



Техническое описание платформы
Waves Enterprise
Выпуск 1.6.0

<https://wavesenterprise.com>

янв. 26, 2024

1	Документация блокчейн-платформы Waves Enterprise	1
2	Системные требования	3
3	Развертывание платформы в ознакомительном режиме (Sandbox)	4
4	Развертывание платформы с подключением к Mainnet	13
5	Развертывание платформы в частной сети	18
6	Примеры конфигурационных файлов ноды	39
7	Лицензии блокчейн-платформы Waves Enterprise	48
8	Комиссии в сети «Waves Enterprise Mainnet»	50
9	Инструментарий gRPC	53
10	Методы REST API	62
11	Разработка и применение смарт-контрактов	135
12	JavaScript SDK	149
13	Обмен конфиденциальными данными	170
14	Управление ролями участников	174
15	Подключение и удаление нод	176
16	Запуск ноды с созданным снимком данных	178
17	Архитектура	180
18	Протокол работы блокчейна Waves-NG	184
19	Подключение новой ноды к сети	186
20	Активация функциональных возможностей	189

21	Анкоринг	192
22	Механизм создания снимка данных	195
23	Смарт-контракты	198
24	Транзакции блокчейн-платформы	207
25	Атомарные транзакции	231
26	Алгоритмы консенсуса	234
27	Криптография	243
28	Роли участников	246
29	Клиент	248
30	Генераторы	257
31	Сервисы авторизации и подготовки данных	259
32	Официальные ресурсы и контакты	290
33	Словарь терминов	291
34	Что нового в Waves Enterprise	296

Документация блокчейн-платформы Waves Enterprise

Блокчейн-платформа Waves Enterprise - это комплексная система распределенного реестра, позволяющая формировать как публичные, так и приватные сети блокчейна для решения задач в корпоративном и государственном секторах.

1.1 Что такое блокчейн?

Блокчейн - непрерывная последовательная цепочка взаимосвязанных блоков, содержащих какую-либо информацию. Эта цепочка пополняется новыми блоками. Процесс создания блока называется *майнингом*. Каждый блок содержит хэш-сумму данных предыдущего блока. Это делает невозможным последующее изменение содержимого любого из блоков, поскольку для этого необходимо изменить содержимое блоков на протяжении всей цепочки на всех узлах.

На корпоративном уровне технология блокчейна используется для создания **систем распределенного реестра**. Система распределенного реестра не имеет единого центра управления, а данные одновременно хранятся на всех узлах сети. Для обновления данных применяются алгоритмы консенсуса - автоматизированного подтверждения наличия одной и той же копии данных на всех узлах сети.

Такая система позволяет обеспечить безопасность передаваемых данных и решить проблему доверия между участниками сети.

1.2 Для чего предназначена блокчейн-платформа WE?

Блокчейн-платформа Waves Enterprise позволяет решать широкий спектр задач бизнеса и государственных структур:

- Ускорение делопроизводства — благодаря автоматизации бизнес-процессов и уменьшению количества посредников.
- Защита данных от изменений извне — с помощью шифрования и многоэтапной проверки каждой операции в сети.

- Реализация собственной бизнес-логики любой сложности — за счет широких возможностей по разработке смарт-контрактов и удобных инструментов интеграции с блокчейном.
- Достижение взаимного доверия между участниками бизнес-процессов — благодаря гарантированному учету мнения большинства в децентрализованной сети.

1.3 Решения на базе блокчейн-платформы WE

Компанией Waves Enterprise были реализованы следующие сервисы на базе блокчейн-платформы:

- *Waves Enterprise Mainnet* - основная блокчейн-сеть Waves Enterprise.
- *Voting* - блокчейн-сервис для организации и проведения электронных голосований. Он позволяет проводить прозрачные опросы, корпоративные голосования и референдумы, полностью защищенные от фальсификаций.
- *Data oracles* - сервис для безопасного перенаправления в блокчейн данных из внешних источников, необходимых для реализации бизнес-логики и смарт-контрактов.

С частными проектами, реализованными на базе блокчейн-платформы Waves Enterprise, вы можете ознакомиться [на нашем официальном сайте](#).

Системные требования

Ниже приведены аппаратные и системные требования к компьютеру, на котором разворачивается нода блокчейн-платформы Waves Enterprise.

Вариант	vCPU	RAM	SSD	Режим работы JVM
Минимальные требования	2+	4Gb	50Gb	java -Xmx2048M -jar
Рекомендуемые требования	2+	4+ Gb	50+ Gb	java -Xmx4096M -jar

Подсказка: «Xmx» - флаг, определяющий максимальный размер доступной для JVM памяти.

Требования к окружению для платформы Waves Enterprise

- Oracle Java SE 11 (64-bit) или OpenJDK 11 и выше
- Docker CE
- Docker-compose

Развертывание платформы в ознакомительном режиме (Sandbox)

Для ознакомления с блокчейн-платформой Waves Enterprise вам доступна бесплатная версия, запускающаяся в Docker-контейнере. Для ее установки и использования не требуется лицензия, высота блокчейна ограничена 30000 блоками. При времени раунда блока, равном 30 секундам, время полноценной работы платформы в ознакомительном режиме составляет 10 дней.

При развертывании в ознакомительном режиме вы получите локальную версию блокчейн-платформы, которая позволяет протестировать основные функции:

- отправка транзакций;
- прием данных из блокчейна;
- установка и вызов смарт-контрактов;
- передача конфиденциальных данных между нодами;
- тестирование мониторинга ноды при помощи InfluxDB и Grafana.

Взаимодействие с платформой может осуществляться как через клиентское приложение, так и через интерфейсы gRPC и REST API.

3.1 Установка платформы

Перед началом установки убедитесь, что на вашей машине установлены Docker Engine и Docker Compose. Также ознакомьтесь с [системными требованиями](#) к блокчейн-платформе.

Обратите внимание, что для выполнения команд на ОС Linux могут потребоваться права администратора (префикс `sudo` с последующим вводом пароля администратора).

1. Создайте рабочую директорию и поместите в нее файл **docker-compose.yml**. Этот файл вы можете скачать из официального репозитория [Waves Enterprise в GitHub](#), выбрав самый свежий релиз платформы, либо в терминале при помощи утилиты `wget`:

```
wget https://github.com/waves-enterprise/WE-releases/releases/download/v1.6.0/docker-  
compose.yml
```

- Откройте терминал и перейдите в директорию, содержащую скачанный файл `docker-compose.yml`. Запустите Docker-контейнер для развертывания платформы:

```
docker run --rm -ti -v $(pwd):/config-manager/output wavesenterprise/config-manager:v1.6.
↵0
```

Дождитесь сообщения об окончании развертывания:

```
INFO [launcher] WE network environment is ready!
```

В результате будут созданы 3 ноды с автоматически сгенерированными учетными данными. Информация о нодах доступна в файле `./credentials.txt`:

```
node-0
blockchain address: 3Nzi7jJYn1ek6mMvtKbPhehxMQarAz9YQvF
public key:        7cLSA5AnvZgiL8CnoffwxXPkpQhvviJC9eyWBKSUsi58
keystore password: 0EtrVSL9gzj087jYx-gIoQ
keypair password:  JInWk1kauuZDHGXfJ-rNXQ
API key:           we

node-1
blockchain address: 3Nxz6BYyk6CYrqH4Zudu5UYoHU6w7NXbZMs
public key:        VBkFFQmaHzv3YMiWLhh4qsCn4prUvteWsjgiiHEpWEp
keystore password: FsUp3xiX_NF-bQ9gw6t0sA
keypair password:  Qf2rBgBT9pnozLP0k01yYw
API key:           we

node-2
blockchain address: 3NtT9onn8VH1DsbioPVBuhU4pnuCtBtbsTr
public key:        8YkDPLsek5VF5bNY9g2dxAthd9AMmmRyvMPTv1H9iEpZ
keystore password: T77fAroHavbWCS6Uir2oFg
keypair password:  bELB4EU1GDd5rS-RId_6pA
API key:           we
```

- Запустите готовую конфигурацию:

```
docker-compose up -d
```

Сообщение при успешном запуске нод и сервисов:

```
Creating network "platf_we-network" with driver "bridge"
Creating node-2      ... done
Creating postgres    ... done
Creating node-0      ... done
Creating node-1      ... done
Creating auth-service ... done
Creating crawler     ... done
Creating data-service ... done
Creating frontend    ... done
Creating nginx-proxy  ... done
```

После успешного запуска контейнеров клиент платформы будет доступен в браузере локально по адресу **127.0.0.1** или **localhost**. Интерфейс REST API ноды располагается по адресу **127.0.0.1/node-0** или **localhost/node-0**.

Обратите внимание, что по умолчанию для локального nginx-сервера платформы предоставляется порт **80:80**. Если на вашей ОС этот порт занят другим приложением, измените параметр ports секции **nginx-proxy** в файле **docker-compose.yml**, выбрав доступный порт:

```
nginx-proxy:
  image: nginx:latest
  hostname: nginx-proxy
  container_name: nginx-proxy
  ports:
    - "81:80"
```

После этого клиент и REST API будут доступны по адресу **127.0.0.1:81** или **localhost:81**.

4. Для остановки запущенных нод выполните команду:

```
docker-compose down
```

3.2 Последующие действия

3.2.1 Настройка мониторинга ноды в ознакомительном режиме

Нода, запущенная в ознакомительном режиме, имеет возможность подключения мониторинга ее показателей при помощи **InfluxDB** и **Grafana**.

Для установки и конфигурации мониторинга прежде всего остановите запущенные контейнеры при помощи команды `docker-compose down`.

1. Установите Docker-образ Grafana:

```
docker run -d --name=grafana -p 3000:3000 grafana/grafana
```

2. Установите Docker-образ InfluxDB:

```
docker run -d --name influxdb -p 8086:8086 -e INFLUXDB_DB=sandbox_influxdb -e
↪INFLUXDB_ADMIN_USER=sandbox_influxdb_admin -e INFLUXDB_ADMIN_
↪PASSWORD=sandbox_influxdb_pass quay.io/influxdb/influxdb:v2.0.3
```

Подсказка: Обратите внимание, что эти шаги необходимы для предварительной установки Grafana и InfluxDB. Если вы намерены использовать Docker Compose, их можно пропустить: после приведенного ниже изменения файла `docker-compose.yml` и запуска контейнеров образы обоих сервисов будут скачаны и установлены автоматически, если не были установлены ранее.

3. Перейдите в директорию `./configs/nodes/node-0` и откройте файл конфигурации ноды **node.conf** с правами администратора. В конец конфигурационного файла добавьте следующие параметры сервисов мониторинга:

```
# Performance metrics
kamon {
  # Set to "yes", if you want to report metrics
  enable = yes

  # An interval within metrics are aggregated. After it, them will be sent to
```

(continues on next page)

(продолжение с предыдущей страницы)

```

→ the server
metric.tick-interval = 1 second

# Reporter settings
influxdb {
  hostname = "localhost"
  port = 8086
  database = "sandbox_influxdb"
  time-units = "ms"

  authentication {
    user = "sandbox_influxdb_admin"
    password = "sandbox_influxdb_pass"
  }

  environment.host = "node-0"
}
}

# Non-aggregated data (information about blocks, transactions, ...)
metrics {
  enable = yes
  node-id = "node-0"

  influx-db {
    uri = "http://localhost:8086"
    db = "sandbox_influxdb"

    username = "sandbox_influxdb_admin"
    password = "sandbox_influxdb_pass"

    batch-actions = 100
    batch-flash-duration = 1s
  }
}
}

```

- Откройте файл **docker-compose.yml** и добавьте параметры развертывания сервисов мониторинга.

Полный листинг файла **docker-compose.yml**

- Запустите платформу и войдите в клиентское приложение Grafana. Для этого запустите контейнеры платформы при помощи команды **docker-compose up -d** и дождитесь завершения развертывания.

Затем откройте клиент Grafana, доступный на порту **3000**:

<http://127.0.0.1:3000/login>

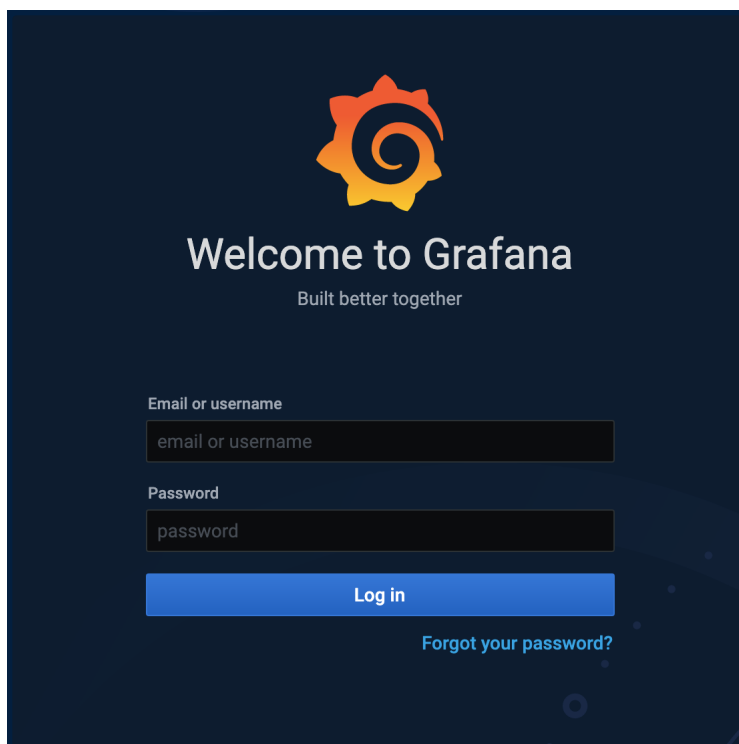
Для аутентификации в клиенте используйте данные, установленные в подкатегории **environment** раздела **services: grafana:** файла **docker-compose.yml**:

```

services:
  grafana:
    image: grafana/grafana:latest
    hostname: grafana

```

(continues on next page)



(продолжение с предыдущей страницы)

```
container_name: grafana
environment:
  GF_SECURITY_ADMIN_USER: 'admin'
  GF_SECURITY_ADMIN_PASSWORD: 'pass'
```

6. Подключите базу данных платформы к Grafana. Для этого нажмите на вкладку **Configuration** (Конфигурация) и перейдите в категорию **Data Sources** (Источники данных):

Добавьте новый источник данных, нажав на кнопку **Add Data Source**, и из предложенного списка выберите **InfluxDB**.

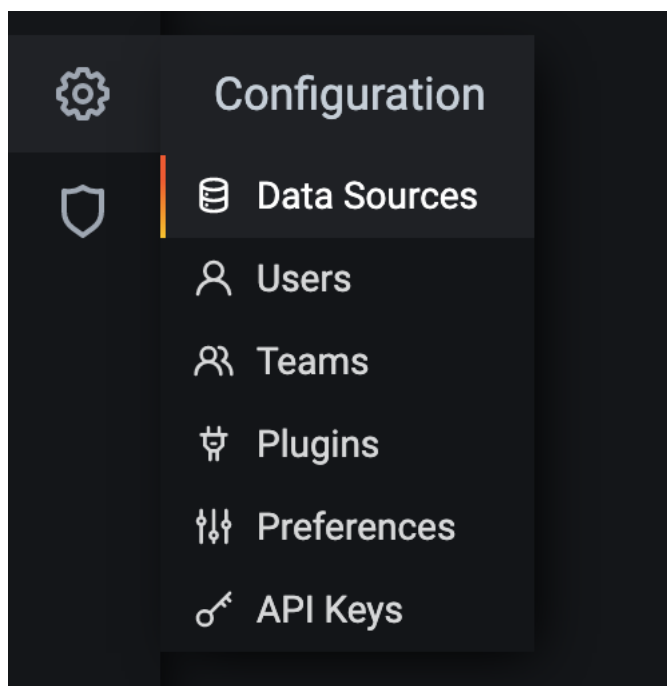
Затем настройте базу данных, используемую платформой.

Секция HTTP, поле URL: **http://influxdb:8086**

Секция InfluxDB Details: введите данные, установленные в подкатегории **environment** раздела **services: influxdb**: файла **docker-compose.yml**:

```
influxdb:
  image: influxdb
  hostname: influxdb
  container_name: influxdb
  environment:
    - INFLUXDB_DB=influxdb // database field
    - INFLUXDB_ADMIN_USER=admin // user field
    - INFLUXDB_ADMIN_PASSWORD=pass // password field
```

Нажмите кнопку **Save & Test**. При успешном подключении базы данных появится сообщение **Data source is working**.



Settings

Name ⓘ InfluxDB Default ☒

Query Language

InfluxQL ▾

HTTP

URL ⓘ http://influxdb:8086

Access Server (default) ▾ [Help >](#)

Whitelisted Cookies ⓘ

Auth

Basic auth	<input type="checkbox"/>	With Credentials ⓘ	<input type="checkbox"/>
TLS Client Auth	<input type="checkbox"/>	With CA Cert ⓘ	<input type="checkbox"/>
Skip TLS Verify	<input type="checkbox"/>		
Forward OAuth Identity ⓘ	<input type="checkbox"/>		

7. Настройте визуализацию данных в Grafana

Для этого нажмите на кнопку **+** (Добавить) на левой панели клиента и перейдите на вкладку **Import** (Импортировать).

В окно **Import via panel json** (Импортировать через json панели) вставьте файл json, содержащий параметры визуализации данных.

Затем нажмите кнопку **Load** (Загрузить). В окне **Options** выберите желаемое имя панели, либо оставьте название **sandbox**, установленное по умолчанию.

Нажмите кнопку **Import** (Импортировать).

После этого вы сможете просматривать показатели вашей демо-платформы на созданной панели. Перейдите во вкладку **Dashboard/Manage** и выберите панель **sandbox** для ее открытия.

Смотрите также

Развертывание платформы в ознакомительном режиме (Sandbox)

Платформа в ознакомительном режиме: устранение ошибок

3.2.2 Платформа в ознакомительном режиме: устранение ошибок

1. Ошибка при запуске контейнера для развертывания платформы:

```
2021-02-07 16:26:59,289 INFO [launcher] ./output/configs/nodes/node-0/accounts.conf
2021-02-07 16:27:07,432 INFO [launcher] ./output/configs/nodes/node-1/accounts.conf
2021-02-07 16:27:19,948 INFO [launcher] ./output/configs/nodes/node-2/accounts.conf
2021-02-07 16:27:28,023 INFO [launcher] Creating blockchain section for the node config
↳files
Traceback (most recent call last):
  File "launcher.py", line 304, in <module>
    create_new_network()
  File "launcher.py", line 228, in create_new_network
    create_blockchain(addresses, nodes)
  File "launcher.py", line 106, in create_blockchain
    network_participants.append(ConfigFactory.from_dict({"public-key": addresses.get_
↳keys()[i],
IndexError: list index out of range
```

Причина: Повторный запуск контейнера.

Решение: Удалите рабочую директорию с файлами платформы и начните заново со скачивания файла *docker-compose.yml*.

2. Ошибка при запуске платформы после успешного развертывания:

```
ERROR: for node-1 Cannot create container for service node-1: Conflict. The container
↳name "/node-1" is already in use by container
↳"47cfd7a517e160d201ae969b24392ca0bc2b9720c73e7324dac45daaa24814cb". You have to remove
↳(or rename) that conCreating node-2 ... error

ERROR: for node-2 Cannot create container for service node-2: Conflict. The container
↳name "/node-2" is already in use by container "ccd28832f1fb5457186e50d5e5Creating node-
↳0 ... error
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tainer to be able to reuse that name.
```

```
ERROR: for node-0 Cannot create container for service node-0: Conflict. The container
↳ postgres ... error
eb8ac184f88195f1a560ee8ef7ade5c46f899d". You have to remove (or rename) that container
↳ to be able to reuse that name.
```

```
ERROR: for postgres Cannot create container for service postgres: Conflict. The
↳ container name "/postgres" is already in use by container
↳ d4bc6d758faafcc9b2bc352b9cbcc5dc909f2959059b7abf17db0088916506d1". You have to remove
↳ (or rename) that container to be able to reuse that name.
```

```
ERROR: for node-1 Cannot create container for service node-1: Conflict. The container
↳ name "/node-1" is already in use by container
↳ 47cfd7a517e160d201ae969b24392ca0bc2b9720c73e7324dac45daaa24814cb". You have to remove
↳ (or rename) that container to be able to reuse that name.
```

```
ERROR: for node-2 Cannot create container for service node-2: Conflict. The container
↳ name "/node-2" is already in use by container
↳ ccd28832f1fb5457186e50d5e58f98ed3b35c944931589a42a0262a205a17393". You have to remove
↳ (or rename) that container to be able to reuse that name.
```

```
ERROR: for node-0 Cannot create container for service node-0: Conflict. The container
↳ name "/node-0" is already in use by container
↳ 7ed421ac8c8c5ca91a916970c1eb8ac184f88195f1a560ee8ef7ade5c46f899d". You have to remove
↳ (or rename) that container to be able to reuse that name.
```

```
ERROR: for postgres Cannot create container for service postgres: Conflict. The
↳ container name "/postgres" is already in use by container
↳ d4bc6d758faafcc9b2bc352b9cbcc5dc909f2959059b7abf17db0088916506d1". You have to remove
↳ (or rename) that container to be able to reuse that name.
```

```
ERROR: Encountered errors while bringing up the project.
```

Причина: Контейнеры отдельных нод или сервисов уже используются запущенными контейнерами.

Решение: Если вам необходимо пересобрать платформу заново, остановите ее при помощи команды `docker-compose down`. При помощи команды `docker stop [ID контейнера]` остановите запущенные контейнеры нод и сервисов. Вы можете ввести несколько ID запущенных контейнеров подряд через пробел или остановить все контейнеры при помощи команды `docker stop $(docker ps -a -q)`. Затем при помощи команды `docker rm [ID контейнера]` удалите их. ID используемых контейнеров доступны в отчетах об ошибках, подобных приведенному выше. Вы можете удалить несколько контейнеров или все используемые контейнеры одной командой при помощи аналогичного синтаксиса.

3. Ошибка при запуске контейнеров:

```
ERROR: for nginx-proxy Cannot start service nginx-proxy: driver failed programming
↳ external connectivity on endpoint nginx-proxy
↳ (86add881e45535e666443cb00e6a6cb66f79a906e412d4f78d2db9d81c6d63d7): Error starting
↳ userland proxy: listen tcp 0.0.0.0:80: bind: address already in use
```

```
ERROR: for nginx-proxy Cannot start service nginx-proxy: driver failed programming
↳ external connectivity on endpoint nginx-proxy
↳ (86add881e45535e666443cb00e6a6cb66f79a906e412d4f78d2db9d81c6d63d7): Error starting
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪userland proxy: listen tcp 0.0.0.0:80: bind: address already in use  
ERROR: Encountered errors while bringing up the project.
```

Причина: Порт 80:80 на вашей машине занят другим приложением.

Решение: Остановите контейнеры при помощи команды `docker-compose down`. Затем измените параметр `ports` секции `nginx-proxy` в файле `docker-compose.yml`, выбрав свободный порт:

```
nginx-proxy:  
  image: nginx:latest  
  hostname: nginx-proxy  
  container_name: nginx-proxy  
  ports:  
    - "81:80"
```

После этого клиент и REST API будут доступны по адресу **127.0.0.1:81** или **localhost:81**. Остальные сервисы будут доступны по адресам со своими прежними портами.

4. Ошибка при переходе по адресу 127.0.0.1 или localhost в браузере Mozilla Firefox:

```
SSL_ERROR_RX_RECORD_TOO_LONG
```

Причина: Вход на localhost по умолчанию выполняется через HTTPS, однако при разворачивании платформы в ознакомительном режиме SSL не предусмотрено.

Решение: Введите полный адрес, используя HTTP: `http://127.0.0.1` или `http://localhost`.

Смотрите также

Развертывание платформы в ознакомительном режиме (Sandbox)

Настройка мониторинга ноды в ознакомительном режиме

Смотрите также

Транзакции блокчейн-платформы

Смарт-контракты

Обмен конфиденциальными данными

Инструментарий gRPC

Методы REST API

Развертывание платформы с подключением к Mainnet

В этом варианте развертывания платформы все ваши транзакции будут отправляться в Mainnet, основную сеть Waves Enterprise. При работе с Mainnet, за каждую транзакцию предусмотрены *комиссии* в WEST.

Для подключения к Mainnet вам достаточно установить одну ноду. В случае если вам необходимо развернуть сеть из нескольких нод с подключением к Mainnet, обратитесь в *службу технической поддержки* для консультации.

4.1 Генерирующий баланс

Обратите внимание, что для работы в сети необходимо иметь на адресе ноды или в лизинге не менее **50 000 WEST** - эта сумма составляет неснижаемый **генерирующий баланс**, необходимый для отправки транзакций и майнинга. Если сумма токенов на адресе становится ниже генерирующего баланса, нода теряет возможность быть выбранной майнером и отправлять транзакции до пополнения баланса. Информация о генерирующем балансе в сети Mainnet обновляется один раз в 1000 блоков. Нода получит возможность быть выбранной в качестве майнера и отправлять транзакции только после обновления генерирующего баланса.

4.2 Создание аккаунта, перевод токенов и подтверждающая транзакция

Перед развертыванием ПО ноды создайте аккаунт WE при помощи *клиента*. Затем выполните следующие шаги:

1. В клиенте создайте блокчейн-адрес при помощи кнопки **Адрес не выбран** в правом верхнем углу приложения, либо при помощи кнопки **Создать адрес** во вкладке **Токены**. Не забудьте записать или запомнить seed-фразу! С ее помощью вы всегда сможете восстановить доступ к вашему адресу при утрате учетных данных. После создания адреса нажмите на кнопку **Использовать адрес**.

2. Переведите на созданный адрес сумму в WEST, превышающую генерирующий баланс. Для этого перейдите на вкладку **Токены** клиента и нажмите на кнопку **Добавить токенов** через **Waves Exchange**. Скопируйте ваш блокчейн-адрес, а затем следуйте подсказкам обменного сервиса для покупки WEST.
3. Передайте в лизинг любое количество токенов WEST на адрес `3NrKDuHjUG7vSCiMMD259msBKcPRm4MvaJu` и сохраните идентификатор этой транзакции - он будет использован для подтверждения вашего баланса и факта владения вашим блокчейн-адресом. Поскольку токены передаются на этот адрес в лизинг, в дальнейшем вы сможете в любое время отозвать их обратно.

4.3 Развертывание ноды

Ознакомьтесь с *системными требованиями* к блокчейн-платформе.

После успешной передачи токенов разверните ноду:

1. Создайте рабочую директорию и поместите в нее файл **docker-compose.yml**. Этот файл вы можете скачать из [официального репозитория Waves Enterprise в GitHub](#), выбрав самый свежий релиз платформы, либо в терминале при помощи утилиты `wget`:

```
wget https://github.com/waves-enterprise/WE-releases/releases/download/v1.6.0/docker-
compose.yml
```

2. Скачайте файл `mainnet.conf` из [официального репозитория Waves Enterprise в GitHub](#), выбрав актуальную версию платформы. Затем переименуйте его в `private_network.conf` и поместите в корень рабочей директории.
3. Разверните вашу ноду при помощи следующей команды:

```
docker run --rm -ti -v $(pwd):/config-manager/output/ wavesenterprise/config-manager:v1.
6.0
```

После развертывания ноды все сгенерированные адреса и пароли будут храниться в файле **credentials.txt** в рабочей директории.

4.4 Подключение ноды к Mainnet

1. Зайдите на сайт [службы технической поддержки Waves Enterprise](#) и зарегистрируйтесь.
2. Создайте заявку **Подключение участника** для юридического или физического лица.
3. Заполните все необходимые поля формы, в частности, публичный ключ подключаемой ноды. Если вы планируете осуществлять майнинг в Mainnet, поставьте флажок **Прошу предоставить права майнинга**.
4. В поле **Подтверждение владения токенами WEST** введите идентификатор транзакции, при помощи которой вы передали токены в лизинг на адрес `3NrKDuHjUG7vSCiMMD259msBKcPRm4MvaJu`.
5. Дождитесь рассмотрения заявки и подтверждения успешной регистрации, после чего запустите ноду, публичный ключ которой вы указали в заявке на подключение:

```
docker-compose up -d node-0
```

После запуска контейнера REST API ноды будет доступен по адресу `http://localhost:6862`. Для остановки вашей ноды выполните команду `docker-compose down`.

