos inteligentes en Nebulas. Nebulas␣
↪debe correr su mÃaquina virtual `nvm` para ejecutar el contrato,␣
↪por lo que se debe pagar por el derecho de uso de ese recurso. En␣
↪este caso `TransactionDataGas` es 60.

### EjecuciÃşn de los _payloads_ (Smart contract deploy & call)

Como el tipo de transacciÃşn `binary` no realiza ningÃžn tipo de␣
↪procesamiento al momento de ejecutar la transacciÃşn, la␣
↪ejecuciÃşn no requiere _gas_. Por el contrario, cuando un␣
↪contrato inteligente implementa una llamada al momento de la␣
↪emisiÃşn de la transacciÃşn, la ejecuciÃşn consumirÃą recursos de␣
↪la computadora de los mineros, y probablemente se haga uso␣
↪tambiÃl'n del almacenamiento de datos en el blockchain.

####ÂăEjecuciÃşn de instrucciones

Cada ejecuciÃşn de un contrato utiliza recursos de los mineros, que␣
↪se deben pagar. El contador de instrucciones V8 realiza el␣
↪cÃąlculo del nÃžmero de instrucciones a procesar. El lÃ■mite de␣
↪nÃžmero de instrucciones a ejecutar impide el uso excesivo de␣
↪recursos y la generaciÃşn de _deadlocks_.

#### Almacenamiento de contratos

La caracterÃ■stica `LocalContractStorage`, que permite almacenar␣
↪objetos de los contratos inteligentes, utiliza una unidad de _gas_
↪ cada 32 bytes que se almacenan. La lectura de esos objetos no␣
↪consume gas.

El lÃ■mite para la ejecuciÃşn de contratos estÃą dada por:

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

Matriz de conteo de *gas*

La matriz de conteo de *gas* para la ejecuciÃşn de los contratos inteligentes estÃą dada por:

| ExpresiÃşn | CÃşdigo de ejemplo | Op. binario | Op. de carga | Op. de almacenamiento | Op. de devoluciÃşn | Op. llamada interna | Conteo de gas | | | | — | :— | —: | —: | —: | —: | —: | —: | | 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 | | |

Consejos

En Nebulas, el *pozo* de transacciones de cada nodo tiene un valor mÃ■nimo y mÃąximo para `gasPrice` y un valor mÃąximo para `gasLimit`. Si el valor de `gasPrice` en la transacciÃşn no estÃą en el rango del parÃąmetro `gasPrice` definido en el pozo de transacciones, o el valor para `gasLimit` es mayor que el definido en ese pozo, la transacciÃşn serÃą rechazada.

ConfiguraciÃşn de los parÃąmetros gasPrice y gasLimit:

`gasPrice`

- mÃ■nimo: el valor mÃ■nimo de `gasPrice` se puede establecer en el archivo de configuraciÃşn. Si no estÃą configurado, el valor por defecto es `1000000` (10^6).
- mÃąximo: el valor mÃąximo para `gasPrice` es de `1000000000000` (10^12).

No se debe superar los valores mÃąximos y mÃ■nimos en la configuraciÃşn del pozo de transacciones.

`gasLimit`

- mÃ■nimo: debe ser mayor a cero.
- mÃąximo: su valor mÃąximo es de `50000000000` (50 * 10^9).

No se debe superar los valores mÃąximos y mÃ■nimos en la configuraciÃşn del pozo de transacciones.

Registros

IntroducciÃşn

Nebulas provee dos tipos de registro: registro de consola & registro *verbose*.

Registro de consola

El **registro de consola** (Console Log o CLog) se usa para para ayudar a los desarrolladores a comprender quÃľ tipo de trabajo **Neb** se estÃą ejecutando, incluyendo inicios y paradas de componentes, recepciÃşn de nuevos bloques en el *blockchain*, sincronizaciÃşn, etc.

CLog escribirÃą todos sus registros en stdout y en archivos de registro al mismo tiempo. Puedes ver ese registro directamente por la salida estÃąndar.

Instrucciones de la consola de registros Nebulas

```
// el nivel de registro puede ser `Info`, `Warning` y `Error`
logging.CLog().Info("")
```

Especificaciones de inicio de servicios

Al iniciarse un servicio Nebulas se deberÃ■a crear un registro de consola, que se debe emitir inmediatamente antes de iniciar el servicio. Para ello, basta con ejecutar el siguiente comando:

```
logging.CLog().Info("Iniciando servicio xxxxxx...")
```

Especificaciones de parada de servicios

Al detenerse un servicio Nebulas se deberÃ■a crear un registro de consola, que se debe emitir inmediatamente antes de detener el servicio. Para ello, basta con ejecutar el siguiente comando:

```
logging.CLog().Info("Deteniendo servicio xxx...")
```

Registro *verbose*

Este tipo de registro (Verbose Log, o VLog) es Ãžtil para depurar cualquier trabajo **Neb**, o para entender su funcionamiento; esto incluye la verificaciÃşn de bloques, el descubrimiento de nuevos nodos, emisiÃşn de tokens, etc.

- VLog escribirÃą sus registros Ãžnicamente en archivos de registro. Puedes consultarlos en tu carpeta de registro si lo deseas.

Es posible filtrar el nivel de *verbosidad* de este registro; los niveles son **Debug < Info < Warn < Error < Fatal.**

Hooking

Por defecto, los *hooks* Function y FileRotate se aÃśaden a la salida de CLog y VLog.

FunctionNameHooker

Este *hook* escribirÃą en los registros el nombre de la funciÃşn llamante y su correspondiente nÃžmero de lÃ■nea en el cÃşdigo fuente. El resultado se verÃ■a asÃ■:

```
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

Este *hook* dividirÃą los registros en pequeÃśas secciones encabezadas por *fecha*. Por defecto, todos los registros se reemplazarÃąn una vez por hora. La carpeta de registros deberÃ■a verse de forma similar a esta:

```
> neb-2018010415.log neb-2018010416.log neb.log -&gt; /path/to/neb-
↪2018010415.log
```

Si tienes alguna sugerencia con respecto a los registros, siÃľntete libre de enviar un issue a nuestro repositorio wiki. ÂąGracias!

Sistema de direcciones Nebulas

El sistema de direcciones Nebulas fue diseÃśado cuidadosamente. Como puedes ver mÃąs abajo, las direcciones de cuentas y contratos inteligentes comienzan con una letra "n".

Direcciones de cuentas

De una forma similar a Bitcoin y Ethereum, Nebulas adopta el algoritmo de curvas elÃ■pticas como su sistema bÃąsico de encriptaciÃşn para las cuentas Nebulas. La direcciÃşn se deriva de una clave pÃžblica que a su vez deriva de una clave privada, encriptada con la contraseÃśa del usuario. AdemÃąs, nuestro diseÃśo de *checksum* apunta a prevenir que un usuario accidentalmente envÃ■e *NAS* a la cuenta incorrecta debido a un error de tipeo.

La fÃşrmula especÃ■fica de cÃąlculo de direcciones es la siguiente:

```
1.  contenido = ripemd160(sha3_256(clave pÃžblica))
    longitud: 20 bytes
                         +---------+---------+------------------+
2.  checksum = sha3_256( |  0x19   +  0x57   |      contenido   |␣
↪)[:4]
                         +---------+---------+------------------+
    longitud: 4 bytes

                        +---------+----------+-----------------+------
↪------+
```

```
3.  direcciÃşn = base58( |    0x19   |  0x57    |      contenido |
↪checksum  | ïijL'
                        +---------+---------+-----------------+------
↪------+
    longitud: 35 caracteres
```

0x57 es un *type code* de un byte que indica que la direcciÃşn es de una cuenta; **0x19** es un *relleno* de un byte.

En esta etapa, Nebulas adopta el estÃąndar de codificaciÃşn base58, similar al de Bitcoin.

Una direcciÃşn vÃąlida se ve asÃ■: *n1TV3sU6jyzR4rJ1D7jCAmtVGSntJagXZHC*.

Direcciones de contratos inteligentes

El cÃąlculo de estas direcciones varÃ■a ligeramente del cÃąlculo anterior. Para mÃąs informaciÃşn, vÃľase contratos inteligentes y rpc.sendTransaction.

La fÃşrmula especÃ■fica de cÃąlculo de contratos inteligentes es la siguiente:

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

                         +---------+---------+-----------------+-----
↪-------+
3.  direcciÃşn = base58 ( |  0x19   |  0x58    |      contenido    |
↪checksum  | ïijL'
                         +---------+---------+-----------------+-----
↪-------+
    longitud: 35 caracteres
```

0x58 es un *type code* de un byte que indica que la direcciÃşn es de un contrato inteligente; **0x19** es un *relleno* de un byte.

Una direcciÃşn de contrato inteligente vÃąlida es, por ejemplo: *n1sLnoc7j57YfzAVP8tJ3yK5a2i56QrTDdK*

DIP (TBD)

CÃşmo unirse a la mainnet de Nebulas

IntroducciÃşn

The Nebulas Mainnet 2.0 (Nebulas Nova) ha sido recientemente lanzada. Este tutorial le enseÃśarÃą cÃşmo unirse y cÃşmo trabajar con la Mainnet de Nebulas.

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

Compilacion

El archivo ejecutable de la mainnet de Nebulas y las bibliotecas dependientes se deben compilar primero. Varios mÃşdulos importantes se enumeran a continuaciÃşn:

- **NBRE:** el Nebulas Blockchain Runtime Environment es la plataforma para ejecutar la RepresentaciÃşn del Protocolo de Nebulas, como el DIP, el NR, etc.
- **NEB:** el proceso principal de la mainnet de Nebulas. NEB y NBRE se ejecutan en procesos independientes y se comunican a travÃľs de IPC.

Las instrucciones sobre cÃşmo compilar los mÃşdulos se pueden encontrar en los tutoriales.

ConfiguraciÃşn

PodrÃąs encontrar los archivos de configuraciÃşn de la mainnet en `mainnet/conf`.

Esa carpeta contiene los siguientes archivos:

genesis.conf

Permite configurar los parÃąmetros del bloque inicial (*genesis block*), incluyendo:

- **meta.chain_id**: Identidad de la cadena.
- **consensus.dpos.dynasty**: DinastÃ■a inicial de validadores.
- **token_distribution**: AsignaciÃşn inicial de tokens.

IMPORTANTE: No se deben realizar cambios sobre el archivo genesis.conf.

config.conf

Permite configurar los parÃąmetros del runtime.

Para mÃąs informaciÃşn sobre este archivo, vÃľase `template.conf`.

Nota: la informaciÃşn del nodo semilla oficial debe verse tal como se muestra aquÃ■ debajo:

```
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"]
```

Resumen de la API

Endpoint principal:

- GetNebState: Devuelve informaciÃşn sobre el cliente.
- GetAccountState: Devuelve el balance y el *nonce* de la cuenta.
- Call: Ejecuta un contrato inteligente de forma local, sin enviar datos al chain.
- SendRawTransaction: Permite enviar una transacciÃşn firmada.
- GetTransactionReceipt: Obtiene informaciÃşn del recibo de una transacciÃşn mediante su hash.

LÃľase mÃąs sobre estos mÃľtodos aquÃ■.

Tutoriales

En inglÃľs

1. InstalaciÃşn, (contribuciÃşn de Victor).
2. CÃşmo enviar una transacciÃşn, (contribuciÃşn de Victor).
3. CÃşmo escribir un contrato inteligente en Javascript, (contribuciÃşn de otto).
4. IntroducciÃşn al almacenamiento de contratos inteligentes, (contribuciÃşn de Victor).
5. InteracciÃşn con Nebulas por medio de la API RPC, (contribuciÃşn de Victor).

CÃşmo contribuir

ÂąSiÃľntete libre de unirte a la Mainnet de Nebulas! Si has encontrado un error, por favor envÃ■a un aviso, o si eres desarrollador, crea un pull request; de ese modo podremos corregir los errores o aÃśadir tu contribuciÃşn a esta pÃągina lo antes posible.

CÃşmo unirse a la testnet de Nebulas

IntroducciÃşn

Estamos encantados de lanzar la Testnet de Nebulas. Simula la red de Nebulas y NVM, y permite a los desarrolladores interactuar con Nebulas sin pagar el costo del gas.

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

Compilacion

El archivo ejecutable de la mainnet de Nebulas y las bibliotecas dependientes se deben compilar primero. Varios mÃşdulos importantes se enumeran a continuaciÃşn:

- **NBRE:** el Nebulas Blockchain Runtime Environment es la plataforma para ejecutar la RepresentaciÃşn del Protocolo de Nebulas, como el DIP, el NR, etc.
- **NEB:** el proceso principal de la mainnet de Nebulas. NEB y NBRE se ejecutan en procesos independientes y se comunican a travÃľs de IPC.

Las instrucciones sobre cÃşmo compilar los mÃşdulos se pueden encontrar en los tutoriales.

ConfiguraciÃşn

PodrÃąs encontrar los archivos de configuraciÃşn de la testnet en `testnet/conf`.

Esa carpeta contiene los siguientes archivos:

genesis.conf

Permite configurar los parÃąmetros del bloque inicial (*genesis block*), incluyendo:

- **meta.chain_id:** identidad de la cadena.
- **consensus.dpos.dynasty:** dinastÃ■a inicial de validadores.
- **token_distribution:** asignaciÃşn inicial de tokens.

IMPORTANTE: No se deben realizar cambios sobre el archivo genesis.conf.

config.conf

Permite configurar los parÃąmetros del runtime.

Para mÃąs informaciÃşn sobre este archivo, vÃľase `template.conf`.

Nota: la informaciÃşn del nodo semilla oficial debe verse tal como se muestra aquÃ■ debajo:

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

Resumen de la API

Test Endpoint:

- GetNebState: returns nebulas client info.
- GetAccountState: devuelve el balance y el *nonce* de la cuenta.
- LatestIrreversibleBlock: devuelve el Ãžltimo bloque irreversible.
- Call: ejecuta un contrato inteligente de forma local, sin enviar datos al chain.
- SendRawTransaction: permite enviar una transacciÃşn firmada.
- GetTransactionReceipt: obtiene informaciÃşn del recibo de una transacciÃşn mediante su hash.

LÃľase mÃąs sobre estos APIs aquÃ■.

Pedir Tokens

Cada email puede reclamar tokens todos los dÃ■as aquÃ■.

Tutoriales

1. Installation (thanks Ariel)
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

ÂąSiÃľntete libre de unirte a la Mainnet de Nebulas! Si has encontrado un error, por favor envÃ■a un aviso, o si eres desarrollador, crea un pull request; de ese modo podremos corregir los errores o aÃśadir tu contribuciÃşn a esta pÃągina lo antes posible.

Nebulas node environment

Introduction

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

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

Hardware configuration

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

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

Environment

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

NTP

Install the NTP service to keep system time in sync.

Ubuntu install steps:

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

Centos install steps:

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

Contribution

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

DApp Development

Smart Contract

Lenguajes

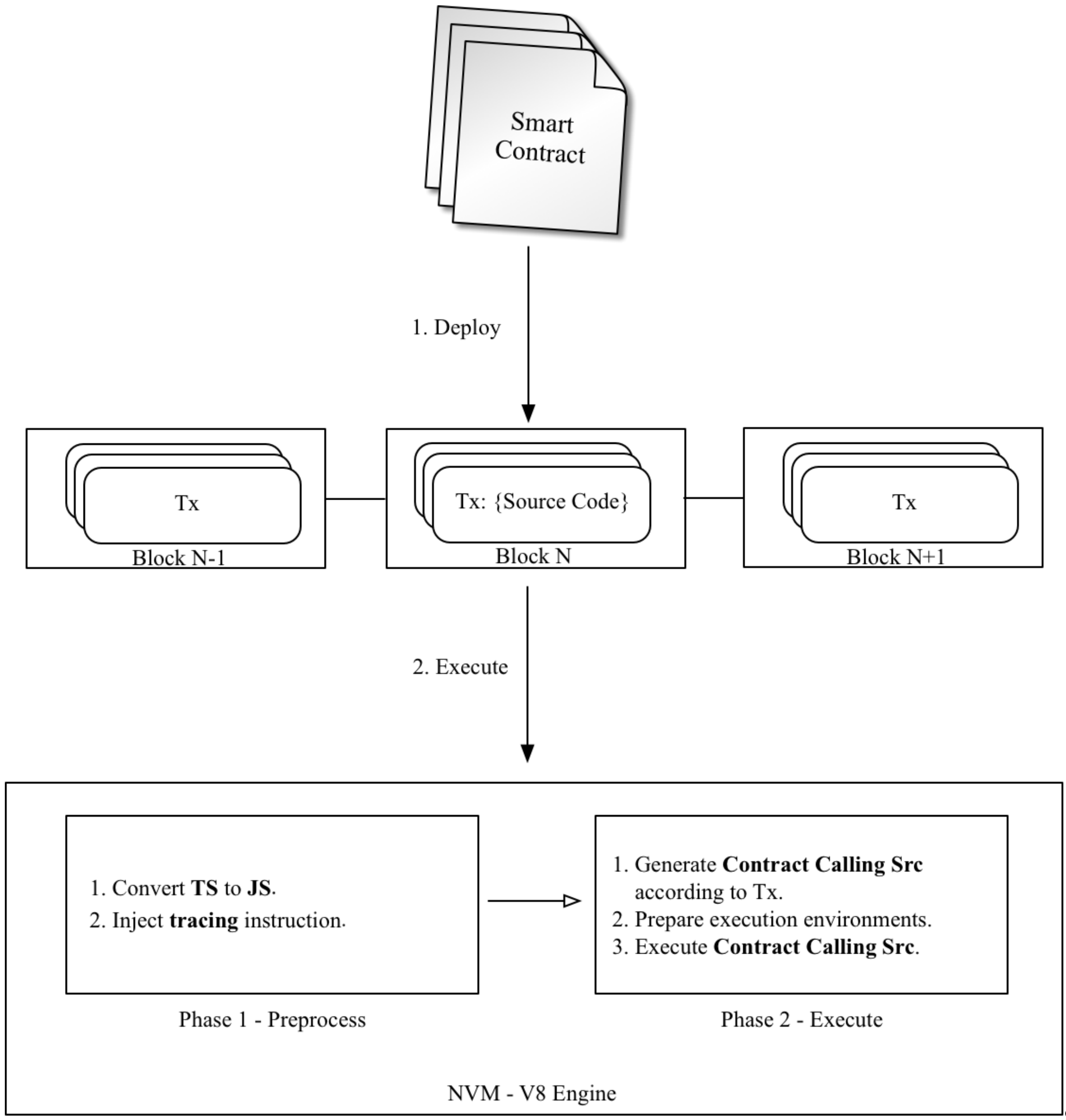
Los contratos inteligentes en la plataforma Nebulas pueden ser escritos en dos lenguajes:

- JavaScript
- TypeScript

Ambos lenguajes estÃąn integrados mediante Chrome V8, un motor javascript desarrollado por el proyecto Chromium (Google Chrome y navegadores Chromium derivados).

Modelo de ejecuciÃşn

El diagrama de aquÃ■ abajo representa el modelo de ejecuciÃşn de los contratos inteligentes:



Smart Contract Execution Model

1. El cÃşdigo completo del contrato inteligente, junto a sus argumentos, son empaquetados en la transacciÃşn e implementados en Nebulas.
2. La ejecuciÃşn de todo contrato inteligente se divide en dos fases:
    (a) Pre-proceso: inyectar traceo de ejecuciones y otras medidas de seguridad similares.
    (b) EjecuciÃşn: generar el cÃşdigo ejecutable y lanzarlo.

Contratos

Los contratos en Nebulas son similares a las clases en el contexto de lenguajes de programaciÃşn orientados a objetos. Contienen datos persistentes en variables de estado y funciones que pueden modificar dichas variables.

