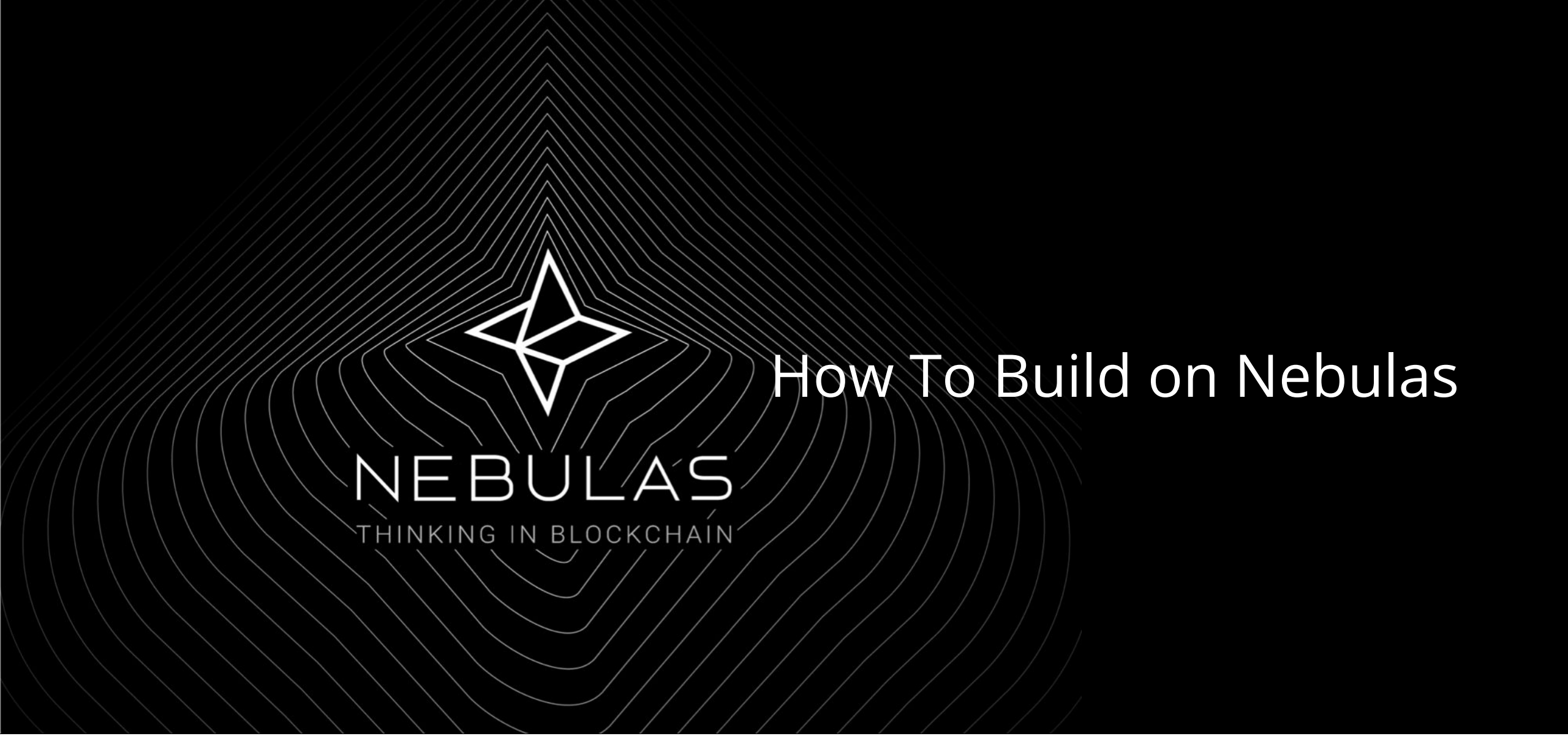


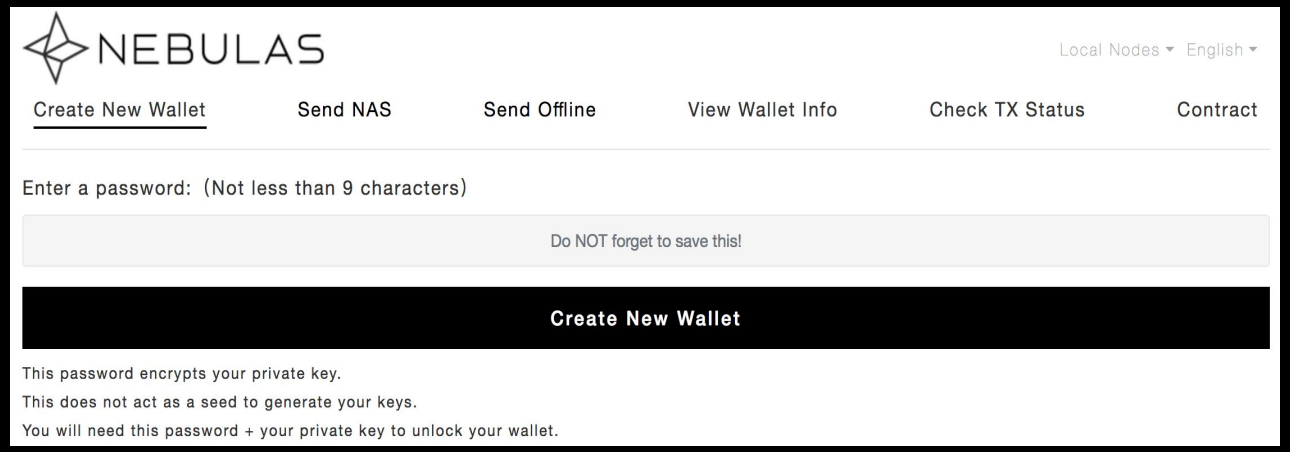
How To Build on Nebulas

Nebulas.io

NEBULAS

CONTENTS

Create Your Wallet



The official Nebulas Web Wallet makes it easy to create your wallet, send transactions and deploy and interact with Smart Contracts.

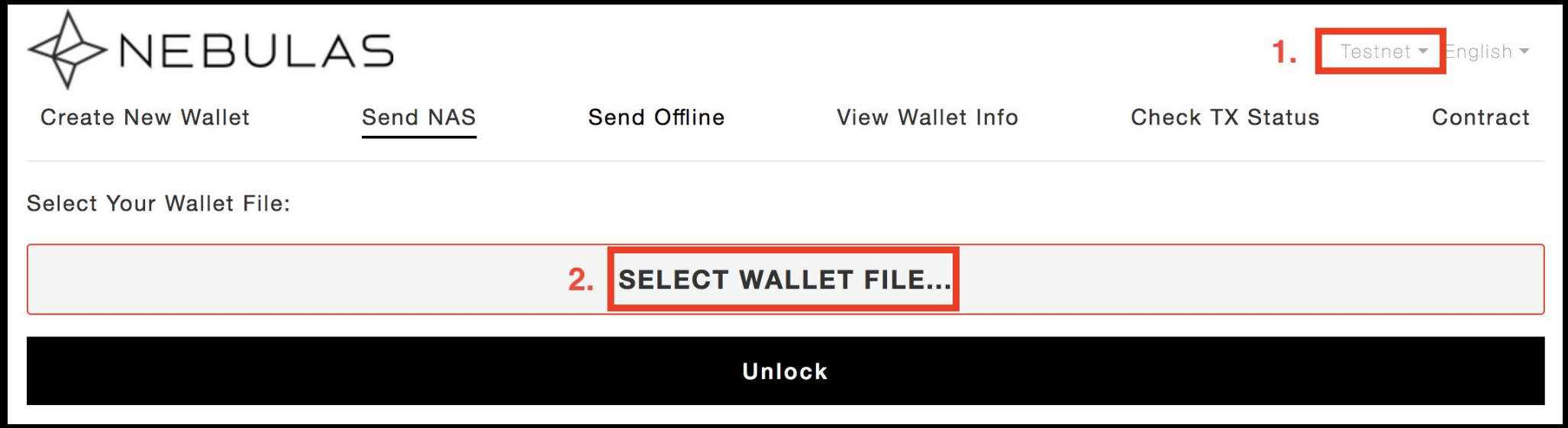
The creation of a wallet will generate a keystore file which contains your address, encrypted private key and additional details about your wallet generation

Wallets *can* be used for mainnet, localnet and testnet.


```
{
  "version":4,
  "id":"ed0d201b-f421-4e49-a03f-e759e50048fb",
  "address":"n1Ni6YPvAj7YXmChKwA3kHWzNsHEsp1F4ta",
  "crypto":{
    "ciphertext":"e2083cbe5ba03f7b6a3f3ce.....0135603287cb85",
    "cipherparams":{
      "iv":"ff7e9d417c8e0edc3aaa3dd8961c20c1"
    },
    "cipher":"aes-128-ctr",
    "kdf":"scrypt",
    "kdfparams":{
      "dklen":32,
      "salt":"a5a601df63f04f3c05cb584d146.....8aa394559e8802",
      "n":4096,
      "r":8,
      "p":1
    },
    "mac":"bdb04873edd4463f80231d61d79c ..... 73a791861e",
    "machash":"sha3256"
  }
}
```