6. Для осуществления майнинга и отправки транзакций переведите **50 000 WEST** или более на адрес подключенной ноды.

Подсказка: Для просмотра состояния вашей лицензии для работы в Mainnet воспользуйтесь запросом `GET /licenses/status` к ноде.

4.5 Последующие действия

4.5.1 Обновление ноды в Mainnet

С выходом каждого нового релиза платформы мы рекомендуем обновлять ноды, подключенные к Mainnet. Всем пользователям, ноды которых работают в Mainnet, приходит электронное письмо с уведомлением об обновлении версии ноды. Если вы не получили такого письма, обратитесь в [службу технической поддержки](#).

Для обновления ноды выполните следующие действия:

1. Скачайте последнюю версию файла `docker-compose.yml` из [официального репозитория Waves Enterprise в GitHub](#), выбрав последний релиз.
2. Поместите файл `docker-compose.yml` в рабочую директорию вашей ноды, заменив старый файл.
3. Если нода запущена, остановите ее:

```
docker-compose down
```

4. После остановки ноды выполните команду:

```
docker-compose up -d node-0
```

При первом запуске ноды, начиная с версии 1.4.0, будет автоматически запущен мигратор стейта. Миграция выполняется в автоматическом режиме и занимает несколько минут. Если миграция завершилась успешно вы увидите сообщение `Migration finished sucessfully` и запуск ноды будет продолжен.

Внимание: Если вы не используете Docker Compose, свяжитесь со [службой технической поддержки](#) для получения инструкций по обновлению ноды.

Смотрите также

[Развертывание платформы с подключением к Mainnet](#)

[Mainnet: устранение ошибок](#)

[Комиссии в сети «Waves Enterprise Mainnet»](#)

4.5.2 Mainnet: устранение ошибок

При разворачивании платформы с подключением к Mainnet возможно возникновение подобных ошибок на этапе разворачивания ноды:

```
ERROR: for node-1 Cannot create container for service node-1: Conflict. The container
↳name "/node-1" is already in use by container
↳"47cfd7a517e160d201ae969b24392ca0bc2b9720c73e7324dac45daaa24814cb". You have to remove
↳(or rename) that conCreating node-2 ... error

ERROR: for node-2 Cannot create container for service node-2: Conflict. The container
↳name "/node-2" is already in use by container "ccd28832f1fb5457186e50d5e5Creating node-
↳0 ... error
tainer to be able to reuse that name.

ERROR: for node-0 Cannot create container for service node-0: Conflict. The conCreating
↳postgres ... error
eb8ac184f88195f1a560ee8ef7ade5c46f899d". You have to remove (or rename) that container
↳to be able to reuse that name.

ERROR: for postgres Cannot create container for service postgres: Conflict. The
↳container name "/postgres" is already in use by container
↳"d4bc6d758faafcc9b2bc352b9cbcc5dc909f2959059b7abf17db0088916506d1". You have to remove
↳(or rename) that container to be able to reuse that name.

ERROR: for node-1 Cannot create container for service node-1: Conflict. The container
↳name "/node-1" is already in use by container
↳"47cfd7a517e160d201ae969b24392ca0bc2b9720c73e7324dac45daaa24814cb". You have to remove
↳(or rename) that container to be able to reuse that name.

ERROR: for node-2 Cannot create container for service node-2: Conflict. The container
↳name "/node-2" is already in use by container
↳"ccd28832f1fb5457186e50d5e58f98ed3b35c944931589a42a0262a205a17393". You have to remove
↳(or rename) that container to be able to reuse that name.

ERROR: for node-0 Cannot create container for service node-0: Conflict. The container
↳name "/node-0" is already in use by container
↳"7ed421ac8c8c5ca91a916970c1eb8ac184f88195f1a560ee8ef7ade5c46f899d". You have to remove
↳(or rename) that container to be able to reuse that name.

ERROR: for postgres Cannot create container for service postgres: Conflict. The
↳container name "/postgres" is already in use by container
↳"d4bc6d758faafcc9b2bc352b9cbcc5dc909f2959059b7abf17db0088916506d1". You have to remove
↳(or rename) that container to be able to reuse that name.
ERROR: Encountered errors while bringing up the project.
```

Причина: Контейнеры отдельных нод или сервисов уже используются запущенными контейнерами.

Решение: Остановите ноду при помощи команды `docker-compose down`. При помощи команды `docker stop [ID контейнера]` остановите запущенные контейнеры нод и сервисов. Вы можете ввести несколько ID запущенных контейнеров подряд через пробел или остановить все контейнеры при помощи команды `docker stop $(docker ps -a -q)`. Затем при помощи команды `docker rm [ID контейнера]` удалите их. ID используемых контейнеров доступны в отчетах об ошибках, подобных приведенному выше. Вы можете удалить несколько контейнеров или все используемые контейнеры одной командой при помощи аналогичного синтаксиса.

После удаления конкурирующих контейнеров разверните платформу заново.

Смотрите также

Развертывание платформы с подключением к Mainnet

Обновление ноды в Mainnet

Смотрите также

Комиссии в сети «Waves Enterprise Mainnet»

Генераторы

Развертывание платформы в частной сети

Если ваш проект или решение требует независимого блокчейна, вы можете развернуть собственную блокчейн-сеть на базе платформы Waves Enterprise. Специалисты компании помогут вам сконфигурировать поставку платформы под нужды проекта.

Однако если вам потребуется изменить какие-либо параметры или настроить платформу самостоятельно, в данном разделе приведено пошаговое руководство по развертыванию и ручному конфигурированию платформы для работы в частной сети.

5.1 Получение лицензии для работы в частной сети и сопутствующих файлов

Для развертывания платформы в частной сети вам необходимо получить вид лицензии, соответствующий вашим целям: пробную, коммерческую или некоммерческую.

Для обсуждения деталей вашей лицензии свяжитесь с отделом продаж Waves Enterprise по электронной почте: sales@wavesenterprise.com.

По результатам обсуждения, вам будет прислан YAML-файл для Kubernetes, необходимый для первичного развертывания платформы и инструкции по развертыванию и запуску платформы для вашей конфигурации оборудования.

Перед развертыванием ознакомьтесь с [системными требованиями](#) к блокчейн-платформе.

5.2 Создание аккаунта ноды

После получения лицензии и первичного развертывания вам потребится создать аккаунты для каждой ноды вашей будущей сети.

Генерация ключей производится при помощи утилиты AccountsGeneratorApp, которая входит в пакет *generators*. Этот пакет вы можете скачать из официального репозитория [Waves Enterprise в GitHub](#), выбрав используемую вами версию платформы.

Аккаунт ноды включает в себя адрес и ключевую пару - публичный и приватный ключи. Адрес и публичный ключ будут показаны в командной строке во время создания аккаунта при помощи утилиты **generators**. Приватный ключ ноды записывается в хранилище ключей - файл `keystore.dat`, который размещается в директории ноды.

Для создания аккаунта используется конфигурационный файл `accounts.conf`, содержащий параметры генерации аккаунтов. Этот файл находится в директории каждой ноды.

Чтобы создать аккаунт ноды, перейдите в ее директорию и разместите в ней скачанный файл **generators.jar**. Затем запустите его, введя в качестве аргумента файл `accounts.conf`:

```
java -jar generators-x.x.x.jar AccountsGeneratorApp accounts.conf
```

При создании пары ключей вы можете придумать свой пароль для защиты ключевой пары ноды. В дальнейшем вы сможете использовать его в ручном режиме при каждом старте вашей ноды, либо задать глобальные переменные для запроса пароля при старте системы. Подробная информация об использовании пароля для пары ключей ноды приведена [в описании генератора аккаунтов](#).

Если вы не хотите использовать пароль для защиты ключевой пары, нажмите клавишу **Enter**, оставив поле пустым.

В результате работы утилиты будут выведены следующие сообщения:

```
2021-02-09 16:03:18,940 INFO [main] c.w.g.AccountsGeneratorApp$ - 1 Address:␣
↪3Nu7MwQ1eSmDVwBzrN1nyzR8wqb2yzdUcyN; public key:␣
↪F4ytnnS6H72ypCEpgNKYftGotpdX83ZxtWRX2dyGzDiA
2021-02-09 16:03:18,942 INFO [main] c.w.g.AccountsGeneratorApp$ - Generator done
```

В директории ноды будет создан файл `keystore.dat`, содержащий публичный ключ аккаунта.

5.3 Настройка платформы для работы в частной сети

Для конфигурации платформы используются следующие файлы:

- `node.conf` - основной конфигурационный файл ноды, определяющий ее принципы работы и набор опций.
- `api-key-hash.conf` - конфигурационный файл для генерации значений полей `api-key-hash` и `privacy-api-key-hash`, используется для настройки авторизации ноды при выборе метода авторизации по хешу ключевой строки `api-key`. Принципы работы с этим конфигурационным файлом будут рассмотрены при настройке метода авторизации ноды.

Ниже приведено пошаговое руководство по ручной конфигурации отдельной ноды для работы в частной сети. Если в вашей сети развернуто несколько нод, для каждой из них требуется выполнить аналогичные шаги по конфигурации.

Шаг 1. Общая настройка платформы

На этом этапе выполняется настройка консенсуса, исполнения смарт-контрактов Docker и майнинга. Все необходимые для этого параметры располагаются в файле **node.conf**.

Установка и использование платформы

5.3.1 Общая настройка платформы: настройка консенсуса

Блокчейн-платформа Waves Enterprise поддерживает три алгоритма консенсуса - **PoS**, **PoA** и **CFT**. Подробная информация об используемых алгоритмах консенсуса приведена в статье [Алгоритмы консенсуса](#).

Настройки консенсуса располагаются в блоке consensus секции blockchain:

```
consensus {
  type = ""
  ...
}
```

Выберите предпочитаемый тип консенсуса в поле type. Возможные значения: pos, poa и cft.

type = "pos" или закомментированный блок consensus

Если вы не выберете тип консенсуса в этом поле, оставив его пустым, по умолчанию будет использоваться алгоритм **PoS**. Этот вариант равнозначен выбору значения pos. В этом случае другие поля в блоке consensus не требуются, вам потребуется только настроить работу майнинга с PoS в блоке genesis:

```
consensus {
  type = "pos"
}

...

genesis {
  average-block-delay = "60s"
  initial-base-target = 153722867
  initial-balance = "16250000 WEST"
  ...
}
```

За работу майнинга с PoS отвечают следующие параметры блока genesis в секции blockchain:

- average-block-delay - средняя задержка создания блоков. Значение по умолчанию - **60 секунд**.
- initial-base-target - начальное базовое число для регулирования процесса майнинга. От значения параметра зависит частота формирования блоков - чем выше значение, тем чаще создаются блоки. Также величина баланса майнера влияет на использование данного параметра в майнинге - чем больше баланс майнера, тем меньше становится значение initial-base-target при расчёте очереди ноды-майнера в текущем раунде.
- initial-balance - начальный баланс сети. Чем больше доля баланса майнера от изначального баланса сети, тем меньше становится значение initial-base-target для определения ноды-майнера текущего раунда.

```
type = "poa"
```

Для настройки алгоритма консенсуса PoA добавьте в блок consensus следующие параметры:

```
consensus {
  type = "poa"
  round-duration = "17s"
  sync-duration = "3s"
  ban-duration-blocks = 100
  warnings-for-ban = 3
  max-bans-percentage = 40
}
```

- round-duration - длина раунда майнинга блока в секундах.
- sync-duration - период синхронизации майнинга блока в секундах. Полное время раунда складывается из суммы round-duration и sync-duration.
- ban-duration-blocks - количество блоков, на которые нода-майнер попадает в бан.
- warnings-for-ban - количество раундов, в течение которых нода-майнер получает предупреждения. По окончании этого количества раундов нода попадает в бан.
- max-bans-percentage - процент нод-майнеров от общего числа нод в сети, который может быть помещён в бан.

```
type = "cft"
```

Основные параметры настройки CFT идентичны параметрам консенсуса PoA:

```
consensus {
  type: cft
  warnings-for-ban: 3
  ban-duration-blocks: 15
  max-bans-percentage: 33
  round-duration: 7s
  sync-duration: 2s
  max-validators: 7
  finalization-timeout: 4s
  full-vote-set-timeout: 4s
}
```

В сравнении с PoA, для CFT предусмотрено два дополнительных параметра конфигурации, необходимых для валидации блоков в ходе раунда голосования:

- max-validators – лимит валидаторов, участвующих в голосовании в конкретном раунде.
- finalization-timeout – время, в течение которого майнер ждет финализации последнего блока в цепочке. По прошествии этого времени майнер вернет транзакции обратно в UTX-пул и начнет майнить раунд заново.
- full-vote-set-timeout - опциональный параметр, определяющий, сколько времени после окончания раунда (параметр конфигурационного файла ноды: round-duration) майнер ожидает полный набор голосов от всех валидаторов.

При настройке CFT обратите внимание на следующие рекомендации:

- Параметр `sync-duration` должен быть отличен от нуля. Рекомендуется устанавливать значение **от 1 до 5 секунд**, в зависимости от размера и сложности транзакций.
- Примерный расчет значения параметра `finalization-timeout`: $(\text{round-duration} + \text{sync-duration}) / 2$. Не рекомендуется занижать это значение для ускорения финализации: если майнер наберет необходимое число голосов ранее окончания этого времени, он сразу выпустит финализирующий микроблок.
- Если в сети присутствует большое количество майнеров, ограничьте количество валидаторов раунда параметром `max-validators`. Механизм выбора валидаторов обеспечит равномерную ротацию всех валидаторов по раундам. Слишком большое количество валидаторов может отрицательно повлиять на производительность сети. Рекомендуемый диапазон значений: **от 5 до 10**.
- Если сеть работает под постоянной нагрузкой, установите параметр `full-vote-set-timeout`. До истечения этого периода времени майнер ждет полного набора голосов от валидаторов. Если валидатор сталкивается с какими-либо неполадками, сеть использует время `full-vote-set-timeout` для создания дополнительного временного промежутка, который позволяет отставшему валидатору завершить синхронизацию. Рекомендуемое значение: $\text{sync-duration} * 2$, не может превышать $\text{sync-duration} + \text{finalization-timeout}$.

Смотрите также

Алгоритмы консенсуса

Развертывание платформы в частной сети

Общая настройка платформы: настройка майнинга

Общая настройка платформы: настройка исполнения смарт-контрактов

Установка и использование платформы

5.3.2 Общая настройка платформы: настройка исполнения смарт-контрактов

Если вы планируете разработку и исполнение смарт-контрактов в вашем блокчейне, настройте параметры их исполнения в секции `docker-engine` конфигурационного файла ноды:

```
docker-engine {
  enable = yes
  integration-tests-mode-enable = no
  # docker-host = "unix:///var/run/docker.sock"
  execution-limits {
    startup-timeout = 10s
    timeout = 10s
    memory = 512
    memory-swap = 0
  }
  reuse-containers = yes
  remove-container-after = 10m
  allow-net-access = yes
  remote-registries = [
    {
      domain = "myregistry.com:5000"
      username = "user"
      password = "password"
    }
  ]
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    }
  ]
  check-registry-auth-on-startup = no
  # default-registry-domain = "registry.wavesenterprise.com"
  contract-execution-messages-cache {
    expire-after = 60m
    max-buffer-size = 10
    max-buffer-time = 100ms
  }
  contract-auth-expires-in = 1m
  grpc-server {
    # host = "192.168.97.3"
    port = 6865
  }
}
remove-container-on-fail = yes
}
```

- `enable` – включение обработки транзакций для Docker-контрактов.
- `integration-tests-mode-enable` – режим тестирования Docker-контрактов. При включении этой опции смарт-контракты исполняются локально в контейнере.
- `docker-host` – адрес демона docker (опционально). Если это поле закомментировано, адрес демона для исполнения смарт-контрактов будет взят из системного окружения.
- `startup-timeout` – время, отводимое на создание контейнера контракта и его регистрацию в ноде (в секундах).
- `timeout` – время, отводимое на выполнение контракта (в секундах).
- `memory` – ограничение по памяти для контейнера контракта (в мегабайтах).
- `memory-swap` – выделяемый объем виртуальной памяти для контейнера контракта (в мегабайтах).
- `reuse-containers` – использование одного контейнера для нескольких контрактов, использующих один и тот же Docker-образ. Включение опции - `yes`, отключение - `no`.
- `remove-container-after` – промежуток времени бездействия контейнера, по прошествии которого он будет удален.
- `allow-net-access` – разрешение доступа к сети.
- `remote-registries` – адреса Docker-репозитория и настройки авторизации к ним.
- `check-registry-auth-on-startup` – проверка авторизации для Docker-репозитория при запуске ноды. Включение опции - `yes`, отключение - `no`.
- `default-registry-domain` – адрес Docker-репозитория по умолчанию (опционально). Этот параметр используется, если в имени образа контракта не указан репозиторий.
- `contract-execution-messages-cache` – настройки кэша со статусами исполнения транзакций по docker контрактам;
- `expire-after` – время хранения статуса смарт-контракта.
- `max-buffer-size` и `max-buffer-time` – настройки объема и времени хранения кэша статусов.
- `contract-auth-expires-in` – время жизни токена авторизации, используемого смарт-контрактами для вызовов к ноде.

- `grpc-server` – секция настроек gRPC сервера для работы Docker-контрактов с gRPC API.
- `host` – сетевой адрес ноды (опционально).
- `port` – порт gRPC-сервера. Укажите порт прослушивания gRPC-запросов, использующийся платформой.
- `remove-container-on-fail` – удаление контейнера, если при его старте произошла ошибка. Включение опции - `yes`, отключение - `no`.

Смотрите также

[Развертывание платформы в частной сети](#)

[Разработка и применение смарт-контрактов](#)

[Общая настройка платформы: настройка консенсуса](#)

[Общая настройка платформы: настройка майнинга](#)

Установка и использование платформы

5.3.3 Общая настройка платформы: настройка майнинга

Параметры майнинга в блокчейне находятся в разделе `miner` конфигурационного файла ноды:

```
miner {
  enable = yes
  quorum = 2
  interval-after-last-block-then-generation-is-allowed = 10d
  micro-block-interval = 5s
  min-micro-block-age = 3s
  max-transactions-in-micro-block = 500
  minimal-block-generation-offset = 200ms
}
```

- `enable` - активация опции майнинга. Включение - `yes`, отключение - `no`.
- `quorum` - необходимое количество нод-майнеров для создания блока. Значение 0 позволит генерировать блоки оффлайн и используется только в тестовых целях в сетях с одной нодой. При указании этого значения необходимо учитывать, что собственная нода-майнер не суммируется со значением этого параметра, т.е., если вы указываете `quorum = 2`, то для майнинга нужно минимум 3 ноды-майнера.
- `interval-after-last-block-then-generation-is-allowed` - создание блока только в том случае, если последний блок не старше указанного периода времени (в днях).
- `micro-block-interval` - интервал между микроблоками (в секундах).
- `min-micro-block-age` - минимальный возраст микроблока (в секундах).
- `max-transactions-in-micro-block` - максимальное количество транзакций в микроблоке.
- `minimal-block-generation-offset` - минимальный временной интервал между блоками (в миллисекундах).

Настройки майнинга зависят от планируемого в вашей сети размера транзакций.

Также майнинг в блокчейне тесно связан с выбранным алгоритмом консенсуса. При настройке параметров консенсуса необходимо учитывать следующие параметры секции `miner`:

- `micro-block-interval` - интервал между микроблоками. Значение указывается в секундах.
- `min-micro-block-age` - минимальный возраст микроблока. Значение указывается в секундах и не должно превышать значения параметра `micro-block-interval`.
- `minimal-block-generation-offset` - минимальный временной интервал между блоками. Значение указывается в миллисекундах.

Значения параметров создания микроблоков не должны превышать или как-либо иначе конфликтовать со значениями параметров `average-block-delay` для **PoS** и `round-duration` для **PoA** и **CFT**. Количество микроблоков в блоке не ограничено, но зависит от размера транзакций, попавших в микроблок.

Смотрите также

Развертывание платформы в частной сети

Общая настройка платформы: настройка консенсуса

Общая настройка платформы: настройка исполнения смарт-контрактов

Протокол работы блокчейна Waves-NG

Шаг 2. Тонкая настройка платформы

На этом этапе выполняется настройка инструментария gRPC и REST API ноды, их авторизации, TLS и групп доступа к конфиденциальным данным. Эти настройки могут потребоваться вам в случае изменения предустановленных параметров для конфигурации вашего оборудования или ПО.

Все необходимые параметры также располагаются в в файле конфигурации ноды **node.conf**. Для настройки авторизации также применяется файл **api-key-hash.conf**, необходимый при выборе метода авторизации по хэшу заданной строки *api-key*.

Также для настройки TLS вам потребуется утилита **keytool**, которая входит в состав Java SDK или JRE.

5.3.4 Тонкая настройка платформы: настройка авторизации для gRPC и REST API

Авторизация необходима для обеспечения доступа к gRPC и REST API инструментам ноды. Для этого блокчейн-платформа Waves Enterprise поддерживает два типа авторизации:

- по хэшу ключевой строки *api-key*;
- по JWT-токену (OAuth 2).

Внимание: Авторизация по хэшу *api-key* является простым средством доступа к ноде, однако уровень безопасности этого метода авторизации сравнительно низок. Злоумышленник может получить доступ к ноде в случае попадания к нему строки *api-key*. Если вы хотите повысить уровень безопасности в вашей сети, рекомендуем воспользоваться авторизацией по JWT-токену через сервис авторизации.

Для настройки авторизации предусмотрена секция `auth` конфигурационного файла ноды.

```
type = "api-key"
```

Авторизация по хэшу ключевой строки `api-key` используется в ноде по умолчанию. При выборе метода авторизации по хэшу ключевой строки `api-key` секция `auth` содержит следующие параметры:

```
auth {
  type = "api-key"

  # Hash of API key string
  api-key-hash = "G3PZAsY6EA8esgpKxB2UYTQJZJPzc14gLnNbm2xvcDf6"

  # Hash of API key string for PrivacyApi routes
  privacy-api-key-hash = "G3PZAsY6EA8esgpKxB2UYTQJZJPzc14gLnNbm2xvcDf6"
}
```

- `api-key-hash` - хэш от ключевой строки доступа к REST API.
- `privacy-api-key-hash` - хэш от ключевой строки доступа к методам **privacy**.

Для заполнения этих параметров вам потребуется утилита `ApiKeyHash` пакет `generators-x.x.x.jar`, который вы можете скачать из официального репозитория [Waves Enterprise в GitHub](#), выбрав используемую вами версию платформы.

Поместите этот файл в корневую папку платформы, а также создайте файл `api-key-hash.conf`:

```
apikeyhash-generator {
  waves-crypto = yes
  api-key = "some string for api-key"
}
```

В этом файле введите строку, которую вы хотите хэшировать и использовать для авторизации, в параметре `api-key`.

Готовый файл `api-key-hash.conf` введите в качестве аргумента при запуске утилиты `ApiKeyHash` пакета `generators`:

```
java -jar generators-x.x.x.jar ApiKeyHash api-key-hash.conf
```

Пример вывода:

```
Api key: some string for api-key
Api key hash: G3PZAsY6EA8esgpKxB2UYTQJZJPzc14gLnNbm2xvcDf6

2021-02-11 16:31:21,586 INFO [main] c.w.g.ApiKeyHashGenerator$ - Generator done
```

Полученное значение `Api key hash` укажите в параметрах `api-key-hash` и `privacy-api-key-hash` в секции `auth` конфигурационного файла ноды, как указано выше.

```
type = "oauth2"
```

При выборе авторизации по JWT-токену секция `auth` конфигурационного файла ноды выглядит так:

```
auth {
  type: "oauth2"
  public-key: "AuthorizationServicePublicKeyInBase64"
}
```

Публичный ключ для `oAuth` генерируется при первичном развертывании ноды. Он находится в файле `./auth-service-keys/jwtRS256.key.pub`. Скопируйте строку, находящуюся между `-----BEGIN PUBLIC KEY-----` и `-----END PUBLIC KEY-----` и вставьте ее в качестве параметра `public-key` секции `auth` конфигурационного файла ноды.

Подсказка: Интерфейсы REST API и gRPC используют одинаковые значения `api-key` для авторизации по ключевой строке и `public-key` для авторизации по JWT-токену.

Смотрите также

Развертывание платформы в частной сети

Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды

Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным

Тонкая настройка платформы: настройка TLS

5.3.5 Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды

Параметры работы gRPC и REST API для каждой ноды находятся в секции `api` конфигурационного файла:

```
api {
  rest {
    # Enable/disable REST API
    enable = yes

    # Network address to bind to
    bind-address = "0.0.0.0"

    # Port to listen to REST API requests
    port = 6862

    # Enable/disable TLS for REST
    tls = no

    # Enable/disable CORS support
    cors = yes

    # Max number of transactions
    # returned by /transactions/address/{address}/limit/{limit}
    transactions-by-address-limit = 10000
```

(continues on next page)

(продолжение с предыдущей страницы)

```
distribution-address-limit = 1000
}

grpc {
  # Enable/disable gRPC API
  enable = yes

  # Network address to bind to
  bind-address = "0.0.0.0"

  # Port to listen to gRPC API requests
  port = 6865

  # Enable/disable TLS for GRPC
  tls = no
}
```

Блок rest { }

Блок rest { } предназначен для настройки интерфейса REST API ноды. Он включает следующие параметры:

- enable - активация опции REST API на ноды. Включение опции - yes, отключение - no.
- bind-address - сетевой адрес ноды, на котором будет доступен REST API интерфейс.
- port - порт прослушивания REST API запросов.
- tls - включение/отключение TLS для REST API запросов. Включение - yes, отключение - no.
- cors - поддержка кросс-доменных запросов к REST API. Включение опции - yes, отключение - no.
- transactions-by-address-limit - максимальное количество транзакций, возвращаемых методом GET /transactions/address/{address}/limit/{limit}.
- distribution-address-limit - максимальное количество адресов, указываемых в поле limit и возвращаемых методом GET /assets/{assetId}/distribution/{height}/limit/{limit}.

Блок grpc { }

Блок grpc { } предназначен для настройки gRPC-инструментария ноды. Он включает следующие параметры:

- enable - активация gRPC-интерфейса на ноды.
- bind-address - сетевой адрес ноды, на котором будет доступен gRPC-интерфейс.
- port - порт прослушивания gRPC запросов.
- tls - включение/отключение TLS для gRPC запросов. Для включения требуется *настройка TLS ноды*.

Смотрите также*Развертывание платформы в частной сети**Тонкая настройка платформы: настройка авторизации для gRPC и REST API**Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным**Тонкая настройка платформы: настройка TLS***5.3.6 Тонкая настройка платформы: настройка TLS**

Для работы с TLS, помимо его включения в конфигурационном файле ноды, необходимо получить файл keystore при помощи стандартной утилиты **keytool**:

```
keytool \
-keystore we.jks -storepass 123456 -keypass 123456 \
-genkey -alias we -keyalg RSA -validity 9999 \
-dname "CN=Waves Enterprise,OU=security,O=WE,C=RU" \
-ext "SAN=DNS:welocal.dev,DNS:localhost,IP:51.210.211.61,IP:127.0.0.1"
```

- keystore - имя файла keystore.
- storepass - пароль от keystore, указывается в конфиге в секции keystore-password.
- keypass - пароль от приватного ключа, указывается в конфиге в секции private-key-password.
- alias - произвольное имя.
- keyalg - алгоритм генерации ключевой пары.
- validity - срок действия в днях.
- dname - уникальное имя по стандарту X.500, связанное с alias в keystore.
- ext - расширения, применяемые при генерации ключа, указываются все возможные имена хостов и IP-адреса для работы сертификата в различных сетях.

В результате работы keytool будет получен keystore с именем we.jks. Чтобы подключиться к ноды с включенным TLS, также необходимо выпустить клиентский сертификат:

```
keytool -export -keystore we.jks -alias we -file we.cert
```

Полученный файл сертификата we.cert необходимо импортировать в хранилище доверенных сертификатов. При работе ноды в одной сети с пользователем, достаточно указать относительный путь к файлу we.jks в файле конфигурации ноды, как это показано выше.

В случае, если нода находится в другой сети, импортируйте сертификат we.cert в keystore:

```
keytool -importcert -alias we -file we.cert -keystore we.jks
```

Затем также укажите относительный путь к we.jks в секции tls файла конфигурации ноды.