Escribir un contrato

Un contrato, tanto en JavaScript como en TypeScript, debe ser siempre un prototipo de objeto o bien una clase.

Todo contrato debe incluir una funciÃşn `init`, que se ejecutarÃą por Ãžnica vez durante la implementaciÃşn en el blockchain.

Las funciones cuyo nombre lleva el prefijo _ son de tipo `private` y no se pueden ejecutar directamente en una transacciÃşn (ya que estÃąn fuera de scope). Todas las demÃąs se consideran `public` y se pueden ejecutar directamente en una transacciÃşn.

Debido a que los contratos se ejecutan en Chrome V8, las variables de instancia se cargan en memoria, por lo que no es recomendable guardarlas a todas en un state trie. Con este fin, Nebulas expone los objetos `LocalContractStorage` y `GlobalContractStorage` que les permiten a los desarrolladores definir cuÃąles son los campos que se requiere almacenar en un *state trie*. Estos campos se deben definir en el `constructor` del contrato, antes de cualquier otra funciÃşn.

El siguiente es un ejemplo de contrato inteligente:

```
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 + ".");
                }
        }
}
```

Alcance

En Nebulas se definen dos tipos de alcance: `public` y `private`:

- `public`: toda funciÃşn cuyo nombre coincide con la expresiÃşn regular `^[a-zA-Z$][A-Za-z0-9_$]*$` es pÃžblica âĂŤcon excepciÃşn de `init`âĂŤ. Las funciones pÃžblicas se pueden llamar directamente desde `Transaction`.
- `private`: toda funciÃşn cuyo nombre coincide con el prefijo _ es privada. Una funciÃşn privada sÃşlo puede ser llamada por una funciÃşn pÃžblica, u otra funciÃşn privada, pero nunca directamente desde `Transaction`.

Objetos globales

console

El mÃşdulo `console` brinda una consola de depuraciÃşn similar a la de JavaScript que poseen los navegadores modernos.

La consola global se puede utilizar sin necesidad de realizar una llamada a `require('console')`.

console.info([...args])

- ...args <any>

  La funciÃşn console.info() es un alias de `console.log()`.

console.log([...args])

- ...args <any>

  Imprime los parÃąmetros `args` en el registro de Nebulas, a nivel de `info`.

console.debug([...args])

- ...args <any>

  Imprime los parÃąmetros `args` en el registro de Nebulas, a nivel de `debug`.

console.warn([...args])

- ...args <any>

  Imprime los parÃąmetros `args` en el registro de Nebulas, a nivel de `warn`.

console.error([...args])

- `...args <any>`

  Imprime los parÃąmetros `args` en el registro de Nebulas, a nivel de `error`.

LocalContractStorage

El mÃşdulo `LocalContractStorage` provee un *state trie* basado en la capacidad de almacenamiento. Acepta sÃşlo pares de valores clave de tipo cadena.

Todos los datos se almacenan en un state trie privado, asociado a la direcciÃşn del contrato actual. ÃŽnicamente el contrato puede acceder a la misma.

```
interface Descriptor {
        // serializar el valor a una cadena;
        stringify?(value: any): string;

        // de-serializar el valor desde una cadena;
        parse?(value: string): any;
}

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

interface ContractStorage {
        //obtener y devolver el valor de una clave desde Native␣
↪Storage.
        rawGet(key: string): string;
        // establecer clave y par de valores en Native Storage,
        // devolver 0 si no hay errores.
        rawSet(key: string, value: string): number;

        // definir una propiedad llamada `fieldname` para el objeto␣
↪`obj` con descriptor.
        // el descriptor por defecto es JSON.parse/JSON.stringify.
        // devolver lo siguiente:
        defineProperty(obj: any, fieldName: string, descriptor?:␣
↪Descriptor): any;

        // definir las propiedades del objeto `obj` desde `props`.
        // el descriptor por defecto es JSON.parse/JSON.stringify.
        // devolver lo siguiente:
        defineProperties(obj: any, props: DescriptorMap): any;

        // define a StorageMap property named `fieldname` to `obj`␣
↪with descriptor.
        // el descriptor por defecto es JSON.parse/JSON.stringify.
```

```
        // devolver lo siguiente:
        defineMapProperty(obj: any, fieldName: string, descriptor?:␣
↪Descriptor): any;

        // define StorageMap properties to `obj` from `props`.
        // el descriptor por defecto es JSON.parse/JSON.stringify.
        // devolver lo siguiente:
        defineMapProperties(obj: any, props: DescriptorMap): any;

        // borrar una clave en Native Storage.
        // devolver 0 si no hay errores.
        del(key: string): number;

        // obtener un valor mediante una clave desde Native Storage,
        // de-serializar el valor llamando al mÃl'todo `descriptor.
↪parse`.
        get(key: string): any;

        // establecer clave y par de valores en Native Storage,
        // el valor debe serializarse a una cadena mediante una␣
↪llamada a `descriptor.stringify`.
        // devolver 0 si no hay errores.
        set(key: string, value: any): number;
}

interface StorageMap {
        // borrar clave desde Native Storage, devolver 0 si no hay␣
↪errores.
        del(key: string): number;

        // obtener valor mediante clave desde Native Storage,
        // de-serializar el valor mediante una llamada a␣
↪`descriptor.parse`.
        get(key: string): any;

        // establecer clave y par de valores en Native Storage,
        // el valor debe serializarse a una cadena mediante una␣
↪llamada a `descriptor.stringify`.
        // devolver 0 si no hay errores.
        set(key: string, value: any): number;
}
```

BigNumber

El mÃşdulo `BigNumber` hace uso de bignumber.js, una librerÃ■a JavaScript para realizar operaciones aritmÃl'ticas sobre nÃžmeros decimales y no decimales de precisiÃşn arbitraria.

El contrato inteligente puede hacer uso de este mÃşdulo `BigNumber` para calcular directamente el valor de una transacciÃşn, o para cualquier otra operaciÃşn que requiera el manejo de grandes nÃžmeros de precisiÃşn arbitraria.

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

Blockchain

El mÃşdulo `Blockchain` expone un objeto que permite obtener las transacciones y los bloques ejecutados por el contrato inteligente que realiza la llamada. Also, the NAS can be transferred from the contract and the address check is provided.

Blockchain API:

```
// current block
Blockchain.block;

// transacciÃşn actual; el valor de la transacciÃşn/gasPrice/
↪gasLimit se convierte automÃąticamente en un objeto BigNumber.
Blockchain.transaction;

// transferir NAS desde un contrato inteligente a una direcciÃşn␣
↪dada
Blockchain.transfer(address, value);

// verificar la direcciÃşn
Blockchain.verifyAddress(address);
```

Propiedades

- `block`: bloque actual para la ejecuciÃşn del contrato
  - `timestamp`: timestamp del bloque
  - `seed`: semilla aleatoria
  - `height`: altura del bloque
- `transaction`: transacciÃşn actual para la ejecuciÃşn del contrato
  - `hash`: hash de la transacciÃşn
  - `from`: direcciÃşn del emisor de la transacciÃşn
  - `to`: direcciÃşn del destinatario de la transacciÃşn
  - `value`: valor de la transacciÃşn (objeto BigNumber)
  - `nonce`: nonce de la transacciÃşn
  - `timestamp`: timestamp de la transacciÃşn
  - `gasPrice`: gasPrice de la transacciÃşn (objeto BigNumber)

    - `gasLimit`: gasLimit de la transacciÃşn, (objeto BigNumber)

- `transfer(address, value)`: transferir NAS desde un contrato inteligente a una direcciÃşn
    - parÃąmetros:
        - `address`: direcciÃşn Nebulas en donde se recibirÃąn los NAS
        - `value`: valor a transferir (objeto BigNumber)
    - valor devuelto:
        - `0`: transferencia exitosa
        - `1`: transferencia fallida
- `verifyAddress(address)`: verificar direcciÃşn
    - parÃąmetros:
        - `address`: direcciÃşn que se desea chequear
    - return:
        - `1`: direcciÃşn vÃąlida
        - `0`: direcciÃşn invÃąlida

Ejemplo de uso:

```
"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;
```

Evento

El mÃşdulo `Event` almacena los eventos de la ejecuciÃşn de un contrato inteligente dado. Los eventos registrados se almacenan en el *trie* de eventos en la cadena, desde donde se pueden recuperar por medio del mÃľtodo `FetchEvents` utilizando para ello el hash de la transacciÃşn.

Todos los tÃşpicos de evento de contrato llevan el prefijo `chain.contract.` antes del tÃşpico asociado al contrato.

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

- `topic`: tÃşpico definido por el usuario
- `obj`: objeto JSON

VÃľase el ejemplo en `SampleContract` (mÃąs arriba).

Math.random

- `Math.random()` devuelve un nÃžmero pseudoaleatorio de coma flotante en el intervalo (0, 1]. El uso tÃ■pico es:

```
"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)`: si es necesario, se puede utilizar este mÃľtodo para reinicializar la semilla de nÃžmeros aleatorios. **El argumento `myseed` debe ser de tipo string**.

```

"use strict";
var BankVaultContract = function \(\) {};

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

        game:function(subscript, myseed){

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

                  console.log(Math.random());

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

                            if (i == 8) {
                                          // reinicializar la semilla␣
↪de aleatorios con `myseed`
                                          Math.random.seed(myseed);
                            }

                            var rand = parseInt(Math.random()*arr.
↪length);
```

```
                                var t = arr[rand];
                                arr[rand] =arr[i];
                                arr[i] = t;
                        }
                        return arr[parseInt(subscript)];
                },
};

module.exports = BankVaultContract;
```

Date

```
"use strict";

var BankVaultContract = function () {};

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

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

module.exports = BankVaultContract;
```

Notas

- MÃľtodos no soportados: `toDateString()`, `toTimeString()`, `getTimezoneOffset()`, `toLocaleXXX()`.
- `new Date()` y `Date.now()` devuelven el timestamp del bloque actual en milisegundos.
- `getXXX` devuelve el resultado de `getUTCXXX`.

accept

Este mÃľtodo permite realizar una transferencia binaria hacia un contrato inteligente.

Siempre y cuando `to` sea una direcciÃşn de contrato inteligente que declara el mÃľtodo `accept()` y se ejecuta correctamente, la transferencia se realizarÃą sin errores.

```
"use strict";
var DepositeContent = function (text) {
    if(text){
```

```
            var o = JSON.parse(text);
            this.balance = new BigNumber(o.balance);// informaciÃşn␣
↪de balance
            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

Resumen

El siguiente estÃąndar permite la implementaciÃşn de una API estandarizada para la utilizaciÃşn de tokens dentro de contratos inteligentes, como asÃ■ tambiÃľn su transferencia y su uso por terceros dentro del blockchain.

MotivaciÃşn

Una interfaz estÃąndar permite crear tokens desde una aplicaciÃşn de una forma rÃąpida, para ser usada en distintos contextos: desde una cartera hasta una casa de cambios descentralizada.

MÃľtodos

name

Devuelve el nombre del token; por ejemplo: `"MyToken"`.

```
// Devuelve una cadena con el nombre del token.
function name()
```

symbol

Devuelve el sÃ■mbolo del token; por ejemplo: "TK".

```
// Devuelve una cadena con el sÃ■mbolo del token.
function symbol()
```

decimals

Devuelve el nÃžmero de decimales que utiliza el token; por ejemplo: `8`.

Este nÃžmero indica, ademÃąs, la unidad indivisible y mÃ■nima, y el coeficiente a ser utilizado por otras funciones. Por ejemplo, si esta funciÃşn devuelve `8`, serÃą necesario multiplicar por `0.00000001` cualquier otro valor para obtener su representaciÃşn coloquial, ya que las funciones devuelven normalmente los valores en unidades mÃ■nimas.

```
// Devuelve un nÃžmero con la cantidad de decimales que utiliza el␣
↪token.
function decimals()
```

totalSupply

Devuelve el suministro total de tokens.

```
// Devuelve una cadena que representa el suministro total de tokens,
↪ expresado mediante su unidad mÃąs pequeÃśa (lo cual se puede␣
↪consultar mediante el mÃľtodo decimals.
function totalSupply()
```

balanceOf

Devuelve el balance de una direcciÃşn, expresado en unidades mÃ■nimas (vÃľase *decimals*).

```
// Devuelve una cadena que representa el balance de la direcciÃşn␣
↪dada.
function balanceOf(address)
```

transfer

Transfiere la cantidad de tokens indicada por `value` (expresado en unidades mÃ■nimas; vÃľase *decimals* para mÃąs informaciÃşn) a la direcciÃşn `address`. Devuelve un boolean `true` si no hay errores; de lo contrario se lanza un `error`.

```
// Devuelve `true` si la transferencia se realiza exitosamente; en␣
↪otro caso, se lanza un error.
function transfer(address, value)
```

Nota

Las transferencias cuyo valor es 0 son transferencias vÃąlidas, que tambiÃľn disparan el evento `Transfer`.

transferFrom

Transfiere la cantidad de tokens indicada por `value` (expresado en unidades mÃ■nimas; vÃľase *decimals* para mÃąs informaciÃşn) desde la direcciÃşn `from` a la direcciÃşn `to`. Devuelve un boolean `true` si no hay errores; de lo contrario se lanza un `error`.

Este mÃľtodo tiene sentido en el contexto de un contrato inteligente, en el que es necesario delegar el envÃ■o de tokens.

```
// Devuelve `true` si la transferencia se realiza exitosamente; de
↪lo contrario devuelve error.
function transferFrom(from, to, value)
```

Nota

Las transferencias cuyo valor es 0 son transferencias vÃąlidas, que tambiÃľn disparan el evento `Transfer`.

approve

Otorga a `spender` el derecho de realizar mÃžltiples extracciones sobre la cuenta actual, hasta alcanzar el valor especificado mediante el parÃąmetro `value`.

El parÃąmetro `currentValue` especifica el valor actual para `value`, y se establece en 0 antes de la primera llamada; el parÃąmetro existe para impedir ataques.

Con cada nueva llamada, este mÃľtodo sobreescribe el valor permitido con el nuevo valor pasado mediante `value`.

```
// Devuelve `true` si la transacciÃşn se ejecuta correctamente; si
↪no, se devuelve un error.
function approve(spender, currentValue, value)
```

Nota

Para prevenir ataques, el valor por defecto para `value` y `currentValue` es 0.

allowance

Devuelve el valor remanente que `spender` puede todavÃ■a extraer de la cuenta `owner`.

```
// Devuelve una cadena que representa el valor remanente que
↪spender puede todavÃ■a extraer de la cuenta owner.
function allowance(owner, spender)
```

Eventos

transferEvent

Debe dispararse cuando se transfieren tokens, incluyendo operaciones por un valor nulo (0).

Todo contrato inteligente, luego de generar nuevos tokens, **debe** lanzar un evento Transfer con el parÃąmetro `from` indicando el valor `totalSupply`.

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

approveEvent

Este evento se debe disparar luego de una llamada al mÃľtodo `approve(spender, currentValue, value)`.

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

ImplementaciÃşn

Los ejemplos de implementaciones estÃąn disponibles en el archivo NRC20.js.

```
'use strict';

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

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

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

    get: function (key) {
        return this.allowed[key];
```

```
    },

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
        this.transferEvent(true, from, to, value);
    },

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

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

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

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

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

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

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

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

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

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

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

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

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

        this.allowed.set(from, owned);

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

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

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

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

module.exports = StandardToken;
```

NRC721

Abstract

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

Motivation

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

Methods

name

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

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

balanceOf

Returns the number of tokens owned by `owner`.

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

ownerOf

Returns the address of the owner of the tokens.

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

transferFrom

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

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

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

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

approve

Set or reaffirm the approved address for an token.

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

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

setApprovalForAll

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

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

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

getApproved

Get the approved address for a single token.

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

isApprovedForAll

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

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

Events

_transferEvent

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

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

_approveEvent

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

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

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

Implementation

Example implementations are available at

- NRC721BasicToken.js

```
'use strict';

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

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

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

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

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

var StandardToken = function () {
    LocalContractStorage.defineProperties(this, {
        _name: null,
    });
```

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

    });
};

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

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

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

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

    /**
     * Set or reaffirm the approved address for an token.
```

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

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

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

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

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

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

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

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

```
            this._removeTokenFrom(from, tokenId);
            this._addTokenTo(to, tokenId);
            this._transferEvent(true, from, to, tokenId);
        } else {
            throw new Error("permission denied in transferFrom.");
        }

    },

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

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

    /**
      * Internal function to add a token ID to the list of a given␣
↪address
      * @param to address representing the new owner of the given␣
↪token ID
```

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

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

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

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

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

```
            }
        });
    }

};

module.exports = StandardToken;
```

Tools

All the developing 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 tool

- Web-wallet
- WebExtensionWallet

Nebpay

- JavaScript SDK
- iOS SDK
- Android SDK

Nebulas API

- Go
- Python
- PHP
- ruby
- NET
- unity3d
- swift

Static scan tool

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

Command line tool

- A CLI Tool forÂăNebulas

test tools

- NebTest will automate unit testing ofÂănebulasÂăsmart contracts

other

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

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

RPC

Las **Llamadas a Procedimiento Remoto** (Remote Procedure Calls, RPC) brindan una abstracciÃşn Ãžtil para crear servicios y aplicaciones distribuidas.

Nebulas brinda tanto gRPC como RESTful API para que sus usuarios interactuen con el sistema.

- grpc expone una implementaciÃşn concreta del protocolo gRPC, construido en capas sobre HTTP/2. Estas librerÃ■as permiten la comunicaciÃşn entre clientes y servidores usando cualquier combinaciÃşn de lenguajes soportados.
- grpc-gateway es un *plugin* de protoc. Lee las definiciones de servicio de gRPC, y genera un servidor reverse-proxy server que traduce una API RESTful JSON a gRPC. Se utiliza para mapear gRPC a HTTP.

Puntos finales (endpoints)

Puntos finales por defecto:

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

API gRPC

Podemos correr el ejemplo para gRPC, el cliente de pruebas:

```
go run main.go
```

El cliente de pruebas obtiene los estados de cuentas de la direcciÃşn del emisor, crea una transacciÃşn de emisor a receptor, y chequea el estado de cuentas de la direcciÃşn del receptor.

La salida del registro podrÃ■a verse asÃ■:

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

HTTP

TambiÃľn ofrecemos HTTP para acceder a la API RPC desde el navegador. El archivo de mapeo tiene la extensiÃşn **gw.go**.

Es posible actualizar los mÃľtodos **rpc_listen** y **http_listen** en **conf/default/config.conf** para cambiar el puerto RPC/HTTP.

Ejemplo

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

Si todo marcha bien, la respuesta deberÃ■a verse asÃ■:

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

Si ocurre un error de grpc, la respuesta se verÃą asÃ■:

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

RPC methods

- *GetNebState*
- *GetAccountState*

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

RPC API Referencia

Devuelve el estado del neb.

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

ParÃąmetros

ninguno

Devuelve

- `chain_id`: ID del blockchain
- `tail`: Hash de la cola del neb actual
- `lib`: Hash de la librerÃ■a neb actual
- `height`: Altura del bloque de cola neb actual
- `protocol_version`: La versiÃşn del protocolo neb actual.
- `synchronized`: Estado de la sincronizaciÃşn del par.
- `version`: VersiÃşn neb.

Ejemplo para HTTP

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

// Resultado
```

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

Devuelve el estado de la cuenta. Se devuelve el balance y el nonce de la direcciÃşn dada.