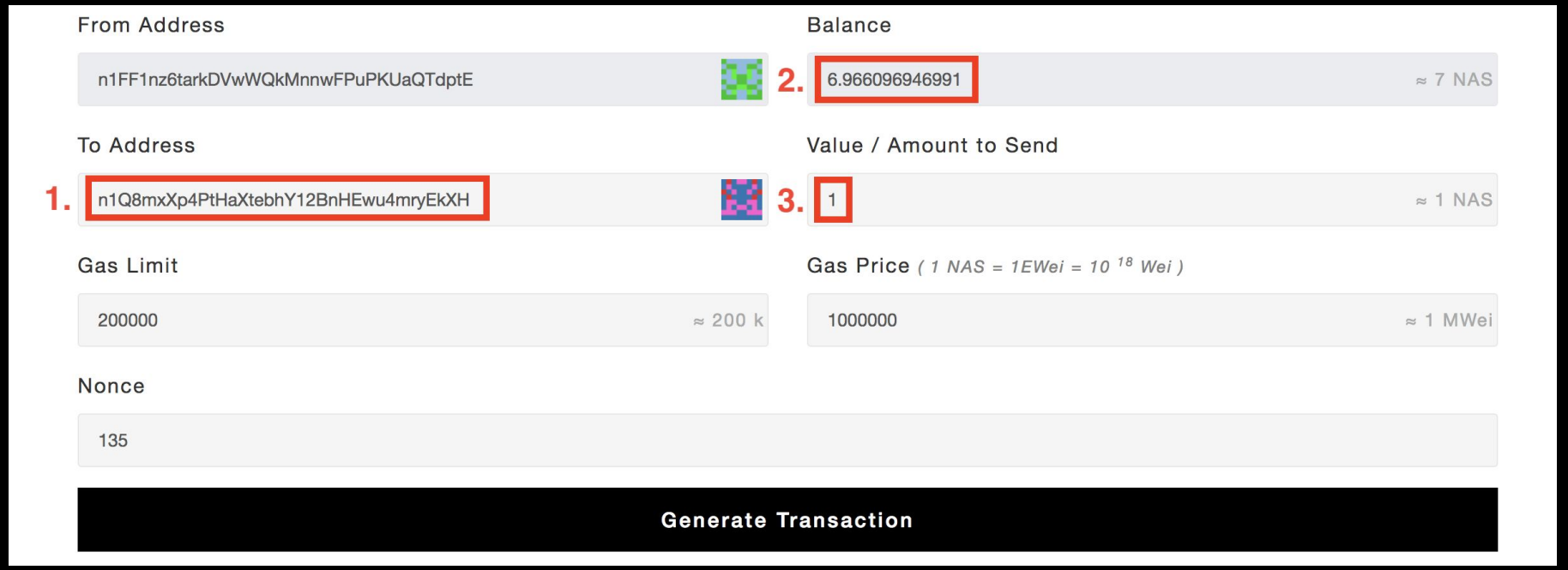

The above is a sample keystore file

Unlock Wallet



1. Network Types：Mainnet, Testnet or LocalNet. In our example, we will use the testnet.
2. Select wallet keystore file and enter password to unlock

Signing Transactions



1. After unlocking, enter the To Address.
2. Once unlocked, your Balance will be shown.
3. Enter the Amount of funds you want to send.

Nebulas coins (NAS) are divisible by up to 18 zeros. This mean the minimum amount of NAS you can send is: 0.000000000000000001

Gas Price and Gas Limit:

Gas is how transactions are paid. Depending on the transaction, the gas fee varies based on size of TX and if going to a smart contract, depending on the required functions.

Nonce Height:

Nonce is how many transactions have already occurred on this address. The nonce height must be entered correctly otherwise your transaction will not be accepted into the blockchain. The Nonce will be automatically updated in the web wallet.

Generate Transaction:

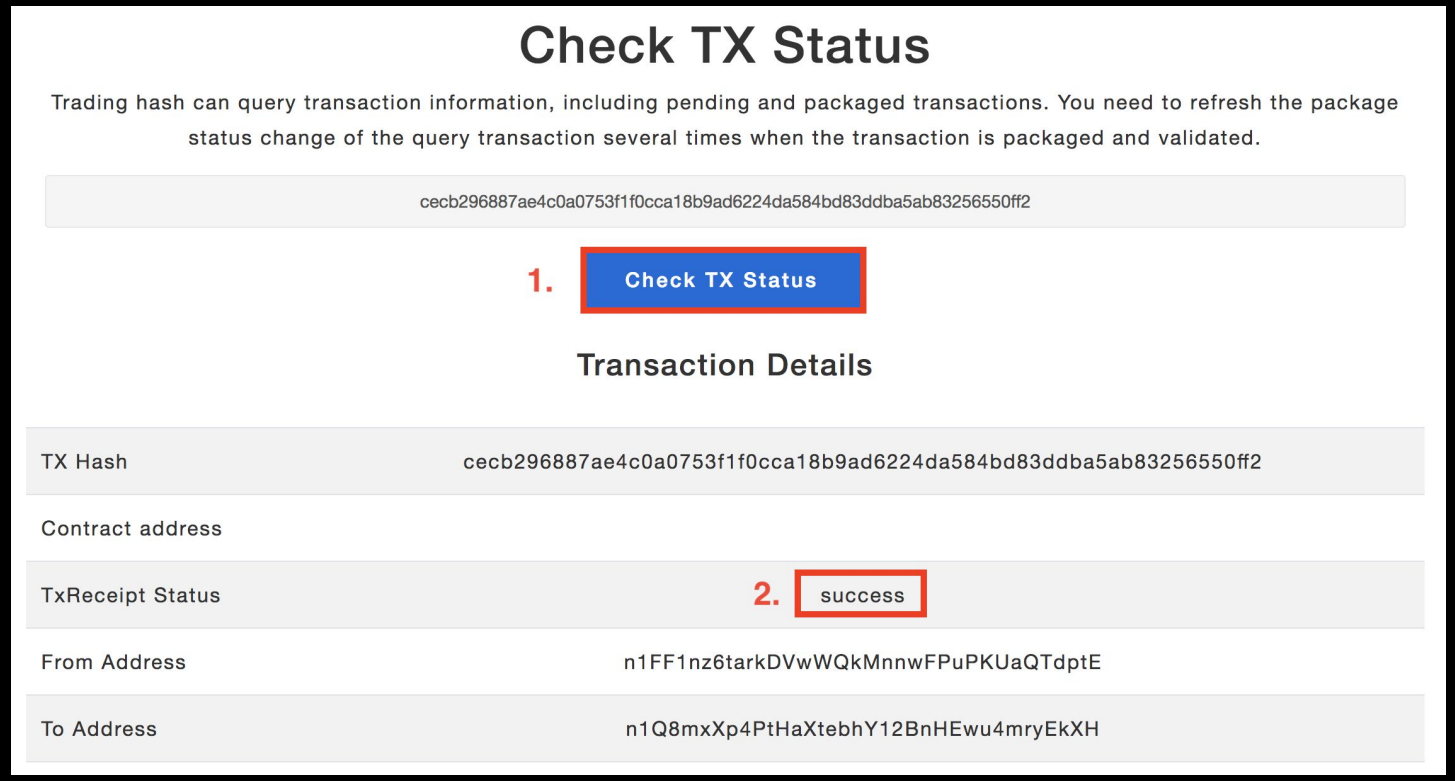
Once you enter the to address, the value you want to send and the gas fees (usually default), press the Generate Transaction button.

Submitting Transactions

1. Raw Transaction Information
2. Signed TX Data (base64 encoded)
3. Generated QR Code
4. Once you click Send Transaction, your transaction will be sent to a node. Transactions usually complete within 15 seconds after submission.
5. Once your Transaction is submitted, you will receive a Transaction Hash (txhash) which you can use to verify your transaction. This acts as your receipt.

- After you receive your txhash, you can click on it to review the status of the transaction and to verify if your transaction has been successfully included into the blockchain.
- Nebulas mints blocks every 15 seconds.
- All data stored on the chain is base64 encoded.

Reviewing Transaction



1. To review the status of the transaction, Click "Check TX Status".
2. Transaction status will usually become successful within 15 seconds after submission.

If your transaction remains as pending, verify you had enough funds to cover the amount being sent plus the gas fee.

Transactions status can be one of the following:

- Pending – Your transaction is awaiting inclusion into a minted block.
- Success – Your transactions has been included into a minted block.
- Failed – Your transaction failed. This may be due to a insufficient gas fee or improper data submission.