Секция tls содержит следующие параметры:

```
tls {
type = EMBEDDED
keystore-path = ${node.directory}"/we_tls.jks"
keystore-password = ${TLS_KEystore_PASSWORD}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
private-key-password = ${TLS_PRIVATE_KEY_PASSWORD}
}
```

- `type` - состояние режима TLS. Возможные опции: `DISABLED` (отключен, в этом случае остальные опции не указываются или комментируются) и `EMBEDDED` (включен, сертификат подписывается провайдером ноды и упаковывается в JKS-файл (keystore) - при этом директория, в которой располагается сертификат, и параметры доступа к сертификату и keystore указывается пользователем вручную в последующих полях).
- `keystore-path` - относительный путь к keystore, размещаемому в директории ноды: `${node.directory}/we_tls.jks`.
- `keystore-password` - пароль для keystore. Укажите пароль, который вы задали ранее флагом `storepass` для утилиты **keytool**.
- `private-key-password` - пароль для приватного ключа. Укажите пароль, который вы задали ранее флагом `keypass` для утилиты **keytool**.

Смотрите также

Развертывание платформы в частной сети

Тонкая настройка платформы: настройка авторизации для gRPC и REST API

Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды

Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным

5.3.7 Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным

Если вы используете API-методы **privacy** для управления конфиденциальными данными, настройте параметры доступа к конфиденциальным данным, для которых предназначена секция `privacy` конфигурационного файла ноды (пример с использованием БД PostgreSQL и включенным периодическим удалением файлов, не попавших в блокчейн):

```
privacy {
storage {
  vendor = postgres
  schema = "public"
  migration-dir = "db/migration"
  profile = "slick.jdbc.PostgresProfile$"
  jdbc-config {
    url = "jdbc:postgresql://postgres:5432/node-1"
    driver = "org.postgresql.Driver"
    user = postgres
    password = wenterprise
    connectionPool = HikariCP
    connectionTimeout = 5000
    connectionTestQuery = "SELECT 1"
    queueSize = 10000
    numThreads = 20
  }
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

}

cleaner {
  enabled: yes
  interval: 10m
  confirmation-blocks: 100
  pending-time: 72h
}
}

```

Перед его изменением определитесь с базой данных, которую вы планируете использовать для хранения конфиденциальных данных. Блокчейн-платформа Waves Enterprise поддерживает взаимодействие с БД PostgreSQL или Amazon S3.

При использовании СУБД PostgreSQL, потребуется установка JDBC-интерфейса. При использовании Amazon S3, информация должна храниться на сервере Minio.

После установки подходящей для вашего проекта СУБД, перейдите к настройке блока секции `privacy`. Укажите используемую вами СУБД в параметре `vendor`:

- `postgres` - для PostgreSQL;
- `s3` - для Amazon S3.

Если вы не используете API-методы **privacy**, укажите `none` и закомментируйте или удалите остальные параметры.

```
vendor = postgres
```

При использовании СУБД PostgreSQL, блок `storage` секции `privacy` выглядит так:

```

storage {
  vendor = postgres
  schema = "public"
  migration-dir = "db/migration"
  profile = "slick.jdbc.PostgresProfile$"
  jdbc-config {
    url = "jdbc:postgresql://postgres:5432/node-1"
    driver = "org.postgresql.Driver"
    user = postgres
    password = wenterprise
    connectionPool = HikariCP
    connectionTimeout = 5000
    connectionTestQuery = "SELECT 1"
    queueSize = 10000
    numThreads = 20
  }
}

```

- `schema` - используемая схема взаимодействия между элементами в рамках БД. По умолчанию применяется схема `public`, однако если в вашей БД предусмотрена иная схема - укажите ее название.
- `migration-dir` - директория для миграции данных.
- `profile` - имя профиля для доступа к JDBC.

- `url` - адрес БД PostgreSQL.
- `driver` - имя драйвера JDBC (Java SataBase Connectivity), позволяющим Java-приложениям взаимодействовать с БД.
- `user` - имя пользователя для доступа к БД.
- `password` - пароль для доступа к БД.
- `connectionPool` - имя пула соединений, по умолчанию HikariCP.
- `connectionTimeout` - время бездействия соединения до его разрыва (в миллисекундах).
- `connectionTestQuery` - тестовый запрос для проверки соединения с БД. Для PostgreSQL рекомендуется отправлять `SELECT 1`.
- `queueSize` - размер очереди запросов.
- `numThreads` - количество одновременных подключений к БД.

Во время установки БД под управлением PostgreSQL создайте аккаунт для доступа к БД. Затем введите указанный вами логин и пароль в поля `user` и `password`. При установке JDBC, задайте имя профиля, которое затем укажите в поле `profile`.

В поле `url` укажите адрес используемой вами БД в следующем формате:

```
jdbc:postgresql://<POSTGRES_ADDRESS>:<POSTGRES_PORT>/<POSTGRES_DB>
```

- `POSTGRES_ADDRESS` - адрес хоста PostgreSQL.
- `POSTGRES_PORT` - номер порта хоста PostgreSQL.
- `POSTGRES_DB` - наименование БД PostgreSQL.

Вы можете указать адрес БД вместе с данными аккаунта, используя параметры `user` и `password`:

```
privacy {
  storage {
    ...
    url = "jdbc:postgresql://yourpostgres.com:5432/privacy_node_0?user=user_privacy_node_
    ↪0@company&password=7nZL7Jr41q0WUHz5qKdypA&sslmode=require"
    ...
  }
}
```

В этом примере `user_privacy_node_0@company` - имя пользователя, `7nZL7Jr41q0WUHz5qKdypA` - его пароль. Также вы можете использовать команду `sslmode=require` для требования пароля при авторизации.

```
vendor = s3
```

При использовании СУБД Amazon S3, блок `storage` секции `privacy` выглядит так:

```
storage {
  vendor = s3
  url = "http://localhost:9000/"
  bucket = "privacy"
  region = "aws-global"
  access-key-id = "minio"
  secret-access-key = "minio123"
```

(continues on next page)

(продолжение с предыдущей страницы)

```

path-style-access-enabled = true
connection-timeout = 30s
connection-acquisition-timeout = 10s
max-concurrency = 200
read-timeout = 0s
}

```

- url - адрес сервера Minio для хранения данных. По умолчанию, Minio использует порт 9000.
- bucket - имя таблицы БД S3 для хранения данных.
- region - название региона S3, значение параметра - aws-global.
- access-key-id - идентификатор ключа доступа к данным.
- secret-access-key - ключ доступа к данным в хранилище S3.
- path-style-access-enabled = true - неизменяемый параметр указания пути к таблице S3.
- connection-timeout - период бездействия до разрыва соединения (в секундах).
- connection-acquisition-timeout - период бездействия при установлении соединения (в секундах).
- max-concurrency - число параллельных обращений к хранилищу.
- read-timeout - период бездействия при чтении данных (в секундах).

В процессе установки сервера Minio вам будет предложено определить логин и пароль для доступа к данным. Введите логин в поле access-key-id, а пароль - в поле secret-access-key.

Секция cleaner

Секция cleaner предназначена для настройки периодического удаления конфиденциальных данных, которые хранятся в БД, но по тем или иным причинам не попали в блокчейн (например, в случае отката транзакции). Эта секция включает следующие параметры:

- enabled - включение/отключение периодического удаления файлов, которые не попали в блокчейн.
- interval - интервал очистки файлов.
- confirmation-blocks - период времени в блоках, в течение которого который существует хэш-транзакция данных в блокчейне, после чего она удаляется.
- pending-time - максимальный период времени, за который сохраняется файл с данными, не попадая в блокчейн.

Смотрите также

Развертывание платформы в частной сети

Тонкая настройка платформы: настройка авторизации для gRPC и REST API

Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды

Тонкая настройка платформы: настройка TLS

5.3.8 Тонкая настройка платформы: настройка анкоринга

Если вы планируете использовать *анкоринг* данных из вашей сети в более крупную сеть, настройте параметры передачи данных в блоке `anchoring` конфигурационного файла ноды. В терминологии конфигурационного файла, `targetnet` - это блокчейн, в который ваша нода будет выполнять транзакции анкоринга из текущей сети.

```
anchoring {
  enable = yes
  height-range = 30
  height-above = 8
  threshold = 20
  tx-mining-check-delay = 5 seconds
  tx-mining-check-count = 20

  targetnet-authorization {
    type = "oauth2" # "api-key" or "oauth2"
    authorization-token = ""
    authorization-service-url = "https://client.wavesenterprise.com/
    ↪authServiceAddress/v1/auth/token"
    token-update-interval = "60s"
    # api-key-hash = ""
    # privacy-api-key-hash = ""
  }

  targetnet-scheme-byte = "v"
  targetnet-node-address = "https://client.wavesenterprise.com:6862/NodeAddress"
  targetnet-node-recipient-address = ""
  targetnet-private-key-password = ""

  wallet {
    file = "node-1_mainnet-wallet.dat"
    password = "small"
  }

  targetnet-fee = 10000000
  sidechain-fee = 5000000
}
```

Параметры анкоринга

- `enable` - включение или отключение анкоринга (*yes / no*);
- `height-range` - интервал блоков, по прошествии которого нода приватного блокчейна отправляет в Targetnet транзакции для анкоринга;
- `height-above` - число блоков в Targetnet, по прошествии которого нода приватного блокчейна создаёт подтверждающую анкоринг транзакцию с данными первой транзакции. Рекомендуется устанавливать значение, не превышающее максимальную величину отката блоков в Targetnet (`max-rollback`);
- `threshold` - число блоков, которое отнимается от текущей высоты приватного блокчейна. В транзакцию для анкоринга, отправляемую в Targetnet, попадёт информация из блока на высоте `current-height - threshold`. Если устанавливается значение `0`, в транзакцию анкоринга записывается значение блока на текущей высоте блокчейна. Рекомендуется устанавливать значение, близкое к максимальной величине отката в приватном блокчейне (`max-rollback`);

- `tx-mining-check-delay` - время ожидания между проверками доступности транзакции для анкоринга в Targetnet;
- `tx-mining-check-count` - максимальное количество проверок доступности транзакции для анкоринга в Targetnet, по выполнении которых транзакция считается не поступившей в сеть.

В зависимости от настроек майнинга в сети Targetnet, расстояние между транзакциями анкоринга может меняться. Установленное значение `height-range` задаёт приблизительный интервал между транзакциями анкоринга. Реальное время попадания транзакций анкоринга в смайненый блок сети Targetnet может превышать время, потраченное на майнинг количества блоков `height-range` в сети Targetnet.

Параметры авторизации при использовании анкоринга

- `type` - тип авторизации при использовании анкоринга: * `api-key` - авторизация по `api-key-hash`; * `auth-service` - авторизация по JWT-токену через [сервис авторизации](#).

В случае выбора авторизации по `api-key-hash`, достаточно указать значение ключа в параметре `api-key`. Если вы выбираете авторизацию по JWT-токену, необходимо указать `type = "auth-service"`, а также раскомментировать и заполнить параметры ниже:

- `authorization-token` - постоянный токен авторизации;
- `authorization-service-url` - URL-адрес сервиса авторизации;
- `token-update-interval` - интервал обновления авторизационного токена.

Параметры для доступа Targetnet

Для ноды, которая будет отправлять транзакции анкоринга в Targetnet, генерируется отдельный файл `keystore.dat` с ключевой парой для доступа в Targetnet.

- `targetnet-scheme-byte` - байт сети Targetnet (Waves Enterprise Mainnet - **V**);
- `targetnet-node-address` - полный сетевой адрес ноды вместе с номером порта в сети Targetnet, на который будут отправляться транзакции для анкоринга. Адрес необходимо указывать вместе с типом соединения (`http/https`), номером порта и параметром `NodeAddress`: `http://node.weservices.com:6862/NodeAddress`;
- `targetnet-node-recipient-address` - адрес ноды в сети Targetnet, на который будут записываться транзакции для анкоринга, подписанные ключевой парой данного адреса;
- `targetnet-private-key-password` - пароль от приватного ключа ноды для подписи транзакций анкоринга.

Сетевой адрес и порт для анкоринга в сеть Targetnet вы можете получить у сотрудников технической поддержки Waves Enterprise. Если вы используете несколько приватных блокчейнов с взаимным анкорингом, используйте соответствующие сетевые настройки частных сетей.

Параметры файла с ключевой парой для подписания транзакций анкоринга в Targetnet, (секция ```wallet```)

- `file` - имя файла и путь до каталога хранения файла с ключевой парой для подписания транзакций анкоринга в сети Targetnet. Файл находится на ноды приватной сети;
- `password` - пароль от файла с ключевой парой.

Параметры комиссий

- `targetnet-fee` - комиссия за выпуск транзакции для анкоринга в сети Targetnet;
- `sidechain-fee` - комиссия за выпуск транзакции в текущем приватном блокчейне.

Смотрите также*Развертывание платформы в частной сети**Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды**Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным**Тонкая настройка платформы: настройка TLS***5.3.9 Тонкая настройка платформы: настройка механизма создания снимка данных**

Для настройки *механизма создания снимка данных* в приватном блокчейне предусмотрен блок **node.consensual-snapshot** конфигурационного файла ноды:

```
node.consensual-snapshot {
  enable = yes
  snapshot-directory = ${node.data-directory}"/snapshot"
  snapshot-height = 12000000
  wait-blocks-count = 10
  back-off {
    max-retries = 3
    delay = 10m
  }
  consensus-type = CFT
}
```

В этом блоке настраиваются следующие параметры:

- **snapshot-directory** – директория на диске для сохранения снимка данных. По умолчанию – под-директория **snapshot** в директории с данными ноды;
- **snapshot-height** – высота блокчейна, на которой будет создан снимок данных;
- **wait-blocks-count** – число блоков после завершения создания снимка данных, по прошествии которых нода рассылает своим пирам (адресам из списка **peers** в конфигурационном файле ноды) сообщение о готовности снимка данных;
- **back-off** – секция настроек для повторных попыток создания снимка данных в случае ошибок: **max-retries** – общее число попыток, **delay** – интервал между попытками (в минутах);
- **consensus-type** – тип консенсуса генезис-блока новой сети. Возможные значения: POA, CFT.

Смотрите также*Развертывание платформы в частной сети**Механизм создания снимка данных**Тонкая настройка платформы: настройка механизма создания снимка данных*

Полные примеры конфигурационных файлов для настройки каждой ноды приведены [здесь](#).

5.4 Подписание genesis-блока и запуск сети

После выполнения конфигурации нод вашей сети необходимо создать genesis-блок - первый блок приватного блокчейна, содержащий транзакции, определяющие первоначальный баланс и разрешения ноды.

Genesis-блок подписывается утилитой *GenesisBlockGenerator*, входящей в пакет **generators**. В качестве аргумента она использует настроенный вами конфигурационный файл ноды `node.conf`:

```
java -jar generators-x.x.x.jar GenesisBlockGenerator node.conf
```

В результате работы утилиты поля `genesis-public-key-base-58` и `signature`, находящиеся в блоке `genesis` секции `blockchain` конфигурационного файла ноды, будут заполнены сгенерированными значениями открытого ключа и подписи genesis-блока.

Пример:

```
genesis-public-key-base-58: "4ozcAj...penxrm"
signature: "5QNVGF...7Bj4Pc"
```

После подписания genesis-блока платформа полностью настроена и готова для запуска сети. Вы можете запустить ее согласно указаниям, полученным от специалистов Waves Enterprise.

5.5 Привязка клиента к частной сети

После запуска сети привяжите к ней клиентское приложение Waves Enterprise - с его помощью пользователи сети смогут отправлять транзакции в блокчейн, а также публиковать и вызывать смарт-контракты.

1. Откройте браузер и введите в адресную строку сетевой адрес вашего компьютера с развернутым ПО ноды.
2. Зарегистрируйтесь в веб-клиенте, используя любой действительный электронный адрес, и зайдите в веб-клиент.
3. Откройте страницу **Выберите адрес -> Создать адрес**. Для открытия меню после первого входа необходимо ввести пароль, который вы вводили при регистрации аккаунта.
4. Выберите пункт **Добавить адрес из ключевого хранилища ноды** и нажмите кнопку **Продолжить**.
5. Заполните поля, указанные ниже. Необходимые значения приведены в файле `credentials.txt` для первой ноды в рабочей директории.
 - Имя адреса - укажите название ноды;
 - URL ноды - укажите значение `http://<сетевой адрес компьютера>/<адрес ноды>`;
 - Тип авторизации на ноде - выберите тип авторизации, который вы настроили ранее: по JWT-токену или по `api-key`;
 - Блокчейн-адрес - укажите адрес ноды;
 - Пароль от ключевой пары - укажите пароль от ключевой пары ноды, если задавали его при генерации аккаунта.

Описание клиента приведено на странице [Клиент](#).

Смотрите также

Примеры конфигурационных файлов ноды

Генераторы

Установка и использование платформы

Примеры конфигурационных файлов ноды

6.1 node.conf

В этом примере конфигурации:

- используется алгоритм консенсуса PoA;
- используется вторая версия генезиса;
- включена роль **sender** для участников сети (см. статью [Роли участников](#));
- включен майнинг для трех нод;
- отключен TLS;
- запущены инструменты gRPC и REST API без TLS, а также исполнение смарт-контрактов;
- включена авторизация по хэшу ключевой строки api-key для gRPC и REST API;
- использование методов **privacy** с БД PostgreSQL для хранения конфиденциальных данных;
- функция периодического удаления конфиденциальных данных, не попавших в блокчейн, отключена.

Поля, значения которых вы получите при использовании пакета **generators** или настроите самостоятельно, исходя из конфигурации вашего оборудования и ПО, помечены как /FILL/.

Каждая секция снабжена дополнительным комментарием.

node.conf:

```

node {
  # Type of cryptography
  waves-crypto = yes

  # Node owner address
  owner-address = " /FILL/ "

  # NTP settings
  ntp.fatal-timeout = 5 minutes

  # Node "home" and data directories to store the state
  directory = "/node"
  data-directory = "/node/data"

  # Location and name of a license file
  # license.file = ${node.directory}/node.license"

  wallet {
    # Path to keystore.
    file = "/node/keystore.dat"

    # Access password
    password = " /FILL/ "
  }

  # Blockchain settings
  blockchain {
    type = CUSTOM
    fees.enabled = false
    consensus {
      type = "poa"
      round-duration = "17s"
      sync-duration = "3s"
      ban-duration-blocks = 100
      warnings-for-ban = 3
      max-bans-percentage = 40
    }
    custom {
      address-scheme-character = "E"
      functionality {
        feature-check-blocks-period = 1500
        blocks-for-feature-activation = 1000
        pre-activated-features = { 2 = 0, 3 = 0, 4 = 0, 5 = 0, 6 = 0, 7 = 0, 9 = 0, 10 = 0,
→ 100 = 0, 101 = 0 }
      }

      # Mainnet genesis settings
      genesis {
        version: 2
        sender-role-enabled: true
        average-block-delay: 60s
      }
    }
  }
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```

initial-base-target: 153722867

# Filled by GenesisBlockGenerator
block-timestamp: 1573472578702

initial-balance: 16250000 WEST

# Filled by GenesisBlockGenerator
genesis-public-key-base-58: ""

# Filled by GenesisBlockGenerator
signature: ""

transactions = [
  # Initial token distribution:
  # - recipient: target's blockchain address (base58 string)
  # - amount: amount of tokens, multiplied by 10e8 (integer)
  #
  # Example: { recipient: "3HQSr3VFCiE6JcWwV1yX8xttYbAGKTLV3Gz", amount:
↪30000000 WEST }
  #
  # Note:
  # Sum of amounts must be equal to initial-balance above.
  #
  { recipient: " /FILL/ ", amount: 1000000 WEST },
  { recipient: " /FILL/ ", amount: 1500000 WEST },
  { recipient: " /FILL/ ", amount: 500000 WEST },
]
network-participants = [
  # Initial participants and role distribution
  # - public-key: participant's base58 encoded public key;
  # - roles: list of roles to be granted;
  #
  # Example: {public-key: "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt",
↪roles: [permissioner, miner, connection_manager, contract_developer, issuer]}
  #
  # Note:
  # There has to be at least one miner, one permissioner and one connection_
↪manager for the network to start correctly.
  # Participants are granted access to the network via
↪GenesisRegisterNodeTransaction.
  # Role list could be empty, then given public-key will only be granted
↪access to the network.
  #
  { public-key: " /FILL/ ", roles: [permissioner, sender, miner, connection_
↪manager, contract_developer, issuer]},
  { public-key: " /FILL/ ", roles: [miner, sender]},
  { public-key: " /FILL/ ", roles: []},
]
}
}
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```

# Application logging level. Could be DEBUG | INFO | WARN | ERROR. Default value is INFO.
logging-level = DEBUG

tls {
  # Supported TLS types:
  # • EMBEDDED: Certificate is signed by node's provider and packed into JKS Keystore.
  ↳ The same file is used as a Truststore.
  #           Has to be manually imported into system by user to avoid certificate
  ↳ warnings.
  # • DISABLED: TLS is fully disabled
  type = DISABLED

  # type = EMBEDDED
  # keystore-path = ${node.directory}"/we.tls.jks"
  # keystore-password = ${TLS_KEYSTORE_PASSWORD}
  # private-key-password = ${TLS_PRIVATE_KEY_PASSWORD}
}

# P2P Network settings
network {
  # Network address
  bind-address = "0.0.0.0"
  # Port number
  port = 6864
  # Enable/disable network TLS
  tls = no

  # Peers network addresses and ports
  # Example: known-peers = ["node-1.com:6864", "node-2.com:6864"]
  known-peers = [ /FILL/ ]

  # Node name to send during handshake. Comment this string out to set random node name.
  # Example: node-name = "your-we-node-name"
  node-name = " /FILL/ "

  # How long the information about peer stays in database after the last communication
  ↳ with it
  peers-data-residence-time = 2h

  # String with IP address and port to send as external address during handshake. Could
  ↳ be set automatically if uPnP is enabled.
  # Example: declared-address = "your-node-address.com:6864"
  declared-address = "0.0.0.0:6864"

  # Delay between attempts to connect to a peer
  attempt-connection-delay = 5s
}

# New blocks generator settings
miner {
  enable = yes

```

(continues on next page)

(продолжение с предыдущей страницы)

```

# Important: use quorum = 0 only for testing purposes, while running a single-node
↪network;
# In other cases always set quorum > 0
quorum = 2
interval-after-last-block-then-generation-is-allowed = 10d
micro-block-interval = 5s
min-micro-block-age = 3s
max-transactions-in-micro-block = 500
minimal-block-generation-offset = 200ms
}

# Nodes REST API settings
api {
  rest {
    # Enable/disable REST API
    enable = yes

    # Network address to bind to
    bind-address = "0.0.0.0"

    # Port to listen to REST API requests
    port = 6862

    # Enable/disable TLS for REST
    tls = no
  }

  grpc {
    # Enable/disable gRPC API
    enable = yes

    # Network address to bind to
    bind-address = "0.0.0.0"

    # Port to listen to gRPC API requests
    port = 6865

    # Enable/disable TLS for gRPC
    tls = no
  }

  auth {
    type: "api-key"

    # Hash of API key string
    # You can obtain hashes by running ApiKeyHash generator
    api-key-hash: " /FILL/ "

    # Hash of API key string for PrivacyApi routes
    privacy-api-key-hash: " /FILL/ "
  }
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```

}

#Settings for Privacy Data Exchange
privacy {
  storage {
    vendor = postgres

    # for postgres vendor:
    schema = "public"
    migration-dir = "db/migration"
    profile = "slick.jdbc.PostgresProfile$"
    jdbc-config {
      url = "jdbc:postgresql://postgres:5432/node-1"
      driver = "org.postgresql.Driver"
    }
    user = postgres
    password = wenterprise
    connectionPool = HikariCP
    connectionTimeout = 5000
    connectionTestQuery = "SELECT 1"
    queueSize = 10000
    numThreads = 20
  }

  # for s3 vendor:
  # url = "http://localhost:9000/"
  # bucket = "privacy"
  # region = "aws-global"
  # access-key-id = "minio"
  # secret-access-key = "minio123"
  # path-style-access-enabled = true
  # connection-timeout = 30s
  # connection-acquisition-timeout = 10s
  # max-concurrency = 200
  # read-timeout = 0s
}

cleaner {
  enabled: no

  # The amount of time between cleanups
  # interval: 10m

  # How many blocks the data hash transaction exists on the blockchain, after which it
  → will be removed from cleaner monitoring
  # confirmation-blocks: 100

  # The maximum amount of time that a file can be stored without getting into the
  → blockchain
  # pending-time: 72h
}
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```

# Docker smart contracts settings
docker-engine {
  # Docker smart contracts enabled flag
  enable = yes

  # For starting contracts in a local docker
  use-node-docker-host = yes

  default-registry-domain = "registry.wavesenterprise.com/waves-enterprise-public"
  # Basic auth credentials for docker host
  #docker-auth {
  #  username = "some user"
  #  password = "some password"
  #}

  # Optional connection string to docker host
  docker-host = "unix:///var/run/docker.sock"

  # Optional string to node REST API if we use remote docker host
  # node-rest-api = "node-0"

  # Execution settings
  execution-limits {
    # Contract execution timeout
    timeout = 10s
    # Memory limit in Megabytes
    memory = 512
    # Memory swap value in Megabytes (see https://docs.docker.com/config/containers/
    ↪resource_constraints/)
    memory-swap = 0
  }

  # Reuse once created container on subsequent executions
  reuse-containers = yes

  # Remove container with contract after specified duration passed
  remove-container-after = 10m

  # Remote registries auth information
  remote-registries = []

  # Check registry auth on node startup
  check-registry-auth-on-startup = yes

  # Contract execution messages cache settings
  contract-execution-messages-cache {
    # Time to expire for messages in cache
    expire-after = 60m
    # Max number of messages in buffer. When the limit is reached, the node processes
    ↪all messages in batch
    max-buffer-size = 10
  }

```

(continues on next page)

(продолжение с предыдущей страницы)

```
# Max time for buffer. When time is out, the node processes all messages in batch
max-buffer-time = 100ms
}
}
}
```

6.2 accounts.conf

В этом примере включено шифрование Waves Crypto, использован стандартный идентифицирующий байт сети и отключена опция обновления keystore ноды для генерации 1 ключевой пары.

Пароль, который вам следует ввести самостоятельно, помечен как /FILL/.

accounts.conf:

```
accounts-generator {
  waves-crypto = yes
  chain-id = V
  amount = 1
  wallet = ${user.home}/node/keystore.dat
  wallet-password = "/FILL/"
  reload-node-wallet {
    enabled = false
    url = "http://localhost:6862/utils/reload-wallet"
  }
}
```

6.3 api-key-hash.conf

В этом примере включено шифрование Waves Crypto.

api-key-hash.conf:

```
apikeyhash-generator {
  waves-crypto = yes
  api-key = "some string for api-key"
}
```

6.4 Дополнительные примеры

Дополнительные примеры конфигурационных файлов с комментариями приведены в официальном [GitHub-репозитории Waves Enterprise](#).