Protocol	Method	API
gRpc		GetAccountState
HTTP	POST	/v1/user/accountstate

ParÃąmetros

- `address`: Cadena hexadecimal que representa la direcciÃşn de la cuenta.
- `height`: Estado de la cuenta bloque, con altura. Si no se especifica, utilice 0 como altura de cola.

Devuelve

- `balance`: Balance actual en unidades de 1/(10^18) NAS.
- `nonce`: Ã■ndice numÃľrico de la transacciÃşn actual.
- `type`: Tipo de direcciÃşn; **87** indica direcciones normales y **88** indica direcciones de contratos inteligentes.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/accountstate -d '{"address":
↪"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3"}'

// Resultado
{
    result {
        "balance":"9489999998980000000000"
        "nonce":51
        "type":87
    }
}
```

Devuelve el Ãžltimo bloque irreversible.

Protocol	Method	API
gRpc		LatestIrreversibleBlock
HTTP	GET	/v1/user/lib

ParÃąmetros

none

Devuelve

- `hash`: Cadena hexadecimal que representa el hash del bloque.
- `parent_hash`: Cadena hexadecimal que representa el hash del bloque padre.
- `height`: Altura del bloque.
- `nonce`: Nonce del bloque.
- `coinbase`: Cadena hexadecimal que representa la direcciÃşn coinbase.
- `timestamp`: Timestamp del bloque.
- `chain_id`: ID del blockchain.
- `state_root`: Cadena hexadecimal que representa el estado raÃ■z.
- `txs_root`: Cadena hexadecimal que representa la transacciÃşn raÃ■z.
- `events_root`: Cadena hexadecimal que representa el evento raÃ■z.
- `consensus_root`: Cadena hexadecimal que representa el consenso raÃ■z.
- `Timestamp`: Timestamp del estado de consenso.
- `Proposer`: Proponente del estado de consenso actual.
- `DynastyRoot`: Cadena hexadecimal que representa la dinastÃ■a raÃ■z.
- `miner`: Minero del bloque.
- `is_finality`: Indica si el bloque es *finality*.
- `transactions`: Slice de transacciones del bloque.
- `transaction`: InformaciÃşn de respuesta *GetTransactionReceipt*.

Ejemplo para HTTP

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

// Resultado
```

```
{
    "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

Permite realizar una llamada a las funciones de un contrato inteligente dado. Es importante que el contrato inteligente ya estÃľ implementado en el blockchain.

Las llamadas a mÃľtodos se realizan sobre el nodo actual, y no se transmiten al resto de la red.

Protocol	Method	API
gRpc		Call
HTTP	POST	/v1/user/call

ParÃąmetros

Son los mismos que se utilizan en *SendTransaction*, con atenciÃşn especial a:

- `to`: Cadena hexadecimal que representa la direcciÃşn de la cuenta receptora. **El valor correcto para ‘‘to‘‘ es la direcciÃşn del contrato.**
- `contract`: Objeto `transaction contract` para la llamada al contrato inteligente.
- Subpropiedades (no se requieren los parÃąmetros `source` y `sourceType`):
  - `function`: El nombre de la funciÃşn a llamar.
  - `args`: Los parÃąmetros del contrato. El contenido de `args` es una cadena JSON que contiene una matriz de parÃąmetros.

Devuelve

- `result`: El resultado de la llamada a la funciÃşn del contrato inteligente.
- `execute_err`: Error de ejecuciÃşn (si ocurriÃş un error).
- `estimate_gas`: EstimaciÃşn del gas utilizado.

Ejemplo para HTTP

```
// PeticiÃşn
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]"}}'

// Resultado
{
   "result": "0",
   "execute_err": "insufficient balance",
   estimate_gas: "22208"
}
```

SendRawTransaction

Permite enviar una transacciÃşn firmada. El valor de la transacciÃşn firmada se debe devolver por medio de SignTransactionWithPassphrase.

Protocol	Method	API
gRpc		SendRawTransaction
HTTP	POST	/v1/user/rawtransaction

ParÃąmetros

- `data`: Datos de la transacciÃşn, firmados.

Devuelve

- `txhash`: Cadena hexadecimal que representa el hash de la transacciÃşn.
- `contract_address`: Se devueve Ãžnicamente cuando se utiliza una transacciÃşn de contrato inteligente.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/rawtransaction -d '{"data":"CiCrHtxyyIJks2/
↪RErvBBA862D6iwAaGQ9OK1NisSGAuTBIYGiY1R9Fnx0z0uPkWbPokTeBIHFFKRaosGhgzPLPtjEF5c
↪i9wAiEAAAAAAAAAAADeC2s6dkAAAoAjDd/
↪5jSBToICgZiaW5hcnlAZEoQAAAAAAAAAAAAAAAAAA9CQFIQAAAAAAAAAAAAAAAAAABOIFgBYkGLnnv
↪"}'

// Resultado
{
    "result":{
        "txhash":
↪"f37acdf93004f7a3d72f1b7f6e56e70a066182d85c186777a2ad3746b01c3b52"
    }
}
```

Ejemplo para utilizaciÃşn de contrato

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/rawtransaction -d '{"data":"CiDam3G9Sy5fV6/
↪ZcjasYPwSF39ZJDIHNB0Us94vn6p6ohIaGVfLzJ83pom1DO1gD307f1JdTVdDLzbMXO4aGhlXy8yfN
↪CEbThvI0iKcjHhgBZUB"}'

// Resultado
{
    "result":{
        "txhash":
↪"f37acdf93004f7a3d72f1b7f6e56e70a066182d85c186777a2ad3746b01c3b52
↪",
        "contract_address":
↪"4702b597eebb7a368ac4adbb388e5084b508af582dadde47"
    }
}
```

GetBlockByHash

Obtiene informaciÃşn del encabezado de un bloque por medio de su hash.

Protocol	Method	API
gRpc		GetBlockByHash
HTTP	POST	/v1/user/getBlockByHash

ParÃąmetros

- `hash`: Cadena hexadecimal que representa el hash de la transacciÃşn.
- `full_fill_transaction`: Si su valor es `true`, devuelve un objeto de transacciÃşn completo; si es `false`, sÃşlo sus hashes.

Devuelve

VÃľase `LatestIrreversibleBlock <./#latestirreversibleblock>`__.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getBlockByHash -d '{"hash":
↪"00000658397a90df6459b8e7e63ad3f4ce8f0a40b8803ff2f29c611b2e0190b8
↪", "full_fill_transaction":"true"}'

// Resultado
{
    "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

Permite obtener informaciÃşn del encabezado de un bloque dado mediante su altura (parÃąmetro `height`).

Protocol	Method	API
gRpc		GetBlockByHeight
HTTP	POST	/v1/user/getBlockByHeight

ParÃąmetros

- `height`: Altura del hash de la transacciÃşn.
- `full_fill_transaction`: Si su valor es `true`, devuelve un objeto de transacciÃşn completo; si es `false`, sÃşlo sus hashes.

Devuelve

VÃľase LatestIrreversibleBlock.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getBlockByHeight -d '{"height": 256, "full_
↪fill_transaction": true}'

// Resultado
{
    "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

Obtiene informaciÃşn del recibo de una transacciÃşn mediante su hash.

Si la transacciÃşn no existe o no se ha empaquetado todavÃ■a en el blockchain, devolverÃą un error `not found`.

Protocol	Method	API
gRpc		GetTransactionReceipt
HTTP	POST	/v1/user/getTransactionReceipt

ParÃąmetros

`hash` Hex string of transaction hash.

Devuelve

- `hash`: Cadena hexadecimal que representa el hash de la transacciÃşn.
- `chainId`: ID del *chain* de la transacciÃşn.
- `from`: Cadena hexadecimal que representa la direcciÃşn de la cuenta del emisor.
- `to`: Cadena hexadecimal que representa la direcciÃşn de la cuenta del receptor.
- `value`: Valor de la transacciÃşn.
- `nonce`: Nonce de la transacciÃşn.
- `timestamp`: Timestamp de la transacciÃşn.
- `type`: Tipo de transacciÃşn.
- `data`: Datos del *payload* de la transacciÃşn.
- `gas_price`: Precio del gas utilizado por la transacciÃşn.
- `gas_limit`: LÃ■mite del gas utilizado por la transacciÃşn.
- `gas_used`: Gas utilizado por la transacciÃşn.
- `contract_address`: DirecciÃşn del contrato de la transacciÃşn.
- `status`: Estado de la transacciÃşn:
  - 0: fallÃş.
  - 1: se ejecutÃş sin errores.
  - 2: pendiente.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getTransactionReceipt -d '{"hash":
↪"cda54445ffccf4ea17f043e86e54be11b002053f9edbe30ae1fbc0437c2b6a73
↪"}'

// Resultado
{
    "result":{
        "hash":
↪"cda54445ffccf4ea17f043e86e54be11b002053f9edbe30ae1fbc0437c2b6a73
↪",
        "chainId":100,
```

```
            "from":"n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
            "to":"n1PxKRaJ5jZHXwTfgM9WqkZJJVXBxRcggEE",
            "value":"1000000000000000000",
            "nonce":"53",
            "timestamp":"1521964742",
            "type":"binary",
            "data":null,
            "gas_price":"1000000",
            "gas_limit":"20000",
            "contract_address":"",
            "status":1,
            "gas_used":"20000"
        }
}
```

GetTransactionByContract

Obtiene informaciÃşn de la transacciÃşn mediante una direcciÃşn de contrato. Si ese contrato no existe o no estÃą empaquetado en el blockchain, se devuelve un error `not found`.

Protocol	Method	API
gRpc		GetTransactionByContract
HTTP	POST	/v1/user/getTransactionByContract

ParÃąmetros

- `address`: Cadena hexadecimal que representa la direcciÃşn de la cuenta del contrato.

Devuelve

- El resultado es el mismo que se obtiene mediante el mÃľtodo *GetTransactionReceipt*.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getTransactionByContract -d '{"address":
↪"n1sqDHGjYtX6rMqFoq5Tow3s3LqF4ZxBvE3"}'

// Resultado
{
    "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

Permite suscribirse a tÃşpicos (eventos) generados por transacciones y bloques. La peticiÃşn se hace mediante una conexiÃşn *keep-alive*.

Protocol	Method	API
gRpc		Subscribe
HTTP	POST	/v1/user/subscribe

ParÃąmetros

- `topics`: Matriz de cadenas con los nombres de los tÃşpicos (eventos) a los cuales nos suscribiremos. Estos pueden ser:
  - `chain.pendingTransaction`: Transacciones pendientes en `transaction_pool`.
  - `chain.latestIrreversibleBlock`: ActualizaciÃşn del Ãžltimo bloque irreversible.
  - `chain.transactionResult`: EjecuciÃşn y envÃ■o de una transacciÃşn.
  - `chain.newTailBlock`: DefiniciÃşn de un nuevo `tail block`.
  - `chain.revertBlock`: ReversiÃşn de un bloque.

Devuelve

- `topic`: Nombre del evento o eventos suscritos.
- `data`: Datos del evento o eventos suscritos.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/subscribe -d '{"topics":["chain.linkBlock",
↪ "chain.pendingTransaction"]}'

// Resultado
{
    "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

Obtiene el valor actual del gas.

Protocol	Method	API
gRpc		GetGasPrice
HTTP	GET	/v1/user/getGasPrice

ParÃąmetros

ninguno.

Devuelve

- `gas_price`: valor del gas al momento de la peticiÃşn. La unidad es 10^-18 NAS.

Ejemplo para HTTP

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

// Resultado
{
    "result":{
        "gas_price":"1000000"
    }
}
```

EstimateGas

Obtiene un estimado de la cantidad de gas necesario para una transacciÃşn dada.

Protocol	Method	API
gRpc		EstimateGas
HTTP	POST	/v1/user/estimateGas

ParÃąmetros

- Se usan los mismos parÃąmetros que en el mÃľtodo *SendTransaction*.

Devuelve

- `gas`: estimado de la cantidad de gas necesario para la transacciÃşn.
- `err`: mensaje de error correspondiente a la transacciÃşn que se ejecutarÃą.

Ejemplo para HTTP

```
// PeticiÃşn
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"}'

// Resultado
{
    "gas":"20000",
    "err":""
}
```

GetEventsByHash

Devuelve la lista de eventos de una transacciÃşn identificada mediante su hash.

Protocol	Method	API
gRpc		GetEventsByHash
HTTP	POST	/v1/user/getEventsByHash

ParÃąmetros

- `hash`: Cadena hexadecimal que representa el hash de la transacciÃşn.

Devuelve

- `events`: Lista de eventos, conteniendo, cada uno de ellos:
  - `topic`: TÃşpico del evento.
  - `data`: Datos del evento.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/getEventsByHash -d '{"hash":
↪"ec239d532249f84f158ef8ec9262e1d3d439709ebf4dd5f7c1036b26c6fe8073
↪"}'

// Resultado
{
    "result":{
        "events":[{
            "topic":"chain.transactionResult",
            "data":"{
                \"hash\":\
↪"d7977f96294cd232781d9c17f0f3212b48312d5ef0f556551c5cf48622759785\
↪",
                \"status\":1,
                \"gas_used\":\"22208\",
                \"error\":\"\"
            }"
        }]
    }
}
```

GetDynasty

Obtiene una lista con las direcciones de los mineros de la dinastÃ■a actual.

Protocol	Method	API
gRpc		GetDynasty
HTTP	POST	/v1/user/dynasty

ParÃąmetros

- `height`: altura del bloque.

Devuelve

- `miners`: matriz que contiene las direcciones de los mineros.

Ejemplo para HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/user/dynasty -d '{"height": 1}'

// Resultado
{
    {
        "result":{
            "miners":[
                "n1FkntVUMPAsESuCAAPK711omQk19JotBjM",
                "n1JNHZJEUvfBYfjDRD14Q73FX62nJAzXkMR",
                "n1Kjom3J4KPsHKKzZ2xtt8Lc9W5pRDjeLcW",
                "n1TV3sU6jyzR4rJ1D7jCAmtVGSntJagXZHC",
                "n1WwqBXVMuYC3mFCEEuFFtAXad6yxqj4as4",
                "n1Zn6iyyQRhqthmCfqGBzWfip1Wx8wEvtrJ"
            ]
        }
    }
}
```

Management RPC

AdemÃąs de la interfaz Neb API RPC, Nebulas ofrece otras API de administraciÃşn. La consola Neb da soporte tanto para API como para interfaces administrativas. *Management RPC* hace uso de gRPC y HTTP, lo que permite vincular, ademÃąs, las interfaces Neb API RPC.

Nebulas brinda tanto gRPC como RESTful API para que sus usuarios interactuen con el sistema.

Nuestro archivo proto de administraciÃşn define todas las API administrativas. **Recomendamos usar las interfaces de acceso vÃ■a consola, o restringir la RPC administrativa sÃşlo para acceso local**.

Endpoint administrativo por defecto del RPC:

Referencia

Ã■ndice

- *NodeInfo*

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

NodeInfo

Devuelve informaciÃşn del nodo p2p.

ParÃąmetros

Ninguno.

Devuelve

- `id`: ID del nodo.
- `chain_id`: ID del blockchain.
- `coinbase`: Coinbase.
- `peer_count`: NÃžmero de pares conectados al momento de la consulta.
- `synchronized`: Estado de sincronizaciÃşn del nodo.
- `bucket_size`: El tamaÃśo del *bucket* de la tabla de rutas del nodo.
- `protocol_version`: VersiÃşn del protocolo de red.
- `RouteTable*[] route_table`: Matriz routeTable de la red.

```
message RouteTable {
    string id = 1;
    repeated string address = 2;
}
```

Ejemplo HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/admin/nodeinfo

// Resultado
{
    "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

Devuelve una lista de cuentas.

ParÃąmetros

Ninguno.

Devuelve

- `addresses`: Lista de cuentas.

Ejemplo HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/admin/accounts

// Resultado
{
    "result":{
        "addresses":[
            "n1FkntVUMPAsESuCAAPK711omQk19JotBjM",
            "n1JNHZJEUvfBYfjDRD14Q73FX62nJAzXkMR",
            "n1Kjom3J4KPsHKKzZ2xtt8Lc9W5pRDjeLcW",
            "n1NHcbEus81PJxybnyg4aJgHAaSLDx9Vtf8",
            "n1QZMXSZtW7BUerroSms4axNfyBGyFGkrh5",
            "n1TV3sU6jyzR4rJ1D7jCAmtVGSntJagXZHC",
            "n1WwqBXVMuYC3mFCEEuFFtAXad6yxqj4as4",
            "n1Z6SbjLuAEXfhX1UJvXT6BB5osWYxVg3F3",
            "n1Zn6iyyQRhqthmCfqGBzWfip1Wx8wEvtrJ"
        ]
    }
}
```

NewAccount

Crea una nueva cuenta protegida con una *passphrase*.

ParÃąmetros

- `passphrase`: La *passphrase* de la cuenta a crear.

Devuelve

- `address`: DirecciÃşn de la cuenta creada.

Ejemplo HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/new -d '{"passphrase":"passphrase
↪"}'

// Resultado

{
    "result":{
        "address":"n1czGUvbQQton6KUWga4wKDLLKYDEn39mEk"
    }
}
```

UnLockAccount

Desbloquea una cuenta protegida con *passphrase*. Transcurrido el tiempo de desbloqueo por defecto, la cuenta se bloquearÃą nuevamente.

ParÃąmetros

- `address`: DirecciÃşn de la cuenta a desbloquear.
- `passphrase`: *Passphrase* de la cuenta a desbloquear.
- `duration`: DuraciÃşn del desbloqueo, en nanosegundos.

Devuelve

- `result`: Boolean con el resultado del desbloqueo.

Ejemplo HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/unlock -d '{"address":
↪"n1czGUvbQQton6KUWga4wKDLLKYDEn39mEk","passphrase":"passphrase",
↪"duration":"1000000000"}'

// Resultado
{
    "result":{
        "result":true
    }
}
```

LockAccount

Bloquea una cuenta dada.

ParÃąmetros

- `address`: DirecciÃşn de la cuenta a bloquear.

Devuelve

- `result`: Boolean que indica el resultado del bloqueo.