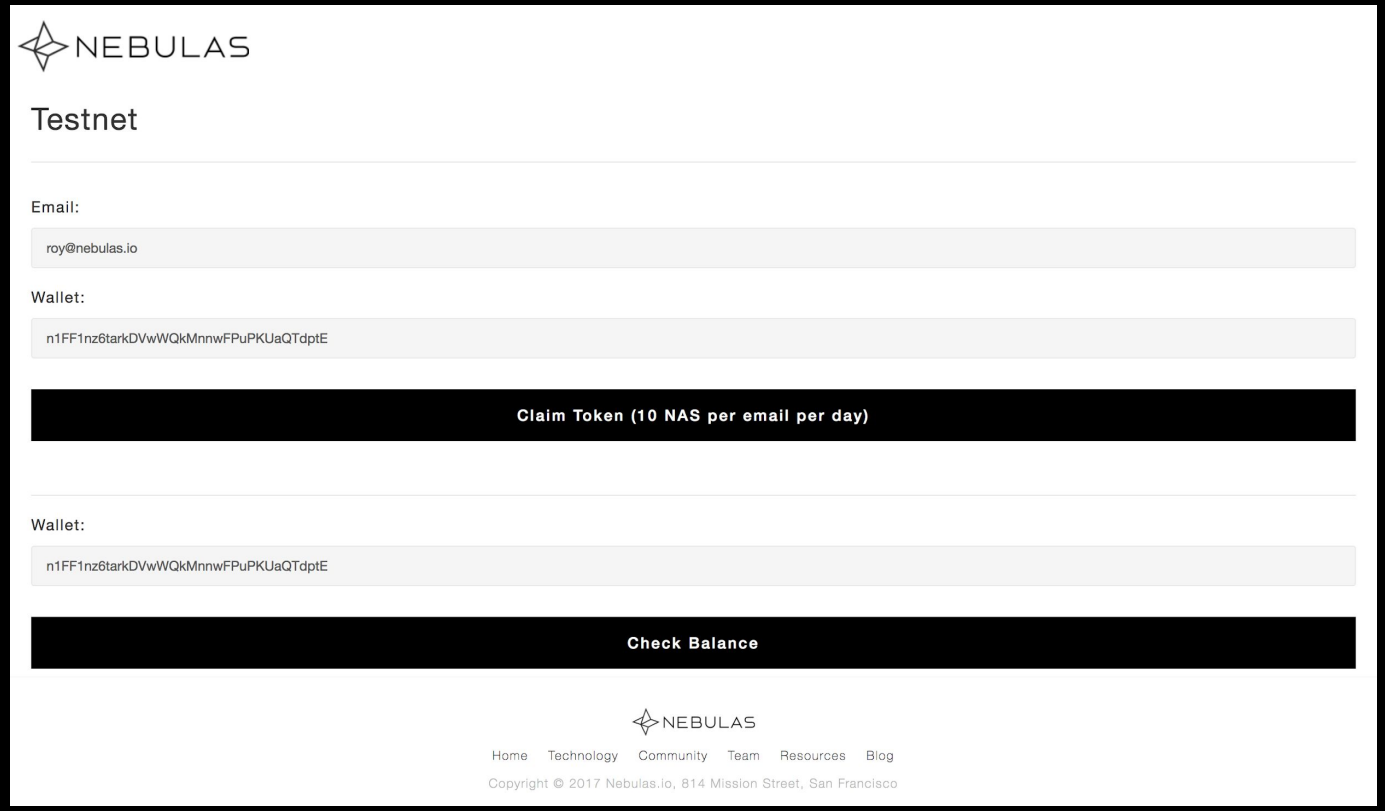
Testnet Faucet

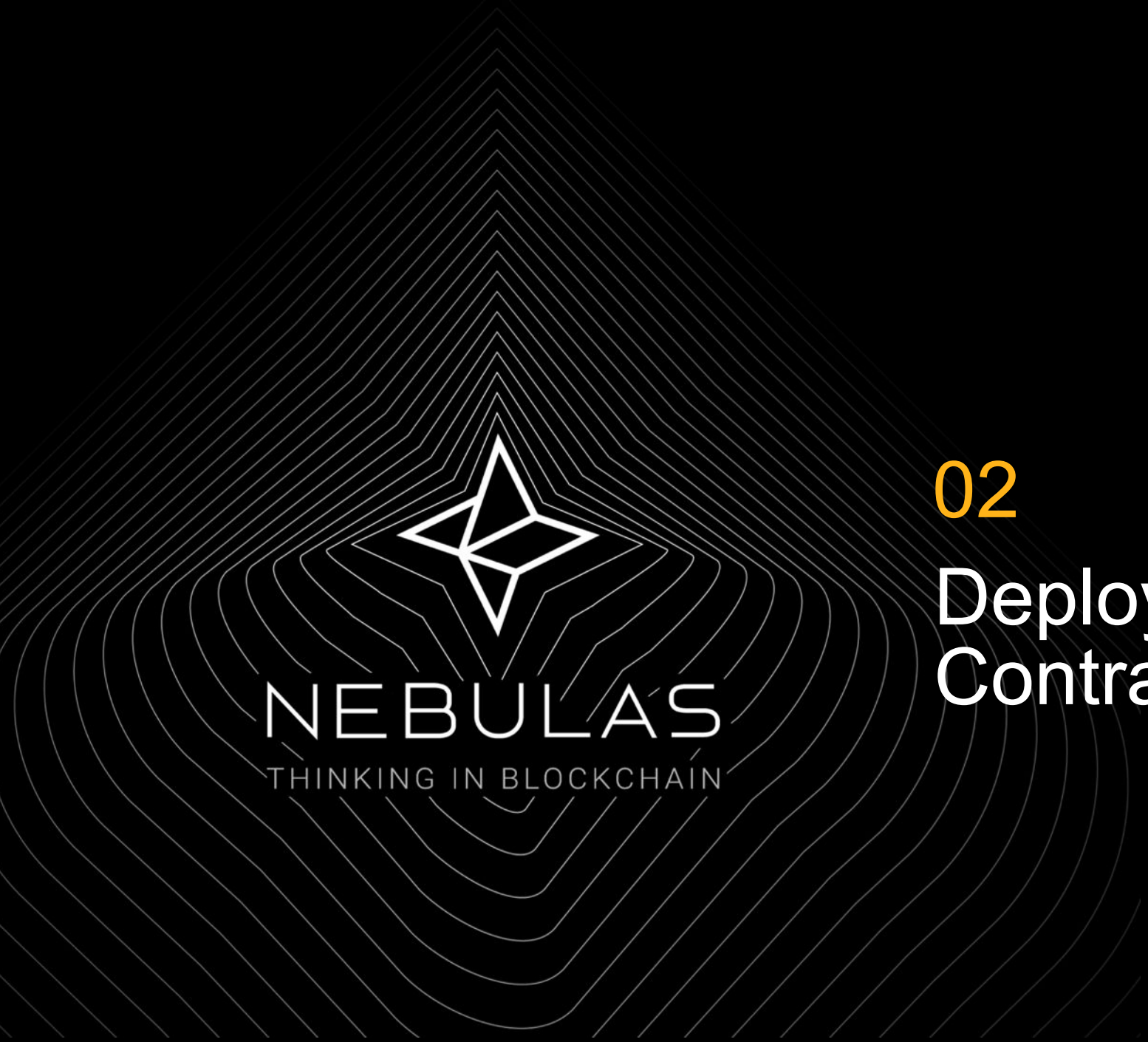
To claim testnet token/gas visit:
https://testnet.nebulas.io/claim/

To claim mainnet token/gas visit:
https://faucet.nebulas.ru/

The purpose of testnet tokens is to test functionality on the test version of Nebulas.

Testnet tokens are good only for the testnet and hold no value on the mainnet.