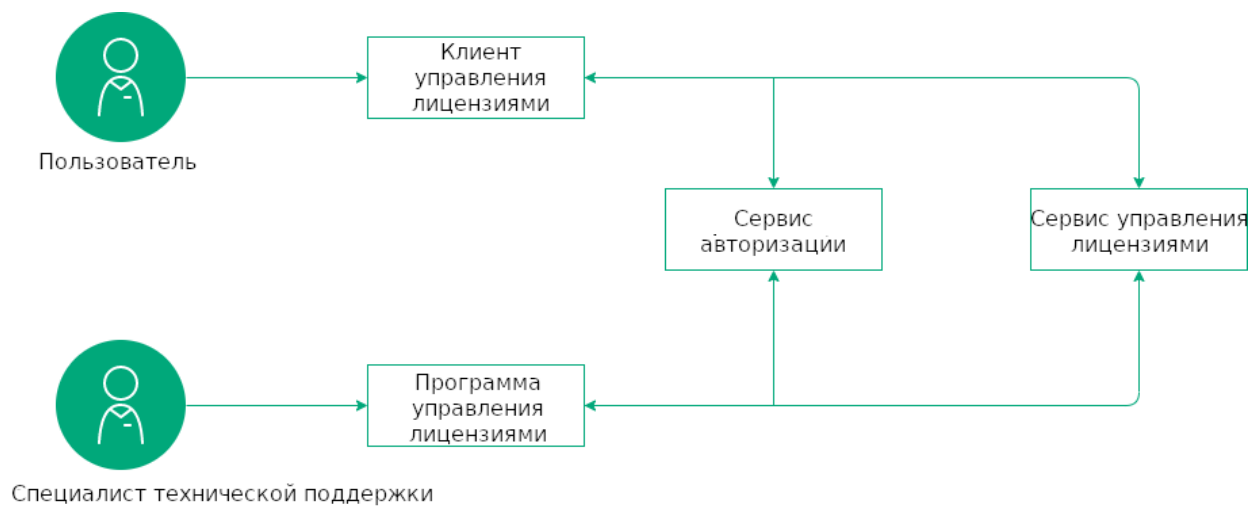
Смотрите также

[Развертывание платформы в частной сети](#)

[Генераторы](#)

Лицензии блокчейн-платформы Waves Enterprise

Блокчейн-платформа Waves Enterprise является коммерческим продуктом, ориентированным на использование в корпоративном и государственном секторах и распространяемым при помощи пользовательских лицензий. Схема получения лицензии на использование платформы выглядит так:



Для доступа к полученным лицензиям и управления ими предусмотрен [сервис управления лицензиями](#). Особенности работы с ним описаны в руководствах по установке платформы:

[Развертывание платформы с подключением к Mainnet](#)

[Развертывание платформы в частной сети](#)

7.1 Виды лицензий

Для ознакомления с возможностями платформы вам не потребуется лицензия. Детальное описание функциональности платформы и ее порядок ее установки в ознакомительном режиме приведены в статье *Развертывание платформы в ознакомительном режиме (Sandbox)*.

Для полноценного использования платформы доступны следующие виды лицензий:

- **Пробная лицензия** - позволяет ознакомиться с платформой и технологией в рамках реализации пилотного проекта партнера. Выдается по договору на срок пилотного проекта, либо на время разработки и отладки продукта.
- **Коммерческая лицензия** - позволяет использовать платформу для реализации коммерческих проектов. Выдается на срок, определяемый договорными отношениями с партнёром.
- **Некоммерческая лицензия** - позволяет использовать платформы в реализации проектов, не ставящих целью извлечение прибыли. Выдается на срок, определяемый договорными отношениями с партнёром.
- **Лицензия для работы в сети Mainnet** - специальная лицензия, позволяющая использовать блокчейн-сеть Waves Enterprise Mainnet для обмена данными и выполнения операций партнера. При работе в Mainnet предусмотрены *комиссии* за проводимые транзакции.

Каждый вид лицензии распространяется на одну ноду.

7.2 Сроки действия лицензий

Лицензии также различаются по сроку действия, который обговаривается при заключении договора:

- 3 месяца - стандартный срок действия пробной лицензии.
- Аренда на время использования платформы - определяется временными рамками проекта, реализуемого с использованием блокчейн-платформы.
- 1 год.
- 2 года.
- Бессрочная лицензия.

После истечения срока действия лицензии нода, на которую распространяется действие лицензии, теряет возможность формировать новые блоки и отправлять новые транзакции в сеть.

Для обсуждения количества лицензий и нод в вашей сети, а также других условий партнерства с Waves Enterprise свяжитесь с отделом продаж Waves Enterprise по электронной почте: sales@wavesenterprise.com.

Смотрите также

Комиссии в сети «Waves Enterprise Mainnet»

Комиссии в сети «Waves Enterprise Mainnet»

Но- мер тран- зак- ции	Название транзак- ции	Комис- сия	Описание	
1	<i>Genesis transaction</i>	отсут- ствует	Первоначальная привязка баланса к адресам создаваемых при старте блокчейна нод	
3	<i>Issue Transaction</i>	1 WEST	Выпуск токенов. Комиссия взимается только в WEST	
4	<i>Transfer Transaction</i>	0.01 WEST	Перевод токенов	
5	<i>Reissue Transaction</i>	1 WEST	Перевыпуск токенов	
6	<i>Burn Transaction</i>	0.05 WEST	Сжигание токенов	
8	<i>Lease Transaction</i>	0.01 WEST	Передача токенов в аренду	
9	<i>Lease Cancel Transaction</i>	0.01 WEST	Отмена аренды токенов	
10	<i>Create Alias Transaction</i>	1 WEST	Создание псевдонима	
11	<i>MassTransfer Transaction</i>	0.05 WEST	Массовый перевод токенов. Указана мини- мальная комиссия	размер комиссии за- висит от количества адресов в транзакции
12	<i>Data Transaction</i>	0.05 WEST за ки- лобайт	Транзакция с данными в виде полей с парой ключ-значение. Комиссия всегда взимается с автора транзакции. Указана минимальная ко- миссия	размер комиссии за- висит от размера дан- ных
13	<i>SetScript Transaction</i>	0.5 WEST	Транзакция	привязывающая скрипт с RIDE- контрактом к аккау- нту
14	<i>Sponsorship Transaction</i>	1 WEST	Установка или отмена спонсорства	
15	<i>SetAssetScript Transaction</i>	1	Транзакция	привязывающая

Смотрите также

licenses

Развертывание платформы с подключением к Mainnet

Инструментарий gRPC

Блокчейн-платформа Waves Enterprise предоставляет возможность взаимодействия с блокчейном при помощи gRPC-интерфейса.

gRPC - это высокопроизводительный фреймворк для удаленного вызова процедур (Remote Procedure Call, RPC), разработанный корпорацией Google. Фреймворк работает поверх HTTP/2. Для передачи данных между клиентом и сервером используется формат сериализации **protobuf**, описывающий применяемые типы данных.

Официально gRPC поддерживает 10 языков программирования. Список поддерживаемых языков доступен в [официальной документации gRPC](#).

9.1 Предварительная настройка gRPC-интерфейса

Перед использованием gRPC-интерфейса:

1. определитесь с языком программирования, который вы будете применять для взаимодействия с нодой;
2. установите фреймворк gRPC для вашего языка программирования в соответствии с [официальной документацией gRPC](#);
3. скачайте и распакуйте пакет protobuf-файлов [we-proto-x.x.x.zip](#) для используемой вами версии платформы, а также плагин [protoc](#) для компиляции protobuf-файлов;
4. убедитесь, что gRPC-интерфейс *запущен и настроен в конфигурационном файле ноды*, с которой будет производиться обмен данными.

Для взаимодействия с нодой через gRPC-интерфейс по умолчанию предусмотрен порт **6865**.

9.2 Для чего предназначен gRPC-интерфейс платформы

Вы можете использовать gRPC-интерфейс каждой ноды для следующих задач:

9.2.1 gRPC: отслеживание событий в блокчейне

gRPC-интерфейс имеет возможность отслеживания определенных групп событий, происходящих в блокчейне. Информация о выбранных группах событий собирается в потоки, которые поступают в gRPC-интерфейс ноды.

Набор полей, предназначенный для сериализации и передачи данных о событиях в блокчейне, приведен в файлах, которые находятся в каталоге **messagebroker** пакета **we-proto-x.x.x.zip**:

- **messagebroker_blockchain_events_service.proto** - основной protobuf-файл;
- **messagebroker_subscribe_on_request.proto** - файл, содержащий поля с параметрами запроса;
- **messagebroker_blockchain_event.proto** - файл, содержащий поля ответов с данными групп событий и сообщениями об ошибках.

Для отслеживания определенной группы событий в блокчейне отправьте запрос **SubscribeOn(startFrom, transactionTypeFilter)**, который инициализирует подписку на выбранную группу событий.

Параметры запроса:

startFrom - момент начала отслеживания событий:

- **CurrentEvent** - начало отслеживания от текущего события;
- **GenesisBlock** - получение всех событий выбранной группы, начиная от генезис-блока;
- **BlockSignature** - начало отслеживания от указанного блока.

transactionTypeFilter - фильтрация выводимых событий по транзакциям, которые производятся в ходе этих событий:

- **Any** - выводить события со всеми типами транзакций;
- **Filter** - выводить события с типами транзакций, указанных в виде списка;
- **FilterNot** - выводить события со всеми транзакциями кроме тех, которые указаны в этом параметре в виде списка.

Вместе с запросом **SubscribeOnRequest** отправляются данные авторизации: JWT-токен или ключевая фраза **api-key**, в зависимости от используемого метода авторизации.

Информация о событиях

После успешной отправки запроса на gRPC-интерфейс будут приходить данные следующих групп событий:

1. **MicroBlockAppended** - успешный майнинг микроблока:
 - **transactions** - полные тела транзакций из полученного микроблока.
2. **BlockAppended** - успешное завершение раунда майнинга с формированием блока:
 - **block_signature** - подпись полученного блока;
 - **reference** - подпись предыдущего блока;
 - **tx_ids** - список ID транзакций из полученного блока;
 - **miner_address** - адрес майнера;

- `height` - высота, на которой расположен полученный блок;
- `version` - версия блока;
- `timestamp` - время формирования блока;
- `fee` - сумма комиссий за транзакции внутри блока;
- `block_size` - размер блока (в байтах);
- `features` - список изменений блокчейна, за которые голосовал майнер в ходе раунда.

3. **RollbackCompleted** - откат блока:

- `return_to_block_signature` - подпись блока, до которого произошел откат;
- `rollback_tx_ids` - список ID транзакций, которые будут удалены из блокчейна.

4. **AppendedBlockHistory** - информация о транзакциях сформированного блока. Данный тип событий поступает на gRPC-интерфейс до достижения текущей высоты блокчейна, если в запросе в качестве отправной точки для получения событий указаны `GenesisBlock` или `BlockSignature`. После достижения текущей высоты начинают выводиться текущие события по заданным фильтрам.

Данные ответа:

- `signature` - подпись блока;
- `reference` - подпись предыдущего блока;
- `transactions` - полные тела транзакций из блока;
- `miner address` - адрес майнера;
- `height` - высота, на которой расположен блок;
- `version` - версия блока;
- `timestamp` - время формирования блока;
- `fee` - сумма комиссий за транзакции внутри блока;
- `block_size` - размер блока (в байтах);
- `features` - список изменений блокчейна, за которые голосовал майнер в ходе раунда.

Информация об ошибках

Для вывода информации об ошибках в ходе отслеживания событий в блокчейне предусмотрено сообщение `ErrorEvent` со следующими вариантами ошибок:

- `GenericError` - общая или неизвестная ошибка с текстом сообщения;
- `MissingRequiredRequestField` - не заполнено обязательное поле при формировании запроса `SubscribeOnRequest`;
- `BlockSignatureNotFoundError` - в блокчейне отсутствует подпись запрошенного блока;
- `MissingAuthorizationMetadata` - при формировании запроса `SubscribeOn` не введены данные авторизации;
- `InvalidApiKey` - при авторизации по `api-key`, неверная ключевая фраза;
- `InvalidToken` - авторизации по OAuth, неверный JWT-токен.

Смотрите также

Инструментарий gRPC

9.2.2 gRPC: получение параметров конфигурации ноды

Для получения параметров конфигурации ноды предусмотрен метод и одноименный запрос `NodeConfig`. Этот метод упакован в protobuf-файл `util_node_info_service.proto`.

Запрос `NodeConfig` не требует ввода дополнительных параметров. В ответе выводятся следующие параметры конфигурации ноды, к которой был осуществлен запрос:

- `version` - используемая версия блокчейн-платформы;
- `crypto_type` - используемый криптографический алгоритм;
- `chain_id` - идентифицирующий байт сети;
- `consensus` - используемый алгоритм консенсуса;
- `minimum_fee` - минимальная комиссия за транзакции;
- `additional_fee` - дополнительная комиссия за транзакции;
- `max_transactions_in_micro_block` - максимальное установленное количество транзакций в микроблоке;
- `min_micro_block_age` - минимальное время существования микроблока (в секундах);
- `micro_block_interval` - интервал формирования микроблоков (в секундах);
- `pos_round_info`: при использовании алгоритма консенсуса PoS, выводится параметр `average_block_delay` (время средней задержки создания блоков, в секундах);
- `poa_round_info`: при использовании алгоритма консенсуса PoA, выводятся параметры `round_duration` (длина раунда майнинга блока, в секундах) и `sync_duration` (период синхронизации майнинга блока, в секундах).

Смотрите также

Инструментарий gRPC

9.2.3 gRPC: получение информации о транзакции по ее ID

Для получения информации о транзакции по ее ID на gRPC-интерфейс ноды предназначены два запроса, упакованных в файл `contract_transaction_service.proto`:

- `TransactionExists` - запрос о существовании транзакции с указанным ID;
- `TransactionInfo` - запрос информации о транзакции с указанным ID.

Оба запроса требуют ввода параметра `tx_id` - ID транзакции, о которой запрашивается информация.

Ответ на запрос `TransactionExists`:

- `exists` - булев тип данных: `true` - существует, `false` - не существует.

В ответе на запрос `TransactionInfo` содержится следующая информация о транзакции:

- `height` - высота блокчейна, на которой была произведена транзакция;

- `transaction` - название транзакции.

Смотрите также

Инструментарий gRPC

9.2.4 gRPC: получение информации о состоянии смарт-контракта

Для получения параметров конфигурации ноды предусмотрен запрос `ContractExecutionStatuses`. Поля этого запроса содержатся в protobuf-файле `util_contract_status_service.proto`.

Запрос `ContractExecutionStatuses` требует ввода параметра `tx_id` - ID транзакции вызова смарт-контракта, информацию о состоянии которого необходимо получить.

Информация о состоянии смарт-контракта

В ответе на запрос `ContractExecutionStatuses` выводятся следующие данные смарт-контракта:

- `sender` - участник, отправивший смарт-контракт в блокчейн;
- `tx_id` - ID транзакции вызова смарт-контракта;
- `Status` - информация об исполнении смарт-контракта: 0 - успешно исполнен (SUCCESS), 1 - исполнен с ошибкой (ERROR), 2 - не исполнен (FAILURE);
- `code` - код ошибки в ходе выполнения смарт-контракта;
- `message` - сообщение об ошибке;
- `timestamp` - время вызова смарт-контракта;
- `signature` - подпись смарт-контракта.

Смотрите также

Инструментарий gRPC

9.2.5 gRPC: получение информации о размере UTX-пула

Запрос о размере UTX-пула `UtxInfo` отправляется в виде подписки: после его отправки ответ от ноды приходит раз в 1 секунду. Это запрос не требует ввода дополнительных параметров и находится в файле `transaction_public_service.proto`.

В ответ на запрос выводится сообщение `UtxSize`, которое содержит два параметра:

- `size` - размер UTX-пула в килобайтах;
- `size_in_bytes` - размер UTX-пула в байтах.

Смотрите также

Инструментарий gRPC

9.2.6 gRPC: формирование и проверка электронной подписи данных (PKI)

Для сетей, работающих с использованием ГОСТ-криптографии, gRPC-интерфейс имеет возможность формирования отсоединенной электронной подписи для передаваемых данных, а также ее проверки. Для этого предусмотрено два метода, упакованных в protobuf-файл **contract_pki_service.proto**:

- **Sign** - формирование отсоединённой ЭП для данных, передаваемых в запросе.
- **Verify** - проверка отсоединённой ЭП для данных, передаваемых в запросе.

Формирование отсоединенной ЭП

Метод **Sign** требует ввода следующих параметров:

- **input_data** - данные, для которых требуется ЭП (в виде массива байт в кодировке **base64**);
- **keystore_alias** - имя хранилища для закрытого ключа ЭП;
- **password** - пароль хранилища для закрытого ключа;
- **sig_type** - формат ЭП. Поддерживаемые форматы: 1 - CAdES-BES, 2 - CAdES-X Long Type 1, 3 - CAdES-T.

Ответ метода содержит поле **signature** со сформированной электронной подписью в формате **base64**.

Проверка отсоединенной ЭП

Метод **Verify** требует ввода следующих параметров:

- **input_data** - данные, закрытые ЭП (в виде массива байт в кодировке **base64**);
- **signature** - электронная подпись в виде массива байт в кодировке **base64**;
- **sig_type** - формат ЭП. Поддерживаются значения: 1 - CAdES-BES, 2 - CAdES-X Long Type 1, 3 - CAdES-T.
- **extended_key_usage_list** - список объектных идентификаторов (OID) криптографических алгоритмов, которые используются при формировании ЭП (*опциональное поле*).

Ответ метода содержит поле **status** с булевым типом данных: **true** - подпись действительна, **false** - подпись скомпрометирована.

Проверка УКЭП

Метод **verify** имеет возможность проверки усиленной квалифицированной электронной подписи (УКЭП). Для корректной проверки УКЭП установите на вашу ноду корневой сертификат ЭЦП удостоверяющего центра (УЦ), при помощи которого будет осуществляться валидация подписи.

Корневой сертификат устанавливается в хранилище сертификатов **cacerts** используемой вами виртуальной машины Java (JVM) при помощи утилиты **keytool**:

```
sudo keytool -import -alias certificate_alias -keystore path_to_your_JVM/lib/security/
cacerts -file path_to_the_certificate/cert.cer
```

После флага `-alias` укажите предпочтительное вам имя сертификата в хранилище.

Хранилище сертификатов `cacerts` расположено в поддиректории `/lib/security/` вашей виртуальной машины Java. Чтобы узнать путь к виртуальной машине на Linux, воспользуйтесь следующей командой:

```
readlink -f /usr/bin/java | sed "s:bin/java::"
```

Затем добавьте к полученному пути `/lib/security/cacerts` и вставьте полученный абсолютный путь к **cacerts** после флага `-keystore`.

После флага `-file` укажите абсолютный или относительный путь к полученному сертификату ЭЦП удостоверяющего центра.

Пароль по умолчанию для **cacerts** - `changeit`. При необходимости, вы можете изменить его при помощи утилиты **keytool**:

```
sudo keytool -keystore cacerts -storepasswd
```

Смотрите также

Инструментарий gRPC

9.2.7 gRPC: реализация методов шифрования

gRPC-интерфейс ноды предусматривает возможность зашифровать произвольные данные при помощи алгоритмов шифрования, применяемых в блокчейн-платформе Waves Enterprise, а также дешифровать их. Для этого предусмотрен набор запросов, упакованных в файл **contract_crypto_service.proto**:

- **EncryptSeparate** - шифрование данных уникальными ключами СЕК отдельно для каждого получателя, каждый СЕК шифруется (*оборачивается*) отдельным ключом КЕК;
- **EncryptCommon** - шифрование данных единым ключом СЕК для всех получателей, каждый ключ СЕК шифруется (*оборачивается*) отдельным ключом КЕК для каждого получателя;
- **Decrypt** - дешифровка данных.

Подсказка: Дешифровка данных возможна, если ключ получателя сообщения находится в `keystore` ноды.

Запросы и ответы шифрования

Запросы **EncryptSeparate** и **EncryptCommon** требуют ввода следующих данных:

- `sender` - адрес отправителя данных;
- `password` - пароль к зашифрованным данным;
- `encryption_data` - шифруемые данные (в виде массива байт в кодировке **base64**);
- `recipients_public_keys` - публичные ключи получателей-участников сети;
- `crypto_algo` - используемый алгоритм шифрования. Доступные значения: 1 - ГОСТ 28147-89, 2 - ГОСТ 34.12-2015, 3 - AES.

В ответе на запрос **EncryptSeparate** поступают следующие данные для каждого получателя:

- `encrypted_data` - зашифрованные данные;

- `public_key` - публичный ключ получателя;
- `wrapped_key` - результат шифрования ключа для получателя.

В ответе на запрос `EncryptCommon` поступают следующие данные:

- `encrypted_data` - зашифрованные данные;
- `recipient_to_wrapped_structure` - структура в формате «ключ : значение», содержащая публичные ключи получателей с соответствующими результатами шифрования ключей для каждого из них.

Запрос и ответ дешифровки

При запросе `Decrypt` вводятся следующие данные:

- `recipient` - публичный ключ получателя из `keystore` ноды;
- `password` - пароль к зашифрованным данным;
- `encrypted_data` - зашифрованные данные;
- `wrapped_key` результат шифрования ключа для указанного получателя;
- `sender_public_key` - публичный ключ отправителя данных;
- `crypto_algo` - используемый алгоритм шифрования. Доступные значения: 0 - алгоритм неизвестен, 1 - ГОСТ 28147-89, 2 - ГОСТ 34.12-2015, 3 - AES.

В ответ на запрос `Decrypt` поступает поле `decrypted_data`, содержащая расшифрованные данные в виде массива байт в кодировке **base64**.

Смотрите также

[Инструментарий gRPC](#)

9.2.8 gRPC: отправка транзакций в блокчейн

gRPC-интерфейс поддерживает возможность отправки транзакций в блокчейн посредством отправки запроса `Broadcast`, упакованного в файл **`transaction_public_service.proto`**.

Запрос `Broadcast` требует ввода следующих параметров:

- `version` - версия транзакции;
- `transaction` - название транзакции вместе предназначенным для нее набором параметров.

Для каждой транзакции предусмотрен отдельный `protobuf`-файл, описывающий поля запросов и ответов. Эти поля универсальны для запросов по gRPC и REST API и приведены в статье [Транзакции блокчейн-платформы](#).

Смотрите также

Инструментарий gRPC

Описание транзакций

Комиссии в сети «Waves Enterprise Mainnet»

Для каждой из этих задач предусмотрен собственный набор методов, упакованный в соответствующие protobuf-файлы. С детальным описанием каждого набора методов вы можете ознакомиться в статьях выше.

gRPC-методы ноды, в отличие от методов REST API, не требуют авторизации. Также все методы, упакованные в **protobuf**-файлы, размещенные в директории **contract**, доступны как для ноды, так и для смарт-контрактов. При использовании в смарт-контрактах эти методы требуют авторизации.

Смотрите также

Сервисы gRPC, используемые смарт-контрактом

Внимание: Блокчейн-платформа Waves Enterprise постепенно отказывается от методов REST API. В новых версиях платформы методы REST API развиваться не будут.


REST API позволяет пользователям удалённо взаимодействовать с нодой через запросы и ответы в формате JSON. Работа с API происходит по протоколу https. В качестве интерфейса к REST API применяется фреймворк Swagger.

10.1 Использование REST API

Все вызовы методов REST API — это https-запросы GET, POST или DELETE к URL `https://yournetwork.com/node-N/api-docs/swagger.json`, содержащие соответствующие наборы параметров. Нужные группы запросов с выбираются в интерфейсе Swagger посредством выбора маршрутов (routes) - URL к отдельным методам REST API. В конце каждого маршрута предусмотрена точка доступа (endpoint) - обращение к методу.

Пример запроса о размере UTX-пула:

Route **Endpoint**


[GET/transactions/unconfirmed/size](#)

Для использования практически всех методов REST API требуется авторизация по api-key или JWT-токену.

При авторизации по api-key укажите значение выбранной ключевой фразы, а при авторизации по JWT-токену - значение **access**-токена.

При этом, для методов, связанных с доступом к ноде, предусмотрена исключительно авторизация по api-key:

- доступ к keystore ноды (например, метод sign);
- работа с группами доступа к приватным данным;
- доступ к конфигурации ноды.

Если используется авторизация по JWT-токену, доступ к этим методам будет закрыт.

10.2 Для чего предназначен REST API платформы

Вы можете использовать интерфейс REST API для выполнения следующих задач:

10.2.1 REST API: работа с транзакциями

Для работы с транзакциями предусмотрены методы группы transactions.

Подписание и отправка транзакций

REST API ноды использует JSON-представление транзакции для отправки запросов.

Основные принципы работы с транзакциями приведены в разделе [Транзакции блокчейн-платформы](#). Описание полей для каждой транзакции приведено в разделе [Описание транзакций](#).

Для подписания транзакций предназначен метод **POST /transactions/sign**. Этот метод подписывает транзакцию закрытым ключом отправителя, сохраненным в keystore ноды. Для подписания запросов ключом из keystore ноды обязательно укажите пароль к ключевой паре в поле password.

Пример запроса на подписание [транзакции 3](#):

POST /transactions/sign:

```
{
  "type": 3,
  "version": 2,
  "name": "Test Asset 1",
  "quantity": 100000000000,
  "description": "Some description",
  "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
  "decimals": 8,
  "reissuable": true,
  "password": "1234",
  "fee": 100000000
}
```

Метод **POST /transactions/sign** в ответе возвращает поля, необходимые для публикации транзакции.

Пример ответа с [транзакцией 3](#):

POST /transactions/sign:

```
{
  "type": 3,
  "id": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
  "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "fee": 100000000,
  "timestamp": 1549378509516,
  "proofs": [
    ↪ "NqZGcbcbQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFAknTMEGuFrEndCjXBtrueLWaqbJhpeiG
    ↪ " ],
  "version": 2,
  "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
  "name": "Test Asset 1",
  "quantity": 10000,
  "reissuable": true,
  "decimals": 8,
  "description": "Some description",
  "chainId": 84,
  "script": "base64:AQa3b8tH",
  "height": 60719
}
```

Для публикации транзакции предназначен метод **POST /transactions/broadcast**. На вход этого метода подаются поля ответа метода **sign**. Также транзакция может быть отправлена в блокчейн при помощи других инструментов, приведенных в статье [Транзакции блокчейн-платформы](#).

Помимо отдельных методов подписания и отправки транзакций, предусмотрен комбинированный метод **POST /transactions/signAndBroadcast**. Этот метод подписывает и отправляет транзакцию в блокчейн без промежуточной передачи информации между методами.

Пример запроса и ответа метода с [транзакцией 112](#):

POST /transactions/signAndBroadcast:

Запрос:

```
{
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "policyName": "Policy for sponsored v1",
  "password": "sfgKYBFCF@#$fsdf()*%",
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  ],
  "fee": 100000000,
  "description": "Privacy for sponsored",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 112
}

```

Ответ:

```

{
  "senderPublicKey": "3X6Qb6p96dY4drVt3x4XyHKCRvree4QDqNZyDWHzjJ79",
  "policyName": "Policy for sponsored v1",
  "fee": 100000000,
  "description": "Privacy for sponsored",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 112,
  "version": 2,
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
  "proofs": [
    ↪ "3vDVjp6UJeN9ahtNcQWt5WDVqC9KqdEsrr9HTToHfoXfd1HtVwnUPPtJKM8tAsCtby81XYQReLj33hLEZ8qbGA3V
    ↪ "
  ],
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAooHUoLSAQvxBSqjE91WK3LwWGjiiCxx"
  ],
  "id": "EeymzQcM2LrsgGDFfxGn8DhahJbFYmorcBrEh8phv5S",
  "timestamp": 1585307711344
}

```

Информация о транзакциях

Группа `transactions` также включает следующие методы получения информации о транзакциях в блокчейне:

GET /transactions/info/{id}

Получение информации о транзакции по ее идентификатору {id}. Идентификатор транзакции указывается в ответе методов **POST /transactions/sign** или **POST /transactions/signAndBroadcast**.

Метод возвращает данные транзакции, аналогичные ответам методов **POST /transactions/broadcast** и **POST /transactions/signAndBroadcast**.

Пример ответа:

POST /transactions/signAndBroadcast:

```
{
  "type": 4,
  "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
  "sender": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
  "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMHm3Uki7pLw",
  "recipient": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
  "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
  "amount": 100000,
  "fee": 100000,
  "timestamp": 1549365736923,
  "attachment": "string",
  "signature":
  → "GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm
  → ",
  "height": 7782
}
```

GET /transactions/address/{address}/limit/{limit}

Метод возвращает данные последних {limit} транзакций адреса {address}.

Для каждой транзакции возвращаются данные, аналогичные ответам методов **POST /transactions/broadcast** и **POST /transactions/signAndBroadcast**.

Пример ответа для одной транзакции:

GET /transactions/address/{address}/limit/{limit}:

```
[
  [
    {
      "type": 2,
      "id":
      → "4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CddnyBW
      → ",
      "fee": 100000,
      "timestamp": 1549365736923,
      "signature":
      → "4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CddnyBW
      → "
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    ↪ ",
    "sender": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMH3Uki7pLw",
    "recipient": "3N9iRMou3pgmyPbFZn5QZQvBTQBkL2fR6R1",
    "amount": 1000000000
  }
]
]

```

GET /transactions/unconfirmed

Метод возвращает данные всех транзакций из UTX-пула ноды.