Ejemplo HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/account/lock -d '{"address":
↪"n1czGUvbQQton6KUWga4wKDLLKYDEn39mEk"}'

// Resultado
{
    "result":{
        "result":true
    }
}
```

SignTransactionWithPassphrase

Firma una transacciÃşn utilizando una *passphrase*. La cuenta de la direcciÃşn del emisor debe estar desbloqueada antes de realizar esta llamada.

ParÃąmetros

- `passphrase`: *passphrase* de la cuenta de origen.
- `transaction`: Ã■dem a *SendTransaction*.

Devuelve

- `data`: Datos de la transacciÃşn firmada.

Ejemplo HTTP

```
// PeticiÃşn
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":"20000000000","gasLimit
↪":"2000000"}, "passphrase":"passphrase"}'

// Resultado
{
    "result":{
        "data":
↪"CiBOW15yoZ+XqQbMNr4bQdJCXrBTehJKukwjcfW5eASgtBIaGVduKnw+6lM3HBXhJEzzuvv3yNdYA
↪BwhwhqUkp/
↪gEJtE4kndoc7NdSgqD26IQqa0Hjbtg1JaszAvHZiW+XH7C+Ky9XTKRJKuTOc446646d/
↪Sbz/nxQE="
    }
}
```

SendTransactionWithPassphrase

EnvÃa una transacciÃşn ya firmada con el mÃľtodo *SignTransactionWithPassphrase*.

ParÃąmetros

- `passphrase`: *Passphrase* de la cuenta origen.
- `transaction`: Ãdem a *SendTransaction*.

Devuelve

- `txhash`: Hash de la transacciÃşn.
- `contract_address`: DirecciÃşn del contrato inteligente.

Ejemplo HTTP

```
// PeticiÃşn
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":"20000000000","gasLimit
↪":"2000000"},"passphrase":"passphrase"}'
```

```
// Resultado
{
    "result":{
        "hash":
↪"143eac221da8079f017bd6fd6b6a08ea0623114c93c638b94334d16aae109666
↪",
        "contract_address":""
    }
}
```

SendTransaction

EnvÃ■a una transacciÃşn. Los parÃąmetros `from`, `to`, `value`, `nonce`, `gasPrice` y `gasLimit` son obligatorios.

ParÃąmetros

- `from`: Requerido. Cadena hexadecimal que representa la direcciÃşn de la cuenta origen.
- `to`: Requerido. Cadena hexadecimal que representa la direcciÃşn de la cuenta destino.
- `value`: Requerido. Valor a enviar junto a esta transacciÃşn.
- `nonce`: Requerido. Nonce de la transacciÃşn.
- `gas_price`: Requerido. Precio del gas para esta transacciÃşn.
- `gas_limit`: Requerido. LÃ■mite de gas para esta transacciÃşn.
- `binary`: Opcional. Cualquier tipo de datos binarios con un tamaÃśo mÃąximo de 64 bytes.
- `type`: Opcional. Tipo de payload para la transacciÃşn. Si se especifica un tipo, el necesario pasar el parÃąmetro correspondiente; si no se especifica, el tipo se determinarÃą de acuerdo al contrato y a sus datos binarios. EnumeraciÃşn de tipos:
  - `binary`: TransacciÃşn normal con datos binarios.
  - `deploy`: ImplementaciÃşn de contrato inteligente.
  - `call`: Llamada a funciÃşn de contrato inteligente.
- `contract`: Opcional. Objeto que contiene el contrato a implementar o la funciÃşn a llamar mediante esta transacciÃşn. Propiedades:
  - `source`: CÃşdigo fuente del contrato a implementar.
  - `sourceType`: Abreviatura del lenguaje del cÃşdigo fuente a implementar. Acepta los siguientes valores:
    - `js`: CÃşigo fuente en Javascript.

    * ts: CÃşdigo fuente en Typescript.

  - function: La llamada a funciÃşn del contrato inteligente.
  - args: ParÃąmetros para el contrato inteligente. El contenido es una matriz de cadenas codificada en un objeto JSON.

Importante

- Al implementar un contrato inteligente, los parÃąmetros to y from deben ser idÃľnticos, y deben corresponder a la direcciÃşn del contrato inteligente.
- nonce: Para obtener el valor correcto, debe sumarse 1 al nonce actual de la direcciÃşn asignada al contrato inteligente. El valor actual del nonce se puede obtener mediante GetAccountState.
- Los parÃąmetros gasPrice y gasLimit son necesarios para cada transacciÃşn. Recomendamos hacer uso de los mÃľtodos GetGasPrice y EstimateGas para obtener los valores actuales.
- El parÃąmetro contract sÃşlo se requiere para las implementaciones (o llamadas a funciones) de contratos inteligentes.
- Cuando se implementa un contrato inteligente, los parÃąmetros source y sourceType deben especificarse; args en este contexto es opcional y se usa Ãžnicamente cuando la funciÃşn de inicializaciÃşn lo requiere.
- El campo function se usa para llamar a un mÃľtodo de un contrato inteligente.

Devuelve

- txhash: Hash de la transacciÃşn.
- contract_address: Valor devuelto Ãžnicamente al implementar un contrato inteligente.

Ejemplo para una transacciÃşn corriente

```
// PeticiÃşn
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":"20000000000",
↪"gasLimit":"2000000"}'

// Resultado
{
    "result":{
```

```
        "txhash":
↪"fb5204e106168549465ea38c040df0eacaa7cbd461454621867eb5abba92b4a5
↪",
        "contract_address":""
    }
}
```

Ejemplo para una implementaciÃşn de contrato inteligente

```
// PeticiÃşn
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":"20000000000","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":""}}'

// Resultado
{
    "result":{
        "txhash":
↪"3a69e23903a74a3a56dfc2bfbae1ed51f69debd487e2a8dea58ae9506f572f73
↪",
        "contract_address":"n21Y7arNbUfLGL59xgnA4ouinNxyvz773NW"
    }
}
```

SignHash

Permite firmar el hash de un mensaje.

ParÃąmetros

- `address`: DirecciÃşn del emisor.
- `hash`: Hash del mensaje, utilizando el algoritmo sha3-256.
- `alg`: Algoritmo a emplear para la firma.

Devuelve

- `data`: Datos de la transacciÃşn firmada.

Ejemplo de una transacciÃşn corriente

```
// PeticiÃşn
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}'

// Resultado
{
    "result":{
        "data":
↪"a7HHsLRvKTNazD1QEogY+Fre8KmBIyK+1Na4zv0Z72puFVkY9uZD6nGixGx/
↪6s1x6Baq7etGwlDNxVvHsoGWbAA="
    }
}
```

StartPprof

Inicia pprof.

ParÃąmetros

- `listen`: IP y puerto a monitorear.

Devuelve

- `result`: Boolean que indica si pprof se iniciÃş correctamente.

Ejemplo HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X POST http://
↪localhost:8685/v1/admin/pprof -d '{"listen":"0.0.0.0:1234"}'

// Resultado
{
    "result":{
        "result":true
    }
}
```

GetConfig

Devuelve los parÃąmetros de configuraciÃşn en uso por la consola Neb.

ParÃąmetros

Ninguno.

Devuelve

- `config`: Matriz con los parÃąmetros de configuraciÃşn.

Ejemplo HTTP

```
// PeticiÃşn
curl -i -H 'Content-Type: application/json' -X GET http://
↪localhost:8685/v1/admin/getConfig

// Resultado
{
    "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"
```

```
            }
        }
    }
}
```

Consola REPL

Nebulas brinda una consola Javascript interactiva, capaz de invocar todas las funciones de la API y de administrar mÃľtodos RPC. Se conecta al nodo local por defecto, sin que sea necesario especificar el host.

Iniciar la consola

Utilice el comando:

```
./neb console
```

En caso de no especificar ningÃžn archivo de configuraciÃşn, el sistema de arranque leerÃą el archivo `conf/default/config.conf`. Si ese archivo no estÃą disponible, o si es necesario especificar un archivo de configuraciÃşn distinto, debe iniciar la terminal con este comando:

```
./neb -c <config file> console
```

InteracciÃşn con la consola

La consola puede utilizar la interfaz `admin.setHost` para especificar los nodos a los que se conecta. Cuando reciÃľn se inicia la consola, o cuando no se especifica un host, la terminal interactÃža por defecto con el nodo local. **De esa manera, es necesario iniciar un nodo local antes de iniciar la consola.**

```
> admin.setHost("https://testnet.nebulas.io")
```

Consejos

La *testnet* sÃşlo iniciarÃą la interfaz RPC de la API, de modo que sÃşlo estarÃąn disponibles los esquemas API.

Uso de la consola

Tenemos dos esquemas, API y admin, con los cuales acceder a los comandos de la consola. Los usuarios pueden acelerar el ingreso de comandos usando la tecla `TAB`, tal como en una shell de linux.

```
> 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
```

Algunos mÃľtodos de carÃącter administrativo requieren el ingreso de una contraseÃśa. El usuario puede ingresar la contraseÃśa en la misma lÃ■nea de comandos o bien aguardar a que el proceso lo requiera. **Se recomienda esperar a que el comando lo solicite, ya que de este modo no serÃą visible en la lÃ■nea de comandos.**

Ingresar la contraseÃśa directamente:

```
> admin.unlockAccount("n1UWZa8yuvRgePRPgp8a2jX4J9UwGXfHp6i",
↪"passphrase")
{
    "result": {
        "result": true
    }
}
```

Esperar a que el proceso la solicite:

```
> admin.unlockAccount("n1UWZa8yuvRgePRPgp8a2jX4J9UwGXfHp6i")
Unlock account n1UWZa8yuvRgePRPgp8a2jX4J9UwGXfHp6i
Passphrase:
{
    "result": {
        "result": true
    }
}
```

Interfaces que solicitan contraseÃśa:

```
admin.newAccount
admin.unlockAccount
admin.signHash
```

```
admin.signTransactionWithPassphrase
admin.sendTransactionWithPassphrase
```

Los parÃạmetros de la lÃ■nea de comandos son consistentes con los parÃạmetros de la interfaz RPC.

VÃľase:

- NEB RPC
- NEB RPC_Admin

Salir de la consola

Para salir de la consola, sÃşlo basta con presionar `ctrl-C` o tipear el comando `exit`.

Archivos de configuraciÃşn

Existen cuatro tipos de archivos de configuraciÃşn en Nebulas:

- Nodos normales.
- Nodos para minerÃ■a: el archivo tiene mÃạs entradas que el destinado a nodos normales.
- Super nodo: algunos lÃ■mites de conexiones son mayores.
- Nodos de firmas: no sincroniza informaciÃşn con ningÃžn nodo; sÃşlo se dedica a realizar firmas y a desbloquear.

A continuaciÃşn se ofrecen los contenidos de esos cuatro tipos de archivo, a modo de ejemplo.

Nodos normales

```
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
}
```

Nodos para minerÃ■a

```
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 nodos

```
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
}
```

Nodos de firmas

```
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
}
```

Infrastructure

Network Protocol

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

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

Protocol

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

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

- Magic Number: 32 bits (4 chars)
  - The protocol‘s magic number, a numerical constant or text value used to identify the protocol.
  - Default: 0x4e, 0x45, 0x42, 0x31
- Chain ID: 32 bits
  - The Chain ID is used to distinguish the test network from the main network.
- Reserved: 24 bits

  - reserved field.
  - The first bit indicates whether the network message is compressed.
  - compressed: {0x80, 0x0, 0x0}; uncompressed: {0x0, 0x0, 0x0}
- Version: 8 bits
  - The version of the Message Name.
- Message Name: 96 bits (12 chars)
  - The identification or the name of the Message.
- Data Length: 32 bits
  - The total length of the Data.
- Data Checksum: 32 bits
  - The CRC32 checksum of the Data.
- Header Checksum: 32 bits
  - The CRC32 checksum of the fields from Magic Number to Data Checksum, totally 256 bits.
- Data: variable length, max 512M.
  - The message data.

We always use Big-Endian on the message protocol.

Handshaking Messages

- Hello

the handshaking message when a peer connects to another.

```
version: 0x1

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

- OK

the response message for handshaking.

```
version: 0x1

data: struct {
    string node_id // the node id, generated by underlying libp2p.
```

```
    string node_version // the client version, x.y.z schema, eg. 0.
↪1.0.
}
```

- Bye

the message to close the connection.

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

Networking Messages

- NetSyncRoutes

request peers to sync route tables.

```
version: 0x1
```

- NetRoutes

contains the local route tables.

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

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

Nebulas Messages

TBD.

Crypto Design Doc

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

Hash

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

Keystore

The Nebulas Keystore is designed to manage userâĂŹs keys.

Key

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

Provider

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

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

Signature

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

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

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

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

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

NVM - Nebulas Virtual Machine

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

go-nebulas now support two kinds of Virtual Machines:

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

Nebulas V8 Engine

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

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

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

```
"use strict";

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

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

```
        if (deposit == null) {
            return 0;
        }
        if (Blockchain.block.height < deposit.height) {
            return 0;
        }
        var balance = new BigNumber(deposit.balance);
        var value = new BigNumber(amount);
        if (balance.lessThan(value)) {
            return 0;
        }
        var result = Blockchain.transfer(Blockchain.transaction.
↪from, value);
        if (result > 0) {
            deposit.balance = balance.dividedBy(value).toString();
            this.bankVault.put(Blockchain.transaction.from,␣
↪deposit);
        }
        return result;
    }
};

module.exports = BankVaultContract;
```

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

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

LLVM

TBD.

permission_control_in_smart_contract

What Is Permission Control Of Smart Contract

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

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

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

Owner Permission Control

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

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

Explanation:

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

Other Permission Control

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

```
'use strict';
var Mixin = function () {};
Mixin.UNPAYABLE = function () {
   if (Blockchain.transaction.value.gt(0)) {
       return false;
   }
   return true;
};
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.UNPAYABLE : 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);
    })
```

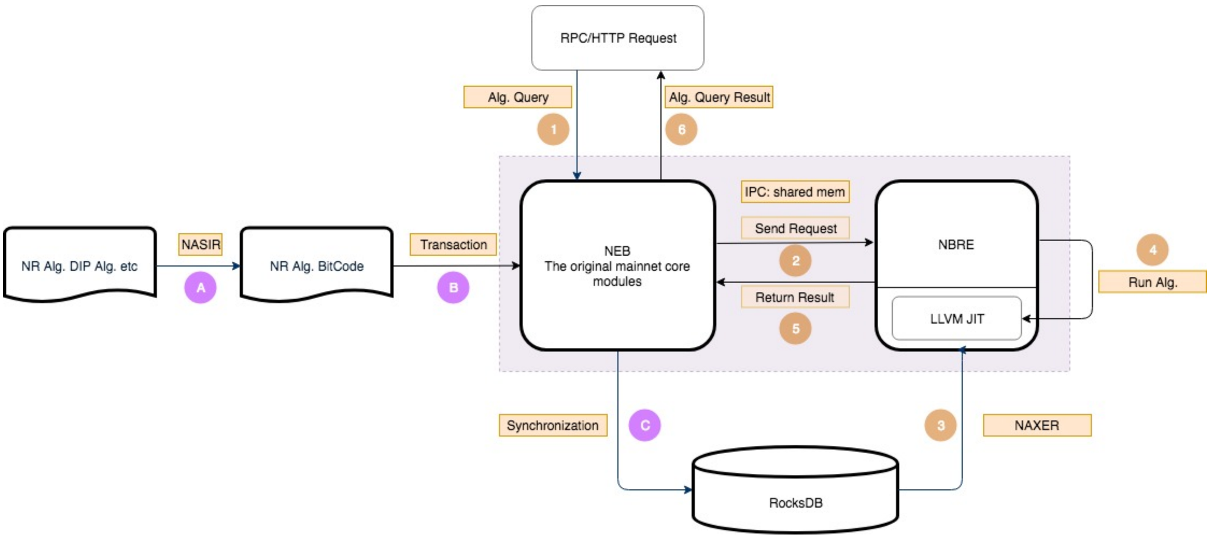
Tips:

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

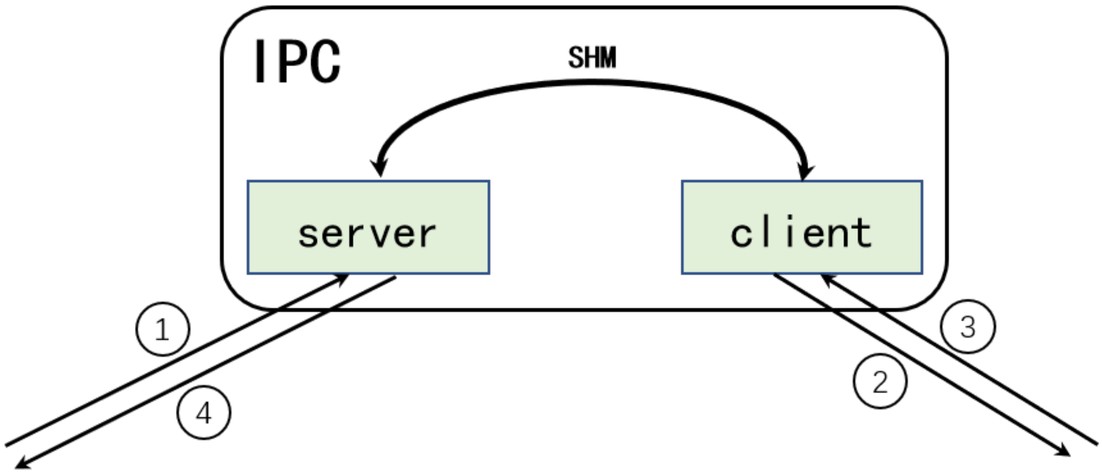
IPC is the messenger for NEB and NBRE interaction.

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

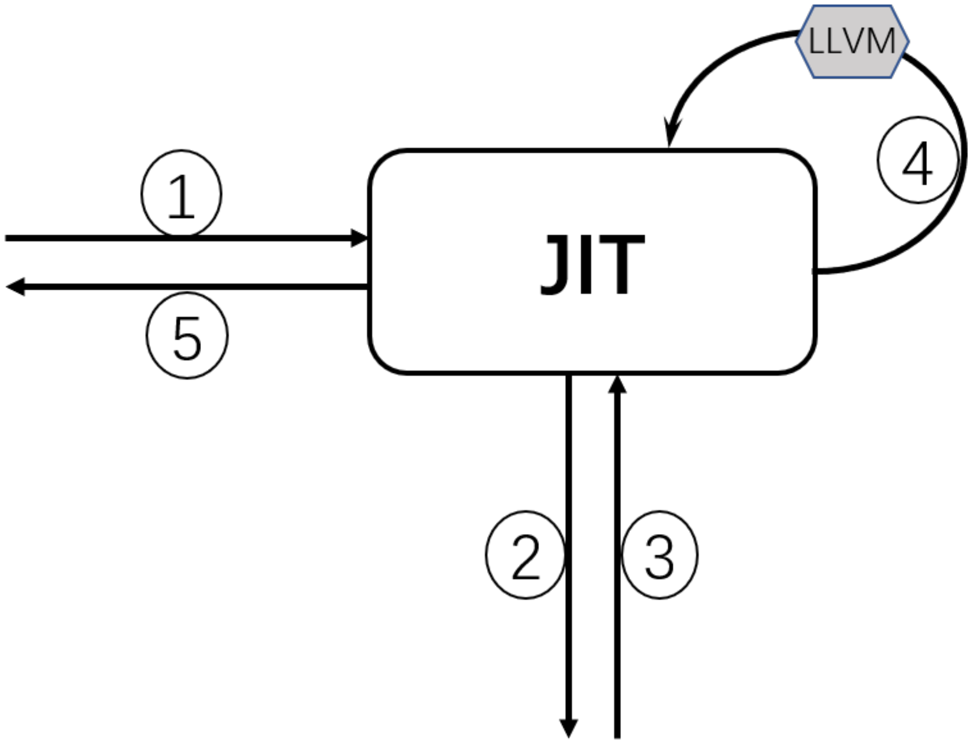
NBRE's updates are performed by adding algorithms and interface programs to the database. When a new function is updated or called, the corresponding program will be loaded into the JIT in the database.

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.

How to Develop

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
- GDBèřČèŕŢGOçĺŃåžŔ

neb-dont-generate-coredump-file

OverView

During Testing, neb may be crash, and we want to get the coredump file which could help us to find the reason. However, neb don't generate coredump file by default. We can find the crash log in /var/log/apport.log when a crash occurred:

```
"called for pid 10110, signal 11, core limit 0, dump mode 1 "
```

The coredump file is very very important, it can serve as useful debugging aids in several situations, and help us to debug quickly. Therefore we should make neb to generate coredump file.

Set the core file size

We can use `ulimit -a` command to show core file size. If it's size is zero, which means coredump file is disabled, then we should set a value for core file size. for temporarily change we can use `ulimit -c unlimited` , and for permanently change we can edit `/etc/security/limits.conf` file, it will take effect after reboot or command `sysctl -p`.