02

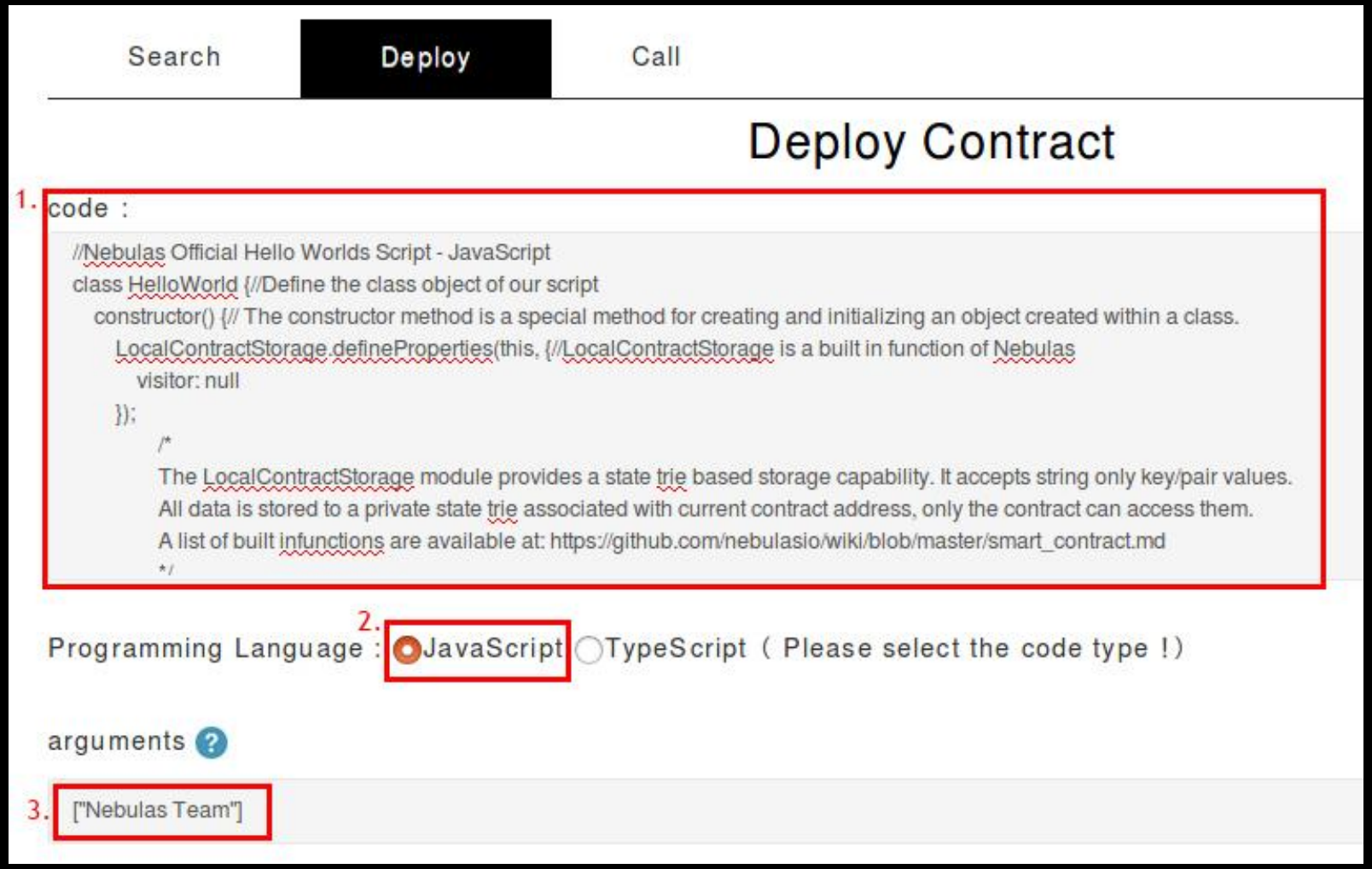
Deploying Smart Contracts

NEBULAS Hello World – Contract Breakdown

This and more are available for you to review at https://github.com/Nebulas-Learning/

```
1  //Nebulas Official Hello Worlds Script - JavaScript
2  class HelloWorld {//Define the class object of our script
3      constructor() {// The constructor method is a special method for creating and initializing an object created within a class.
4          LocalContractStorage.defineProperties(this, {//LocalContractStorage is a built in function of Nebulas
5              visitor: null
6          });
7          /*
8          The LocalContractStorage module provides a state trie based storage capability. It accepts string only key/pair values.
9          All data is stored to a private state trie associated with current contract address, only the contract can access them.
10         A list of built infunctions are available at: https://github.com/nebulasio/wiki/blob/master/smart_contract.md
11         */
12     }
13     init(visitor) {//All Nebulas smart contracts must contain the init class. Any Arguments entered during deployment will be attached to the init function and the variables (e.g. visitor).
14         this.visitor = visitor;//Defining visitor variable as this.visitor
15     }
16     greetings(city) {//creating the greetings class/function that can be called by the user.
17         //This is a string that we will be printing (hello world).
18         //Event.Trigger("greetings", "Here is " + city + ". Hello World! By " + this.visitor + ".")//Prints via RPC - need local node.
19         return "greetings", "Here is " + city + ". Hello World! By " + this.visitor + ".";//Prints via execution - for web-wallet
20     }
21     who() {//creating the who class/function that can be called by the user.
22         return this.visitor;//This will return the name of the visitor which is defined during deployment.
23     }
24 }
25
26 module.exports = HelloWorld;//All contracts must use module.exports. It must be the class/protocol object defined in the script. In our example, it is "HelloWorld"
```

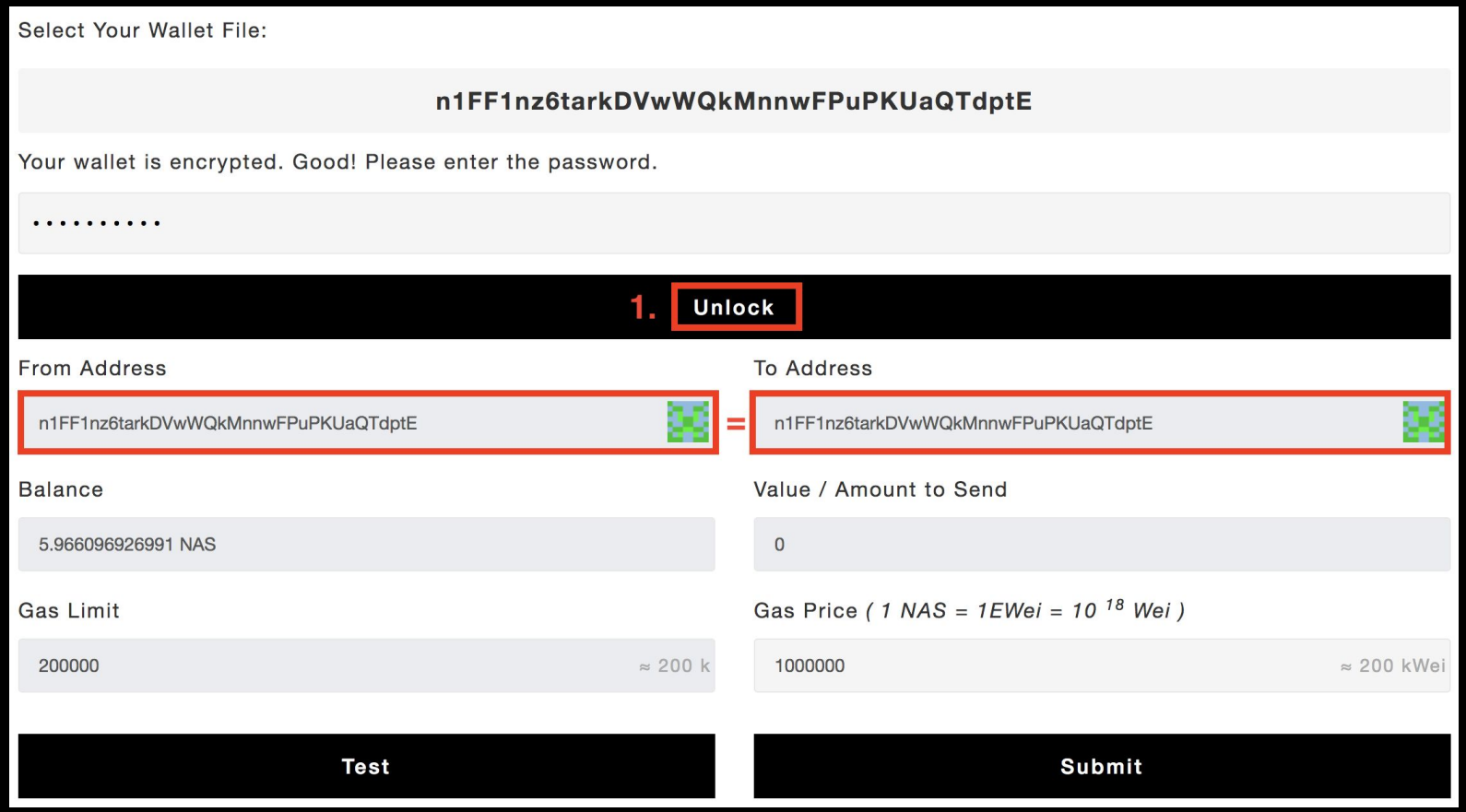
Hello World – Deploy a Contract



1. Sample Smart Contract
2. Select Contract Type
3. Enter any arguments

- Contracts must be either a class or a prototype object.
- All smart contracts must have the init function
- The “module.exports” function must be called at the end of the script and must contain the class/object name.
- Nebulas currently supports Smart Contracts written in Javascript and Typescript
- Arguments must be entered in JSON array format.