Для каждой транзакции возвращаются данные, аналогичные ответам методов **POST /transactions/broadcast** и **POST /transactions/signAndBroadcast**.

Пример ответа для одной транзакции:

GET /transactions/unconfirmed:

```

[
  {
    "type": 4,
    "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
    "sender": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMH3Uki7pLw",
    "recipient": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
    "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
    "amount": 100000,
    "fee": 100000,
    "timestamp": 1549365736923,
    "attachment": "string",
    "signature":
    ↪ "GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm
    ↪ "
  }
]

```

GET /transactions/unconfirmed/size

Метод возвращает количество транзакций, находящихся в UTX-пуле в виде числа.

Пример ответа:

GET /unconfirmed/info/{id}:

```
{
  "size": 4
}
```

GET /unconfirmed/info/{id}

Метод возвращает данные транзакции, находящейся в UTX-пуле, по ее {id}.

В ответе метода содержатся данные транзакции, аналогичные ответам методов **POST /transactions/broadcast** и **POST /transactions/signAndBroadcast**.

Пример ответа:

GET /unconfirmed/info/{id}:

```
{
  "type": 4,
  "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
  "sender": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8",
  "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMHM3Uki7pLw",
  "recipient": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8",
  "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
  "amount": 100000,
  "fee": 100000,
  "timestamp": 1549365736923,
  "attachment": "string",
  "signature":
  ↪ "GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjetsUm
  ↪ ",
  "height": 7782
}
```

POST /transactions/calculateFee

Метод возвращает сумму комиссии за отправленную транзакцию.

В запросе указываются параметры, аналогичные запросу **POST /transactions/broadcast**. В ответе метода возвращается идентификатор ассета, в котором взимается комиссия (null для WAVES).

Пример ответа:

GET /unconfirmed/info/{id}:

```
{
  "feeAssetId": null,
  "feeAmount": 10000
}
```

Смотрите также

Методы REST API

Транзакции блокчейн-платформы

Описание транзакций

10.2.2 REST API: формирование и проверка электронной подписи данных (PKI)

Для формирования и проверки электронных подписей предусмотрена группа методов `pki`.

Принцип работы этой методов `POST /pki/sign` и `POST /pki/verify` аналогичен методам *gRPC-методов* `contract_pki_service.proto`.

Все методы группы доступны только для сетей с ГОСТ-криптографией.

GET /pki/keystoreAliases

Метод возвращает список с именами всех доступных хранилищ закрытых ключей ЭП.

Пример ответа:

GET /pki/keystoreAliases:

```
{
  [
    "3Mq9crNkTFf8oRPyisgtf4TjBvZxo4BL2ax",
    "e19a135e-11f7-4f0c-9109-a3d1c09812e3"
  ]
}
```

POST /pki/sign

Метод формирует отсоединённую ЭП для данных, передаваемых в запросе. Запрос состоит из следующих полей:

- `inputData` - данные, для которых требуется ЭП (в виде массива байт в кодировке **base64**);
- `keystoreAlias` - имя хранилища для закрытого ключа ЭП;
- `password` - пароль хранилища для закрытого ключа;

- sigType - формат ЭП. Поддерживаемые форматы: 1 - CAdES-BES, 2 - CAdES-X Long Type 1, 3 - CAdES-T.

Пример запроса:

POST /pki/sign:

```
{
  "inputData" : "SGVsbG8gd29ybGQh",
  "keystoreAlias" : "key1",
  "password" : "password",
  "sigType" : 1,
}
```

Метод возвращает поле signature, содержащее сгенерированную отсоединенную ЭП.

Пример ответа:

POST /pki/sign:

```
{
  "signature" :
  → "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQjndjfvnksdnjfn="
}
```

POST /pki/verify

Метод предназначен для проверки отсоединённой ЭП для данных, передаваемых в запросе. Запрос состоит из следующих полей:

- inputData - данные, закрытые ЭП (в виде массива байт в кодировке **base64**);
- signature - электронная подпись в виде массива байт в кодировке **base64**;
- sigType - формат ЭП. Поддерживаются значения: 1 - CAdES-BES, 2 - CAdES-X Long Type 1, 3 - CAdES-T.
- extended_key_usage_list - список объектных идентификаторов (OID) криптографических алгоритмов, которые используются при формировании ЭП (*опциональное поле*).

Пример запроса:

POST /pki/verify:

```
{
  "inputData" : "SGVsbG8gd29ybGQh",
  "signature" : "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQ=",
  "sigType" : "CAdES_X_Long_Type_1",
  "extendedKeyUsageList": [
    "1.2.643.7.1.1.1.1",
    "1.2.643.2.2.35.2"
  ]
}
```

Ответ метода содержит поле `sigStatus` с булевым типом данных: `true` - подпись действительна, `false` - подпись скомпрометирована.

Пример ответа:

POST /pki/verify:

```
{
  "sigStatus" : "true"
}
```

Проверка УКЭП

Метод `POST /pki/verify` имеет возможность проверки усиленной квалифицированной электронной подписи (УКЭП). Для корректной проверки УКЭП установите на вашу ноду корневой сертификат ЭЦП удостоверяющего центра (УЦ), при помощи которого будет осуществляться валидация подписи.

Корневой сертификат устанавливается в хранилище сертификатов **cacerts** используемой вами виртуальной машины Java (JVM) при помощи утилиты **keytool**:

```
sudo keytool -import -alias certificate_alias -keystore path_to_your_JVM/lib/security/
cacerts -file path_to_the_certificate/cert.cer
```

После флага `-alias` укажите предпочтительное вам имя сертификата в хранилище.

Хранилище сертификатов `cacerts` расположено в поддиректории `/lib/security/` вашей виртуальной машины Java. Чтобы узнать путь к виртуальной машине на Linux, воспользуйтесь следующей командой:

```
readlink -f /usr/bin/java | sed "s:bin/java::"
```

Затем добавьте к полученному пути `/lib/security/cacerts` и вставьте полученный абсолютный путь к **cacerts** после флага `-keystore`.

После флага `-file` укажите абсолютный или относительный путь к полученному сертификату ЭЦП удостоверяющего центра.

Пароль по умолчанию для **cacerts** - `changeit`. При необходимости, вы можете изменить его при помощи утилиты **keytool**:

```
sudo keytool -keystore cacerts -storepasswd
```

Смотрите также

[Методы REST API](#)

[Криптография](#)

10.2.3 REST API: реализация методов шифрования

Для реализации методов шифрования предусмотрены методы REST API группы `crypto`.

Принцип работы этой группы методов аналогичен набору *gRPC-методов* `contract_crypto_service.proto`.

POST /crypto/encryptSeparate

Шифрование данных, переданных в запросе, уникальными ключами CEK отдельно для каждого получателя, каждый CEK шифруется (*оборачивается*) отдельным ключом KEK.

В запросе подаются следующие данные:

- `sender` - адрес отправителя данных;
- `password` - пароль к зашифрованным данным;
- `encryptionText` - шифруемые данные (в виде строки);
- `recipientsPublicKeys` - публичные ключи получателей-участников сети;
- `crypto_algo` - используемый алгоритм шифрования. Доступные значения: `gost-28147` - ГОСТ 28147-89, `gost-3412-2015-k` - ГОСТ 34.12-2015, `aes` - AES.

Если в вашей сети используется шифрование по ГОСТ, вам будут доступны только алгоритмы `gost-28147` и `gost-3412-2015-k`. При отключенном шифровании по ГОСТ доступен только алгоритм шифрования `aes`.

Пример запроса:

POST /crypto/encryptSeparate:

```
{
  "sender": "3MsHHc8LvyjPCKeSst9vsYcsHeQVzH6YJkL",
  "password": "",
  "encryptionText": "some string to encrypt",
  "recipientsPublicKeys": [
    "3MuNFC1Z8Tuy73pMzVUT6yowk4anWA8MNNE"
  ],
  "cryptoAlgo": "aes"
}
```

В ответе метода поступают следующие данные для каждого получателя:

- `encrypted_data` - зашифрованные данные;
- `public_key` - публичный ключ получателя;
- `wrapped_key` - результат шифрования ключа для получателя.

Пример ответа:

POST /crypto/encryptSeparate:

```
{
  "encryptedText": "IZ5Kk5YNspMWl/jm1TizVxD6Nik=",
  "publicKey":
  ↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc
  ↪ ",
  "wrappedKey":
  ↪ "uWVoxJAzruwTDDSbphDS31TjSQX6CSWXivp3x34uE3XtnMqqK9swoaZ3LyAgFDR7o6CfkgzFkWmTen4qAZewPfBbwR
  ↪ "
},
```

POST /crypto/encryptCommon

Шифрование данных, переданных в запросе, единым ключом СЕК для всех получателей, каждый ключ СЕК шифруется (*оборачивается*) отдельным ключом КЕК для каждого получателя.

В запросе **POST /crypto/encryptCommon** подаются данные, аналогичные запросу **POST /crypto/encryptSeparate**.

В ответе метода поступают следующие данные:

- encrypted_data - зашифрованные данные;
- recipient_to_wrapped_structure - структура в формате «ключ : значение», содержащая публичные ключи получателей с соответствующими результатами шифрования ключей для каждого из них.

Пример ответа:**POST /crypto/encryptCommon:**

```
{
  "encryptedText": "NpCCig2i3jzo0xBnfqjfedbti8Y=",
  "recipientToWrappedStructure": {
    ↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc
    ↪ ":
    ↪ "M8pAe8HnKiWLE1HsC1ML5t8b7giWxiHfvagh7Y3F7rZL8q1tqMCJMYJo4qz4b3xjcuuUiV57tY3k7oSig53Aw1Dkkw
    ↪ ",
    ↪ "9LopMj2GqWxBYgnZ2gxaNwxXqXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S
    ↪ ":
    ↪ "Doqn6gPvBBesU2vdwgFYMbDhM4knEGMbqPn8Np76mNRRoZXLdioofyVbSSaTTER4cvXwzEwVMugiy2wuzFwk3zCiT3
    ↪ "
  }
}
```

POST /crypto/decrypt

Дешифровка данных, зашифрованных при помощи криптографического алгоритма, используемого сетью. Дешифровка возможна, если ключ получателя сообщения находится в keystore ноды.

В запросе подаются следующие данные:

- recipient - публичный ключ получателя из keystore ноды;
- password - пароль к зашифрованным данным;
- encryptedText - зашифрованная строка;
- wrappedKey результат шифрования ключа для указанного получателя;
- senderPublicKey - публичный ключ отправителя данных;
- cryptAlgo - используемый алгоритм шифрования. Доступные значения: gost-28147 - ГОСТ 28147-89, gost-3412-2015-k - ГОСТ 34.12-2015, aes - AES.

Если в вашей сети используется шифрование по ГОСТ, для дешифровки будут доступны только алгоритмы gost-28147 и gost-3412-2015-k. При отключенном шифровании по ГОСТ доступен только алгоритм шифрования aes.

Пример запроса:

POST /crypto/decrypt:

```
{
  "recipient": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "password": "12345qwert",
  "encryptedText":
  → "t859AE7idnjPpn3lUiorfzSGwcGPMVd0hQe1HAhoIOMOX0QPBc8TUhn+8pKRCL8evH2Ra9Vc",
  "wrappedKey": "2nfob2yW76xj2rQBWZkzFD2UjYymWqQUCpFqbSWQiSYnuaw6DZoAde8KsTCMxPFVHA",
  "senderPublicKey": "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "cryptoAlgo": "aes"
}
```

В ответ на запрос поступает поле decryptedText, содержащее расшифрованную строку.

Пример ответа:

POST /crypto/decrypt:

```
{
  "decryptedText": "some string for encryption",
}
```


Ответ:

```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrsgLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [
    ↪ "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc
    ↪ "
  ],
  "fee": 110000000,
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
}
```

POST /privacy/sendDataV2

Метод **POST /privacy/sendDataV2** аналогичен методу **POST /privacy/sendData**, однако позволяет приложить файл в окне Swagger, не прибегая к его конверсии в формат **base64**. Поле Data в этой версии метода отсутствует.

Примеры запроса и ответа:

POST /privacy/sendDataV2:

Запрос:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "info": {
    "filename": "Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "Alvanov@org.com",
    "comment": "some comments"
  },
  "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
}
```

Ответ:

```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrsgLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [
```

(continues on next page)

(продолжение с предыдущей страницы)

```

→ "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc
→ "
],
"fee": 110000000,
"id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
"type": 114,
"timestamp": 1571043910570
}

```

GET /privacy/{policy-id}/recipients

Метод предназначен для получения адресов всех участников, записанных в группу {policy-id}.

В ответе метода возвращается массив строк с адресами участников группы доступа.

Пример ответа:

GET /privacy/{policy-id}/recipients:

```

[
  "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]

```

GET /privacy/{policy-id}/owners

Метод предназначен для получения адресов владельцев группы доступа {policy-id}.

В ответе метода возвращается массив строк с адресами владельцев группы доступа.

Пример ответа:

GET /privacy/{policy-id}/owners:

```

[
  "3GCFaCWtvLDnC9yX29YftMbn75gwf dwGsBn",
  "3GGxcmNyq8ZAHZK7or14Ma84khW8peBohJ",
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZYnh84pn",
  "3GKpShRQRTddF1yYhQ58ZnKMTnp2xdEzKqW"
]

```


GET /privacy/{policy-id}/hashes

Метод предназначен для получения массива идентификационных хэшей данных, которые привязаны к группе доступа {policy-id}.

В ответе метода возвращается массив строк с идентификационными хэшами данных группы доступа.

Пример ответа:

GET /privacy/{policy-id}/hashes:

```
[
  "FdfdNBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
  "eedfdNBVqYXrapgJP9atQccdBPagJPwHDKkh6A"
]
```

GET /privacy/{policyId}/getData/{policyItemHash}

Метод предназначен для получения пакета конфиденциальных данных группы доступа {policyId} по идентификационному хэшу {policyItemHash}.

В ответе метода возвращается хэш-сумма конфиденциальных данных.

Пример ответа:

GET /privacy/{policyId}/getData/{policyItemHash}:

```
c29tZV9iYXNlNjRfZW5jb2RlZF9zdHJpbmc=
```

GET /privacy/{policyId}/getInfo/{policyItemHash}

Метод предназначен для получения метаданных для пакета конфиденциальных данных группы {policyId} по идентификационному хэшу {policyItemHash}.

В ответе метода возвращаются следующие данные:

- sender - адрес отправителя конфиденциальных данных;
- policy_id - идентификатор группы доступа;
- type - тип конфиденциальных данных (file);
- info - массив данных о файле: filename - имя файла, size - размер файла, timestamp - временная метка размещения файла в формате *Unix Timestamp* (в миллисекундах), author - автор файла, comment - опциональный комментарий к файлу, hash - идентификационный хэш конфиденциальных данных.

Пример ответа:

GET /privacy/{policyId}/getInfo/{policyItemHash}:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "policy": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "info": {
    "filename": "Contract №100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "Comment"
  },
  "hash": "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1"
}
```

POST /privacy/forceSync

Метод предназначен для принудительного получения пакета конфиденциальных данных. Применяется в случае, если транзакция с конфиденциальными данными для группы доступа присутствует в блокчейне, но по какой-либо причине эти данные не были записаны в хранилище конфиденциальных данных ноды. В этом случае метод позволяет принудительно скачать отсутствующие данные.

Запрос метода содержит следующие данные:

- **sender** - адрес ноды-участника группы доступа, отправляющей запрос;
- **policy** - идентификатор группы доступа;
- **source** - адрес ноды, с которой должны скачиваться отсутствующие данные. В случае, если нода неизвестна, установите параметр на `null` или оставьте поле пустым: в этом случае скачивание файла будет произведено из хранилища первой ноды из списка группы доступа.

Ответ метода содержит поле **result** с результатом получения данных и поле **message** с текстом возможной ошибки. В случае успешного получения, возвращается **success**, конфиденциальные данные записываются в хранилище ноды.

В случае возникновения ошибки, возвращается **error**, в поле **message** приводится описание ошибки.

Примеры запроса и ответа:**POST /privacy/forceSync:**

Запрос:

```
{
  "sender": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8",
  "policy": "my_policy",
  "source": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
}
```

Ответ:

```
{
  "result": "error"
  "message": "Address '3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8' not in
  ↪policy 'my_policy'"
}
```

POST /privacy/getInfos

Метод предназначен для получения массива метаданных конфиденциальных данных по идентификатору группы доступа и идентификационному хэшу.

Запрос метода содержит следующие данные:

- policiesDataHashes - массив данных с двумя элементами для каждой отдельной группы доступа: policyId - идентификатор группы доступа, datahashes - массив хэшей конфиденциальных данных для получения метаданных по каждому из них.

В ответе метода для каждого отдельного хэша конфиденциальных данных возвращается массив данных, аналогичный ответу метода GET /privacy/{policyId}/getInfo/{policyItemHash}.

Примеры запроса и ответа:

POST /privacy/getInfos:

Запрос:

```
{ "policiesDataHashes":
  [
    {
      "policyId": "somepolicyId_1",
      "datahashes": [ "datahash_1", "datahash_2" ]
    },
    {
      "policyId": "somepolicyId_2",
      "datahashes": [ "datahash_3", "datahash_4" ]
    }
  ]
}
```

Ответ:

```
{
  "policiesDataInfo": [
    {
      "policyId": "somepolicyId_1",
      "datasInfo": [
        {
          "hash":
          ↪"e67ad392ab4d933f39d5723aead96c18c491140e119d590103e7fd6de15623f1
          ↪",
          "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
          "type": "file",
          "info": {
```

(continues on next page)

(продолжение с предыдущей страницы)

```

        "filename": "Contract №100/5.doc",
        "size": 2048,
        "timestamp": 1000000000,
        "author": "Alvanov@org.com",
        "comment": "Comment"
      },
    ],
    {
      "hash":
↪ "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1
↪ ",
      "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
      "type": "file",
      "info": {
        "filename": "Contract №101/5.doc",
        "size": 2048,
        "timestamp": 1000000000,
        "author": "Alvanov@org.com",
        "comment": "Comment"
      }
    }
  ]
}

```

Смотрите также*Методы REST API**Обмен конфиденциальными данными***10.2.5 REST API: валидация адресов и псевдонимов участников сети**

Для валидации адресов и псевдонимов в сети предусмотрены следующие методы группы addresses:

GET /addresses/validate/{addressOrAlias}

Валидация заданного адресата или его псевдонима {addressOrAlias} в блокчейн-сети работающей ноды.

Пример ответа:

GET /addresses/validate/{addressOrAlias}:

```
{
  addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
  valid: true
}
```

POST /addresses/validateMany

Валидация нескольких адресов или псевдонимов, передаваемых в поле `addressesOrAliases` в виде массива. Информация в ответе для каждого адреса идентична ответу метода GET `/addresses/validate/{addressOrAlias}`.

Примеры запроса и ответа для одного адреса, одного существующего и одного несуществующего псевдонимов:

POST /addresses/validateMany:

Запрос:

```
{
  addressesOrAliases: [
    "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
    "alias:T:asdfghjk",
    "alias:T:invAliDA11ass99911%~&$$$ "
  ]
}
```

Ответ:

```
{
  validations: [
    {
      addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
      valid: true
    },
    {
      addressOrAlias: "alias:T:asdfghjk",
      valid: true
    },
    {
      addressOrAlias: "alias:T:invAliDA11ass99911%~&$$$ ",
      valid: false,
      reason: "GenericError(Alias should contain only following characters: -.
→0123456789@_abcdefghijklmnopqrstuvwxyz)"
    }
  ]
}
```

Смотрите также

Методы REST API

10.2.6 REST API: подписание и валидация сообщений в блокчейне

Для подписания и валидации сообщений предусмотрены следующие методы группы addresses:

POST /addresses/sign/{address}

Метод подписывает строку, переданную в поле message, приватным ключом адресата {address}, а затем сериализует ее в формат **base58**. В ответе метода возвращается сериализованная строка, публичный ключ и подпись адресата.

Примеры запроса и ответа:

POST /addresses/sign/{address}:

Запрос:

```
{
  "message": "mytext"
}
```

Ответ:

```
{
  "message": "wWshKhJj",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "62PFG855ThsEHUZ4N8VE8kMyHCK9GwnvtTZ3hq6JHYv12BhP1eRjegA6nSa3DAoTTMammhamadvizDUYZAZtKY9S
  ↪ "
}
```

POST /addresses/verify/{address}

Проверка подписи сообщения, выполненной адресатом {address}.

Примеры запроса и ответа:

POST /addresses/verify/{address}:

Запрос:

```
{
  "message": "wWshKhJj",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "5kwwE9sDZzssNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts
  ↪ "
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↔ "
}
```

Ответ:

```
{
  "valid": true
}
```

POST /addresses/signText/{address}

Метод подписывает строку, переданную в поле `message`, приватным ключом адресата `{address}`. В отличие от метода `POST /addresses/sign/{address}`, строка передается в исходном формате.

Примеры запроса и ответа:**POST /addresses/signText/{address}:**

Запрос:

```
{
  "message": "mytext"
}
```

Ответ:

```
{
  "message": "mytext",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
↔ "5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7Zjnrzs9QYRWo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u
↔ "
}
```

POST /addresses/verifyText/{address}

Проверка подписи сообщения, выполненной адресатом `{address}` посредством метода `POST /addresses/signText/{address}`.

Примеры запроса и ответа:

POST /addresses/verifyText/{address}:

Запрос:

```
{
  "message": "mytext",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7Zjnrzs9QYRwo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u
  ↪ "
}
```

Ответ:

```
{
  "valid": true
}
```

Смотрите также*Методы REST API***10.2.7 REST API: информация о конфигурации и состоянии ноды, остановка ноды**

Для получения информации о конфигурации ноды предусмотрены две группы методов:

- **node** - получение основных конфигурационных параметров ноды, информации о состоянии ноды, остановка ноды, изменение уровня логирования;
- **anchoring** - запрос GET /anchoring/config, возвращающий секцию **anchoring** конфигурационного файла ноды.

Для получения основных конфигурационных параметров ноды предусмотрены как методы, требующие авторизации, так и открытые методы.

Группа node:**GET /node/config**

Метод возвращает основные конфигурационные параметры ноды.

Пример ответа:

GET /node/config:

```
{
  {
    "version": "1.3.0-RC7",
    "gostCrypto": false,
    "chainId": "V",
    "consensus": "POA",
    "minimumFee": {
      "3": 0,
      "4": 0,
      "5": 0,
      "6": 0,
      "7": 0,
      "8": 0,
      "9": 0,
      "10": 0,
      "11": 0,
      "12": 0,
      "13": 0,
      "14": 0,
      "15": 0,
      "102": 0,
      "103": 0,
      "104": 0,
      "106": 0,
      "107": 0,
      "111": 0,
      "112": 0,
      "113": 0,
      "114": 0
    },
    "additionalFee": {
      "11": 0,
      "12": 0
    },
    "maxTransactionsInMicroBlock": 500,
    "minMicroBlockAge": 0,
    "microBlockInterval": 1000,
    "blockTiming": {
      "roundDuration": 7000,
      "syncDuration": 700
    }
  }
}
```

GET /node/owner

Метод возвращает адрес и публичный ключ владельца ноды.

Пример ответа:

GET /node/config:

```
{
  "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
  "publicKey": "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt"
}
```

GET /node/status

Метод возвращает информацию о текущем состоянии ноды.

Пример ответа:

GET /node/status:

```
{
  "blockchainHeight": 47041,
  "stateHeight": 47041,
  "updatedTimestamp": 1544709501138,
  "updatedAt": "2018-12-13T13:58:21.138Z"
}
```

Также, при возникновении ошибок с использованием ГОСТ-криптографии на ноды, метод вернет описание ошибки:

GET /node/status:

```
{
  "error": 199,
  "message": "Environment check failed: Supported JCSP version is 5.0.40424, actual is ↵
↵2.0.40424"
}
```

GET /node/version

Метод возвращает версию ноды.

Пример ответа:

GET /node/version:

```
{  
  "version": "Waves Enterprise v0.9.0"  
}
```

GET /node/logging

Метод отображает список логгеров, указанных при конфигурировании ноды, и уровень логирования для каждого из них.

Уровни логирования ноды:

- ERROR - логирование ошибок;
- WARN - логирование предупреждений;
- INFO - логирование событий ноды;
- DEBUG - расширенная информация о событиях по каждому работающему модулю ноды: запись произошедших событий и выполняемых действий;
- TRACE - подробная информация о событиях уровня DEBUG;
- ALL - отображение информации на всех уровнях логирования.

Пример ответа:

GET /node/logging:

```
ROOT-DEBUG  
akka-DEBUG  
akka.actor-DEBUG  
akka.actor.ActorSystemImpl-DEBUG  
akka.event-DEBUG  
akka.event.slf4j-DEBUG  
akka.event.slf4j.Slf4jLogger-DEBUG  
com-DEBUG  
com.github-DEBUG  
com.github.dockerjava-DEBUG  
com.github.dockerjava.core-DEBUG  
com.github.dockerjava.core.command-DEBUG  
com.github.dockerjava.core.command.AbstrDockerCmd-DEBUG  
com.github.dockerjava.core.exec-DEBUG
```

POST /node/logging

Метод предназначен для смены уровня логирования для выбранных логгеров.

Пример запроса:

POST /node/logging:

```
{
  "logger": "com.wavesplatform.Application",
  "level": "ALL"
}
```

POST /node/stop

Метод останавливает ноду, ответа не предусмотрено.

Метод GET /anchoring/config:

Метод выводит секцию anchoring конфигурационного файла ноды.

Пример ответа:

GET /anchoring/config:

```
{
  "enabled": true,
  "currentChainOwnerAddress": "3FWwx4o1177A4oeHAEW5EQ6Bkn4Lv48quYz",
  "mainnetNodeAddress": "https://clinton-pool.wavesenterpriseservices.com:443",
  "mainnetSchemeByte": "L",
  "mainnetRecipientAddress": "3JzVWCSV6v4ucSxtGSjZsvdiCT1FAzwpqrP",
  "mainnetFee": 8000000,
  "currentChainFee": 666666,
  "heightRange": 5,
  "heightAbove": 3,
  "threshold": 10
}
```

Смотрите также

Методы REST API

Примеры конфигурационных файлов ноды

10.2.8 REST API: информация об участниках сети

Для получения информации об участниках сети предусмотрено три группы методов:

- `addresses` - методы, предназначенные для получения информации об адресах участников сети;
- `alias` - получение адреса участника по установленному для него псевдониму или псевдонима по адресу участника;
- `leasing` - запрос `GET /leasing/active/{address}`, выводящий список транзакций лизинга, в которых адрес принимал участие как отправитель или получатель.

Группа `addresses`:

GET /addresses

Получение всех адресов участников, ключевые пары которых хранятся в `keystore` ноды.

Пример ответа:

GET /addresses:

```
[  
  "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",  
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"  
]
```

GET /addresses/seq/{from}/{to}

Получение адресов участников, которые хранятся в `keystore` ноды в заданном диапазоне: от адреса `{from}` до адреса `{to}`.

Формат ответа метода идентичен формату `GET /addresses`.

GET /addresses/balance/{address}

Получение баланса для адреса `{address}`.

Пример ответа:

GET /addresses/balance/{address}:

```
{
  "address": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "confirmations": 0,
  "balance": 100945889661986
}
```

POST /addresses/balance/details

Получение подробной информации о балансе для списка адресов, который указывается в виде массива в поле `addresses` при запросе.

Параметры, возвращаемые в ответе метода:

- `regular` — сумма токенов, принадлежащих непосредственно участнику (**R**);
- `available` — общий баланс участника, за исключением средств, переданных участником в лизинг (**A = R - L**);
- `effective` — общий баланс участника, включая средства, переданные участнику в лизинг, и за вычетом средств, которые участник сам передал в лизинг (**E = R + F - L**);
- `generating` — генерирующий баланс участника, включая средства переданные в лизинг, за последние 1000 блоков.

Переменные в скобках: **L** - средства, переданные участником в лизинг другим участникам, **F** - средства, полученные участником в лизинг.

Пример ответа для одного адреса:

POST /addresses/balance/details:

```
[
  {
    "address": "3M4Bxh2VfzKFXqiQB8bDgRfVnPWzZUQ2MEF",
    "regular": 59899999999400000,
    "generating": 59899999999400000,
    "available": 59899999999400000,
    "effective": 59899999999400000
  }
]
```

GET /addresses/balance/details/{address}

Получение подробной информации о балансе для отдельного адреса. Информация в ответе идентична методу `POST /addresses/balance/details`.