```
<domain>     <type>     <item>          <value>
 * soft     core              unlimited
```

But these ways are't work, neb still can't generate coredump file and `cat /proc/$pid/limits` always "Max core file size 0"

Why? Why? Why? It doesn't Work

1. If the setting is wrong? Just try a c++ programe build, run it and we can find that it can generate coredump.
2. Neb is started by supervisord, is it caused by supervisordïij§
3. Try to start neb without supervisord, then the neb coredump is generated!
4. Yes, the reason is supervisord, then we can google "supervisord+coredump" to solve it.

Solution

Supervisord only set RLIMIT_NOFILE, RLIMIT_NOPROC by set_rlimits , others are seted default 0 1. modify supervisord code options.py in 1293 line

```
vim /usr/lib/python2.6/site-packages/supervisor/options.py

soft, hard = resource.getrlimit(resource.RLIMIT_CORE)
resource.setrlimit(resource.RLIMIT_CORE, (-1, hard))
```

1. restart supervisord and it works .

Other seetings

You can also change the name and path of coredump file by changing file `/proc/sys/kernel/core_pattern`:

```
echo "/neb/app/core-%e-%p-%t" > /proc/sys/kernel/core_pattern

%p: pid
%: '%' is dropped
%%: output one '%'
%u: uid
%g: gid
%s: signal number
%t: UNIX time of dump
%h: hostname
%e: executable filename
%: both are dropped
```

References

- supervisord coredump
- core_pattern

Crash Reporter in Nebulas

In this doc, we introduce the crash reporter in Nebulas, which is used to collect crash reports in Nebulas and send it back to Nebulas Team, so the whole community can help improving the quality of Nebulas.

Overview

We, the Nebulas Team and the Nebulas community, always try our best to ensure the stability of Nebulas, since people put their faith and properties on it. That means critical bugs are unacceptable, and we are aware of that. However, we can't blindly think Nebulas is stable enough or there won't be any bugs. Thus, we have plan B, the crash reporter, to collect crash report and send it back to Nebulas community. We hope the whole community can leverage the crash reports and keep improving Nebulas.

Using crash reporter is a very common practice. For example, Microsoft Windows includes a crash reporting service called Windows Error Reporting that prompts users to send crash reports to Microsoft for online analysis. The information goes to a central database run by Microsoft. Apple also involves a standard crash reporter in macOS, named Crash Reporter. The Crash Reporter can send the crash logs to Apple Inc, for their engineers to review. Open-source community also have their own crash reporter, like Bug Buddy for Gnome, Crashpad for Chrome, Talkback for Mozilla, and etc.

In Nebulas, the crash reporter just works like the other crash reporters. It's aware of the crash, collects necessary information about the crash, and sends it back the Nebulas server. The server is hosted by Nebulas, and accessible for the whole community.

As a opensource, decentralized platform, we are aware of that the crash reporter may violate some users' privacy concern. Thus, we remove all private information in the crash report, like the user name, user id, user's home path and IP address. Furthermore, the crash reporter is optional and users may choose close it if users still have some concerns.

How to use it

To enable or disable the crash reporter, you need to look into the configuration file, `config.conf`, and change `enable_crash_reporter` to `true` to enable it, while `false` to disable it.

How it works

In this section, we would like to share some tech details. If you are not interested in the details, you can ignore this section.

The crash reporter is actually a daemon process, which is started by `neb`. When starting the crash reporter, `neb` will tell it the process id (pid) of `neb` process, and the crash file path. For the crash reporter, it will periodically check if the `neb` process and the crash file exists. At the time it finds the crash file, it will eliminate the private information and send it back to Nebulas.

Currently, the crash report is generated by the `stderr` output from `neb`. We'd like the work with the whole community to collect detailed information in the future.

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£Źçŕ Ğç§■æŰĞå§žäžŐMac OSX åŠŇ Ubuntuçşżçż§ïijŇçőĂå■ŢäżŃçż■åęĆä¡ŢèřČèŕŢGo-Nebulaséąźç ŻőïijŇäÿżèęĄäżŃçż■äÿĽ'çğ■æŰźæşŢèřČèŕŢïijŽdlvåŚ¡äżď'èąŇèřČèŕŢïijŇGogland IDEèřČèŕŢïijŇäżĕåŔŁVisual Studio CodeèřČèŕŢãĂĆ

èřČèŕŢåŹÍDelveåőĽ'èčĚ

åIJÍ Mac OSX äÿŁåőĽ'èčĚDelve

GoogleåőŸæŰźäÿžgolangçŽĎèřČèŕŢä¿Ńå■ŘçŤĺgdbïijŇä¡ĘæŸŕdelveæŸŕæŻť åŘĹéĂĆçŽĎèřČèŕŢåŹĺïij NebulasãĂĆæŹőéĂŽçŽĎgoéąźçŻőæŸŕåŔŕäżěçŽĎïijŇåĚůä¡Şä¡ŞçŐřåřśæŸŕèřČèŕŢGo-NebulaséąźçŻőæŮűïijŇæŮ■çĆźæŮăæşŢåĄIJä¡ŔïijŇäijŽæřÿè£IJhangä¡ŔãĂĆæĹŚäżňå£Ěéążäž ŐgithubäÿŁäÿ binaryïijŇæ■ěéłď'åęĆäÿŃïijŽ

åĚĹçŤĺHomebrewåőĽ'èčĚæIJL'bugçŽĎDelveïijŽ

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

åőĽ'èčĚæ■ď'æIJL'éŮőéćŸçŽĎDelveïijŇåĚűåőđåřśæŸŕäÿžäžĘèőľ'åőČåÿőæĹŚäżňåIJÍMacæIJžåŹĺäÿŁç■ certèŕĄäźęãĂĆåęĆæđIJä¡ăèĞłåůśæĎ£æĎŔçźĄçŘŘçŽĎæĽ'ŃåŁĺåĹŻåżžèŕĄäźęïijŇäź§åŔŕäżěäÿ■çŤĺåőĽ'èčĚ a self-signed certificateãĂŚãĂĆ çňňäžŇæİąrmåŚ¡äżď'æŸŕäÿžäžĘåĹăéŹď'è£Źäÿłæ IJL'éŮőéćŸçŽĎdlv binaryïijŇæĹŚäżňéIJĂèęĄäżŐæžŘçăĄçijŰèŕŚåĞžäÿĂäÿłæ■čçąőçŽĎçĽĹæIJňïijŇåźűäÿŤåĹľ çŤĺHomebewä äÿŃè¡¡æžŘäżčçăĄ

```
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
```

åĹŻåżžäÿĂäÿłäÿť æŮűæŰĞäżűåď'źïijŇäżŐgithubäÿŃè¡¡äżčçăĄãĂĆæşĺæĎŔæŰĞäżűåď'źäÿ■æăĞæşĺçž not foundãĂĆåĚűåőČéČĺåĹĘèŕůæăźæ■őèĞłåůśæIJžåŹĺçŐřåćČèő¿ç¡őãĂĆ

çijŰèŕŚ

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

åžŤèŕěäijŽåĞžçŐřåęĆäÿŃæŔŘçď'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¡çŽĎdlvæŃůèť İè£Ż/usr/local/ debuggerãĂĆè¿ŞåĚěåŚ¡äżď'dlv versionïijŇåęĆæđIJèČ¡æ■čåÿÿè£ŘèąŇïijŇæŸ¿çď'žçĽĹæIJňåŔůïijŇèŕť æŸŐ

åIJÍ Ubuntu äÿŁåőĽ'èčĚDelve

åŕźäžŐUbuntuçşżçž§ïijŇåŔŕäžěçŻť æŐěä¡£çŤĺäÿŃéĺćçŽĎæŇĞäżď åőĽèčĚDelveïijŽ

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

äÿŃè¡¡Go-NebulasåůěçĺŃäżčçăĄ

```
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
```

åĹŻåżžäÿĂäÿłäÿť æŮűæŰĞäżűåď źïijŇäżŐgithubäÿŃè¡¡äżčçăĄãĂĆæşĺæĎŔæŰĞäżűåď źäÿ■æăĞæşĺçž

DelveåŚ¡äżďèąŇèřČèŕŢ åęĆæđIJä¡ääžěåĽ■çŤĺgdbèřČèŕŢè£ĞCçĺŃåžŔïijŇåŕźdlvåŚ¡äżďèąŇèřČèŕŢçŽ
è£ŹéĞŇåŔłäžŃçż■debugéČĺåĹĘãĂĆ

è¿ŞåĚěåęĆäÿŃåŚ¡äżď è£ŻåĚěèřČèŕŢ

```
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£Ŕèąňæŭåèŕŕç ŽĎèŕÏïijŇäijŽè£ŻåĚědebug sessionïijŽ

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

æĹŚäżňæĽ ŞçőŮåĪĺnebçŽĎåĞ¡æŢřåĚěåŔčèő¿ç¡őæŰ■çĆźïijŇè¿ŞåĚěåŚ¡äżď

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

dlvèřČèŕŢåŹĺæŔŘçďžäżčçăĄåřĘåĪĺcmd/neb/main.goçŽĎèąŇåŔů80èąŇåĄIJä¡ŔïijŇæşĺæĎŔè£ŹæŮűn

```
(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:
```

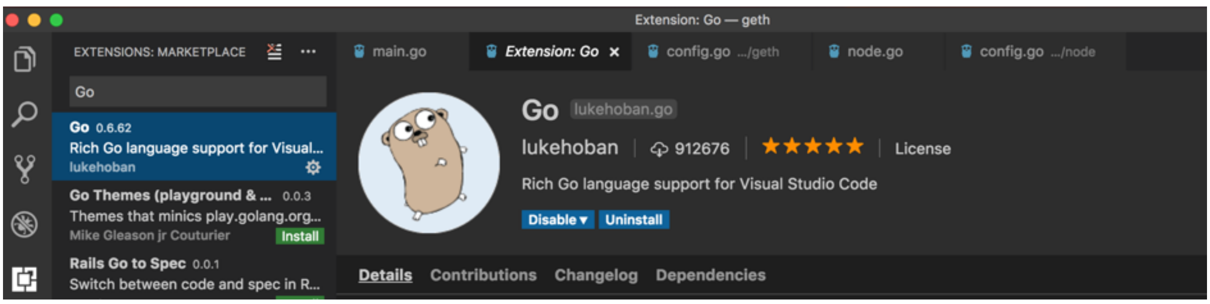
æ§ěçIJŃåŘŸéĞŔïijŇåŘŕçŤÍprintåŚ¡äżďïijŽ

```
(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),
```

æŻť 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¡ŕåĚňåŔÿåŔŚåÿČçŽĎèůĺåźşåŔřäżčçăĄçijŰè¿ŚåůěåĚůïijŇäÿŃè¡¡åIJřåİĂïijŽ //code.visualstudio.com/Download VS CodeéIJĂèęĄåőĽèčĚGoæŔŠäżű



æĽ Şåij ĂæŰĞäżűåď ź/Users/xxx/workspace/blockchain/src/github.com/nebulasio/go-nebulas/ïijŇåIJĺ.vscodeæŰĞäżűåď źäÿŃåĹŻåżžäÿď äÿłæŰĞäżűsettings.jsonåŠŇlaunch.jsonãĂĆ settings.jsonæŰĞäżűåĘĚåőźïijŽ

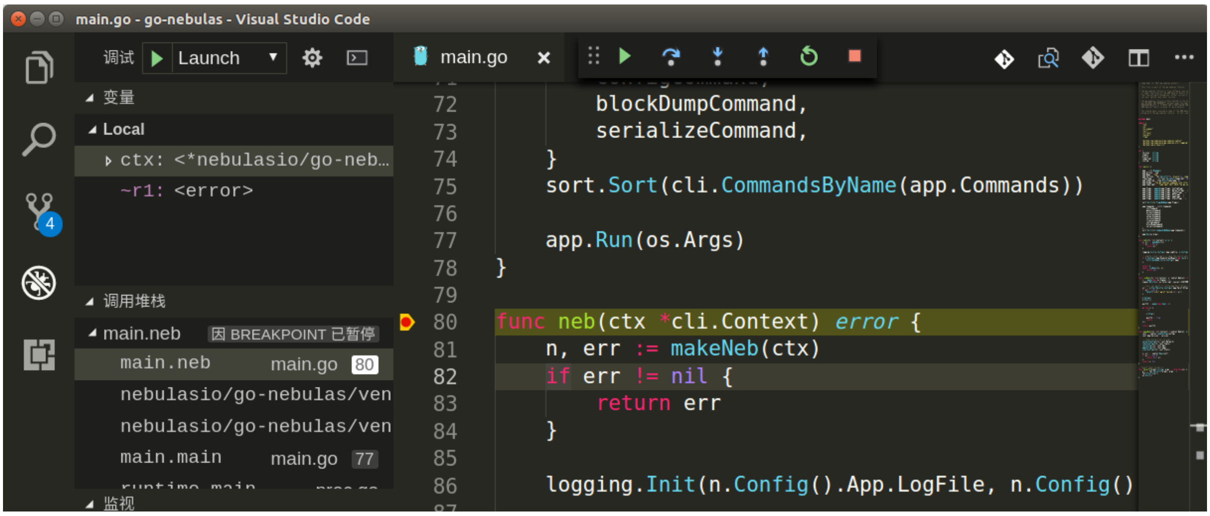
```
// Place your settings in this file to overwrite default and user␣
↪settings.
{
    "go.gopath": "/Users/xxx/workspace/blockchain/",
    "go.formatOnSave": true,
    "go.gocodeAutoBuild": false,
    "go.toolsGopath": "/Users/xxx/workspace/gotools",
    "explorer.openEditors.visible": 0,
}
```

go.toolsGopathæŸŕanalysis toolsåőĽèčĚçŽĎåIJřåĬĂïijŇåŔŕäżěæŇĞåőŽäÿžäżżä¡ŢçŻőå¡ŢïijŇè£ŹäžŻan toolsåŔŕäżěä¿ŻåĚűåőČworkspaceåĚśäžńãĂĆ

launch.jsonæŰĞäżűåĘĚåőźïijŽ

```
{
    "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
        }
    ]
}
```

åIJĺcmd/neb/main.goïijŇnebåĞ¡æŢřäÿ■èő¿ç¡őæŰ■çĆźïijŇF5è£Řè¡ŇïijŇGo-Nebulaséą́źçŻőäijŽè£Žè¡Ňçij́Űèŕ́Śè£Řè¡ŇïijŇåĄĲåĲĺæŰ■çĆźïijŽ



çĎűåŘŐïijŇåřśåŔŕäżěåijĂå£ČçŽĎåŘŕåŁÍNebulasäżčçăĄèřČèŕŢäźŃæŮĚïijĄ

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.

Downloads

Bleeding edge code can be cloned from the branch of their git repositories:

- Mainnet
- Explorer
- Web Wallet
- neb.js

Mainnet

Nebulas mainnet Eeagle Nebulas launched on Mar 30, 2018. It's a basic public chain. There are two features:

- Supports javascript development
- Over 2000 TPS.

Nebulas NOVA launched in the end of 2018. There are three features:

- Nebulas Rank: measure the value of on-chain data
- Nebulas Blockchain Runtime Environment: instant upgrade the core protocols immediately
- Developer Incentive Protocol: provide native on-chain incentive for developers

Click here to learn about Nebulas NOVA. Some articles:

- Nebulas NOVA, To Discover Data Value In the Blockchain World, [Youtube]
- 6 Minutes Learning Nebulas NOVA with 92k Lines of Code by Joel Wang [Youtube]

The third important version will be launch in 2020 with PoD consensus mechanism. Click here to learn about the PoD Node Strategy.

Click here to learn how to join the mainnet.

Testnet

A functional equivalent Nebulas Testnet is available now,allowing developers to interact with Nebulas freely. View: How to join the testnet.

Roadmap

Nebulas releases are here and roadmap are here.

1.4.4 Node Strategy

Nebulas PoD Node Decentralization Strategy - Based on the Proof of Devotion (PoD) Mechanism

V1.0 by Nebulas Foundation, PDF version

Nebulas began it journey with the Vision of "**Let everyone get values from decentralized collaboration fairly.**" With the continued evolution of the âĂIJ**Autonomous Metanet**âĂİ[1], Nebulas is proceeding to itâĂŹs ultimate goal.

[1] Autonomous Metanet: An open collaboration system based on blockchain technology, which is oriented around complex data and interaction.

At the core of NebulasâĂŹ PoD Node Decentralization Strategy is the **Proof of Devotion (PoD)**[2] Mechanism. This idea behind Proof of Devotion is to provide a measurable value of all users based on the size of their contribution to the ecosystem which includes pledging, consensus and governance mechanisms. With PoD, we plan to not just decentralize NebulasâĂŹ blockchain nodes but to also decentralize community governance via the formation of a representative system and government committees.

Nebulas is building a new **Decentralized Autonomous Organization (DAO)**[3] for complex data networks that will fully embrace community, decentralization and autonomy on a contribution measured basis.

Learn more about the Node Strategy and PoD mechanism:

1. PoD Overview

Proof of Devotion (PoD) Mechanism Overview

1.1 Design Objectives

In order to build a sustainable and beneficial public chain, it is necessary to take into account both the speed and irreversibility of the consensus mechanism as well as the fairness of governance.

At present, we face new application scenarios including simple data interactions to complex, multi-level, on-chain functions. This diverse environment is spawning the creation of new user roles as well as significantly increasing the complexity of the system. Communication scenarios have evolved from in person collaboration to collaboration that ecompasess the world. The goal of collaboration has also changed with the end results going from the physical to the virtual world. This results in time spans for collaborative projects becoming longer and more flexible.*[1]*

To ensure a fair governance system within these new scenarios, a new approach to collaboration is required. Traditional centralized governance cannot cope with these new and complex scenarios that we face daily in our technologically evolving world. In this new world filled with complex data interaction patterns and expanding user roles, centralized single evaluation options are difficult to be adaptable and comprehensive leading to considerable limitations.

[2] Proof of Devotion (PoD): A consensus mechanism built on the basis of the size of community contributions. This includes both consensus and governance mechanisms. The establishment of consensus committees through community contributors to achieve nebulasâĂŹ blockchain nodes decentralization; Participation in community governance through the representation of governance committees.

[3] Decentralized Autonomous Organization (DAO): An organization that is represented by public and transparent computer code. Financial transaction records and procedural rules of a distributed autonomous organization are stored within the blockchain.

Existing decentralized collaboration method do not take into account the new distribution of benefits caused by the existence of expanded user roles. As a result, there is an uneven distribution of benefits leading to slow development and eventually, an unsustainable ecosystem.

We must protect the interests of all community members so that value comes from the depth of Nebulas‘ ecosystem which in turn follows our core beliefs. Under the premise of ensuring efficiency and irreversibility first, we have designed PoD to pursue fairness from the perspective of contribution and to protect the interests of the community.

1.2 Composition

NebulasâĂŹ Proof of Devotion (PoD) can provide a simple overview of mechanisms built on the basis and magnitude of community contributions which include both consensus mechanisms and governance mechanisms. See Figure 1.1.

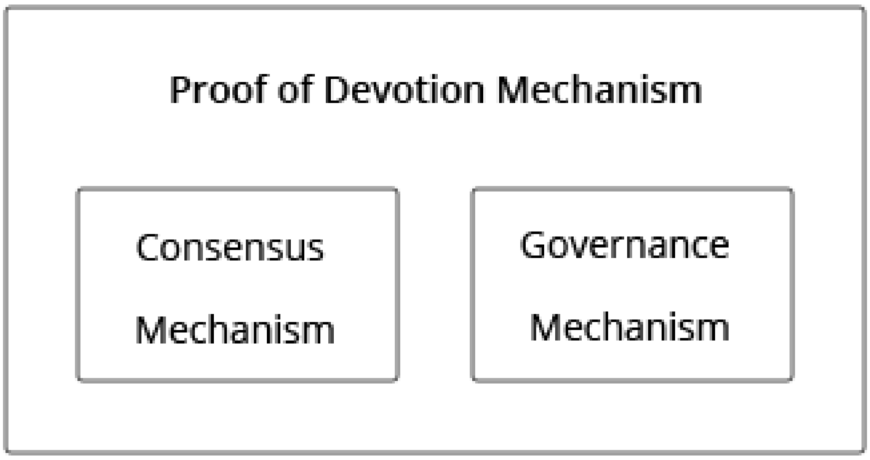


Figure 1.1 PoD Composition

The composition of the PoD mechanism will involve two executive committees split into consensus and governance.