Hello World – Unlocking Your Wallet



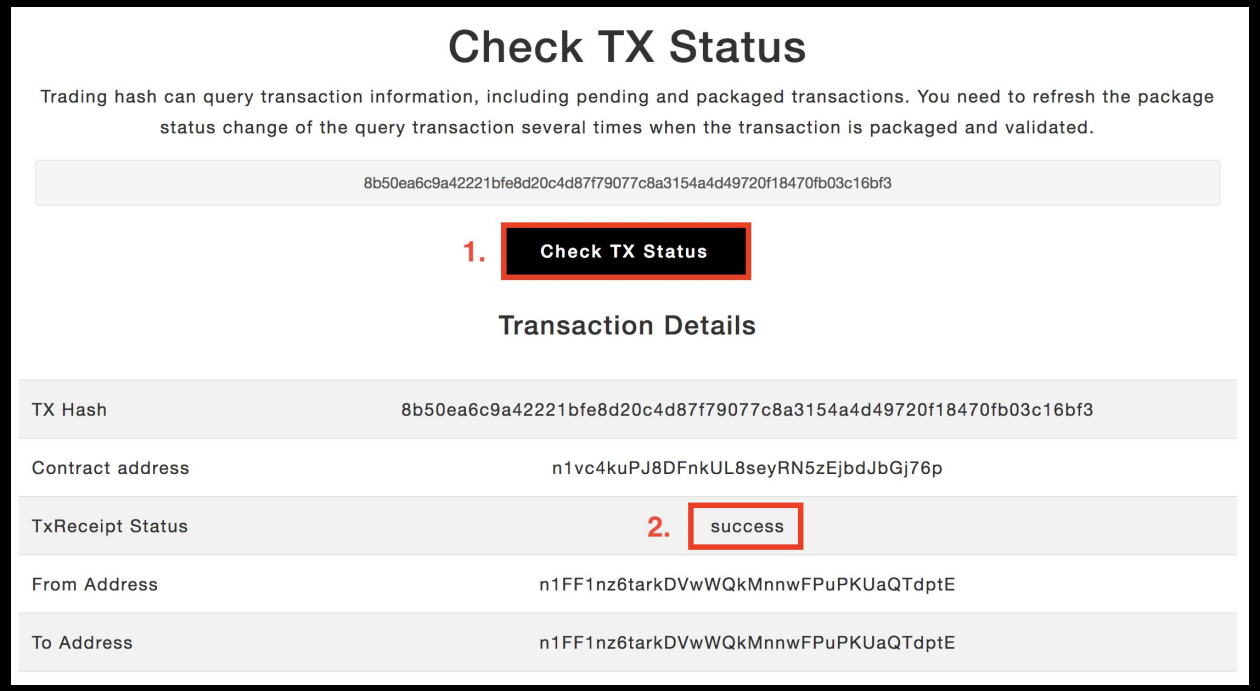
- When deploying a smart contract, the "To Address" must be the same as the address the From Address. Keep "Value/Amount To Send" at 0.
- Depending on the size of your smart contract and the current gas cost, you may need to increase your gas limit.

Hello World – Deployment

1. Upon pressing “Test”, your contract is executed and tested. If it does not pass testing, you will be presented a error message.
2. Upon submission, your Smart Contract will be deployed to the Nebulas Blockchain.
3. Your txhash will be presented after submission. The txhash is how you can locate your smart contract in the future.
4. Upon deployment, you will receive a contract address. This address is required to interact with your contract.

By clicking on the txhash, you can review the status of the submitted transaction.

Hello World – Verifying successful deployment



1. Click “Check TX Status”
2. Transaction status will usually become successful within 15 seconds

If your contract deployment remains “Pending” for a extended period of time, confirm that you have enough gas to pay the fee. Also make sure the contract test was successful.

If your transaction fails, verify the testing of the smart contract in the previous step.

Our TX Hash: 8b50ea6c9a42221bfe8d20c4d87f79077c8a3154a4d49720f18470fb03c16bf3

Hello World –View code

1. To view a deployed contract, you must have the deployment transaction hash.
2. The function we will first call in this smart contract is “greetings”. It will be expecting one argument for the variable “city”.

   The variable “visitor” was submitted during deployment in the previous steps.

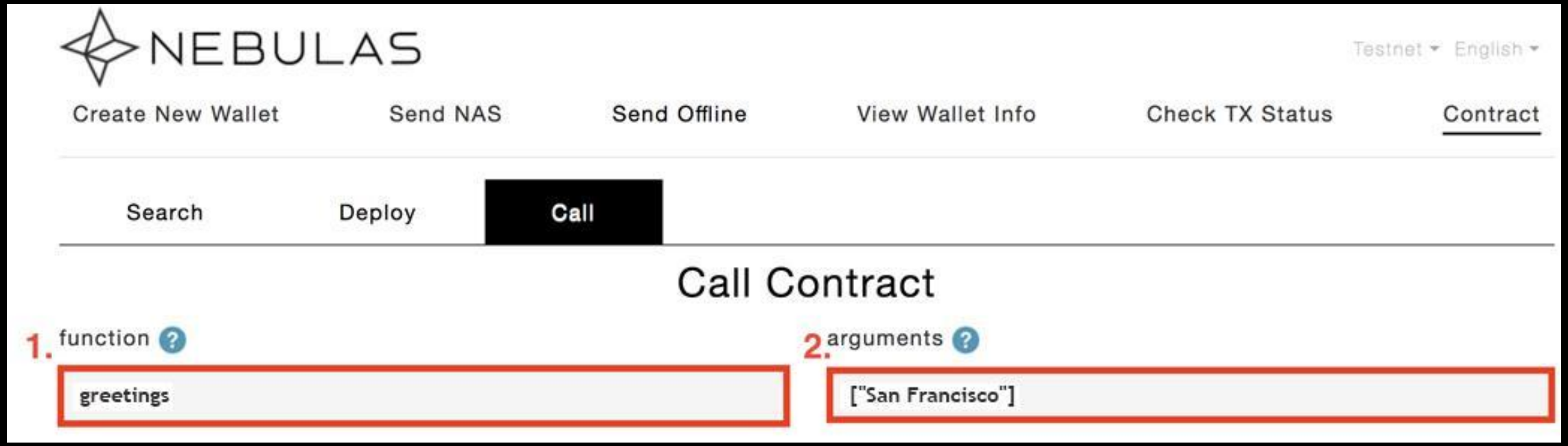
To view a deployed smart contract, you must use the transaction hash when the contract is deployed.

To locate the txhash of a deployed smart contract, use the RPC request “GetTransactionByContract”.

```
curl -i -H 'Content-Type: application/json' -X POST
http://localhost:8685/v1/user/getTransactionByContract
-d '{"address":"n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p"}'
```