Пример ответа:

GET /addresses/balance/details/{address}:

```
[
  {
    "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "regular": 0,
    "generating": 0,
    "available": 0,
    "effective": 0
  }
]
```

GET /addresses/effectiveBalance/{address}

Получение общего баланса адреса, включая средства, переданные в лизинг.

Пример ответа:**GET /addresses/effectiveBalance/{address}:**

```
{
  "address": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "confirmations": 0,
  "balance": 1240001592820000
}
```

GET /addresses/effectiveBalance/{address}/{confirmations}

Получение баланса для адреса {address} после количества подтверждений \geq {confirmations}. Возвращается общий баланс участника, включая средства, переданные участнику в лизинг.

Пример ответа для количества подтверждений ≥ 1 :**GET /addresses/effectiveBalance/{address}/{confirmations}:**

```
{
  "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "confirmations": 1,
  "balance": 0
}
```

GET /addresses/generatingBalance/{address}/at/{height}

Получение генерирующего баланса адреса на указанной высоте блокчейна {height}.

Пример ответа:

GET /addresses/generatingBalance/{address}/at/{height}:

```
{
  "address": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "generatingBalance": 1011543800600
}
```

GET /addresses/scriptInfo/{address}

Получение данных о скрипте, установленном на адресе.

Параметры, возвращаемые в ответе метода:

- address - адрес в формате **base58**;
- script - тело скрипта в формате **base64**;
- scriptText - исходный код скрипта;
- complexity - сложность скрипта;
- extraFee - комиссия за исходящие транзакции, установленные скриптом.

Сложность скрипта - число от 1 до 100, отражающее количество вычислительных ресурсов, требуемое для исполнения скрипта.

Пример ответа:

GET /addresses/scriptInfo/{address}:

```
{
  "address": "3N3keodUiS8WEw9W4BKDNxgNdUpwSnpb3K",
  "script":
  ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKfGzTtXXnnv7upRHXJzZrLSQo8",
  ↪ ",
  "scriptText": "ScriptV1(BLOCK(LET(x,CONST_LONG(1)),FUNCTION_CALL(FunctionHeader(==,
  ↪ List(LONG, LONG)),List(FUNCTION_CALL(FunctionHeader(+,List(LONG, LONG)),List(REF(x,
  ↪ LONG), CONST_LONG(1)),LONG), CONST_LONG(2)),BOOLEAN),BOOLEAN))",
  "complexity": 11,
  "extraFee": 10001
}
```


GET /addresses/publicKey/{publicKey}

Метод возвращает адрес участника на основании его публичного ключа.

Пример ответа:

GET /addresses/publicKey/{publicKey}:

```
{
  "address": "3N4WaaaNAVLMQgVKTRSePgWbUAKvZTjAQbq"
}
```

GET /addresses/data/{address}

Метод возвращает данные, записанные на указанном адресе при помощи *транзакций 12*.

Пример ответа:

GET /addresses/data/{address}:

```
[
  {
    "key": "4yR7b6Gv2rzLrhYBHpgVCmLH42raPGTF4Ggi1N36aWnY",
    "type": "integer",
    "value": 1500000
  }
]
```

GET /addresses/data/{address}/{key}

Метод возвращает данные, записанные на указанном адресе с ключом {key}. Этот ключ указывается в *транзакции 12* в поле data.key.

Пример ответа:

GET /addresses/data/{address}/{key}:

```
{
  "key": "4yR7b6Gv2rzLrhYBHpgVCmLH42raPGTF4Ggi1N36aWnY",
  "type": "integer",
  "value": 1500000
}
```

Группа alias:

GET /alias/by-alias/{alias}

Получение адреса участника по его псевдониму {alias}.

Пример ответа:

GET /alias/by-alias/{alias}:

```
{
  "address": "address:3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
}
```

GET /alias/by-address/{address}

Получение псевдонима участника по его адресу {address}.

Пример ответа:

GET /alias/by-alias/{alias}:

```
[
  "alias:participant1",
]
```

Метод GET /leasing/active/{address}:

Метод возвращает список транзакций создания лизинга, в которых адрес принимал участие как отправитель или получатель.

Пример ответа с одной транзакцией:

GET /alias/by-alias/{alias}:

```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEwoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVwRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
    ↪ "25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPfyegP82sNSfM2bXNX3K7p6D3HD
    ↪",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
  }
]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"height": 1298747
}
```

Смотрите также*Методы REST API*

10.2.9 REST API: информация об используемом алгоритме консенсуса

Для получения информации, относящейся к используемому алгоритму консенсуса, предусмотрены методы группы consensus.

GET /consensus/algo

Метод возвращает название используемого алгоритма консенсуса.

Пример ответа:**GET /consensus/algo:**

```
{
  "consensusAlgo": "Leased Proof-of-Stake (LPoS)"
}
```

GET /consensus/settings

Метод возвращает параметры используемого алгоритма консенсуса, заданные в конфигурационном файле ноды.

Пример ответа:**GET /consensus/settings:**

```
{
  "consensusAlgo": "Proof-of-Authority (PoA)",
  "roundDuration": "25 seconds",
  "syncDuration": "5 seconds",
  "banDurationBlocks": 50,
  "warningsForBan": 3
}
```

GET /consensus/minersAtHeight/{height}

Метод возвращает очередь майнеров на высоте {height}. Доступен при использовании алгоритма *консенсуса PoA*.

Пример ответа:

GET /consensus/minersAtHeight/{height}:

```
{
  "miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrm92sUKcXkd",
    "3N2cQFfUDzG2iujBrFTnD2TAsCnOhDxYu8w",
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
  "height": 1
}
```

GET /consensus/miners/{timestamp}

Метод возвращает очередь майнеров на время {timestamp} (указывается в формате **Unix Timestamp**, в миллисекундах). Доступен при использовании *алгоритма консенсуса PoA*.

Пример ответа:

GET /consensus/miners/{timestamp}:

```
{
  "miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrm92sUKcXkd",
    "3N2cQFfUDzG2iujBrFTnD2TAsCnOhDxYu8w",
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
  "timestamp": 1547804621000
}
```

GET /consensus/bannedMiners/{height}

Метод возвращает список заблокированных майнеров на высоте {height}. Доступен при использовании *алгоритма консенсуса PoA*.

Пример ответа:

GET /consensus/bannedMiners/{height}:

```
{
  "miners": [
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
  "height": 440
}
```

GET /consensus/basetarget/{signature}

Метод возвращает значение базовой сложности (basetarget) создания блока по его подписи {signature}. Доступен при использовании *алгоритма консенсуса PoS*.

GET /consensus/basetarget

Метод возвращает значение базовой сложности (basetarget) создания текущего блока. Доступен при использовании *алгоритма консенсуса PoS*.

GET /consensus/generatingbalance/{address}

Метод возвращает генерирующий баланс, доступный для ноды {address}, включая средства переданные участнику в лизинг. Доступен при использовании *алгоритма консенсуса PoS*.

GET /consensus/generationsignature/{signature}

Метод возвращает значение генерирующей подписи (generation signature) создания блока по его подписи {signature}. Доступен при использовании *алгоритма консенсуса PoS*.

GET /consensus/generationsignature

Возвращает значение генерирующей подписи (generation signature) текущего блока. Доступен при использовании *алгоритма консенсуса PoS*.

Смотрите также

Методы REST API

Алгоритмы консенсуса

10.2.10 REST API: информация о смарт-контрактах

Для получения информации о смарт-контрактах, загруженных в сеть, предусмотрен набор методов группы `contracts`.

GET /contracts

Метод возвращает информацию по всем смарт-контрактам, загруженным в сеть. Для каждого смарт-контракта в ответе возвращаются следующие параметры:

- `contractId` - идентификатор смарт-контракта;
- `image` - имя Docker-образа смарт-контракта, либо его абсолютный путь в репозитории;
- `imageHash` - хэш-сумма смарт-контракта;
- `version` - версия смарт-контракта;
- `active` - статус смарт-контракта на момент отправки запроса: `true` - запущен, `false` - не запущен.

Пример ответа для одного смарт-контракта:

GET /contracts:

```
[
  {
    "contractId": "dmLT1ippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "image": "registry.wvservices.com/wv-sc/may14_1:latest",
    "imageHash": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957",
    "version": 1,
    "active": false
  }
]
```

POST /contracts

Метод возвращает набор полей «ключ:значение», записанных в стейт одного или нескольких смарт-контрактов. ID запрашиваемых смарт-контрактов указываются в поле `contracts` запроса.

Пример ответа для одного смарт-контракта:

POST /contracts:

```
{
  "8vBJhy4eS8oEwCHC3yS3M6nZd5CLBa6XNt4Nk3yEEExG": [
    {
      "type": "string",
      "value": "Only description",
      "key": "Description"
    },
    {
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "type": "integer",
    "value": -9223372036854776000,
    "key": "key_may"
  }
]
}

```

GET /contracts/status/{id}

Метод возвращает статус исполняемой *транзакции 103* на создание смарт-контракта по идентификатору транзакции {id}. Однако если после отправки транзакции в блокчейн нода перезапускается, метод не вернет корректное состояние этой транзакции.

В ответе метода возвращаются следующие параметры:

- sender - адрес отправителя транзакции;
- senderPublicKey - публичный ключ отправителя транзакции;
- txId - ID транзакции;
- status - статус транзакции: успешно попала в блок, подтверждена, отклонена;
- code - код ошибки (при наличии);
- message - сообщение о статусе транзакции;
- timestamp - временная метка в формате **Unix Timestamp**, в миллисекундах;
- signature - подпись транзакции.

Пример ответа:

GET /contracts/status/{id}:

```

{
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNEn2yHRKWU",
  "status": "Success",
  "code": null,
  "message": "Smart contract transaction successfully mined",
  "timestamp": 1558961372834,
  "signature":
  ↪ "4gXy7qtzkaHHH6NkksnZ5pvn8juF65MvjQ9JgVztpgNwLNwuyyr27Db3gCh5YyADqZeBH72EyAkBouUoKvwJ3RQJ
  ↪ "
}

```

GET /contracts/{contractId}

Метод возвращает результат исполнения смарт-контракта по его идентификатору {contractId}.

Пример ответа:

GET /contracts/{contractId}:

```
[
  {
    "key": "avg",
    "type": "string",
    "value": "3897.80146957"
  },
  {
    "key": "buy_price",
    "type": "string",
    "value": "3842"
  }
]
```

POST /contracts/{contractId}

Метод возвращает значения ключей из стека смарт-контракта {contractId}. В запросе указываются следующие данные:

- contractId - идентификатор смарт-контракта;
- limit - ограничение количества выводимых блоков данных;
- offset - количество блоков данных для пропуска в выводе;
- matches - опциональный параметр для составления регулярного выражения, по которому фильтруются ключи.

Пример ответа:

POST /contracts/{contractId}:

```
[
  {
    "type": "string",
    "key": "avg",
    "value": "3897.80146957"
  },
  {
    "type": "string",
    "key": "buy_price",
    "value": "3842"
  }
]
```


GET /contracts/executed-tx-for/{id}

Метод возвращает результат исполнения смарт-контракта по идентификатору *транзакции 105*.

В ответе метода возвращаются данные транзакции 105, а также результаты исполнения в поле `results`.

Пример ответа, смарт-контракт не исполнялся:

GET /contracts/executed-tx-for/{id}:

```
{
  "type": 105,
  "id": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiaRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfoTEuPNV9GrdaunpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
    ↪ "4BoG6wQnYyZWYUKzAwh5n1184tsEWUqUTWmXMEvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLawTSXquAs",
    ↪ ""
  ],
  "version": 1,
  "tx": {
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLZZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
      ↪ "yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxm4BYA4TaqYVw5qxtWzGMPQyVeKYv",
      ↪ ""
    ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
  },
  "results": []
}
```

GET /contracts/{contractId}/{key}

Возвращает значение ключа `{key}` исполненного смарт-контракта по его идентификатору.

Пример ответа:

GET /contracts/{contractId}/{key}:

```
{
  "key": "updated",
  "type": "integer",
  "value": 1545835909
}
```

Смотрите также

Методы REST API

Смарт-контракты

Разработка и применение смарт-контрактов

10.2.11 REST API: информация о блоках сети

Для получения информации о различных блоках сети предусмотрена группа методов blocks.

GET /blocks/height

Метод возвращает номер текущего блока в блокчейне (высоту блокчейна).

Пример ответа:

GET /blocks/height:

```
{
  "height": 7788
}
```

GET /blocks/height/{signature}

Метод возвращает высоту блока по его подписи {signature}.

Ответ метода содержит поле height, как и метод GET /blocks/height.

GET /blocks/first

Метод возвращает информацию о генезис-блоке сети.

В ответе содержатся следующие параметры:

- reference - хэш-сумма генезис-блока;
- blocksize - размер генезис-блока;
- signature - подпись генезис-блока;
- fee - комиссия за транзакции, включенные в генезис-блок;

- `generator` - адрес создателя генезис-блока;
- `transactionCount` - количество транзакций *1* и *101*, включенных в генезис-блок;
- `transactions` - массив с телами транзакций 1 и 101, включенных в генезис-блок;
- `version` - версия генезис-блока;
- `timestamp` - временная метка создания генезис-блока в формате **Unix Timestamp** (в миллисекундах);
- `height` - высота создания генезис-блока (**1**).

Пример ответа:

GET `/blocks/first`:

```
{
  "reference":
  ↪ "67rpwLCuS5DGA8KGZXKsVQ7dnPb9goRLoKfgGbLfQg9WoLUGNY77E2jT11fem3coV9nAkguBACzrU1iyZM4B8roQ",
  ↪ "blocksizes": 1435,
  "signature":
  ↪ "4HENriUyMthzMSqWa5sYPFMATbZpQugTBMk6mXUh5HmnvHfUhmQk6EqmdhGvNFCUvTDrsyiVqkxtm8iiV2xNTSNK",
  ↪ "fee": 0,
  "generator": "3MvQKx98a713B28rdUAtbWJ8DFJEXhnTjKs",
  "transactionCount": 26,
  "transactions": [
    {
      "type": 1,
      "id":
      ↪ "2AdCY254MFSrgxpr6otBisV5Zz7neH8YoM6VGW5egoVJnWD8cJpYZVR42aVKTZnwGT9ee7LCpAGMNSUV86FEAGXu",
      ↪ "fee": 0,
      "timestamp": 1606211535610,
      "signature":
      ↪ "2AdCY254MFSrgxpr6otBisV5Zz7neH8YoM6VGW5egoVJnWD8cJpYZVR42aVKTZnwGT9ee7LCpAGMNSUV86FEAGXu",
      ↪ "recipient": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "amount": 1250000000000000
    },
    {
      "type": 1,
      "id":
      ↪ "5VC2LoFTbrfLkd48bjQkp8CmTyqXJSkjh723qxo9v5pz38tBUjRW9tHLuvwajSvkzQNFxrcC6Yjkgx5R2YR3x5VC",
      ↪ "fee": 0,
      "timestamp": 1606211535610,
      "signature":
      ↪ "5VC2LoFTbrfLkd48bjQkp8CmTyqXJSkjh723qxo9v5pz38tBUjRW9tHLuvwajSvkzQNFxrcC6Yjkgx5R2YR3x5VC",
      ↪ "recipient": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
      "amount": 3000000000000000
    }
  ]
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

      "type": 1,
      "id":
    ↪ "4cmwEkSnBLC3TBTPUiT7HwmdER25X7GzCj2mgiEJ8K149vnNa1orBZUNstwNXtXFyKcQbkRPym39d9wJXTE4wgbU
    ↪ ",
      "fee": 0,
      "timestamp": 1606211535610,
      "signature":
    ↪ "4cmwEkSnBLC3TBTPUiT7HwmdER25X7GzCj2mgiEJ8K149vnNa1orBZUNstwNXtXFyKcQbkRPym39d9wJXTE4wgbU
    ↪ ",
      "recipient": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
      "amount": 100000000000000
    },
    {
      "type": 1,
      "id":
    ↪ "5Etq3o1eWoN3bqR9cYV6149qxAE3ru4CoSCf1Mm5sSJEedcbmLhsbfg8rh4S6ESrAPq7ZEbghEgHjyb3xzUbDDRh
    ↪ ",
      "fee": 0,
      "timestamp": 1606211535610,
      "signature":
    ↪ "5Etq3o1eWoN3bqR9cYV6149qxAE3ru4CoSCf1Mm5sSJEedcbmLhsbfg8rh4S6ESrAPq7ZEbghEgHjyb3xzUbDDRh
    ↪ ",
      "recipient": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
      "amount": 100000000000000
    },
    {
      "type": 110,
      "id":
    ↪ "3HewQJtzuaumzX4TvmN7fxVCgnsWTTaLeQjYBVDDuYoEW2ijWd7JME8h1gtsqepv5SDhHPvoMesVNm96br8WRgF8
    ↪ ",
      "fee": 0,
      "timestamp": 1606211535610,
      "signature":
    ↪ "3HewQJtzuaumzX4TvmN7fxVCgnsWTTaLeQjYBVDDuYoEW2ijWd7JME8h1gtsqepv5SDhHPvoMesVNm96br8WRgF8
    ↪ ",
      "targetPublicKey":
    ↪ "56rV5kcR9SBsxQ9LtNrmp6V72S4BDkZUJaA6ujZswDneDmCTmeSG6UE2FQP1rPXdfpWQNunRw4aijGXxoK3o4puj
    ↪ ",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP"
    },
    {
      "type": 101,
      "id":
    ↪ "5r4uLWn3rwmqbBygNj29iR4YsiV82dYWFeCbepAHhKGXqnn27vE6i811U9H2UZgX8zNQYZciyw3PR6nAdwjSPSp5
    ↪ ",
      "fee": 0,
      "timestamp": 1606211535609,
      "signature":
    ↪ "5r4uLWn3rwmqbBygNj29iR4YsiV82dYWFeCbepAHhKGXqnn27vE6i811U9H2UZgX8zNQYZciyw3PR6nAdwjSPSp5
    ↪ ",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "permissioner"
    }
  ]

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    },
    {
      "type": 101,
      "id":
      ↪ "4pBwjviNLtSPEBY5YB7ZdUXVSFnEk4rgscW8r9QQKxdxQZzjwdq1ZnruMxQo7tomQVJf1Ni6SyVxShrQZhbJaFN
      ↪ ",
      "fee": 0,
      "timestamp": 1606211535608,
      "signature":
      ↪ "4pBwjviNLtSPEBY5YB7ZdUXVSFnEk4rgscW8r9QQKxdxQZzjwdq1ZnruMxQo7tomQVJf1Ni6SyVxShrQZhbJaFN
      ↪ ",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "miner"
    },
    {
      "type": 101,
      "id":
      ↪ "5kwQwLH8oTy1ztF6xxsBxE3MDGio1NJm8F7Mtpynf3QTW9CWCsp5Fio5SxLmPxnB1bUVQHMCHbQCD4wXJLJgjSrp
      ↪ ",
      "fee": 0,
      "timestamp": 1606211535607,
      "signature":
      ↪ "5kwQwLH8oTy1ztF6xxsBxE3MDGio1NJm8F7Mtpynf3QTW9CWCsp5Fio5SxLmPxnB1bUVQHMCHbQCD4wXJLJgjSrp
      ↪ ",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "connection_manager"
    },
    {
      "type": 101,
      "id":
      ↪ "62xS2qkR7chFMSdryTjwB15BKd4CH5Hwn9PbzasZo1Qx6Bwg82nixMPKRQobDy3JW7cLmzMHi97hJk1JSDqhwUgM
      ↪ ",
      "fee": 0,
      "timestamp": 1606211535606,
      "signature":
      ↪ "62xS2qkR7chFMSdryTjwB15BKd4CH5Hwn9PbzasZo1Qx6Bwg82nixMPKRQobDy3JW7cLmzMHi97hJk1JSDqhwUgM
      ↪ ",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "contract_developer"
    },
    {
      "type": 101,
      "id":
      ↪ "2sNwzGbwDL2Es53P8XY5wA9T9wwu3eXJbJUrtXJ9wg49urPjuBejWbidat2z3yZ8JrTpKWWFEsrerCtnC38XuRTJ
      ↪ ",
      "fee": 0,
      "timestamp": 1606211535605,
      "signature":
      ↪ "2sNwzGbwDL2Es53P8XY5wA9T9wwu3eXJbJUrtXJ9wg49urPjuBejWbidat2z3yZ8JrTpKWWFEsrerCtnC38XuRTJ
      ↪ ",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "issuer"
    }
  ]

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    },
    {
      "type": 110,
      "id":
↪ "4hLep3GngPEBH2xEmuUZ323muT8BstFdT552e42z6ZXCKGnF1PABGGjEiCkHfr6hMuyvRJ7axD9qoGeWQCU5yaCk
↪ ",
      "fee": 0,
      "timestamp": 1606211535610,
      "signature":
↪ "4hLep3GngPEBH2xEmuUZ323muT8BstFdT552e42z6ZXCKGnF1PABGGjEiCkHfr6hMuyvRJ7axD9qoGeWQCU5yaCk
↪ ",
      "targetPublicKey":
↪ "5nGi8XoiGjjybPmjLNy1k2bus4yXLaeuA3Hb7BikwD9tboFwFXJYUmt05Joox76c3pp2Mr1LjgodUJuxryCJofQ
↪ ",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c"
    },
    {
      "type": 101,
      "id":
↪ "nj9Xfqm3pPLmuLsWfDZx4htKaNKAYvhen7tF95T9YwdmK1pqkiCjtaV9AxCwzEceViyo5rHPapigxPyCZdBWvRn
↪ ",
      "fee": 0,
      "timestamp": 1606211535604,
      "signature":
↪ "nj9Xfqm3pPLmuLsWfDZx4htKaNKAYvhen7tF95T9YwdmK1pqkiCjtaV9AxCwzEceViyo5rHPapigxPyCZdBWvRn
↪ ",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
      "role": "permissioner"
    },
    {
      "type": 101,
      "id":
↪ "24AmxdGyH3afYRxPXn5zqvU1Fro1MwVQPDqwkjdCKLddSEiKVhyeMHTAVrRpHu83ZDPMYqkf3ty161PrujmGYtef
↪ ",
      "fee": 0,
      "timestamp": 1606211535603,
      "signature":
↪ "24AmxdGyH3afYRxPXn5zqvU1Fro1MwVQPDqwkjdCKLddSEiKVhyeMHTAVrRpHu83ZDPMYqkf3ty161PrujmGYtef
↪ ",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
      "role": "miner"
    },
    {
      "type": 101,
      "id":
↪ "4xsEQoh6Z4wDW6jT9UP3SqA1Yv5trbaGfF4uHajWxayBU8hrw2ZAYmtAWwDFytTdc6yqDepj6GwzxZuFYtq6638v
↪ ",
      "fee": 0,
      "timestamp": 1606211535602,
      "signature":
↪ "4xsEQoh6Z4wDW6jT9UP3SqA1Yv5trbaGfF4uHajWxayBU8hrw2ZAYmtAWwDFytTdc6yqDepj6GwzxZuFYtq6638v
↪ ",

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
    "role": "connection_manager"
  },
  {
    "type": 101,
    "id":
    ↪ "FSNaHMC11W3VskpGYfgxt3fqAMvt6gUmgy61CX8mm93QykuRp2E9Z8BtQc8w22Awc6W8CpXGJn6VcpkcBdAx4Tj
    ↪ ",
    "fee": 0,
    "timestamp": 1606211535601,
    "signature":
    ↪ "FSNaHMC11W3VskpGYfgxt3fqAMvt6gUmgy61CX8mm93QykuRp2E9Z8BtQc8w22Awc6W8CpXGJn6VcpkcBdAx4Tj
    ↪ ",
    "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
    "role": "contract_developer"
  },
  {
    "type": 101,
    "id":
    ↪ "4rfDMTGjbHENy3uiACLmfAHFJWyouhridZHGpynfV8S6aX3XmZHjUSfCvadm3KSzb8eHRq1kmzEaLMxvbqWkUKBY
    ↪ ",
    "fee": 0,
    "timestamp": 1606211535600,
    "signature":
    ↪ "4rfDMTGjbHENy3uiACLmfAHFJWyouhridZHGpynfV8S6aX3XmZHjUSfCvadm3KSzb8eHRq1kmzEaLMxvbqWkUKBY
    ↪ ",
    "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
    "role": "issuer"
  },
  {
    "type": 110,
    "id":
    ↪ "4q5iXHv8jZ1qw5FptfBCz1cic14u1M4zCzE1i5qqEA4z6TQmeVFaqhZRpepFpdyGiSyKH4s6XqKPTgxuEJ8Sp4QQ
    ↪ ",
    "fee": 0,
    "timestamp": 1606211535610,
    "signature":
    ↪ "4q5iXHv8jZ1qw5FptfBCz1cic14u1M4zCzE1i5qqEA4z6TQmeVFaqhZRpepFpdyGiSyKH4s6XqKPTgxuEJ8Sp4QQ
    ↪ ",
    "targetPublicKey":
    ↪ "25GXtqKBAHTCrHuDoXvwQGxNHBdeVcjdLvSmQ7SVFq4FD0MWzV78oRkgoS32AFDQ23DvfGFX6QpRkQRShQ4zMJy
    ↪ ",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ"
  },
  {
    "type": 101,
    "id":
    ↪ "2gjzK3qSp89ywXCjEpvCHKSeYqoBYR2XCKegZ1ngGrQF8cDGXjA19HN8eYTgw8DRoXy62MM138EXXiZyV7oCaZrt
    ↪ ",
    "fee": 0,
    "timestamp": 1606211535599,
    "signature":

```

(continues on next page)

(продолжение с предыдущей страницы)

```

→ "2gzjK3qSp89ywXCjEpvCHKSeYqoBYR2XCKegZ1ngGrQF8cDGXjA19HN8eYtGw8DRoXy62MM138EXXiZyV7oCaZrt
→ ",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "permissioner"
  },
  {
    "type": 101,
    "id":
→ "3zq1bCbeiNt4Z35rVtKwPo2MnW8peEcX2fQtgMseiJSb3TN7TKfU9auLEWKAgRXoNjpbpi9XA4aJw8Ly4gcpEaTv
→ ",
    "fee": 0,
    "timestamp": 1606211535598,
    "signature":
→ "3zq1bCbeiNt4Z35rVtKwPo2MnW8peEcX2fQtgMseiJSb3TN7TKfU9auLEWKAgRXoNjpbpi9XA4aJw8Ly4gcpEaTv
→ ",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "miner"
  },
  {
    "type": 101,
    "id":
→ "Aikgzt9ChSDfK4foF9oQJ8qRjV5cRyqF9okU9gr9JdpXh2LpyVB7GW4XSjmyc4MK9btPh3xd2whFDoCr8J5F4Hs
→ ",
    "fee": 0,
    "timestamp": 1606211535597,
    "signature":
→ "Aikgzt9ChSDfK4foF9oQJ8qRjV5cRyqF9okU9gr9JdpXh2LpyVB7GW4XSjmyc4MK9btPh3xd2whFDoCr8J5F4Hs
→ ",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "connection_manager"
  },
  {
    "type": 101,
    "id":
→ "48EGdWC133vQeydqMSXjmXJKB6L2brnu8Sh5W8r4anKCaUQZp5iKGrpVUAwsiUHfHrMXGA52roeoqo7abUHQbbVw
→ ",
    "fee": 0,
    "timestamp": 1606211535596,
    "signature":
→ "48EGdWC133vQeydqMSXjmXJKB6L2brnu8Sh5W8r4anKCaUQZp5iKGrpVUAwsiUHfHrMXGA52roeoqo7abUHQbbVw
→ ",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "contract_developer"
  },
  {
    "type": 101,
    "id":
→ "FwNbJyr2Est9DFi5uch1ZfkQjDg13asqSsAdm37381aMWMrdaxcjQXmpKus1rxDcxZd5YnD4MNkz1ZpPgZ8nupn
→ ",
    "fee": 0,
    "timestamp": 1606211535595,
    "signature":

```

(continues on next page)

(продолжение с предыдущей страницы)

```

→ "FwNbJyr2Est9DFi5uch1ZfkQjDg13asqSsAdm37381aMWMrdaxcjQXmpKus1rxDcxZd5YnD4MNkz1ZpPgZ8nupn
→ ",
    "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
    "role": "issuer"
  },
  {
    "type": 110,
    "id":
→ "ps5vGHxv4DfTFnTXsqeS22hXQQm8uBf1mwnc7gtDvGxGGfEhDq8DvnCjtKukYmuEW6adz5NQGLbaqbMJK7ChYdA
→ ",
    "fee": 0,
    "timestamp": 1606211535610,
    "signature":
→ "ps5vGHxv4DfTFnTXsqeS22hXQQm8uBf1mwnc7gtDvGxGGfEhDq8DvnCjtKukYmuEW6adz5NQGLbaqbMJK7ChYdA
→ ",
    "targetPublicKey":
→ "5fbBNmkW9LJBUNFJW6vsjnmBzGf2AMwdqgHNVne2iYPMNW2wtDJGmF4PGnqyzTYJyYN3kWNwd4cFf9xBZ8Qi9Hki
→ ",
    "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn"
  },
  {
    "type": 101,
    "id":
→ "5BG3AhFnGbDcSDJ88KmXViU2tCxs4VnhXGjgocn2ZCcvcJtbxGjso4DKPkcajUNJBhPZHqgMmEKugVxqBMjNf2YY
→ ",
    "fee": 0,
    "timestamp": 1606211535594,
    "signature":
→ "5BG3AhFnGbDcSDJ88KmXViU2tCxs4VnhXGjgocn2ZCcvcJtbxGjso4DKPkcajUNJBhPZHqgMmEKugVxqBMjNf2YY
→ ",
    "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
    "role": "permissioner"
  },
  {
    "type": 101,
    "id":
→ "HYoFXRgsyHGTa9JTnCDpJtBu6hr61LTYTA2zGPkUAVaTn6mhHfSKoVJbn91DN2gtqZxNreQnrV4GGnMR4cFikAE
→ ",
    "fee": 0,
    "timestamp": 1606211535593,
    "signature":
→ "HYoFXRgsyHGTa9JTnCDpJtBu6hr61LTYTA2zGPkUAVaTn6mhHfSKoVJbn91DN2gtqZxNreQnrV4GGnMR4cFikAE
→ ",
    "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
    "role": "contract_developer"
  },
  {
    "type": 101,
    "id":
→ "4snBMYD3dDw9pivJM2YFSJBPPtK4K43YGL8Qjw4APadgZCtqsR4yoo3CZC4bgf5ZffwVWQzVmfSjxpzsiwCjNju
→ ",
    "fee": 0,

```

(continues on next page)

(продолжение с предыдущей страницы)

```

        "timestamp": 1606211535592,
        "signature":
↪ "4snBMYD3dDw9pivJM2YFSJBPPtK4K43YGL8Qjw4APadgZCtqsR4yoo3CZC4bgf5ZffwVWQzVmfSjxpzsiwCjNju
↪",
        "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
        "role": "issuer"
    }
],
"version": 1,
"poa-consensus": {
    "overall-skipped-rounds": 0
},
"timestamp": 1606211535610,
"height": 1
}

```

GET /blocks/last

Метод возвращает содержимое текущего блока блокчейна.