- The consensus mechanism shall be implemented by the Consensus Committee. The consensus committee is selected from all the available nodes via a comprehensive ranking algorithm.
- The governance mechanism shall be implemented by the Governance Committee. The governance committee is composed of the most dedicated contributors of the Consensus Committee.

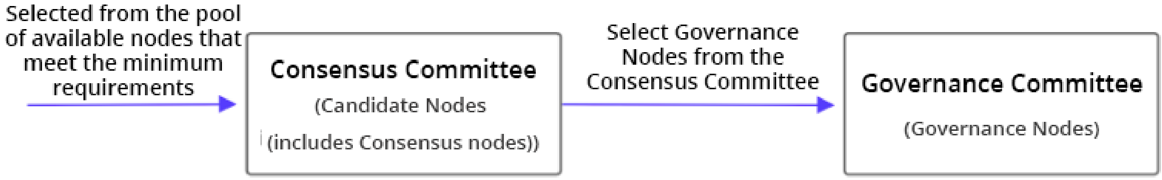


Figure 1.2 PoD Executive Committee

1.3 Incentive Allocation

Since the launch of the Nebulas mainnet on March 31, 2018, DPoS*[2]* has been used as the interim consensus mechanism until the release of PoD. The DPoS consensus mechanism generates 8,219.1744 NAS in revenue per day; generating 2,999,941 NAS per year.

This collected revenue will be used exclusively for the Nebulas PoD Node Decentralization Strategy.

The incentive ratio of the two PoD Executive Committee (consensus and governance) will be about 5:1. Which equates to:

- The total incentive amount for the consensus mechanism per year will be: 2,499,951 NAS
- The total incentive amount for the governance mechanism per year will be: 499,999 NAS

The incentive for the consensus mechanism will be evenly divided by the consensus nodes who have generated blocks during the active polling cycle (selection) of the consensus committee. Any candidate nodes that have not been selected during the polling cycle (selection) will not receive any incentive during this period.

The incentive for the governance mechanism will be evenly divided by the governance nodes who has participated in all voting proposals during the governance cycle. Any governance nodes that do not participate in **ALL** voting proposals during the governance cycle will not receive any governance incentive during this period.

1.4 Contribution Measurement: NAX

The measurement for the weight of contribution to the Nebulas ecosystem is the NAX*[3]* Smart Asset. As a smart asset, NAX can only be obtained by **decentralized staking (dStaking)***[4]* the NAS asset. As per the ***Nebulas NAX White Paper*** (Github, PDF), NAX adopts a dynamic distribution model where the daily total issuance quantity is related to the pledge rate of the entire Nebulas ecosystem; the number of NAX obtained by an address is related to the quantity of NAS pledged and the age/duration of the pledge (the longer, the better), which can be considered a measure of the contribution of that address to the community and ecosystem. Therefore, NAX can be considered effective proof of those who contribute to the Nebulas ecosystem.

The Go Nebulas community collaboration platform will also utilize NAX as an ecosystem contribution incentive to encourage community members to continue to build communities.

---

[1] Orange Paper: Nebulas Governance

[2] **Delegated Proof of Stake Consensus (DPoS):** Delegates are chosen by stakeholder votes, and delegates then decide on the issue of consensus in a democratic way. This includes but is not limited to: All network parameters, cost estimates, block intervals, transaction size, etc.

[3] **NAX:** This smart asset is generated by decentralized pledging and is the first token on nextDAO. Users on the Nebulas blockchain can obtain NAX by pledging NAS. NAX adopts

dynamic distribution strategy where the actual issuance quantity is related to the global pledge rate, the amount of NAS pledged individually and the age of the pledge.

[4] **dStaking Decentralized Pledge:** Unlike traditional pledges (staking) that requires the transfer of assets to smart contracts, decentralized pledges record the user's pledge while the assets remain at the user's personal address.

2. Consensus

This chapter will introduce the Consensus Mechanism of PoD mechanism, follow here:

The consensus mechanism utilizes smart contract management which is primarily comprised of node selection rules and the consensus algorithm. This smart contract jointly completes the block generation and ensures the normal operation of the mainnet.

The average block time on the Nebulas mainnet is 15 seconds. During each polling period, the 21 selected consensus nodes take turns generating 10 blocks each. As a result, one polling cycle is 210 blocks which takes about 52.5 minutes. The consensus mechanism execution process during each polling cycle is shown in the following figure:

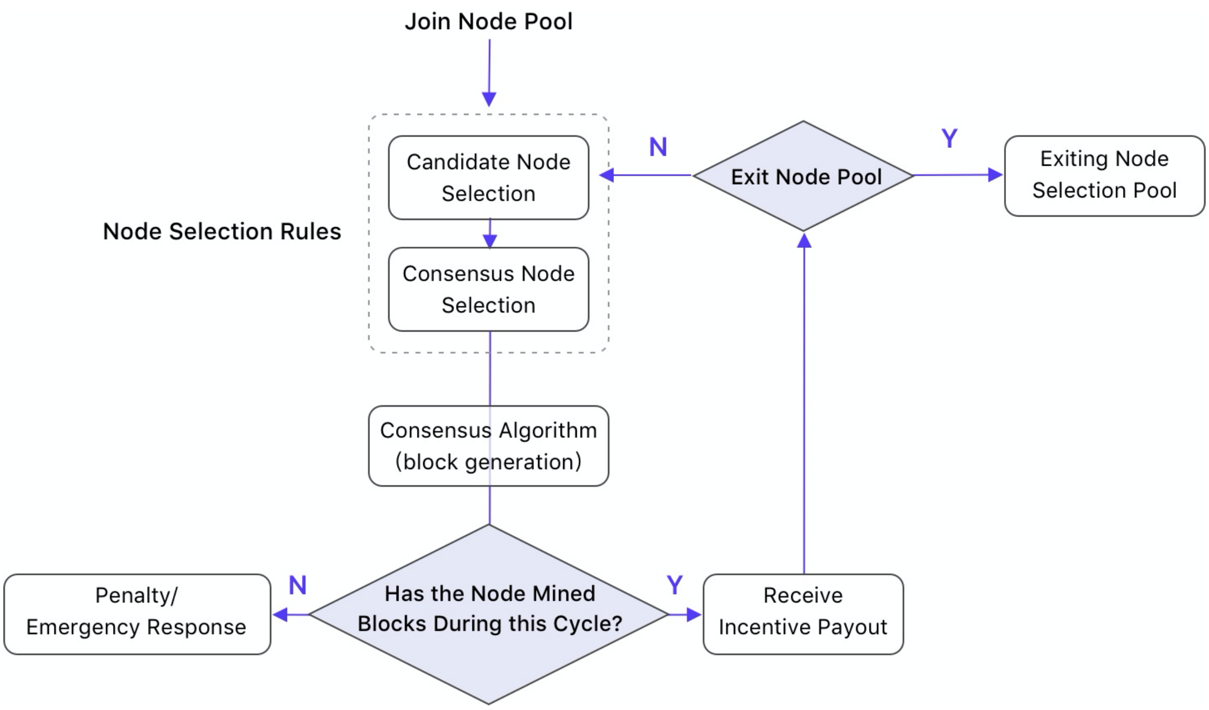


Figure 2.1 The consensus mechanism execution process for each polling cycle

2.1 Minimum Requirements for Node Selection

Any individual or organization can apply to become a consensus node and must meet all of the following eligibility requirements to participate in the candidate selection process:

- The server meets the minimum requirements (see Appendix A - recommended hardware configuration);
- The server is guaranteed to be in operation;
- The node pledge (vote) is not less than 100,000 NAX;
- Pledge of 20,000 NAS as deposit;
- No record of severe level abuse or manipulation on the network (see *2.5.1 penalties*)

2.2 Node Selection Rules

The node selection rule consists of two steps:

1. **Candidate node selection:** During each polling cycle, among all nodes that meet the minimum selection requirements, a total of 51 nodes are selected according to the comprehensive candidate node ranking algorithm via smart contract;
2. **Consensus node selection:** During each polling cycle, the algorithm selects 21 consensus nodes which are selected in a consistent method and best represents the user‘s rights and interests in a group of candidate nodes which is based on the consensus node selection algorithm via smart contract. The consensus nodes are responsible for block

generation and can obtain consensus incentives as long as they participate in the process of block generation (online, creating blocks, not manipulating the network, etcâĂę).

The candidate node and the consensus node together constitute the consensus committee. The selection process is shown in the following figure:

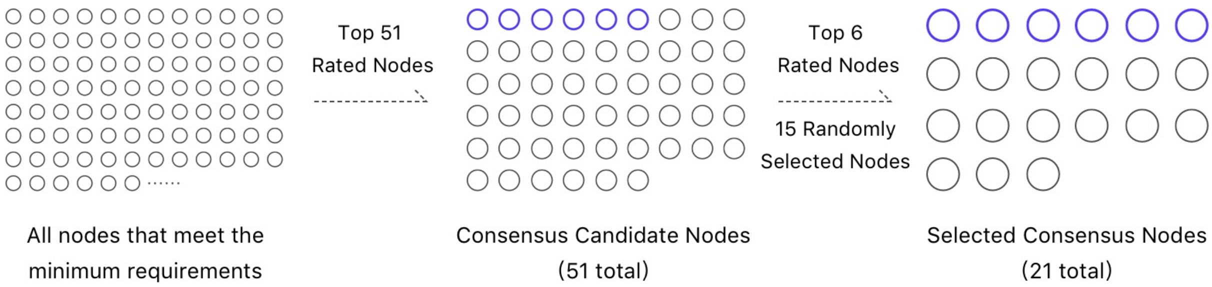


Figure 2.2 Node Selection Process

2.2.1 Candidate Node Comprehensive Ranking Algorithm

Under the premise of meeting the minimum requirements of becoming a candidate node (*2.1 Minimum requirements for node selection*), all nodes are ranked by the comprehensive candidate node ranking algorithm; the top 51 nodes selected via the ranking algorithm will be selected as candidate nodes.

The candidate node ranking algorithm references to two primary factors: NAX poll number V(i) and block stabilization index S(i). The final candidate node ranking index âĂIJR(i)âĂİ is:

R(i) = V(i) ÃŮ S(i)

Assuming that S(i) is the same for multiple nodes and the NAX vote V(i) is also the same for multiple nodes, the first node to reach the required NAX vote V(i) is selected.

2.2.1.1 NAX Votes

Number of NAX votes V(i): All community members and node operators can pledge NAX to further support the activation of a node which helps the node improve their overall ranking.

2.2.1.2 Block Generation Stability Index

Block generation stability index S(i): This rating is determined by the ratio of successful block generation when itâĂŹs chosen as a consensus node. If this node has not yet had the chance to be a consensus node, initial value of S(i) is 0.8. During each polling cycle, a candidate node has three possible values:

1. Not participating in block generation;
2. Successful/accepted block generation;

3. Invalid block generation.

A. Not participating in block generation

The S(i+1) of the following cycle of candidate nodes that have not participated in block generation is:

S(i+1) = S(i) + 0.01

If the node continues to not participate in the generation of blocks, the S(i) rating is reduced to the initial value S=0.8 at its lowest level.

B. Successful block generation

Consensus nodes need to generate 10 blocks per polling cycle (polling cycles consist of 210 blocks). If a node generates 10 blocks successfully during the cycle, the S(i+1) of the next cycle is:

S(i+1) = S(i) + 0.1 (S<=1)

S(i) is gradually increased to the maximum level of 1. In general, functioning nodes with stable block generation will reach the maximum level of S(i)=1.

C. Invalid block generation

If the node generates a invalid block, the S(i+1) of the next cycle is:

S(i+1) = S(i) ÃŮ (10 - C) / 10

Where C is the number of invalid blocks. The larger C, the lower S(i) value. If S(i) falls to the K threshold (K initial value is 0.5), the consensus node cannot be selected as a candidate node for the next 20 polling cycles, as detailed in *2.5.1 Penalties*.

2.2.2 Consensus Node Selection Algorithm

Consensus nodes are selected from the 51 candidate nodes that are initially selected during the polling cycle. The selection method is as follows:

1. The top 6 candidate nodes are selected automatically as per their score (detailed above).
2. The remaining 15 candidate nodes are selected from the candidate pool containing the 45 additional nodes according to the following formula:

   RConsensus = (R(i) / Sum(R)) ÃŮ Random()

   The formula explanation is as follows:

   RConsensusïijŽConsensus node ranking index.

   R(i)ïijŽCandidate nodes ranking index. R(i) is the derived score of two primary factors: NAX poll number V(i) and block stabilization index S(i); as a result, R(i) is treated as the community support rate of the node and its contribution of historical blocks generation.

   Sum(R): The sum score of 51 candidate nodes ranking index; as a result, R(i)/Sum(R) can be treated as an individual node contribution ratio within the 51 candidate nodes.

   Random()ïijŽA random probability.

2.3 Consensus Algorithm

The consensus algorithm is based on the well understood and mature DPoS consensus mechanism where the block generation of the following polling cycle is scheduled to be produced by the nodes within the consensus committee; the selected 21 consensus nodes take turns to generate blocks. After the polling cycle is complete, the next selected 21 consensus nodes take turns to generate blocks in the following cycle.

Byzantine fault tolerant BFT*[1]* operation is used to ensure the consistency and stability of the blockchain and the PoD mechanism.

2.3.1 Block Generation Order

The order of block generation of the 21 consensus nodes is randomly selected via a Verifiable Random Function (VRF*[2]*) in one polling cycle. The consensus nodes and the order responsible for the block generation remain unchanged during each polling cycle.

2.3.2 Packaging of Generated Blocks

Consensus nodes package transactions that are contained within the transaction cache pool when it is time to generate a new block. The specific methods is as follows:

1. Consensus nodes package blocks strictly according to the predefined order and duration of the polling cycle.
2. Package as many transactions as possible within packing time-frame;
3. Transactions with a higher overall GasPrice (when compared to other pending transactions) will take priority;
4. A non-verifiable transaction is disregarded when it‘s execution fails.

2.3.3 On-chain Confirmation

The on-chain confirmation for consensus nodes guarantees the consistency and security of the chain as well as penalizing any nodes that may harm the integrity of the blockchain. The Nebulas blockchain utilizes the following rules:

1. The longest subchain is chosen as the optimal chain.
2. The optimal chain is selected according to hash order of previous blocks if the subchains are with the same length.
3. The use of BFT operation across the network for irreversible transactions requires the confirmation from âĚŤ+1 of consensus nodes within the network;
4. The penalty mechanism should be adopted for attacks such as generating blocks when unexpected and attempting double-spends (see *2.5.1 Penalties*).

2.4 Exit Mechanism

Voting for PoD nodes is a fair and free service. All members of the community can withdraw support via NAX for a node or apply to exit the node pool at any time.

2.4.1 Withdrawal of NAX Support for a Node (votes)

All members or organizations within the community may at any time withdraw their support for a community operated node. When support/votes are withdrawn, the node operators NAX support level is immediately reduced (*2.2.1.1 NAX votes*, V(i)) and will affect their ranking in following rounds of node selection. As stated in the minimum requirements (*2.1 Minimum requirements for node selection*), if the total amount of NAX support for a node drops under 100,000 NAX, they cannot be selected as a consensus node.

Quantity of votes to withdraw: The voters may choose how much NAX to withdraw for their support. Community members or organizations can only apply to revoke their own NAX.

NAX return time-frame: Once a withdrawal request has been issued, NAX is returned to the voterâĂŹs original address after 120 polling cycles (approximately 5 days) has passed.

2.4.2 Exiting the Node Pool

All nodes can exit the pool at any time. Once the exit request has been issued, the node will immediately lose its candidacy for following cycles.

Return of NAS security deposit: All NAS deposit required for candidacy is returned in one sum (partial refund is not an option).

NAS security deposit return time-frame: Once the exit request has been issued, the security deposit is returned after 820 polling cycles (approximately 1 month) and will be returned to the original address.

Return of NAX deposit: Any NAX that has been voted/issued for a node which is exiting the pool will be returned to the corresponding address after 120 polling cycles (approximately 5 days).

2.5 Penalties and Emergency Response

2.5.1 Penalties

2.5.1.1 Block Generation Penalties

In order to maintain the security of the PoD system, the corresponding Penalties are carried out according to the situation; the more malicious act of a node, the higher the punishment.

The three block generation penalties for the consensus node are as follows:

Table 2.1: Consensus Mechanism Safety Rating Table

Medium and Severe punishment process:

1. When security issues occur, restrictions are automatically executed and will freeze the NAS that is held in collateral to the corresponding penalty.
2. During the voting phase of the next governance cycle, the governance committee votes to determine whether the node punishment is justified.
   (a) If the governance committee votes that the punishment is justified, the NAS that has been frozen will be donated to the Go Nebulas Community Collaboration Fund (See 3.2.2 Community Assets).
   (b) If the governance committee votes that the node did not cause intentional harm to the network, the block generation stability index S(i) of the node will be restored to the level prior to the punishment and the NAS will be unfrozen.

See the 3.2.3 Penalties for consensus mechanism and 3.3.3 Processing of voting results of 3.2 Governance scope.

2.5.1.2 Governance Penalties

In addition to the block generation penalties (listed above), when a consensus node is selected as a governance node, the governance node must complete all governance tasks (taking part in votes). If the governance node does not take part in the governance process for two consecutive governance cycles, it cannot be selected for the next 820 polling cycles (approximately one month). See 3.4.1 Individual governance node penalties.

2.5.2 Emergency Response

In the event of an attack on the Nebulas mainnet from a hacker or other unforeseen threats/emergencies and in order to ensure that the network can quickly respond to these attacks and reduce the harm of them, the Nebulas Foundation has reserved emergency smart contract management methods. The Nebulas Foundation can immediately blacklist the address in question and prohibit transfers from blacklisted addresses.

The entire process is open and transparent. The Nebulas Foundation will thoroughly review the incident and openly accept the supervision from the community.

---

[1] **BFT (Byzantine Fault Tolerance):** It is a fault-tolerant technique in the field of distributed computing. Byzantine fault-tolerant comes from the Byzantine Fault problem. The Byzantine Fault problem models the real world, where computers and networks can behave unpredictably due to hardware errors, network congestion or disruption, and malevolence. Byzantine fault-tolerant techniques are designed to handle real-world abnormal behavior and meet the specification requirements of the problems to be addressed.

[2] **VRF (Verifiable Random Function):** Verifiable random functions: It is an encryption scheme that maps the input to a verifiable pseudo-random output. The program was proposed by Micali (the founder of Algorand), Rabin and Vadhan in 1999. To date, VRF has been widely used in various encryption scenarios, protocols and systems.

3. Governance

This chapter will introduce the Governance Mechanism of PoD mechanism, follow here:

Nebulas focuses on the contribution of different roles to the diverse ecosystem via decentralized collaboration and the utilized governance mechanism is an important portion of PoD mechanism.

The governance mechanisms are a range of tools for community self-governmenance via the organization of community collaboration and management of community assets by the governance committee.

3.1 Governance Committee

The implementation of governance mechanisms is managed by the Governance Committee and is made up of governance nodes.

Governance cycle: One governance cycle will occur every 820 consensus node polling cycles (approximately 1 month).