NEBULAS

Hello World – Calling the Contract

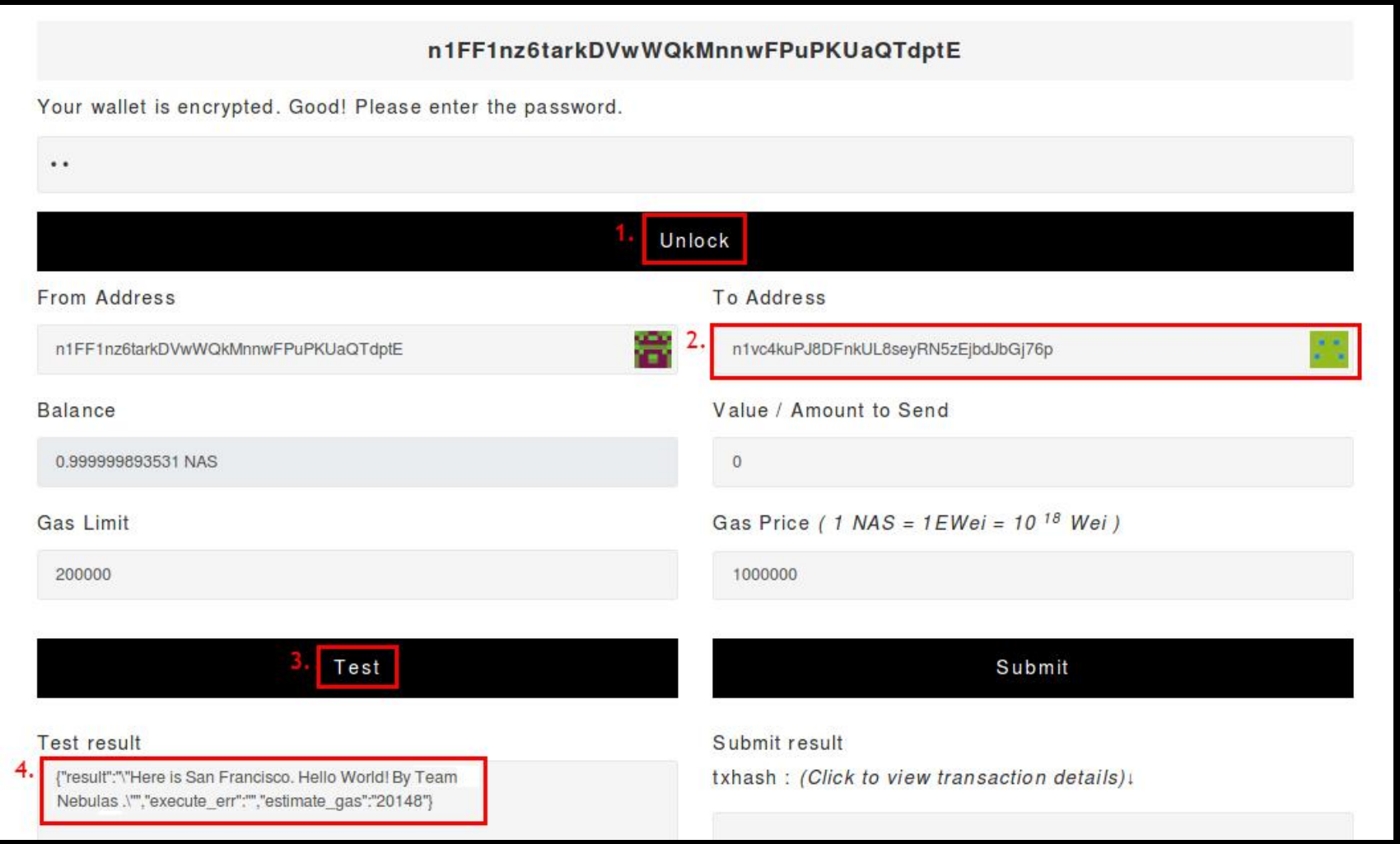


1. Enter the name of the function you want to enter. In our case, the function is called “greetings”
2. Enter any arguments that are required for the function. This is where we enter the “city” name. In our example, it is San Francisco – in array format

If your contract has multiple functions, you can enter the different name of each.

All arguments must be entered in array format.

Hello World – Unlock wallet and Enter Contract Address and query result



1. The calling of the smart contract is similar to the deployment process. Simply unlock your wallet.
2. Enter the correct smart contract address that we were given earlier during the initial deployment.
3. Test the arguments (previous slide) against the deployed smart contract.
4. Our result shows the hello world string including the visitor which we passed during deployment and the city which are passing during this request.
   visitor=Team Nebulas
   city=San Francisco

Our deployed contract address: n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p

NEBULAS

Hello World –View code, prepare to query contract information

1. View contract code based on transaction hash.
2. Now, we will call the who function in the contract.

The who function will return the visitor passed during the initialization/deployment process earlier on. During deployment, we entered "Team Nebulas" as a argument.

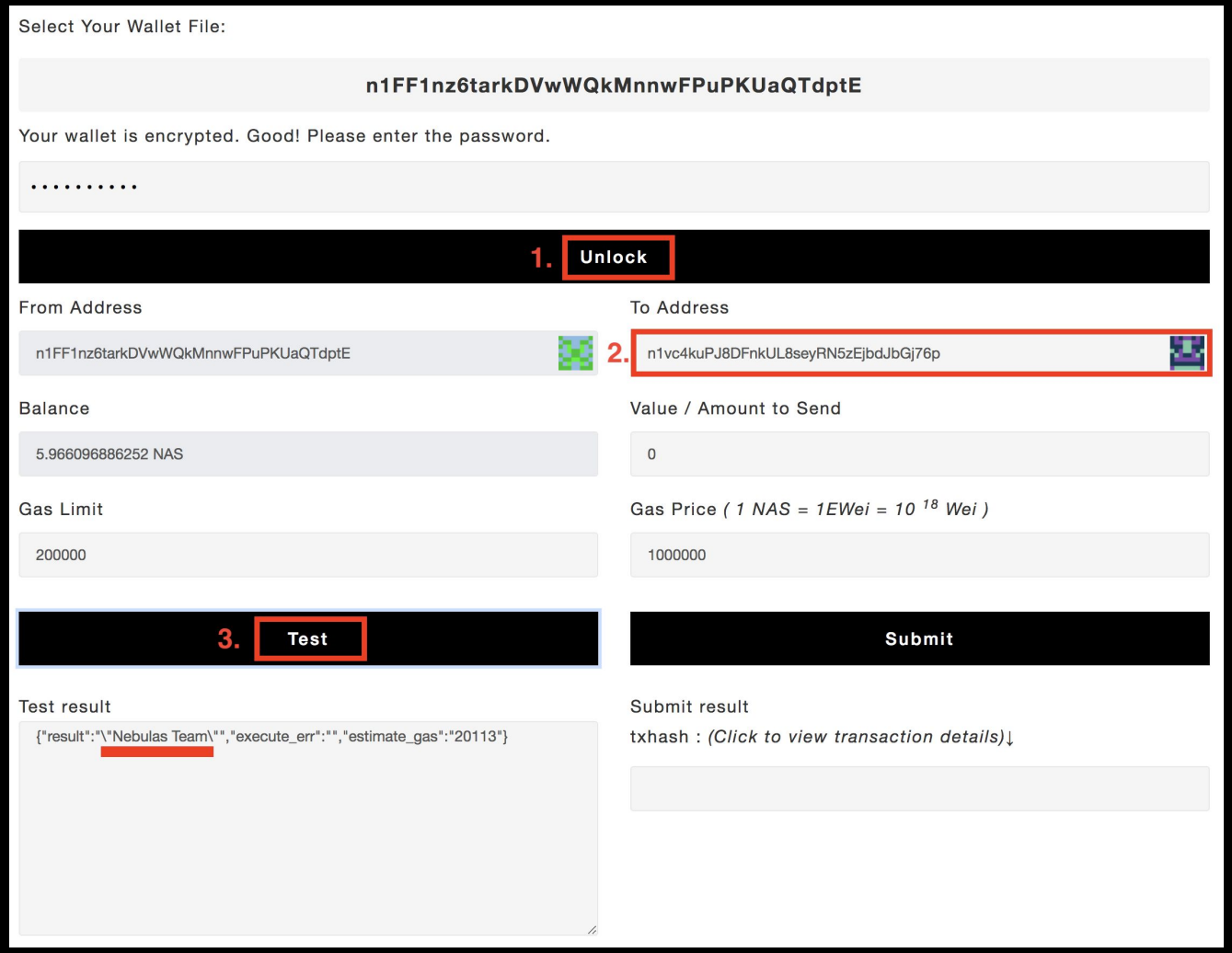
NEBULAS

Hello World – Fill in the parameters for obtaining contract information

1. Fill in the name of the contract function to be executed. In our example, we will execute the function “who”.
2. Fill in function parameters in array format. If you do not have any arguments, enter “[]” for empty/no arguments.

NEBULAS

Hello World –Return variable

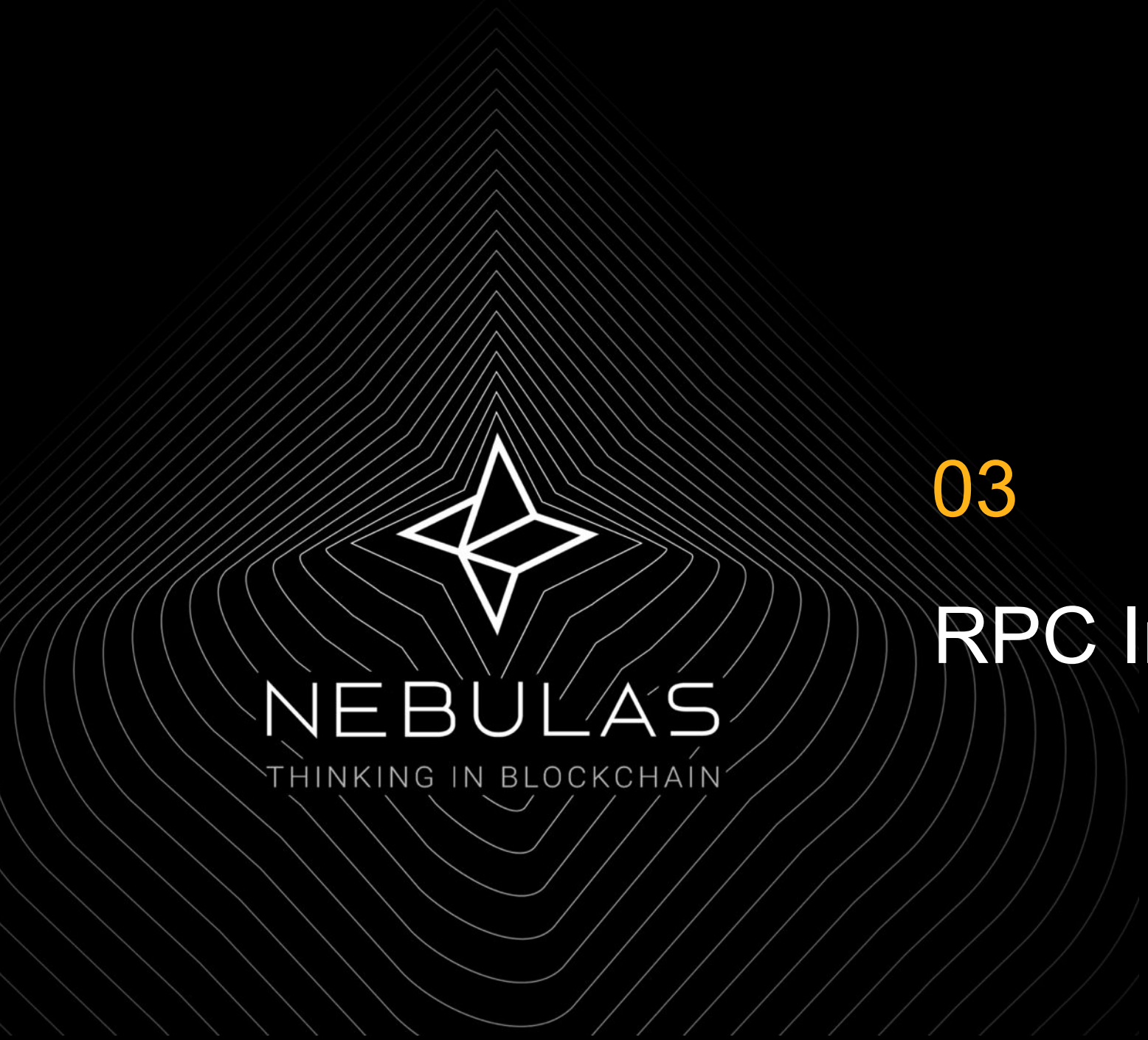


1. Unlock your wallet.
2. Enter he destination the contract address to call.
3. Clicking on the test will call the contract function and return the results of the who function.