Текущий блок находится в процессе создания, пока он не будет принят нодами-майнерами, количество транзакций в нем может меняться.

В ответе метода возвращаются следующие параметры:

- reference - хэш-сумма блока;
- blocksize - размер блока;
- features - *функциональные возможности*, запущенные на момент создания блока;
- signature - подпись блока;
- fee - комиссия за транзакции, включенные в блок;
- generator - адрес создателя блока;
- transactionCount - количество транзакций *1* и *101*, включенных в блок;
- transactions - массив с телами транзакций, включенных в блок;
- version - версия блока;
- poa-consensus.overall-skipped-rounds - количество пропущенных раундов майнинга, при использовании алгоритма консенсуса *PoA*;
- timestamp - временная метка создания блока в формате **Unix Timestamp** (в миллисекундах);
- height - высота создания блока.

Пример ответа для пустого текущего блока:

GET /blocks/last:

```
{
  "reference":
  ↪ "hT5RcPT4jDVoNspfZkNhKqfGuMbrizjpG4vmPecVfWgWaGMoAn5hgPBjPC9696TL8wGDKJzkwewiqe8m26C4aPd
  ↪ ",
  "blocksize": 226,
  "features": [],
  "signature":
  ↪ "5GAM7jfqScw4g3g7PCNNtz5xG3JzjJnW4Ap2soThirSx1AmUQHQMjz8VMtkFEzK7L447ouKHfj2gMvZyP5u94Rps
  ↪ ",
  "fee": 0,
  "generator": "3Mv79dyPX2cvLtrXn1MDDWiCZMBrkW9d97c",
  "transactionCount": 0,
  "transactions": [],
  "version": 3,
  "poa-consensus": {
    "overall-skipped-rounds": 1065423
  },
  "timestamp": 1615816767694,
  "height": 1826
}
```

GET /blocks/at/{height}

Метод возвращает содержимое блока на высоте height.

В ответе метода возвращаются следующие параметры:

- reference - хэш-сумма блока;
- blocksize - размер блока;
- features - *функциональные возможности*, запущенные на момент создания блока;
- signature - подпись блока;
- fee - комиссия за транзакции, включенные в блок;
- generator - адрес создателя блока;
- transactionCount - количество транзакций, включенных в блок;
- transactions - массив с телами транзакций, включенных в блок;
- version - версия блока;
- poa-consensus.overall-skipped-rounds - количество пропущенных раундов майнинга, при использовании алгоритма консенсуса *PoA*;
- timestamp - временная метка создания блока в формате **Unix Timestamp** (в миллисекундах);
- height - высота создания блока.

Пример ответа:

GET /blocks/at/{height}:

```
{
  "reference":
  ↪ "hT5RcPT4jDVoNspfZkNhKqfGuMbrizjpG4vmPecVfWgWaGMoAn5hgPBjPC9696TL8wGDKJzkwewiqe8m26C4aPd
  ↪ ",
  "blocksize": 226,
  "features": [],
  "signature":
  ↪ "5GAM7jfqScw4g3g7PCNntz5xG3JzjJnW4Ap2soThirSx1AmUQHQMjz8VMtkFEzK7L447ouKHfj2gMvZyP5u94Rps
  ↪ ",
  "fee": 0,
  "generator": "3Mv79dyPX2cvLtrXn1MDDWiCZMBrkW9d97c",
  "transactionCount": 0,
  "transactions": [],
  "version": 3,
  "poa-consensus": {
    "overall-skipped-rounds": 1065423
  },
  "timestamp": 1615816767694,
  "height": 1826
}
```

GET /blocks/seq/{from}/{to}

Метод возвращает содержимое блоков от высоты {from} до высоты {to}.

Для каждого блока возвращаются параметры, идентичные методу GET /blocks/at/{height}.

GET /blocks/seqext/{from}/{to}

Метод возвращает содержимое блоков с расширенной информацией о транзакциях от высоты {from} до высоты {to}.

В остальном, для каждого блока возвращаются параметры, идентичные методу GET /blocks/at/{height}.

GET /blocks/signature/{signature}

Метод возвращает содержимое блока по его подписи {signature}.

В ответе метода возвращаются параметры, идентичные методу GET /blocks/at/{height}.

GET /blocks/address/{address}/{from}/{to}

Метод возвращает содержимое всех блоков, сформированных адресатом {address} от высоты {from} до высоты {to}.

В ответе метода для каждого блока возвращаются параметры, идентичные методу GET /blocks/at/{height}.

GET /blocks/child/{signature}

Метод возвращает унаследованный блок от блока с подписью {signature}.

В ответе метода возвращаются параметры, идентичные методу GET /blocks/at/{height}.

GET /blocks/headers/at/{height}

Метод возвращает заголовок блока на высоте {height}.

В ответе метода возвращаются следующие параметры:

- reference - хэш-сумма блока;
- blocksize - размер блока;
- features - *функциональные возможности*, запущенные на момент создания блока;
- signature - подпись блока;
- fee - комиссия за транзакции, включенные в блок;
- generator - адрес создателя блока;
- pos-consensus.base-target - коэффициент, регулирующий время выпуска блока, при использовании алгоритма консенсуса *PoS*;
- pos-consensus.generation-signature - подпись, необходимая для валидации майнера блока;
- poa-consensus.overall-skipped-rounds - количество пропущенных раундов майнинга, при использовании алгоритма консенсуса *PoA*;
- version - версия блока;
- timestamp - временная метка создания блока в формате **Unix Timestamp** (в миллисекундах);
- height - высота создания блока.

Пример ответа:

GET /blocks/at/{height}:

```
{
  "reference":
  ↪ "5qWJh9aQ2hkwnBWygGYmrBhzMe5inRZ2r6WhEXz3VJsiMtASWkvbsVeZGychZKzcPDbWmpzdhQwNQJ19PfK2dst9
  ↪ ",
  "blocksize": 589,
  "features": [
    0
  ],
  "signature":
  ↪ "4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv
  ↪ ",
  "fee": 5000000,
  "generator": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "pos-consensus": {
    "base-target": 249912231,
    "generation-signature": "LM83w6eWQHnLJF2D9RQNdNcHAdnZLCLWrn5bfcoqcZy"
  },
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"poa-consensus": {  
  "overall-skipped-rounds": 2  
},  
"transactionCount": 2,  
"version": 12,  
"timestamp": 1568287320962,  
"height": 48260  
}
```

GET /blocks/headers/seq/{from}/{to}

Метод возвращает заголовки блоков с высоты {from} до высоты {to}.

В ответе метода для каждого блока возвращаются параметры, идентичные методу GET /blocks/headers/at/{height}.

GET /blocks/headers/last

Метод возвращает заголовок текущего блока.

В ответе метода для каждого блока возвращаются параметры, идентичные методу GET /blocks/headers/at/{height}.

Смотрите также

Методы REST API

10.2.12 REST API: информация о ролях участников

Для получения информации о ролях участников в сети предназначены методы группы permissions.

Подробнее о ролях участников см. статью [Роли участников](#).

GET /permissions/{address}

Метод возвращает информацию об активных ролях участника {address}, а также время формирования запроса в формате Unix Timestamp (в миллисекундах).

Пример ответа:

GET /permissions/{address}:

```
{
  "roles": [
    {
      "role": "miner"
    },
    {
      "role": "permissioner"
    }
  ],
  "timestamp": 1544703449430
}
```

GET /permissions/{address}/at/{timestamp}

Метод возвращает информацию о ролях участника {address}, активных на момент времени {timestamp}. Время указывается в формате Unix Timestamp (в миллисекундах).

Пример ответа:

GET /permissions/{address}/at/{timestamp}:

```
{
  "roles": [
    {
      "role": "miner"
    },
    {
      "role": "permissioner"
    }
  ],
  "timestamp": 1544703449430
}
```

POST /permissions/addresses

Метод возвращает роли для нескольких адресов, активные на указанный момент времени.

В запросе передаются следующие данные:

- addresses - список адресов в виде массива строк;
- timestamp - время в формате Unix Timestamp (в миллисекундах).

Пример запроса с двумя адресами:

POST /permissions/addresses:

```
{
  "addresses": [
    "3N2cQFfUDzG2iujBrFTnD2TAsCNoHDxYu8w", "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7"
  ],
  "timestamp": 1544703449430
}
```

В ответе метода возвращается массив данных `addressToRoles`, в котором указаны роли для каждого адреса, а также время `timestamp`.

Пример ответа для двух адресов:

POST /permissions/addresses:

```
{
  "addressToRoles": [
    {
      "address": "3N2cQFfUDzG2iujBrFTnD2TAsCNoHDxYu8w",
      "roles": [
        {
          "role": "miner"
        },
        {
          "role": "permissioner"
        }
      ]
    },
    {
      "address": "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7",
      "roles": [
        {
          "role": "miner"
        }
      ]
    }
  ],
  "timestamp": 1544703449430
}
```

Смотрите также

Методы REST API

Роли участников

Управление ролями участников

10.2.13 REST API: информация об ассетах и балансах адресов

Для получения информации об ассетах и балансах адресов предусмотрены методы группы `assets`.

GET /assets/balance/{address}

Метод возвращает баланс всех ассетов адреса.

В ответе возвращаются следующие параметры:

- `address` - адрес участника;
- `balances` - объект с балансами участника:
 - `assetId` - ID ассета;
 - `balance` - баланс ассета;
 - `quantity` - общее количество выпущенных токенов ассета;
 - `reissuable` - перевыпускаемость ассета;
 - `issueTransaction` - тело [транзакции 3](#) создания ассета;
 - `minSponsoredAssetFee` - минимальное значение комиссии для спонсорских транзакций;
 - `sponsorBalance` - средства, выделенные для оплаты транзакций по спонсируемым ассетам.

Пример ответа:

GET /assets/balance/{address}:

```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "balances": [
    {
      "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
      "balance": 4879179221,
      "quantity": 48791792210,
      "reissuable": true,
      "minSponsoredAssetFee" : 100,
      "sponsorBalance" : 1233221,
      "issueTransaction" : {
        "type" : 3,
        ...
      }
    },
    {
      "assetId": "49KfHPJcKvSAvNKwM7CTofjKHZL87SaSx8eyADbjv5Wi",
      "balance": 10,
      "quantity": 10000000000,
      "reissuable": false,
      "issueTransaction" : {
        "type" : 3,
        ...
      }
    }
  ]
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    }
  }
]
}

```

POST /assets/balance

Метод возвращает баланс всех активов для нескольких адресов, переданных в поле `addresses`. Параметры, передаваемые в ответе для каждого адреса, идентичны методу `GET /assets/balance/{address}`.

Пример ответа для одного адреса:

GET /assets/balance/{address}:

```

{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "balances": [
    {
      "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
      "balance": 4879179221,
      "quantity": 48791792210,
      "reissuable": true,
      "minSponsoredAssetFee" : 100,
      "sponsorBalance" : 1233221,
      "issueTransaction" : {
        "type" : 3,
        ...
      }
    },
    {
      "assetId": "49KfHPJcKvSAvNKwM7CTofjKHZL87SaSx8eyADBjv5Wi",
      "balance": 10,
      "quantity": 10000000000,
      "reissuable": false,
      "issueTransaction" : {
        "type" : 3,
        ...
      }
    }
  ]
}

```

GET /assets/balance/{address}/{assetId}

Метод возвращает баланс адреса в указанном {assetId}.

Пример ответа:

GET /assets/balance/{address}/{assetId}:

```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUD",
  "balance": 4879179221
}
```

GET /assets/details/{assetId}

Метод возвращает описание ассета {assetId}.

Пример ответа:

GET /assets/details/{assetId}:

```
{
  "assetId" : "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ",
  "issueHeight" : 140194,
  "issueTimestamp" : 1504015013373,
  "issuer" : "3NCBMxgdghg4tUhEEffSXY11L6hUi6fcBpd",
  "name" : "name",
  "description" : "Sponsored asset",
  "decimals" : 1,
  "reissuable" : true,
  "quantity" : 1221905614,
  "script" : null,
  "scriptText" : null,
  "complexity" : 0,
  "extraFee": 0,
  "minSponsoredAssetFee" : 100000
}
```

GET /assets/{assetId}/distribution

Метод возвращает количество токенов ассета на всех адресах, использующих указанный ассет.

Пример ответа:

GET /assets/details/{assetId}:

```
{
  "3P8GxcTEyZtG6LEfnn9knp9wu8uLKrAFHCb": 1,
  "3P2voHxcJg79csj4YspNq1akepX8TSmGhTE": 1200
}
```

Смотрите также*Методы REST API*

10.2.14 REST API: работа с узлами блокчейна

Для работы с узлами блокчейна предусмотрена группа методов peers:

POST /peers/connect

Метод предназначен для подключения новой ноды участника к вашей ноды.

Пример запроса:**POST /peers/connect:**

```
{
  "host": "127.0.0.1",
  "port": "9084"
}
```

Пример ответа:**POST /peers/connect:**

```
{
  "hostname": "localhost",
  "status": "Trying to connect"
}
```

GET /peers/connected

Метод возвращает список подключенных нод.

Пример ответа:

GET /peers/connected:

```
{
  "peers": [
    {
      "address": "52.51.92.182/52.51.92.182:6863",
      "declaredAddress": "N/A",
      "peerName": "zx 182",
      "peerNonce": 183759
    },
    {
      "address": "ec2-52-28-66-217.eu-central-1.compute.amazonaws.com/52.28.66.217:6863",
      "declaredAddress": "N/A",
      "peerName": "zx 217",
      "peerNonce": 1021800
    }
  ]
}
```

GET /peers/all

Метод возвращает список всех известных нод.

Пример ответа:**GET /peers/all:**

```
{
  "peers": [
    {
      "address": "/13.80.103.153:6864",
      "lastSeen": 1544704874714
    }
  ]
}
```

GET /peers/suspended

Метод возвращает список приостановленных нод.

Пример ответа:

GET /peers/suspended:

```
[
  {
    "hostname": "/13.80.103.153",
    "timestamp": 1544704754619
  }
]
```

POST /peers/identity

Метод возвращает публичный ключ ноды, к которому подключается ваша нода для передачи конфиденциальных данных.

В запросе передаются следующие параметры:

- **address** - блокчейн-адрес, который соответствует параметру `privacy.owner-address` в конфигурационном файле ноды;
- **signature** - электронная подпись от значения поля `address`.

Пример запроса:**POST /peers/identity:**

```
{
  "address": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "signature":
  → "6RwMUQcwrxtKDgM4ANes9Amu5EJgyfF9Bo6nTpXyD89ZKMAcpCM97igbWf2MmLXLdqNxdUc68fd5TyRBEB6nqf"
  → ""
}
```

Ответ метода содержит параметр `publicKey`- публичный ключ ноды, связанный с параметром `privacy.owner-address` в его конфигурационном файле. Если выключен режим проверки *handshakes*, то параметр `publicKey` не отображается.

Пример ответа:**POST /peers/identity:**

```
{
  "publicKey": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8"
}
```

GET /peers/hostname/{address}

Метод возвращает имя хоста и IP-адрес ноды по ее адресу в сети Waves Enterprise.

Пример ответа:

GET /peers/hostname/{address}:

```
{
  "hostname": "node1.we.io",
  "ip": "10.0.0.1"
}
```

GET /peers/allowedNodes

Получение актуального списка разрешенных участников сети на момент запроса.

GET /peers/allowedNodes:

```
{
  "allowedNodes": [
    {
      "address": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMdrAEJpj",
      "publicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY"
    },
    {
      "address": "3JLp8wt7rEUdn4Cca5Hp9jZ7w8T5XDAKicd",
      "publicKey": "J3ffCciVu3sustgb5vxmEHczACMR89Vty5ZBLbPn9xyg"
    },
    {
      "address": "3JRY1cp7atRMBd8QQoswRpH7DLawM5Pnk3L",
      "publicKey": "5vn4UcB9En1XgY6w2N6e9W7bqFshG4SL2RLFqEWEbWxG"
    }
  ],
  "timestamp": 1558697649489
}
```

Смотрите также

Методы REST API

10.2.15 REST API: хэширование, работа со скриптами и отправка вспомогательных запросов

Для хэширования, работы со скриптами и отправки вспомогательных запросов к ноде предусмотрена группа методов `utils`:

Хэширование: `utils/hash`

POST `/utils/hash/fast`

Метод возвращает хэш-сумму строки, переданной в запросе.

Пример ответа:

POST `/utils/hash/fast:`

```
{
  "message": "ridethewaves!",
  "hash": "DJ35ymschUFDmqCnDJewjcnVExVkWgX7mJDxhFy9X8oQ"
}
```

POST `/utils/hash/secure`

Метод возвращает двойную хэш-сумму строки, переданной в запросе.

Пример ответа:

POST `/utils/hash/secure:`

```
{
  "message": "ridethewaves!",
  "hash": "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"
}
```

Работа со скриптами: `utils/script`

Данная группа методов предназначена для конвертации кода скриптов в формат **base64** и их декодирования. Скрипты привязываются к аккаунтам при помощи транзакций [13](#) (привязка скрипта к адресу) и [15](#) (привязка скрипта к ассету для адреса).

POST /utils/script/compile

Метод конвертирует код скрипта в формат **base64**.

Пример запроса:

POST /utils/script/compile:

```
let x = 1
(x + 1) == 2
```

В ответе метода возвращаются следующие параметры:

- **script** - тело скрипта в формате **base64**;
- **complexity** - сложность скрипта: число от 1 до 100, отражающее количество вычислительных ресурсов, требуемое для его исполнения;
- **extraFee** - комиссия за исходящие транзакции, установленные скриптом.

Пример ответа:

POST /utils/script/compile:

```
{
  "script":
  ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKfGzTtXXnnv7upRHXJzZrLSQo8",
  ↪ ",
  "complexity": 11,
  "extraFee": 10001
}
```

POST /utils/script/estimate

Метод предназначен для декодирования и оценки сложности скрипта, переданного в запросе в формате **base64**.

В ответе метода возвращаются следующие параметры:

- **script** - тело скрипта в формате **base64**;
- **scriptText** - код скрипта;
- **complexity** - сложность скрипта: число от 1 до 100, отражающее количество вычислительных ресурсов, требуемое для его исполнения;
- **extraFee** - комиссия за исходящие транзакции, установленные скриптом.

Пример ответа:

POST /utils/script/compile:

```
{
  "script":
  ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKfgzTtXXnnv7upRHXJzZrLSQo8",
  ↪ ",
  "scriptText": "FUNCTION_CALL(FunctionHeader(==,List(LONG, LONG)),List(CONST_LONG(1),↪
  ↪ CONST_LONG(2)),BOOLEAN)",
  "complexity": 11,
  "extraFee": 10001
}
```

Вспомогательные запросы**GET /utils/time**

Метод возвращает текущее время ноды в двух форматах:

- system - системное время на машине ноды;
- ntp - сетевое время.

Пример ответа:**POST /utils/script/compile:**

```
{
  "system": 1544715343390,
  "NTP": 1544715343390
}
```

POST /utils/reload-wallet

Метод перезагружает keystore ноды. Применяется в случае, если новая ключевая пара была добавлена в keystore без перезапуска ноды.

Пример ответа:**POST /utils/reload-wallet:**

```
{
  "message": "Wallet reloaded successfully"
}
```

Смотрите также

Методы REST API

10.2.16 REST API: Отладка блокчейна

Для отладки блокчейн-сети предусмотрены методы группы debug:

GET /debug/blocks/{howMany}

Метод отображает размер и полный хеш последних блоков. Количество блоков указывается при запросе.

Пример ответа:

GET /debug/blocks/{howMany}:

```
[
  {
    "226": "7CkZxrAjU8bnat8CjVAPagobNYazyv1HASubmp7YYqGe"
  },
  {
    "226": "GS3y9fUHAKCamq52TPsjizDVir8J7iGoe8P2XZLasxsC"
  },
  {
    "226": "B9LmhGGDdvcfUA9JEWvyVrT9sazZE6gibpAN13xUN7KV"
  },
  {
    "226": "Byb9MHtwYf3MFyi2tbhQ3GTdCct5phKq9REkbjQTzdne"
  },
  {
    "226": "HSxSHbiV4tZc8RaN6jxdhgkAhjxuLn76uHxerMRUefA"
  }
]
```

GET /debug/info

Метод отображает общую информацию о блокчейне, необходимую для отладки и тестирования.

Пример ответа:

GET /debug/info:

```
{
  "stateHeight": 74015,
  "extensionLoaderState": "State(Idle)",
  "historyReplierCacheSizes": {
    "blocks": 13,
    "microBlocks": 2
  },
  "microBlockSynchronizerCacheSizes": {
    "microBlockOwners": 0,
    "nextInventories": 0,
    "awaiting": 0,
    "successfullyReceived": 0
  },
  "scoreObserverStats": {
    "localScore": 42142328633037120000,
    "scoresCacheSize": 4
  },
  "minerState": "mining microblocks"
}
```

POST /debug/rollback

Метод откатывает блокчейн до заданной высоты, удаляя все блоки после нее. В запросе передаются следующие параметры:

- `rollbackTo` - высота, до которой необходимо откатить блокчейн;
- `returnTransactionsToUtx` - возвращение транзакций, которые содержатся в откатываемых блоках, в UTX-пул: `true` - вернуть, `false` - удалить.

Примеры запроса и ответа:**POST /debug/rollback:**

Запрос:

```
{
  "rollbackTo": 100,
  "returnTransactionsToUtx": true
}
```

Ответ:

```
{
  "BlockId":
  → "4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv"
  →
}
```

POST /debug/validate

Метод валидирует транзакции по их идентификатору и измеряет затраченное время в миллисекундах. В запросе передается id транзакции.

Пример ответа:

POST /debug/validate:

```
{
  "valid": false,
  "validationTime": 14444
}
```

GET /debug/minerInfo

Метод отображает информацию о майнере.

Пример ответа:

GET /debug/minerInfo:

```
[
  {
    "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
    "miningBalance": 1248959867200000,
    "timestamp": 1585923248329
  }
]
```

GET /debug/historyInfo

Метод отображает историю последнего блока.

Пример ответа:

GET /debug/historyInfo:

```
{
  "lastBlockIds": [
    "37P4fvexYHPuZNPRRqYbRYxGz7x3r5jFznck7amaS6aWnHL5oQqrqCzsSh1HvYKnd2ZhU6n6sWYPb3hxsY8FBfmZ",
    "5RRu1qtesz4KvrVp4fxzQHebq2fRanNsg3HJKwD4uChqySm7vFHCdHKU6iZYXJDVmfSxiE9Maeb6sM2JireaWLbx",
    "3Lo27JfjekcZnJsYEe7st7evDZ6TgmCUBtiZrSxUCobKL48DZQ4dXMfp89WYjEykh15HEHSXzqMSTQigE8vEcN2r",
    "r4RuXEXAqgfDMKVXRWmZcGMaWKDsAvVxfXDtw8d6bamLR61J1gaoesargYSoZQqRbDrBcefLprk7D78fA728719",
    "3F4Up46crZbpKVWUeieL6GeSrVMYm7JJ7aX6aHD6B8wedFggSKv8d3H39Qy9MLEauFBU9m3qZV1U8emhmnqwmLbg"
  ]
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    ↪ ",
    "QSuBkEtVe9nik5T5S33ogeCbgKy7ihBkS2pwYayK23m4ANier83ThpajEzvpbyPy9pPWzc5St8mYUKxXDscKuRC
    ↪ ",
    "4udpNnz3e1M1GbVZxtwfg8gpF6EbiKxRCRBwi6iRMyLsvh5J2Ec9Wqyu2sq2KYL75o12yiP8TszworeUfuxNmJ5g
    ↪ ",
    "5BZY4RZAJjM5KKCaHpyUsXnb4uunnM5kcfTojc5QzQo3vyP2w3YD4qrALizkkQQR4ziS77BoAGb56QCecUtHFFN
    ↪ ",
    "5JwfLaF1oGxRXVCdDbFuKpxrvxgLCGU3kCFwxUhlL8G3xV211MrKBuAuQ4MaC5uN574uV9U8M6HfHTMERnfr5jGJ
    ↪ ",
    "4bysMhz14E1rC7dLYScfVVqPmHqzi8jdchcnkruJmCNL86TwV2cbF7G9YVchvTrv9qbQZ7JQownV59gRRcD26zm16
    ↪ "
  ],
  "microBlockIds": []
}

```

GET /debug/configInfo

Метод полностью выводит используемый конфигурационный файл ноды.

Пример ответа:

GET /debug/configInfo:

```

{
  "node": {
    "anchoring": {
      "enable": "no"
    },
    "blockchain": {
      "consensus": {
        "type": "pos"
      },
      "custom": {
        "address-scheme-character": "K",
        "functionality": {
          "blocks-for-feature-activation": 10,
          "feature-check-blocks-period": 30,
          "pre-activated-features": {
            ...
          }
        }
      },
      "wallet": {
        "file": "wallet.dat",
        "password": ""
      },
      "waves-crypto": "yes"
    }
  }
}

```

DELETE /debug/rollback-to/{signature}

Метод откатывает блокчейн до блока с указанной подписью {signature}.

Пример ответа:

DELETE /debug/rollback-to/{signature}:

```
{
  "BlockId":
  ↪ "4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv
  ↪ "
}
```

GET /debug/portfolios/{address}

Метод отображает текущий баланс по транзакциям, находящимся в UTX-пуле ноды {address}.