Governance node selection: Governance nodes are selected from the consensus committee and the selected 51 consensus nodes where the largest number of block generators for the past 820 consensus node polling cycles are eligible to become governance nodes in the governance cycle. If there are equally qualified nodes available to become governance nodes and not enough spaces are left, the node(s) which have achieved the number of generated blocks first will be selected.

3.2 Governance Scope

3.2.1 Community Collaboration

The proposal operation of the Nebulas community is an important part of the continuation of the Autonomous Metanet. All proposals and projects of the Nebulas community are public information which are displayed and managed via the Go Nebulas collaboration platform (go.nebulas.io). All community members can put forward their own ideas, opinions and suggestions on the future development of the Nebulas via this platform. Ideas and suggestions include but are not limited to *[1]*:

1. Research and development of the Nebulas mainnet;
2. Community collaboration process optimization, governance recommendations, etcâĂę;
3. Improvement suggestions and bug reports for existing Nebulas community products;
4. Development and maintenance of community eco-products;
5. Community operations and market expansion.

For a proposals to go from idea to implementation, it will go through multiple steps including:

- proposal;
- Project establishment;
- project execution;
- project acceptance.

Each step needs to be voted for and approved by the Governance Committee. The Governance Committee has three types of voting tasks in each governance cycle:

1. **Proposal voting:** Vote on the proposals submitted from the Nebulas community and decide whether to approve the project to the next phase.
2. **Project establishment voting:** Vote on the establishment and budget of projects that have successfully passed the proposal process.
3. **Project acceptance voting:** Review and vote on projects that have been established, completed and issue funding.

The Governance Committee process is as follows:

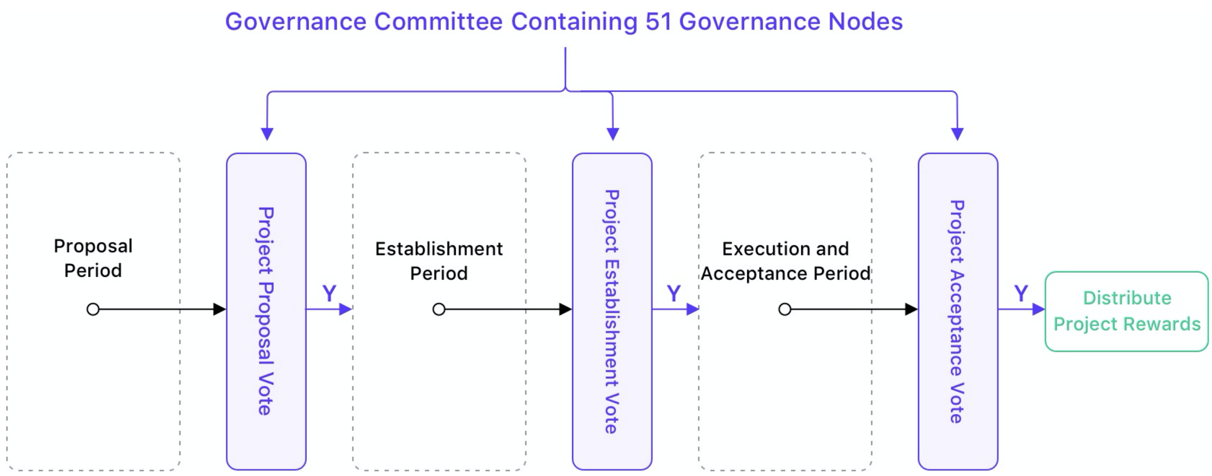


Figure 3.1 Governance Committee Voting Process

Proposal period: All community members are welcome to create and share proposals on Go Nebulas (go.nebulas.io).

Project establishment period: Projects that successfully complete the proposal period will proceed to the establishment period. All projects are separated into two categories:

1. **No budget required for this project:** For example, proposals that include the discussion of improving existing Nebulas projects. These include suggestions on adjusting the structure of the governance organization and the adjustment of the mainnet parameters. After a proposal is voted on and approved, the relevant person in charge may accelerate the implementation of this proposal.
2. **Proposals that require a budget:** This process is facilitated by the Go Nebulas Operations team, which handles project budgets and fund release. Project creators can submit budgets, project objectives, execution steps and expected duration. Creators can also apply to be the project owner or elect a community member to operate the project. Projects should be submitted in accordance with the standard template available on Go Nebulas.

Project execution and acceptance period: The execution and acceptance period is an internal operational process of Go Nebulas; the governance Committee does not directly participate in this process. The process is divided into three stages:

1. **Set budget period:** The project creator or the Go Nebulas operation team can be set as the project creator. Project creators set the reward for successful completion of the project and members of the community are welcome to participate in projects;
2. **Execution period:** The project creator confirms the project owner/manager. At this point, the project owner begins to execute the project and once progressing and upon completion, submits the project results;
3. **Project Review Period:** Once a project is marked as completed, the project creator and the Go Nebulas Operations Team will review project and its results. Afterwords, a recommendation on whether the project has been successfully completed or not will be given to the Governance Committee. The committee then decides what further action, if any is required. If none is required, the project will receive its funding as decided in the budget period.

3.2.2 Community Assets

The Governance Committee is responsible for managing the use of the public community assets. Public community assets include:

1. **Use and distribution of the Go Nebulas Community Collaboration Fund:** The primary source of this fund is the DPoS revenue generated from Nebulas since the launch of the Nebulas mainnet on March 30, 2018. Some of these assets have been used for programs such as the Nebulas Incentive Program. The remaining assets will be used for the Go Nebulas Community Collaboration Fund after node decentralization. Since the maximum amount of NAS issued per governance cycle is capped at no more than $30,000 USDT, the actual use of the governance mechanism within six months of its launch is $180,000 USDT equivalent NAS.
2. **Incentive allocation of the Nebulas PoD Node decentralization strategy:** The incentive for Nebulas PoD Node Decentralization Strategy includes two parts: consensus incentive and governance incentive. The source is 8,219.1744 NAS revenue generated daily via DPoS. For the specific allocation method, review section 1.3 Incentive allocation.

The use of public assets, changes to the allocation program, etc... will require the use of the proposal process and can be implemented only after the adoption of the resolution by the Governance Committee.

3.2.3 Penalties for Consensus Mechanism

During the voting phase of governance cycle, the governance committee will also need to vote on the results of medium and severe security violations.

1. If the governance committee votes that the punishment is justified, the NAS that has been frozen will be donated to the Go Nebulas Community Collaboration Fund.
2. If the governance committee votes that the node did not cause intentional harm to the network, the block generation stability index S(i) of the node will be restored to the level prior to the punishment and the NAS will be unfrozen.

3.3 Governance Method: Vote

3.3.1 Voting Cycle

Governance nodes must vote within 120 polling cycles (about 5 days) after the end of the previous governance cycle. Not participating in voting is considered an Abstain vote.

3.3.2 Voting Methods

The voting of governance nodes is conducted on the public chain with the results viewable to all. All governance nodes are expected to participate in **all** governance periods. All votable items will have the following options (must choose one):

- For
- Against
- Abstain

Each proposal can only be voted for once by each governance node by utilizing 1 NAX per item being voted. NAX used for voting is destroyed and will not be returned.

3.3.3 Processing of Voting Results

The adoption of a proposal or item requires the following conditions:

Table 3.1: Processing of Voting Results Table

* If a single project budget will exceed the maximum dollar value, it is suggested to split the proposed project into a multi-phase project.

** If the total amount of all approved projects during the governance cycle exceeds the maximum budget, projects are ranked by their support rate. Any proposal that is approved but funding is not available for the current governance cycle is deferred to the next governance cycle.

3.4 Penalty Mechanism

3.4.1 Individual Governance Node Penalties

If a consensus node becomes a governance node for two consecutive governance cycles without taking part in governance voting, the node will not be able to be selected as a governance node for 820 consensus polling cycles (approximately one month).

3.4.2 Governance Failure

1. If there are fewer than 26 (of the 51 selected) governance nodes participating in the voting during a governance cycle, the cycle will be declared invalid; no decision made will be executed and all governance incentives will be donated to the Go Nebulas Community Collaboration Fund (See *3.2.2 Community Assets*).
2. If there is no proposal or project in a governance cycle, i.e. there is nothing to vote on, the cycle is declared invalid and all governance incentives will be donated to the Go Nebulas Community Collaboration Fund.

---

[1] Go.nebulas.io help Documentation

Appendix

Appendix A. Recommended Hardware Configuration for Node Operation

Monthly recommended configuration server expenditure is approximately: $150 USDT/month

- CPUïijŽ>=4-Core minimum (Recommended 8-Core)
- RAMïijŽ>=16G
- Disk: >= 600G SSD
- NTP: NTP service is required on the server to ensure correct time synchronization for all operational nodes.

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

ItâĂŹs recommended to build and deploy nodes via docker:

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

```
sudo docker-compose build

sudo docker-compose up -d
```

Appendix B. Node Multi-User Participation

Nodes can be operated by an individual, business entity or even a group of individuals acting as a single entity. **The distribution of node incentives is determined by the primary node operator.**

Supporting a node participant is the autonomous ideology of the community members and those who choose to support node operators should only make this decision after fully examining the operation of the node. PoD can only guarantee the pledging and withdrawal of any pledged NAX to a node and is not responsible for the commitment of the node operator to their supporters.

In order to facilitate the participation of community users, the Nebulas Foundation will form a demonstration multi-user participation node. This node will be operated and maintained by the Nebulas Foundation. All community members can support this node by pledging NAX to its existence and the benefits (minus the basic cost of server operation) of the node will be equally distributed to those who are involved in its co-construction based on NAX pledge quantity.

Appendix C. Earnings Simulation

Assuming that a node is among the 51 candidate nodes every day for a month, the maximum consensus incentive for the month is approximately 9,920 NAS.

There are over 700 polling cycles per month and considering the existence of random factors in the selection algorithm, the average revenue per node is expected to be about 3,307 NAS. This however can vary greatly depending on multiple factors as detailed in this paper.

Assuming that all 51 governance nodes selected each month participate, the nodes incentive is estimated to be 816 NAS per month per node.

Appendix D. Parameter Table

D.1 Basic Parameters

- Average block time: 15 seconds
- Polling cycle: 210 block height, approx. 52.5 minutes
- Governance cycle: 820 polling cycles (approximately 1 month)
- Consensus nodes: 21
- Candidate nodes: 51 (with 21 consensus nodes)
- Governance nodes: 51 (consensus nodes who generated the largest number of valid blocks per governance cycle)
- Deposit: 20,000 NAS
- Candidate node minimum pledge (vote): 100,000 NAX
- NAS Pledge return time (Once the exit request has been issued): 820 polling cycles (approximately 1 month)
- NAX return time (Once the withdrawal request or the exit request has been issued): 120 polling cycles (approximately 5 days)

D.2 Parameters Related to the Consensus Mechanism

- Number of blocks generated per node within each polling cycle: 10
- Initial Value of Block Generation Stability Index S(i): 0.8
- Max Value of Block Generation Stability Index S(i): 1
- Trigger threshold for penalty mechanism via Block Generation Stability Index S(i): 0.5

- Penalty (Medium): 5% of NAS deposit
- Penalty (Severe): All NAS deposit plus all NAX pledged to that node
- Consensus mechanism penalty duration (low and medium security level): Candidate node cannot be selected for 20 polling cycles (approximately 1 day)
- Consensus mechanism penalty duration (severe security level): Permanent

D.3 Governance Mechanism Parameters

- Governance node voting time: 120 polling cycles (approximately 5 days)
- Governance node minimum participation: 26
- Required proposal approval rate: Greater than 50%
- Required approval rate for the project establishment voting, project acceptance voting, and penalties for consensus mechanism voting: Greater than 67%
- Governance penalty trigger: Not participating in ALL voting proposals (at minimum level) for two governance cycles constantly
- Governance penalty duration: 820 polling cycles (approximately 1 month)

D.4 Incentive Allocation Parameters

- Daily bookkeeping Income (entire network): 8,219.1744 NAS
- Annual bookkeeping income (entire network): 2,999,941 NAS
- Total annual consensus mechanism incentives (entire network): 2,499,951 NAS
- Total annual incentives for governance mechanisms (entire network): 499,999 NAS
- Single project budget: Cannot be greater than $15,000 USDT
- Maximum amount of funds released per governance cycle: Cannot greater than $30,000 USDT

Another post: The launch of NebulasâĂŹ Proof of Devotion consensus protocol has begun on Ambcrypto.

Roadmap

In order to best complete the decentralized transition of the mainnet nodes, the Nebulas PoD Node Decentralization Strategy will gradually open the node applications to all. Initially, we invite active project parties, partners and community members within the current Nebulas ecosystem to deploy nodes and explore the governance processes in advance to the public release to provide valuable advice for testing and improvement. The roadmap is as follows:

- **Early January 2020** - Launch of test bounty program on the testnet;
- **January 2020** - Node application by invitation, consensus mechanism initiated;
- **End of February 2020** - Open application for community nodes;

- **End of March 2020** - First governance vote, governance mechanism initiated.

1.4.5 CÃşmo colaborar

Âą Tu contribuciÃşn vale mucho!

Nebulas aims for a continuously improving ecosystem, which means we need help from the community. We need your contributions! It is not limited exclusively to programming, but also bug reports and translations, spreading the tenets of Nebulas, answering questions, and so on.

- *1. Community Collabration Platform: Go.nebulas.io*
- *2. Code*
  - *2.1 Mainnet Development*
  - *2.2 Bug Reporting*
- *3. Documentation*
  - *3.1 Wiki & Translation*
  - *3.2 Writing*
- *4. User Groups*
- *5. Donations*

1. Community Collabration Platform: Go.nebulas.io

Estas colaboraciones no se limitan sÃşlo al desarrollo del cÃşdigo, sino tambiÃľn a reportar errores, traducir documentos, difundir los principios de Nebulas y responder preguntas de usuarios novatos.

La lista de proyectos se puede ver aquÃ■.

2. CÃşdigo

2.1 Desarrollo sobre *mainnet*

El desarrollo sobre la *mainnet* todavÃ■a se estÃą gestando, y necesitamos la ayuda de la comunidad para resolver determinados problemas complejos asociados con la industria del blockchain.

Para mÃąs informaciÃşn al respecto, consulta los siguientes recursos:

- Nuestro repositorio en Github.
- Nuestra hoja de ruta.

2.2 Reporte de errores

ÂąValoramos siempre cualquier reporte de errores!

Si encuentras un error en nuestro cÃşdigo, por favor, repÃşrtalo inmediatamente a la comunidad de Nebulas a travÃľs de este formulario. Puedes reportar errores en la *testnet* de Nebulas, como asÃ■ tambiÃľn en nuestra *mainnet*, en nebPay, en neb.js, en nuestra cartera y tambiÃľn en otras herramientas y documentos.

Consulta el programa de recompensas aquÃ■.

En Nebulas seguimos el Sistema de ValoraciÃşn de Riesgos OWASP para calcular la recompensa correspondiente en base al grado de severidad del error encontrado. Puedes encontrar mÃąs informaciÃşn al respecto en el artÃ■culo correspondiente de nuestra wiki.

Si tienes alguna sugerencia sobre cÃşmo corregir algÃžn error, o cÃşmo mejorar un proyecto relacionado, no dudes en hacÃľrnoslo saber. Si lo deseas, puedes tambiÃľn participar en el desarrollo y proteger, asÃ■, el valor de nuestros activos en forma directa. Juntos hacemos de Nebulas un ecosistema mÃąs sano, seguro y robusto.

3. DocumentaciÃşn

3.1 Wiki y traducciones

ÂąLas traducciones son sumamente importantes para que el mundo conozca Nebulas!

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 White Paper and Yellow Paper, the Nebulas design principle introduction document, and more. Your contribution will significantly help numerous Nebulas developers and community members.

Agradecemos a todos los miembros de nuestra comunidad que, desde distintos puntos del planeta, participan en la traducciÃşn de la documentaciÃşn de Nebulas.

Puedes traducir prÃącticamente cualquier documento disponible, incluyendo esta wiki, la documentaciÃşn tÃľcnica de nuestra *mainnet*, la lista de preguntas frecuentes de nuestro sistema de DApps, el *whitepaper* (o el *yellowpaper*), la introducciÃşn a los principios de diseÃśo, y mÃąs. Tu contribuciÃşn ayudarÃą de forma significativa a los desarrolladores y a los demÃąs miembros de la comunidad.

Ten en cuenta que algunos documentos podrÃ■an requerir una formaciÃşn acadÃľmica en matemÃąticas, ciencias de la computaciÃşn, criptografÃ■a u otras especialidades.

Traducir la wiki

Por favor, lee la guÃ■a para usuarios de la Wiki de Nebulas

Recursos adicionales

Para editar localmente, necesitarÃąs utilizar reST para los archivos .rst y Pandoc Markdown para editar los archivos .md.

Haz clic aquÃ■ para conocer la diferencia entre ambos formatos.

- CÃşmo utilizar Markdown, guÃ■a escrita por John Gruber.
- GuÃ■a tÃľcnica de la sintaxis Markdown, escrita por iA Writer.

DespuÃľs

Cuando estÃľs editando alguna pÃągina en Github, debes hacer clic en ÂńPreview changesÂż para ver el resultado de los cambios.

DespuÃľs de que su contribuciÃşn ha sido aceptada, puede ver el proceso de compilaciÃşn aquÃ■.

3.2 RedacciÃşn de documentos

Los desarrolladores de nuestra comunidad necesitan documentaciÃşn para poder comprender y hacer uso de las distintas funciones que expone Nebulas. Agradeceremos a toda persona de nuestra comunidad que se anime a redactar documentos tÃľcnicos, introducciones, tutoriales y listas de preguntas frecuentes.

AdemÃąs de ello, los miembros de nuestra comunidad necesitan guÃ■as introductorias y otros tipos de guÃ■as para usuarios generales acerca de las distintas herramientas que ofrece Nebulas.

Tu contribuciÃşn serÃą de suma ayuda para toda la comunidad, y probablemente sea traducida a otros idiomas con el fin de llegar a la mayor base de usuarios posible.

4. Grupos de usuarios

La comunicaciÃşn es clave para construir una comunidad vibrante, y para compartir ideas y comentarios acerca de Nebulas.

Nuestro proyecto hace uso de distintos canales para mantener conectada a nuestra comunidad global. Por favor, ingresa a nuestra pÃągina dedicada a la comunidad para mÃąs informaciÃşn al respecto.

comunidad: community.nebulas.io (for developers and non-developers)

Reddit: reddit.com/r/nebulas, reddit.com/r/nasdev (for developers)

Telegram: English (for non-developers).

Agradeceremos a nuestra comunidad de desarrolladores la creaciÃşn de un canal de IRC para lograr una comunicaciÃşn mÃąs fluida. TambiÃľn serÃą bienvenidos aquellos canales orientados a los hispanohablantes.

5. Donaciones

Agradecemos profundamente cualquier donaciÃşn por parte de nuestra comunidad cuyo fin sea el desarrollo de Nebulas. Tanto NAS como ETH son bienvenidos.

TambiÃľn alentamos a los miembros de nuestra comunidad a respaldarnos en tÃľrminos materiales. Por ejemplo: la cesiÃşn de espacios para organizar reuniones y conferencias, guÃ■as turÃ■sticas para los asistentes, fotografÃ■a, etcÃľtera. Tu contribuciÃşn puede ser pÃžblica o anÃşnima. Para cualquier consulta, envÃ■a un correo electrÃşnico a contact@nebulas.io.

1.4.6 Programa de Recompensas

Hoy en dÃ■a casi todos los proyectos son publicados en la PÃągina de Proyecto de Nebulas junto con sus recompensas correspondientes, y se espera que los usuarios apliquen para reclamar un proyecto o partes de Ãľl. Este proceso se aplica a la wiki y al Programa de recompensas de errores de NAT. Por ahora, el Programa de recompensas de errores de Nebulas solo requiere que envÃ■es un formulario con la informaciÃşn relevante.