Notice how we do not need to submit the transaction to receive the result since we are querying for a data result.

If we are pushing new data to the blockchain, the transaction must be submitted.

Our Deployed Smart Contract Address: n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p



03

RPC Interactions

NEBULAS Hello World – Query the result of the executed function with “Event.Trigger”

In the example smart contract, the function greetings has two methods of reviewing the data - return and Event.Trigger

return easily prints the results in our demo

Event.Trigger requires a running local node - this method also requires the data to be submitted to the blockchain. This method was commented out since we do not have a running local node.

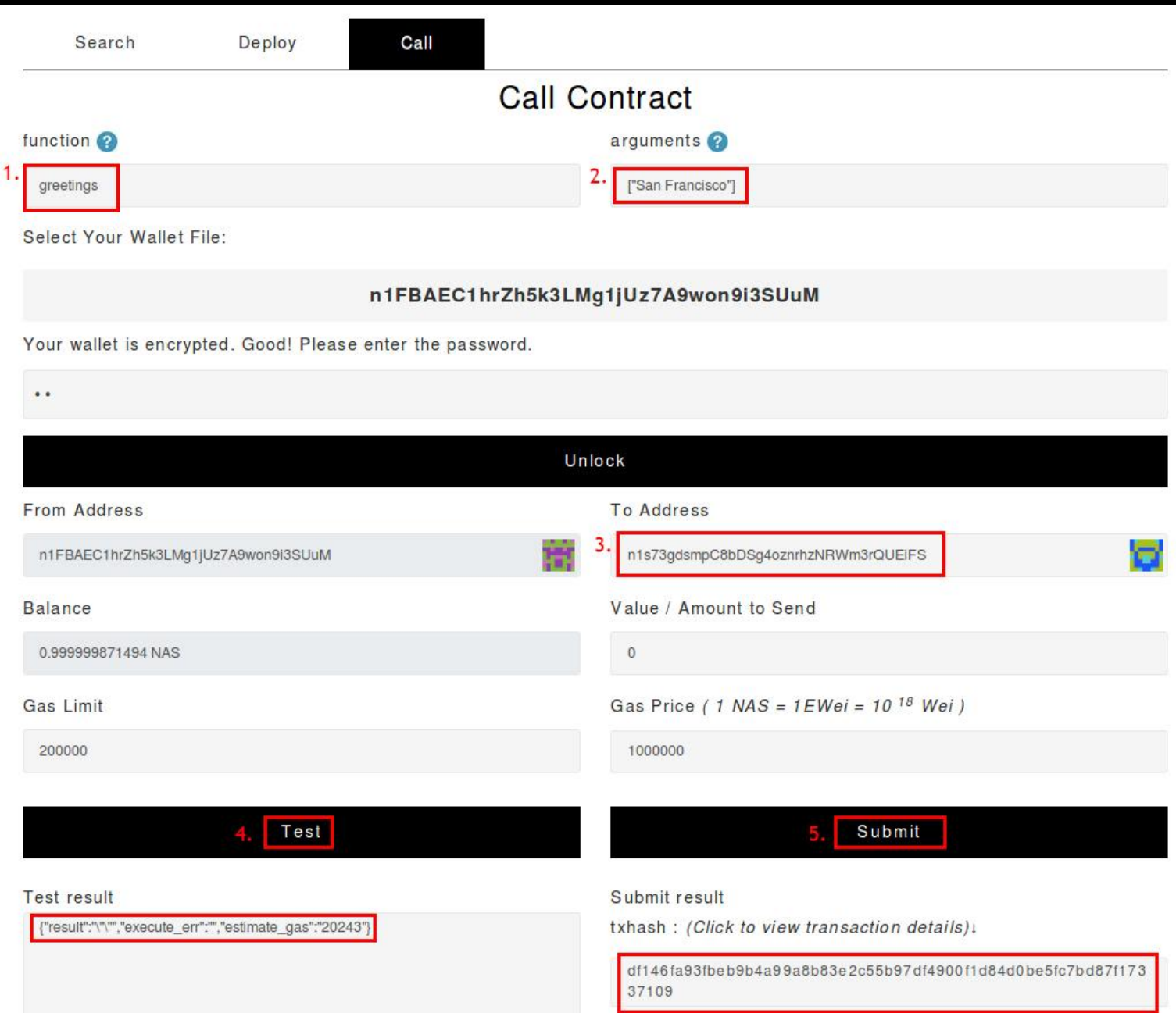
```
16    greetings(city) {//creating the greetings class/function that can be called by the user.
17        //This is a string that we will be printing (hello world).
18        //Event.Trigger("greetings", "Here is " + city + ". Hello World! By " + this.visitor + ".")//Prints via RPC
          - need local node.
19        return "greetings", "Here is " + city + ". Hello World! By " + this.visitor + ".";//Prints via execution -
          for web-wallet
```

To execute the greetings function with the Event.Trigger method with our example smart contract, simply uncomment line 18 and comment out line 19 and deploy the revised smart contract.

Why use Event.Trigger?

- can return a array of data
- Includes more information about the result
- Can include multiple events that took place during the execution.

NEBULAS Hello World – Submit transaction to execute contract function



1. Function to call - greetings.
2. Arguments to include - [“San Francisco”]
3. The address of the revised smart contract
4. Test the function and arguments against the deployed smart contract. Notice how the result will be empty.
5. Submit the transaction to the blockchain. Verify the transaction is successful by clicking on the txhash.

When testing a transaction that utilizes the Event.Trigger method, the result will be empty.
The Event.Trigger data will only be available after the transaction is submitted and accepted into the blockchain.
Upon successful submission into the chain, the Event.Trigger can be reviewed via RPC.

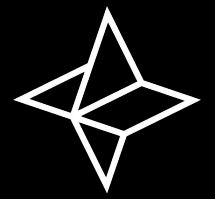
Hello World – Query the result of the executed function

RPC Request

```
› curl -i -H 'Content-Type: application/json' -X POST
  https://testnet.nebulas.io/v1/user/getEventsByHash
  -d '{"hash":"c32b … c3b6"}'
```

RPC Result


```
{
   "result":{
      "events":[
         {
            "topic": "chain.contract.greetings",
            "data":"\"Here is San Francisco. Hello World! By Nebulas Team.\""
         }
         ,{
            "topic":"chain.transactionResult",
            "data":"{\"hash\":\”c32b…c3b6\",\"status\":1,\"gas_used\":\"20229\",\"error\":\"\"}"
         }
      ]
   }
}
```

NEBULAS

THINKING IN BLOCKCHAIN

All RPC requests containing data must be submitted in JSON format.

All returned data will be in JSON format.

RPC requests must be executed on a running go-nebulas node.

RPC Requests and Interactions Details

Full documentation available at:

https://github.com/nebulasio/wiki/blob/master/rpc.md

https://github.com/nebulasio/wiki/blob/master/rpc_admin.md

Nebulas has two RPC request types.

- Public - /v1/user/
- Permissioned - /v1/admin/

The public methods can be ran locally and remotely. They do not require a password and do not interact with the wallet.

The permissioned methods need to be ran on the local machine and are used to interact with a wallet and the local node.

Some requests use GET but most use POST.

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

RPC can be used instead of the web wallet. This is useful when building your Dapp and the server side needs to talk with the blockchain.

In the following examples, we will use RESTful HTTP requests via CURL.

- The data payload of all requests must be JSON format.
- The response from the server is always JSON format.