Programa de Recompensas de la Wiki de Nebulas

Anteriormente, los usuarios que creaban o modificaban el contenido de la Wiki de Nebulas tenÃ■an derecho a ganar una recompensa en forma de NAS. Hoy en dÃ■a, el proceso es muy diferente.

Para calificar para la recompensa de wiki, vaya a la pÃągina de proyecto mencionada y busque “wiki“, o simplemente haga clic en aquÃ■ para ver todos los resultados disponibles.

Programa de Recompensas para Bug Hunters de Nebulas

El programa **Nebulas Bug Bounty** apunta a consolidar en Nebulas un ecosistema saludable y seguro. Para ello, hemos puesto a disposiciÃşn de los *buscadores de errores* una serie de recompensas que premian cada error que se encuentre.

Este programa estÃą implementado por el ComitÃl' TÃl'cnico de Nebulas (*Nebulas Technical Committee*, o NTC), en uniÃşn con el equipo tÃl'cnico de Nebulas y los miembros de su comunidad.

NTC alienta a su comunidad a informar de cualquier tipo de vulnerabilidad a travÃl's del proceso que se describe mÃąs abajo; de ese modo, cada miembro de la comunidad tiene la chance de participar en la construcciÃşn del ecosistema de Nebulas y de recibir a cambio una o mÃąs recompensas.

CategorÃ■as

El programa divide las recompensas en dos categorÃ■as: recompensas por errores comunes (*common bug bounty*) y recompensas por errores especiales (*special bug bounty*).

Recompensas por errores comunes

SerÃąn otorgadas a todas aquellas personas que encuentren vulnerabilidades en la mainnet de Nebulas, en la testnet de Nebulas, en nebPay, en la cartera web, en la librerÃ■a neb.js y en otros componentes similares.

Recompensas por errores especiales

Se otorgarÃąn a quienes descubran vulnerabilidades en las llamadas a funciones intercontratos (*inter-contract call functions*) y similares.

DeterminaciÃşn de las recompensas

El ComitÃl' TÃl'cnico de Nebulas (*Nebulas Technical Committee*) determinarÃą el monto de las recompensas de acuerdo a la gravedad de la vulnerabilidad descubierta, de acuerdo al mÃl'todo de evaluaciÃşn de riesgos OWASP, basÃąndose dos criterios: **impacto** y **probabilidad**. No obstante, el valor final de las recompensas estarÃąn sujetas a la decisiÃşn del comitÃl'.

1

Criterios

Impacto

- Alto: errores que afectan la seguridad de los activos.
- Medio: errores que afectan la estabilidad del sistema.
- Bajo: otros errores que no afectan la seguridad de los activos ni la estabilidad del sistema.

####Âă Probabilidad

- Alta: el error podrÃ■a ser descubierto por cualquier persona que realice una operaciÃşn determinada, independientemente de si el error fue reportado o no.
- Media: sÃşlo algunas personas podrÃ■an encontrar el error (tales como desarrolladores que analicen el cÃşdigo); los usuarios comunes no podrÃ■an desencadenar el problema.
- Baja: cubre sÃşlo el 1% (o menos) de la base de usuarios de Nebulas âĂŤpor ejemplo, propietarios de un modelo poco usual de dispositivo AndroidâĂŤ o cualquier otro caso excepcional.

Montos

Para asegurar que la persona que reporta el error obtenga una recompensa adecuada, estable en el tiempo y proporcional al error hallado, el valor en NAS se ajustarÃą segÃžn la paridad con el dÃşlar estadounidense.

Los montos de las recompensas se dividen en cinco categorÃ■as:

- Errores crÃ■ticos: US$ 1000 o mÃąs (sin lÃ■mite superior)
- Errores de probabilidad alta: US$ 500 o mÃąs
- Errores de probabilidad media: US$ 250 o mÃąs
- Errores de probabilidad baja: US$ 100 o mÃąs
- Mejoras: US$30 o mÃąs

Testnet

Las recompensas para aquellos errores especiales que se encuentren en la testnet (como por ejemplo los vinculados a las llamadas de funciones inter-contratos) se han incrementado de forma acorde, y los montos, en NAS, estarÃąn expresados en dÃşlares estadounidenses.

Reporta un error

Por favor, envÃ■anos tu reporte a travÃľs de este enlace.

Notas

1. Por favor, revisa que tu reporte sea lo mÃąs claro y conciso posible, ya que la evaluaciÃşn de la recompensa estarÃą basada en el contenido del formulario.
2. En el caso de que varias personas descubran un error al mismo tiempo, se valorarÃąn los reportes de acuerdo a su orden cronolÃşgico. Los usuarios de la comunidad son libres de discutir acerca de los errores, pero la discusiÃşn en sÃ■ misma no se considera como reporte; por ello, debe utilizarse el formulario mencionado anteriormente.
3. El programa de recompensas (*Nebulas Bug Bounty Program*) es de largo plazo. El ComitÃľ TÃľcnico de Nebulas se reserva el derecho a la interpretaciÃşn final de este programa, y el derecho de ajustar o cancelar el alcance de las recompensas, el monto y el criterio de selecciÃşn.
4. El ComitÃľ TÃľcnico de Nebulas evaluarÃą y confirmarÃą los reportes de errores en forma posterior a su envÃ■o. Los tiempos de evaluaciÃşn dependerÃąn de la severidad del problema y la dificultad resultante para solucionarlos. El resultado de la evaluaciÃşn se enviarÃą a quien lo reportÃş lo antes posible.
5. Para evitar que los errores reportados puedan ser explotados por personas inescrupulosas, quienes reporten un error deben hacerlo Ãžnicamente a travÃľs del formulario mencionado mÃąs arriba.
6. Quienes reporten errores deben mantener los mismos en total confidencialidad al menos hasta pasados 30 dÃ■as luego de enviar el reporte a Nebulas, y no deben revelar su naturaleza a terceros. Tal periodo de confidencialidad podrÃą ser extendido de forma unilateral por el equipo de Nebulas. Si quien reporta el error lo revela a terceros, de tal suerte que Nebulas o sus usuarios se vean perjudicados, tal persona deberÃą costear los gastos derivados del perjuicio a cada una de las partes involucradas.
7. El ComitÃľ TÃľcnico de Nebulas alienta a los miembros de la comunidad a discutir cualquier otro tÃşpico en el grupo pÃžblico de discusiÃşn de Nebulas; asimismo, alentamos a la comunidad a unirse a nuestro esfuerzo por solucionar los problemas que se presenten. SiÃľntete libre de unirte a nuestra lista de correo de Nebulas.

1.4.7 Preguntas frecuentes

Este documento harÃą foco en la tecnologÃ■a detrÃąs de la plataforma Nebulas. Para preguntas mÃąs generales, por favor, visita este hilo en nuestro subreddit.

Para una mejor comprensiÃşn de la plataforma Nebulas se recomienda leer nuestro whitepaper tÃľcnico.

Tabla de contenidos

3. Protocolo de Incentivo a Desarrolladores
4. Prueba de DevociÃşn
5. Motor de bÃžsqueda de Nebulas
6. Fundamentos
   (a) Nebulas Name Service
   (b) Lightning Network
   (c) Token Nebulas (NAS)
   (d) Contratos inteligentes
   (e) Almacenamiento binario

Nebulas Rank (NR)

Pondera valor a travÃľs de los parÃąmetros de liquidez y propagaciÃşn de la direcciÃşn. *Nebulas Ranking* apunta a establecer un enfoque de mediciÃşn confiable, computable y determinÃ■stico. Con este sistema de valoraciÃşn veremos mÃąs y mÃąs aplicaciones descollantes en la plataforma Nebulas.

Â£CuÃąndo estarÃą listo el sistema *Nebulas Rank*?

(en desarrollo).

Â£TendrÃąn mÃąs peso las DApps que contengan mÃąs transacciones?

(en desarrollo).

Â£CÃşmo logra diferenciar *Nebulas Rank* entre DApps de calidad y DApps con muchas transacciones?

(en desarrollo).

Â£El algoritmo de *Nebulas Ranking* es de cÃşdigo abierto?

SÃ■.

Â£QuiÃľnes pueden colaborar en el desarrollo del algoritmo?

En esta etapa es el equipo de Nebulas el responsable de ese desarrollo. Con el tiempo el cÃşdigo se abrirÃą a las colaboraciones comunitarias, lo que permitirÃą que la comunidad tome decisiones y vote para determinar el futuro del algoritmo.

Â£Es posible engaÃśar a *Nebulas Rank*?

Implementaremos controles estrictos para evitar la manipulaciÃşn; por supuesto, *Nebulas Rank* evolucionarÃą continuamente para cubrir las necesidades de la comunidad.

Nebulas Force (NF)

Ofrece soporte para la actualizaciÃşn de protocolos centrales y contratos inteligentes en los blockchains. Provee la capacidad de auto-evoluciÃşn que caracteriza a Nebulas y sus aplicaciones. Con este sistema, los desarrolladores pueden construir aplicaciones enriquecidas en lapsos reducidos; al mismo tiempo, permite que esas aplicaciones se pueden adaptar dinÃąmicamente a los cambios de mercado.

Â£CuÃąndo estarÃą disponible *Nebulas Force*?

(en desarrollo).

Â£Es posible actualizar los contratos inteligentes?

Si.

Â£De quÃľ manera el sistema de actualizaciÃşn de contratos inteligentes de *Nebulas Force* es mejor que otras soluciones ya disponibles o que estÃąn prÃşximas a lanzarse?

(en desarrollo).

Â£Es posible actualizar el protocolo del blockchain de Nebulas sin realizar un *fork*?

SÃ■.

Â£Es posible actualizar el cÃşdigo de la mÃąquina virtual *Nebulas Virtual Machine*?

SÃ■.

Protocolo de Incentivo a Desarrolladores

DiseÃśado para mejorar la contrucciÃşn de nuestro ecosistema, los incentivos en el token NAS ayudarÃąn a los desarrolladores a crear mayor valor en Nebulas.

Â£CuÃąndo estarÃą disponible este protocolo?

(en desarrollo).

Â£HabrÃą un lÃ■mite en la cantidad de recompensas que una sola dApp puede recibir?

(en desarrollo).

Â£Los desarrolladores podrÃąn lanzar sus propias Ofertas Iniciales de Criptodivisa (ICO)?

(en desarrollo).

Â£RecibirÃąn recompensas Ãžnicamente las dApp mejor posicionadas en *Nebulas Rank*?

(en desarrollo).

Â£Con cuÃąnta frecuencia se entregarÃąn las recompensas?

(en desarrollo).

Â£CÃşmo harÃąn para impedir las trampas en este sistema?

La forma en que el sistema estÃą pensado hace realmente difÃ■cil manipularlo. Como los contratos inteligentes sÃşlo se pueden llamar de forma pasiva, serÃ■a muy ineficiente para un usuario, en tÃľrminos de costos, intentar manipular el sistema.

Puedes obtener informaciÃşn mÃąs detallada en nuestro whitepaper tÃľcnico.

Prueba de DevociÃşn

Para construir un ecosistema saludable Nebulas propone tres puntos clave en su algoritmo de consenso: rapidez, irreversibilidad y equidad. Al adoptar las ventajas de *Proof of Stake* y *Proof of Importance* y al aprovechar *Nebulas Rank*, nuestro algoritmo *Proof of Devotion* se convertirÃą en el sistema de consenso mÃąs elegido por otros proyectos.

Â£CuÃąndo se implementarÃą el algoritmo *Proof of Devotion*?

(en desarrollo).

Â£QuÃľ algoritmo de consenso se utilizarÃą mientras se desarrolla *Proof of Devotion*?

(en desarrollo).

Â£CÃşmo se eligen los *contables*?

El algoritmo *Proof of Devotion* PoD hace uso de *Nebulas Rank* para determinar quÃľ nodos son elegibles. A partir de esa preselecciÃşn, se elige un nodo al azar que realizarÃą la proposiciÃşn del nuevo bloque; los demÃąs nodos pre-seleccionados realizarÃąn la validaciÃşn.

Â£Los *contables* tendrÃąn que realizar un depÃşsito?

SÃ■: una vez que un nodo se elige para realizar una validaciÃşn, debe realizar un depÃşsito para completar la operaciÃşn.

Â£CuÃąntos validadores habrÃą por vez?

(en desarrollo).

Â£QuÃľ tipo de mecanismos anti-fraude se implementarÃąn?

(en desarrollo).

Motor de bÃžsqueda de Nebulas

Nebulas desarrollÃş un motor de bÃžsqueda para aplicaciones descentralizadas, basado en su sistema de ranking de valor. Al usar este motor, los usuarios podrÃąn encontrar fÃącilmente distintas aplicaciones descentralizadas en el mercado.

Â£CuÃąndo estarÃą disponible este sistema?

(en desarrollo).

Â£SerÃą posible buscar dApps que no estÃąn en la plataforma Nebulas?

(en desarrollo).

Â£El motor de bÃžsqueda estarÃą tambiÃľn descentralizado?

(en desarrollo).

Â£HabrÃą un control de *Nebulas Rank* sobre los resultados de la bÃžsqueda?

(en desarrollo).

Â£QuÃľ tipo de informaciÃşn se podrÃą buscar?

Hemos pensado en distintas formas de realizar bÃžsquedas en el blockchain:

- Indizar sitios web relevantes y establecer un mapeo entre ellos y los contratos inteligentes.
- Analizar el cÃşdigo de los contratos inteligentes de cÃşdigo abierto.
- Establecer estÃąndares para los contratos inteligentes, de modo que la bÃžsqueda resulte mÃąs sencilla.

Fundamentos

Nebulas Name Service

El equipo de Nebulas implementarÃą en su blockchain un sistema de dominios similar al DNS âĂŤllamado *Nebulas Name Service*, o NNSâĂŤ, que serÃą abierto, gratuito e irrestricto. Cualquier desarrollador podrÃą implementar su propio sistema de resoluciÃşn de nombres de dominio de forma independiente o bien basado en NNS.

Â£CuÃąndo estarÃą listo este servicio?

(en desarrollo).

Cuando un nombre de dominio recibe una oferta, Â£cuÃąnto tiempo deben mantener las demÃąs personas las suya?

(en desarrollo).

Â£CÃşmo serÃąn las notificaciones cuando un nombre de dominio reciba una oferta?

(en desarrollo).

Cuando se reserva un nombre de dominio, Â£quiÃl'n recibe el monto ofertado?

(en desarrollo).

Si deseo renovar mi nombre de dominio luego de un aÃśo, Â£debo depositar mÃąs NAS?

(en desarrollo).

Â£SerÃą posible reservar nombres de dominio antes del lanzamiento de NNS?

(en desarrollo).

Lightning Network

Nebulas implementa *lightning network* como la infraestructura de su blockchain, ofreciendo un diseÃśo flexible. Cualquier desarrollador puede utilizar el servicio bÃąsico de *lightning network* para desarrollar aplicaciones que trabajen en un escenario de transacciones frecuentes. AdemÃąs, Nebulas lanzarÃą la primera cartera que da soporte a *lightning network*.

Â£CuÃąndo se implementarÃą *lightning network*?

(en desarrollo).

Token Nebulas (NAS)

La red de Nebulas posee un token nativo, NAS, que juega dos roles: por un lado, como la divisa original de la red, provee liquidez entre los activos de los usuarios, y funciona como un incentivo para los *contables* (*bookkeepers*) y para el programa *Developer Incentive Protocol*; por otro lado, NAS serÃą utilizado para realizar el cÃąlculo de las tarifas a pagar para ejecutar contratos inteligentes.

La unidad mÃ■nima de NAS es 1E-18 (0,000000000000000001 NAS).

Â£QuÃl' sucederÃą con el token ERC20 cuando se lance NAS?

(en desarrollo).

Â£Los desarrolladores de dApps podrÃąn acuÃśar tokens y lanzar ICO?

(en desarrollo).

Contratos inteligentes

Â£QuÃľ lenguajes de programaciÃşn serÃąn soportados una vez que se lance la mainnet?

(en desarrollo).

Â£HabrÃą soporte completo para Solidity?

(en desarrollo).

Â£QuÃľ otros lenguajes de programaciÃşn recibirÃąn soporte, y cuÃąndo?

(en desarrollo).

Almacenamiento binario

Â£CuÃąl es la forma recomendada para almacenar datos binarios en el blockchain de Nebulas? Â£Es algo posible? Â£Es recomendable hacerlo? AdemÃąs, no pude hallar informaciÃşn sobre la funciÃşn *GlobalContractStorage* que se menciona en los documentos; Â£de quÃľ se trata?

Actualmente, se pueden almacenar datos binarios en el blockchain mediante una ÂńtransacciÃşn binariaÂż. El tamaÃśo mÃąximo para dicha transacciÃşn es de 128k. Sin embargo, no aconsejamos el almacenamiento de datos binarios ya que se podrÃ■a almacenar informaciÃşn ilegal.

Si bien `GlobalContractStorage` no se ha implementado aÃžn, podemos decir que permitirÃą compartir datos entre distintos contratos del mismo desarrollador.

ChainID y conexiÃşn

Â£CuÃąl es el *chainID* de las redes testnet y mainnet? Â£CÃşmo configuro la conexiÃşn de red?

ChainID de Nebulas:

- Mainnet: 1

- Testnet: 1001
- Private: 100 por defecto, los usuarios pueden personalizar este valor.

ConexiÃşn de red:

- Mainnet
  - CÃşdigo fuente
  - Wiki
- Testnet
  - CÃşdigo fuente
  - Wiki

ImplementaciÃşn de contratos inteligentes

Â£Es posible implementar contratos inteligentes?

SÃ■, es totalmente posible implementarlo directamente, tal como harÃ■as con un NPM, lo cual es muy sencillo y conveniente.

IDE para contratos inteligentes

Â£Existe algÃžn IDE (similar a Remix, de Solidity) para Nebulas, o alguna manera de explorar los parÃąmetros de los contratos inteligentes?

Puedes utilizar nuestra cartera web para implementar el contrato; Ãl'sta provee una serie de funciones de depuraciÃşn sobre la testnet y permite inspeccionar los resultados.

1.4.8 Acuerdo de licencia

Licencia del proyecto de cÃşdigo abierto Nebulas

La licencia preferida para el proyecto de cÃşdigo abierto Nebulas es la Licencia PÃžblica General Reducida de GNU VersiÃşn 3.0 (âĂIJLGPL v3âĂİ), que permite la comercializaciÃşn y alienta a los desarrolladores o a las empresas a modificar el producto y publicar sus cambios.

No obstante, tambiÃl'n hemos notado que existen otras licencias aun mÃąs amigables con el comercio âĂŤcomo por ejemplo la licencia Licencia de Software de Apache 2.0 (âĂIJApache v2.0âĂİ)âĂŤ. Desde el equipo de Nebulas estamos muy satisfechos al ver que se genera cÃşdigo tanto abierto como propietario dentro de nuestro ecosistema.

Ante esta perspectiva, todavÃ■a estamos decidiendo cuÃąl es la mejor licencia para el ecosistema de Nebulas. Los candidatos son LGPL v3, Apache v2.0 y la licencia MIT. Si se elige esta Ãžltima, aÃśadiremos una enmienda para permitir que nuestros productos se utilicen con mÃąs libertad.

Acuerdo de licencia para colaboradores

Todas las contribuciones a la wiki de Nebulas se harÃąn bajo la licencia Creative Commons License SA 4.0.

Para obtener una lista completa de todos los que contribuyeron a la wiki, haga clic aquÃ■.

- genindex
- modindex
- search