For instructions on how to build/compile Nebulas, visit https://github.com/nebulasio/go-nebulas

NEBULAS

Hello World – RPC Interactions

Deploy Smart Contract Example

```
curl -i -H 'Accept: application/json' -X POST
http://localhost:8685/v1/admin/transactionWithPassphrase -H
'Content-Type: application/json' -d '{"transaction":
{"from":"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE","to":"n1FF1nz6ta
rkDVwWQkMnnwFPuPKUaQTdptE", "value":"0","nonce":8,
"gasPrice":"1000000","gasLimit":"2000000","contract":{"source":"Y2xhc3M.
.....gSGVBQi","sourceType":"js", "args":"[\"Nebulas Team\"]"}},
"passphrase": "passphrase"}'
```

JSON Request

```
{
  "transaction":{
    "from":"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE",
    "to":"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE",
    "value":"0",
    "nonce":8,
    "gasPrice":"1000000",
    "gasLimit":"2000000",
    "contract":{
      "source":"Y2xhc3M.....gSGVBQi",
      "sourceType":"js",
      "args":"[\"Nebulas Team\"]"
    }
  },
  "passphrase":"passphrase"
}
```

During this request, we made a POST request to the local node. The executed RPC command is "transactionWithPassphrase" which is a private request.

When deploying a contract, we must include contract followed by the source, type and arguments.

The contract source data should be submitted in Base64 encoding.

Arguments must be entered in array format and double quotes must be escaped.

Once successfully submitted, you will receive a transaction hash and contract address.

Result of Request

```
{
  "result":{
    "txhash":" 8b50ea6c9a42221bfe8d20c4d87f79077….20f18470fb03c16bf3 ",
    "contract_address":" n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p"
  }
}
```

NEBULAS

Hello World – RPC Interactions

Reviewing Status of Deployed Contract Transaction

```
curl -i -H 'Content-Type: application/json' -X POST
http://localhost:8685/v1/user/getTransactionReceipt -d
'{"hash":" 8b50ea6c9a422……9720f184fb03c16bf3 "}'
```

JSON Request

```
{
  "hash":"8b50ea6c9a42221bfe8d20c4d87f79077
          c8a3154a4d49720f18470fb03c16bf3"
}
```

The same request can be made for a standard transactions (non-contract deployment) as well but no contract address will be given. If data was submitted, it will be base64 encoded.

During this request, we made a POST request to the local node. The executed RPC command is “getTransactionReceipt” which is a public request.

All we need to include is the transaction hash.

The result shows the transaction details including type, submitted data, the new contract address and status.

Status Codes:
1 = Successful
2 = Pending
3 = Failed

Result of Request


```
{
  "result":{
    "hash":"8b50ea6c9a42221bfe8d20c4d87f79077c8a3154a4d49720f18470fb0
3c16bf3",
    "chainId":100,
    "from":"n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p",
    "to":"n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p",
    "value":"0",
    "nonce":"25",
    "timestamp":"1529427936",
    "type":"deploy",
    "data":"eyJTb3VyY2jcmlwdCYWiliwiQ…XJncyl6IltcIkRLXCJdIn0=",
    "gas_price":"1000000",
    "gas_limit":"200000",
    "contract_address":"n1ohRhzEJ11AVEqHysgMuMdStb7r3baPkaZ",
    "status":1,
    "gas_used":"21653",
    "execute_error":"",
    "execute_result":"",
  }
}
```

NEBULAS

Hello World – RPC Interactions

Submit Data to Contract

curl -i -H 'Accept: application/json' -X POST http://localhost:8685/v1/admin/transactionWithPassphrase -H 'Content-Type: application/json' -d '{"transaction":{"from":"n1FBAEC1hrZh5k3LMg1jUz7A9won9i3SUuM","to":"n1ohRhzEJ11AVEqHysgMuMdStb7r3baPkaZ","value":"0","nonce":27,"gasPrice":"1000000","gasLimit":"2000000","contract":{"function":"greetings","args":"[\“San Francisco\"]"}}, "passphrase": " passphrase "}'

JSON Request


```
{
  "transaction":{
    "from":"n1FF1nz6tarkDVwWQkMnnwFPuPKUaQTdptE",
    "to":"n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p",
    "value":"0",
    "nonce":9,
    "gasPrice":"1000000",
    "gasLimit":"2000000",
    "contract":{
      "function":"greetings",
      "args":"[\“San Francisco\"]"
    }
  },
  "passphrase":"passphrase"
}
```


During this request, we made a POST request to the local node. The executed RPC command is “transactionWithPassphrase” which is a private request.

When submitting content to a contract, we must include contract followed by function and arguments. No source or type is required

Arguments must be entered in array format and double quotes must be escaped.

Once successfully submitted, you will receive a transaction hash but no contract address due to no contract being deployed.

If we wanted to update our greetings variable, we can submit a new argument and the newest variable will be displayed via **getEventsByHash.**

Result of Request


```
{
  "result":{
    "txhash":"a8a44ed34a5fa9691c141b8c…….91d696a1d4304a4160aa0eb21",
    "contract_address":""
  }
}
```

NEBULAS

Hello World – RPC Interactions

Return the “Hello World” String

```
curl -i -H 'Content-Type: application/json' -X POST
http://localhost:8685/v1/user/getEventsByHash -d
'{"hash":"
c32bd5f6ffc1ddd3b3c7ef86b04159c1831c7628717e08
04b7d8d2b5feb4c3b6 "}'
```

JSON Request

```
{
  "hash":"c32bd5f6ffc1ddd3b3c7ef86b04159c1831
          c7628717e0804b7d8d2b5feb4c3b6"
}
```

During this request, we made a POST request to the local node. The executed RPC command is “getEventsByHash” which is a public request and returns the events list of transaction.

All we need to include is the transaction hash.

The result shows the transaction details including all the events. The event we are interested in is “chain.contract.greetings” which has our hello world string in the data field.

Result of Request


```
{
  "result":{
    "events":[
      {
        "topic":"chain.contract.greetings",
        "data":"Here is San Francisco. Hello World! By Nebulas Team."
      },
      {
        "topic":"chain.transactionResult",
        "data":{
          "hash":" c32bd5f6ffc1ddd3b3c7ef86b04….7e0804b7d8d2b5feb4c3b6",
          "status":1,
          "gas_used":"20123",
          "error":"",
          "execute_result":"Nebulas Team"
        }
      }
    ]
  }
}
```

NEBULAS

Locating Your Smart Contract TX Hash

Returns the deployment tx hash of a contract.

```
curl -i -H 'Content-Type: application/json' -X POST
http://localhost:8685/v1/user/getTransactionByContract -d
'{"address":"n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p"}'
```

JSON Request

```
{
  "address":"n1vc4kuPJ8DFnkUL8seyRN5zEjbdJbGj76p"
}
```

During this request, we made a POST request to the local node. The executed RPC command is “getTransactionByContract” which is a public request and returns the initial deployment details including the transaction hash.

All we need to include is the contract address.

The result shows the transaction details including the contract tx hash, the contract data in base64 encoding and the status of the transaction.

Result of Request


```
{
  "result":{
    "hash":"e73bb5b94da91896d3f57d5d6b3d2ff87...e96054ec365b7ea07f3a0",
    "chainId":100,
    "from":"n1FBAEC1hrZh5k3LMg1jUz7A9won9i3SUuM",
    "to":"n1FBAEC1hrZh5k3LMg1jUz7A9won9i3SUuM",
    "value":"0",
    "nonce":"9",
    "timestamp":"1529427936",
    "type":"deploy",
    "data":"eyJTb3VyY2VU.....iwiQXJncyI6IltcIkRLXCJdIn0=",
    "gas_price":"1000000",
    "gas_limit":"200000",
    "contract_address":"n1ohRhzEJ11AVEqHysgMuMdStb7r3baPkaZ",
    "status":1,
    "gas_used":"21653",
    "execute_error":"",
    "execute_result":""
  }
}
```

Create Wallet via CLI and RPC

Wallets can be created directly via CLI and RPC request

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

JSON Request

```
{
  " passphrase":"passphrase"
}
```

To create a wallet via command line/terminal, from your compiled directory, enter “./neb account new” and follow the instructions.

During this request, we made a POST request to the local node. The executed RPC command is “/account/new” which is a private request and returns the events list of transaction.

All we need to include is the passphrase of the wallet. You will need the passphrase to interact with the newly created wallet.

Once the request is submitted, you will be returned the address of your new wallet. A new keystore file will be created containing your wallet information.

Result of Request

```
{
  "result":{
    "address":"n1U4iy97kmG3yr6hJtzED4LyujtXxjwQDSy"
  }
}
```

Available SDK's

Nebulas offers SDK packages and wallets for the most popular programming languages and mobile operating systems.

- NasNano Wallet available on the Apple Store
- SDK Package available on Github

- NasNano Wallet available on the Google Store
- SDK Package available on Github

- Web Wallet available for All Browsers across all Operating Systems
- Web Extension Wallet available on the Chrome Web Store

SDK Packages available on Github

All packages available at github.com/nebulasio

NEBULAS

nebulas.io

t.me/nebulasio

twitter.com/nebulasio