Пример ответа:

GET /debug/portfolios/{address}:

```
{
  "balance": 104665861710336,
  "lease": {
    "in": 0,
    "out": 0
  },
  "assets": {}
}
```

POST /debug/print

Метод выводит текущие сообщения логгера, имеющего уровень логирования DEBUG.

Ответ выводится в формате "message" : "string"

GET /debug/state

Метод отображает текущий стейт ноды.

Пример ответа:

GET /debug/state:

```
{
  "3JD3qDmgL1icDaxa3n24YSjxr9Jze5MBVVs": 4899000000,
  "3JPWx147Xf3f9fE89YtfvRhtKWBHy9rWnMK": 17528100000,
  "3JU5tCoswHH7FKPBUowySWBnQwpbZiYyNhB": 300021381800000,
  "3JCJChsQ2CGyHc9Ymu8cnsES6YzjjJELu3a": 75000362600000,
  "3JEW9XnPC8w3qQ4AJyVTDBmsVUp32QKoCGD": 5000000000,
  "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3": 6847000000,
  "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF": 1248938560600000,
  "3JV6V4JEVc3a9uSqRmdUMvMKMfZa16HbGmq": 4770000000,
  "3JZtYeGEZHjb2zQ6EcSEo524PdafPn6vWkc": 900000000,
  "3JMMFLX9d1rmXaBK9AF7Wuwzu4vRkkoVQBC": 4670000000,
  "3JJDPdQSPokKp5jEmzwMzmaPUyopLZjW1C": 800000000,
  "3JWDUsqyJEkValaivNPP8VCAa5zGuxiwD9t": 994280900000
}
```

GET /debug/stateWE/{height}

Метод отображает стейт ноды на указанной высоте {height}.

Пример ответа:**GET /debug/stateWE/{height}:**

```
{
  "3JPWx147Xf3f9fE89YtfvRhtKWBHy9rWnMK": 17528100000,
  "3JU5tCoswHH7FKPBUowySWBnQwpbZiYyNhB": 300020907600000,
  "3JCJChsQ2CGyHc9Ymu8cnsES6YzjjJELu3a": 75000350600000,
  "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3": 6847000000,
  "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF": 1248960085800000,
  "3JWDUsqyJEkValaivNPP8VCAa5zGuxiwD9t": 994280900000
}
```

Смотрите также*Методы REST API*

В каждой статье приведена таблица с адресами методов, а также полями запросов и ответов каждого метода.

Если для описываемых методов REST API требуется авторизация, в начале статьи указан значок .

Если авторизация не требуется, вы увидите значок .

Смотрите также

Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды

Разработка и применение смарт-контрактов

Определение и общее описание работы смарт-контрактов блокчейн-платформы Waves Enterprise приведено в статье [Смарт-контракты](#).

11.1 Подготовка к работе

Перед началом разработки смарт-контракта убедитесь, что на вашей машине установлен пакет ПО для контейнеризации приложений [Docker](#). Принципы работы с Docker изложены в [официальной документации](#).

Также убедитесь, что на используемой вами ноде [настроено исполнение смарт-контрактов](#). Если ваша нода работает в Waves Enterprise Mainnet, на ней по умолчанию настроены установка смарт-контрактов из открытого репозитория и установлены рекомендованные параметры для обеспечения оптимального исполнения смарт-контрактов.

Если вы разрабатываете смарт-контракт для работы в частной сети, разверните собственный [репозиторий для Docker-образов](#) и укажите его адрес и учетные данные на вашем сервере в блоке `remote-registries` конфигурационного файла ноды. В этом блоке вы можете указать несколько репозиториев, если вам необходимо определить несколько мест хранения различных смарт-контрактов. Также вы можете загрузить Docker-образ контракта из репозитория, не указанного в конфигурационном файле ноды, при помощи транзакции 103, инициирующей создание смарт-контракта. Подробнее см. раздел [Создание и установка смарт-контракта](#), а также [описание транзакции 103](#).

При работе в Waves Enterprise Mainnet в конфигурационном файле предустановлен открытый репозиторий Waves Enterprise.

11.2 Разработка смарт-контракта

Смарт-контракты блокчейн-платформы Waves Enterprise могут разрабатываться на любом предпочтительном вам языке программирования и реализовывать любые алгоритмы. Готовый код смарт-контракта упаковывается в Docker-образ с параметрами авторизации смарт-контракта (при использовании REST API) или используемыми **protobuf**-файлами (при использовании gRPC).

Примеры кода смарт-контрактов на Python с применением gRPC и REST API-методов для обмена данными с нодой, а также пошаговое руководство по созданию соответствующих Docker-образов приведены в следующих статьях:

11.2.1 Пример смарт-контракта с использованием gRPC

В этом разделе рассмотрен пример создания простого смарт-контракта на Python. Для обмена данными с нодой смарт-контракт применяет gRPC-интерфейс.

Перед началом работы убедитесь, что на вашей машине установлены утилиты из пакета **grpcio** для Python:

```
pip3 install grpcio
```

Порядок установки и использования gRPC-утилит для других доступных языков программирования приведен на [официальном сайте gRPC](#).

Описание и листинг программы

При инициализации смарт контракта при помощи транзакции 103, для него устанавливается целочисленный параметр `sum` со значением 0.

При вызове каждого вызова смарт-контракта при помощи транзакции 104, он возвращает инкремент параметра `sum` (`sum + 1`).

Листинг программы:

```
import grpc
import os
import sys

from protobuf import common_pb2, contract_pb2, contract_pb2_grpc

CreateContractTransactionType = 103
CallContractTransactionType = 104

AUTH_METADATA_KEY = "authorization"

class ContractHandler:
    def __init__(self, stub, connection_id):
        self.client = stub
        self.connection_id = connection_id
        return
```

(continues on next page)

(продолжение с предыдущей страницы)

```

def start(self, connection_token):
    self.__connect(connection_token)

def __connect(self, connection_token):
    request = contract_pb2.ConnectionRequest(
        connection_id=self.connection_id
    )
    metadata = [(AUTH_METADATA_KEY, connection_token)]
    for contract_transaction_response in self.client.
→ Connect(request=request, metadata=metadata):
        self.__process_connect_response(contract_transaction_response)

def __process_connect_response(self, contract_transaction_response):
    print("receive: {}".format(contract_transaction_response))
    contract_transaction = contract_transaction_response.transaction
    if contract_transaction.type == CreateContractTransactionType:
        self.__handle_create_transaction(contract_transaction_response)
    elif contract_transaction.type == CallContractTransactionType:
        self.__handle_call_transaction(contract_transaction_response)
    else:
        print("Error: unknown transaction type '{}'".format(contract_
→ transaction.type), file=sys.stderr)

def __handle_create_transaction(self, contract_transaction_response):
    create_transaction = contract_transaction_response.transaction
    request = contract_pb2.ExecutionSuccessRequest(
        tx_id=create_transaction.id,
        results=[common_pb2.DataEntry(
            key="sum",
            int_value=0)]
    )
    metadata = [(AUTH_METADATA_KEY, contract_transaction_response.auth_
→ token)]
    response = self.client.CommitExecutionSuccess(request=request,
→ metadata=metadata)
    print("in create tx response '{}'".format(response))

def __handle_call_transaction(self, contract_transaction_response):
    call_transaction = contract_transaction_response.transaction
    metadata = [(AUTH_METADATA_KEY, contract_transaction_response.auth_
→ token)]

    contract_key_request = contract_pb2.ContractKeyRequest(
        contract_id=call_transaction.contract_id,
        key="sum"
    )
    contract_key = self.client.GetContractKey(request=contract_key_request,
→ metadata=metadata)
    old_value = contract_key.entry.int_value

    request = contract_pb2.ExecutionSuccessRequest(
        tx_id=call_transaction.id,

```

(continues on next page)

(продолжение с предыдущей страницы)

```

        results=[common_pb2.DataEntry(
            key="sum",
            int_value=old_value + 1)]
    )
    response = self.client.CommitExecutionSuccess(request=request,
↳ metadata=metadata)
    print("in call tx response '{}'.format(response))

def run(connection_id, node_host, node_port, connection_token):
    # NOTE(gRPC Python Team): .close() is possible on a channel and should be
    # used in circumstances in which the with statement does not fit the needs
    # of the code.
    with grpc.insecure_channel('{}:{}'.format(node_host, node_port)) as
↳ channel:
        stub = contract_pb2_grpc.ContractServiceStub(channel)
        handler = ContractHandler(stub, connection_id)
        handler.start(connection_token)

CONNECTION_ID_KEY = 'CONNECTION_ID'
CONNECTION_TOKEN_KEY = 'CONNECTION_TOKEN'
NODE_KEY = 'NODE'
NODE_PORT_KEY = 'NODE_PORT'

if __name__ == '__main__':
    if CONNECTION_ID_KEY not in os.environ:
        sys.exit("Connection id is not set")
    if CONNECTION_TOKEN_KEY not in os.environ:
        sys.exit("Connection token is not set")
    if NODE_KEY not in os.environ:
        sys.exit("Node host is not set")
    if NODE_PORT_KEY not in os.environ:
        sys.exit("Node port is not set")

    connection_id = os.environ['CONNECTION_ID']
    connection_token = os.environ['CONNECTION_TOKEN']
    node_host = os.environ['NODE']
    node_port = os.environ['NODE_PORT']

    run(connection_id, node_host, node_port, connection_token)

```

Если вы хотите, чтобы транзакции с вызовом вашего контракта могли обрабатываться одновременно, то необходимо в самом коде контракта передать параметр `async-factor`. Контракт передаёт значение параметра `async-factor` в составе gRPC-сообщения `ConnectionRequest`, определенном в файле `contract_contract_service.proto`:

```

message ConnectionRequest {
  string connection_id = 1;
  int32 async_factor = 2;
}

```

Подробнее о параллельном исполнении смарт-контрактов.

Авторизация смарт-контракта с gRPC

Для работы с *gRPC* смарт-контракту необходима авторизация. Чтобы смарт-контракт корректно работал с методами API, выполняются следующие действия:

1. В переменных окружения смарт-контракта должны быть определены следующие параметры:
 - `CONNECTION_ID` – идентификатор соединения, передаваемый контрактом при соединении с нодой;
 - `CONNECTION_TOKEN` – токен авторизации, передаваемый контрактом при соединении с нодой;
 - `NODE` – ip-адрес или доменное имя ноды;
 - `NODE_PORT` – порт gRPC сервиса, развёрнутого на ноде.

Значения переменных `NODE` и `NODE_PORT` берутся из конфигурационного файла ноды секции *docker-engine.grpc-server*. Остальные переменные генерируются нодой и передаются в контейнер при создании смарт контракта.

Создание смарт-контракта

1. В директории, которая будет содержать файлы вашего смарт-контракта, создайте поддиректорию `src` и поместите в нее файл **contract.py** с кодом смарт-контракта.
2. В директории `src` создайте директорию `protobuf` и поместите в нее следующие **protobuf**-файлы:
 - `contract_contract_service.proto`
 - `data_entry.proto`

Эти файлы помещены в архив [we-proto-x.x.x.zip](#), размещенном в официальном GitHub-репозитории Waves Enterprise.

3. Сгенерируйте код gRPC-методов на Python на основе файла `contract_contract_service.proto`:

```
python3 -m grpc.tools.protoc -I. --python_out=. --grpc_python_out=. contract_contract_
↪service.proto
```

В результате будет создано два файла:

- `contract_contract_service_pb2.py`
- `contract_contract_service_pb2_grpc.py`

В файле `contract_contract_service_pb2.py` измените строку `import data_entry_pb2 as data__entry__pb2` следующим образом:

```
import protobuf.data_entry_pb2 as data__entry__pb2
```

Таким же образом измените строку `import contract_contract_service_pb2 as contract__contract__service__pb2` в файле `contract_contract_service_pb2_grpc.py`:

```
import protobuf.contract_contract_service_pb2 as contract__contract__service__pb2
```

Затем сгенерируйте вспомогательный файл `data_entry_pb2.py` на основе `data_entry.proto`:

```
python3 -m grpc.tools.protoc -I. --python_out=. data_entry.proto
```

Все три полученных файла должны находиться в директории **protobuf** вместе с исходными файлами.

4. Создайте shell-скрипт **run.sh**, который будет запускать код смарт-контракта в контейнере:

```
#!/bin/sh

eval $SET_ENV_CMD
python contract.py
```

Поместите файл **run.sh** в корневую директорию вашего смарт-контракта.

5. Создайте сценарный файл **Dockerfile** для сборки и управления запуском смарт-контракта. При разработке на Python основой образа вашего смарт-контракта может служить официальный образ Python `python:3.8-slim-buster`. Обратите внимание, что для обеспечения работы смарт-контракта в контейнере Docker должны быть установлены пакеты `dnsutils` и `grpcio-tools`.

Пример Dockerfile:

```
FROM python:3.8-slim-buster
RUN apt update && apt install -yq dnsutils
RUN pip3 install grpcio-tools
ADD src/contract.py /
ADD src/protobuf/common_pb2.py /protobuf/
ADD src/protobuf/contract_pb2.py /protobuf/
ADD src/protobuf/contract_pb2_grpc.py /protobuf/
ADD run.sh /
RUN chmod +x run.sh
ENTRYPOINT ["/run.sh"]
```

Поместите **Dockerfile** в корневую директорию вашего смарт-контракта.

6. Свяжитесь со [службой технической поддержки Waves Enterprise](#) для помещения вашего смарт-контракта в открытый репозиторий, если вы работаете в Waves Enterprise Mainnet.

Если вы работаете в частной сети, [соберите смарт-контракт самостоятельно и разместите его в собственном репозитории](#).

Как работает смарт-контракт с использованием gRPC

После вызова смарт-контракт с gRPC работает следующим образом:

1. После старта программы выполняется проверка на наличие переменных окружения.
2. Используя значения переменных окружения `NODE` и `NODE_PORT`, контракт создает gRPC-подключение с нодой.
3. Далее вызывается потоковый метод `Connect` gRPC-сервиса `ContractService`. Метод принимает gRPC-сообщение `ConnectionRequest`, в котором указывается идентификатор соединения (полученный из переменной окружения `CONNECTION_ID`). В метаданных метода указывается заголовок `authorization` со значением токена авторизации (полученного из переменной окружения `CONNECTION_TOKEN`).
4. В случае успешного вызова метода возвращается gRPC-поток (`stream`) с объектами типа `ContractTransactionResponse` для исполнения. Объект `ContractTransactionResponse` содержит два поля:
 - `transaction` – транзакция создания или вызова контракта;
 - `auth_token` – токен авторизации, указываемый в заголовке `authorization` метаданных вызываемого метода gRPC сервисов.

Если transaction содержит транзакцию 103), то для контракта инициализируется начальное состояние. Если transaction содержит транзакцию вызова (тип транзакции – 104), то выполняются следующие действия:

- с ноды запрашивается значение ключа sum (метод GetContractKey сервиса ContractService);
- значение ключа увеличивается на единицу, т.е. $sum = sum + 1$;
- новое значение ключа сохраняется на ноде (метод CommitExecutionSuccess сервиса ContractService), т.е. происходит обновление состояния контракта.

Смотрите также

Разработка и применение смарт-контрактов

Инструментарий gRPC

11.2.2 Пример смарт-контракта с использованием REST API

Описание и листинг программы

В разделе рассматривается пример создания и запуска простого смарт-контракта. Контракт выполняет инкремент переданного на вход числа при каждом вызове.

Листинг программы:

```
import json
import os
import requests
import sys

def find_param_value(params, name):
    for param in params:
        if param['key'] == name: return param['value']
    return None

def print_success(results):
    print(json.dumps(results, separators=(',', ':')))

def print_error(message):
    print(message)
    sys.exit(3)

def get_value(contract_id):
    node = os.environ['NODE_API']
    if not node:
        print_error("Node REST API address is not defined")
```

(continues on next page)

(продолжение с предыдущей страницы)

```

token = os.environ["API_TOKEN"]
if not token:
    print_error("Node API token is not defined")
headers = {'X-Contract-Api-Token': token}
url = '{0}/internal/contracts/{1}/sum'.format(node, contract_id)
r = requests.get(url, verify=False, timeout=2, headers=headers)
data = r.json()
return data['value']

if __name__ == '__main__':
    command = os.environ['COMMAND']
    if command == 'CALL':
        contract_id = json.loads(os.environ['TX'])['contractId']
        value = get_value(contract_id)
        print_success([
            "key": "sum",
            "type": "integer",
            "value": value + 1])
    elif command == 'CREATE':
        print_success([
            "key": "sum",
            "type": "integer",
            "value": 0])
    else:
        print_error("Unknown command {0}".format(command))

```

Пошаговое описание работы:

- Программа ожидает получить структуру данных в формате json с полем params;
- считывает значение поля a;
- возвращает результат в виде значения поля «{a}+1» в формате json.

Пример входящих параметров:

```

"params": [
  {
    "key": "a",
    "type": "integer",
    "value": 1
  }
]

```

Авторизация смарт-контракта с REST API

Для работы с *REST API ноды* смарт-контракту необходима авторизация. Чтобы смарт-контракт корректно работал с методами API, выполните следующие действия:

1. В переменных окружения смарт-контракта должны быть определены следующие параметры:
 - `NODE_API` - URL-адрес к REST API ноды;
 - `API_TOKEN` - токен авторизации для смарт-контракта;
 - `COMMAND` - команды для создания и вызова смарт-контракта;
 - `TX` - транзакция, необходимая Docker-контракту для работы (*103 - 107*).
2. Разработчик Docker-контракта присваивает значение переменной `API_TOKEN` заголовку запроса `X-Contract-API-Token`. В переменную `API_TOKEN` нода прописывает JWT-токен авторизации при создании и выполнении контракта.
3. Код контракта должен передавать полученный токен в заголовке запроса (`X-Contract-API-Token`) при каждом обращении к API ноды.

Создание смарт-контракта

1. Поместите в нее файл **contract.py** с кодом в директорию, которая будет содержать файлы вашего смарт-контракта.
2. Создайте shell-скрипт **run.sh**, который будет запускать код смарт-контракта в контейнере:

```
#!/bin/sh

python contract.py
```

Поместите файл **run.sh** в корневую директорию вашего смарт-контракта.

3. Создайте сценарный файл **Dockerfile** для сборки и управления запуском смарт-контракта. При разработке на Python, основой образа вашего смарт-контракта может служить официальный образ Python на основе Alpine Linux `python:alpine3.8`.

Пример Dockerfile:

```
FROM python:alpine3.8
ADD contract.py /
ADD run.sh /
RUN chmod +x run.sh
CMD exec /bin/sh -c "trap : TERM INT; (while true; do sleep 1000; done) & wait"
```

Поместите **Dockerfile** в корневую директорию вашего смарт-контракта.

4. Свяжитесь со *службой технической поддержки Waves Enterprise* для помещения вашего смарт-контракта в открытый репозиторий, если вы работаете в Waves Enterprise Mainnet.

Если вы работаете в частной сети, *соберите смарт-контракт самостоятельно и разместите его в собственном репозитории*.

Смотрите также*Разработка и применение смарт-контрактов**Методы REST API*

11.3 Загрузка смарт-контракта в репозиторий

Если вы работаете в Waves Enterprise Mainnet, свяжитесь со [службой технической поддержки Waves Enterprise](#) для добавления смарт-контракта в открытый репозиторий Waves Enterprise.

При работе в частной сети, загрузите Docker-образ смарт-контракта в собственный репозиторий:

1. Запустите ваш репозиторий в контейнере:

```
docker run -d -p 5000:5000 --name my-registry-container my-registry:2
```

2. Перейдите в директорию, содержащую файлы смарт-контракта и сценарный файл Dockerfile с командами для сборки образа.

3. Соберите образ вашего смарт-контракта:

```
docker build -t my-contract .
```

4. Укажите имя образа и адрес его размещения в репозитории:

```
docker image tag my-contract my-registry:5000/my-contract
```

5. Запустите созданный вами контейнер репозитория:

```
docker start my-registry-container
```

6. Загрузите ваш смарт-контракт в репозиторий:

```
docker push my-registry:5000/my-contract
```

7. Получите информацию о смарт-контракте. Для этого выведите информацию о контейнере:

```
docker image ls | grep 'my-node:5000/my-contract'
```

Таким образом вы получите идентификатор контейнера. Выведите информацию о нем при помощи команды `docker inspect`:

```
docker inspect my-contract-id
```

Пример ответа:

```
{
  "Id": "sha256:57c2c2d2643da042ef8dd80010632ffdd11e3d2e3f85c20c31dce838073614dd",
  "RepoTags": [
    "wenode:latest"
  ],
  "RepoDigests": [],
  "Parent": "sha256:d91d2307057bf3bb5bd9d364f16cd3d7eda3b58edf2686e1944bcc7133f07913",
  "Comment": "",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"Created": "2019-10-25T14:15:03.856072509Z",
"Container": "",
"ContainerConfig": {
  "Hostname": "",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
```

Поле Id - это идентификатор Docker-образа смарт-контракта, который вводится в поле ImageHash транзакции 103 при создании смарт-контракта.

11.4 Размещение смарт-контракта в блокчейне

После загрузки смарт-контракта в репозиторий установите его в сети при помощи транзакции 103. Для этого подпишите транзакцию посредством клиента блокчейн-платформы, метода sign REST API или метода *JavaScript SDK*.

Данные, возвращенные в ответе метода, подаются на вход при публикации транзакции 103.

Ниже приведены примеры подписания и отправки транзакции при помощи методов sign и broadcast. В примерах транзакции подписываются ключом, сохраненным в keystore ноды.

Curl-запрос на подписание транзакции 103:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
→application/json' --header 'X-Contract-API-Token' -d '{ \
  "fee": 100000000, \
  "image": "my-contract:latest", \
  "imageHash":
→"7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
  "contractName": "my-contract", \
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2", \
  "password": "", \
  "params": [], \
  "type": 103, \
  "version": 1 \
}' 'http://my-node:6862/transactions/sign'
```

Ответ метода sign, который передается методу broadcast:

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLZZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 100000000,
  "timestamp": 1550591678479,
```

(continues on next page)

(продолжение с предыдущей страницы)

```

"proofs": [
  ↪ "yecRFZm9iBlyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv
  ↪ ],
  "version": 1,
  "image": "my-contract:latest",
  "imageHash":
  ↪ "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "my-contract",
  "params": [],
  "height": 1619
}

```

Curl-запрос на отправку транзакции 103:

```

curl -X POST --header 'Content-Type: application/json' --header 'Accept:
  ↪ application/json' --header 'X-Contract-API-Token' -d '{ \
{
  "type": 103, \
  "id": "ULcQ9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLZVj4Ky", \
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M", \
  "fee": 500000, \
  "timestamp": 1550591678479, \
  "proofs": [
  ↪ "yecRFZm9iBlyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv
  ↪ ], \
  "version": 1, \
  "image": "my-contract:latest", \
  "imageHash":
  ↪ "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
  "contractName": "my-contract", \
  "params": [], \
  "height": 1619 \
}
}' 'http://my-node:6862/transactions/broadcast'

```

11.5 Исполнение смарт-контракта

После размещения смарт-контракта в блокчейне он может быть вызван при помощи транзакции [104](#) *CallContract Transaction*.

Эта транзакция также может быть подписана и отправлена в блокчейн посредством клиента блокчейн-платформы, метода sign REST API или метода *JavaScript SDK*. При подписании транзакции 104 в поле `contractId` укажите идентификатор транзакции 103 для вызываемого смарт-контракта (поле `id` ответа метода `sign`).

Примеры подписания и отправки транзакции при помощи методов `sign` и `broadcast` с использованием ключа, сохраненного в `keystore` ноды:

Curl-запрос на подписание транзакции 104:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
↪application/json' --header 'X-Contract-API-Token' -d '{ \
"contractId": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLZVj4Ky", \
"fee": 10, \
"sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
"password": "", \
"type": 104, \
"version": 1, \
"params": [ \
  { \
    "type": "integer", \
    "key": "a", \
    "value": 1 \
  } \
] \
}' 'http://my-node:6862/transactions/sign'
```

Ответ метода sign, который передается методу broadcast:

```
{
"type": 104,
"id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
"sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
"senderPublicKey": "2YvzcVLrqLCqouVrFZynjfoEuPNV9GrdauNpgdWXLsq",
"fee": 10,
"timestamp": 1549365736923,
"proofs": [
↪"2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v
↪"
],
"version": 1,
"contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
"params": [
  {
    "key": "a",
    "type": "integer",
    "value": 1
  }
]
}
```

Curl-запрос на отправку транзакции 104:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept:
↪application/json' --header 'X-Contract-API-Token' -d '{ \
"type": 104, \
"id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP", \
"sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58", \
"senderPublicKey": "2YvzcVLrqLCqouVrFZynjfoEuPNV9GrdauNpgdWXLsq", \
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"fee": 10, \
"timestamp": 1549365736923, \
"proofs": [ \
  ↪ "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v
  ↪ " \
], \
"version": 1, \
"contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2", \
"params": [ \
  { \
    "key": "a", \
    "type": "integer", \
    "value": 1 \
  } \
] \
}' 'http://my-node:6862/transactions/broadcast'
```

Смотрите также

Смарт-контракты

Общая настройка платформы: настройка исполнения смарт-контрактов

JavaScript SDK

JavaScript SDK - это библиотека для интеграции приложений с платформой Waves Enterprise. Она решает широкий круг задач, связанных с подписанием и отправкой в блокчейн транзакций.

JavaScript SDK поддерживает:

- работу как в браузере, так и в среде Node.js;
- стандарты шифрования ГОСТ;
- подписание всех типов транзакций платформы Waves Enterprise;
- операции с seed-фразами: создание новой фразы, создание из существующей фразы, шифрование;
- клиентскую реализацию методов ноды `crypto/encryptCommon`, `crypto/encryptSeparate`, `crypto/decrypt`.

Для работы с блокчейном JavaScript SDK использует *методы REST API ноды*. Однако приложения, написанные с помощью этой библиотеки, не взаимодействуют с блокчейном напрямую, а подписывают транзакции локально - в браузере или в Node.js. После локального подписания транзакции отправляются в сеть. Такой способ взаимодействия позволяет разрабатывать многоуровневые приложения и сервисы, взаимодействующие с блокчейном.

Данные от приложения передаются и принимаются в формате *json* по HTTPS-протоколу.

Общая схема работы JavaScript SDK:



Пакет JavaScript SDK, а также инструкции по его установке доступны в [GitHub-репозитории Waves Enterprise](#).

12.1 Содержание

12.1.1 Как работает JavaScript SDK

Авторизация в блокчейне

Для того, чтобы пользователь приложения мог взаимодействовать с блокчейном, необходимо авторизовать его в сети. Для этого в JavaScript SDK предусмотрены методы REST API сервиса авторизации, которые позволяют составить многоуровневый алгоритм со всеми возможными типами запросов, связанных с авторизацией пользователя в блокчейне.

Авторизация может производиться как в браузере, так и в среде Node.js.

При авторизации в браузере, используется интерфейс **Fetch API**.

Для авторизации посредством Node.js, применяется HTTP-клиент **Axios**.

Если используемая приложением нода блокчейна использует метод авторизации OAuth, для его авторизации рекомендуется применять библиотеку **api-token-refresher**. Эта библиотека автоматически обновляет токены доступа при истечении времени их использования. Подробнее о работе с OAuth-авторизацией и применении библиотеки api-token-refresher см. раздел «Применение JS SDK в ноде с OAuth».

Создание seed-фразы

Приложение на базе JS SDK может работать с seed-фразами в следующих вариантах:

- создать новую рандомизированную seed-фразу;
- создать seed-фразу из существующей фразы;
- зашифровать seed-фразу паролем или расшифровать ее.

Примеры работы JS SDK с seed-фразами приведены в разделе «Варианты создания seed-фразы».

Подписание и отправка транзакций

Для приложений на основе JS SDK доступны подписание и отправка в блокчейн любых транзакций платформы. Список всех транзакций приведен в разделе [Описание транзакций](#).

Процесс подписания и отправки транзакций в сеть выглядит следующим образом:

1. Приложение инициирует создание транзакции.
2. Все поля транзакции сериализуются в байт-код при помощи вспомогательного компонента JS SDK transactions-factory.
3. Затем транзакция при помощи компонента signature-generator подписывается приватным ключом пользователя в браузере или в среде Node.js. Подписание транзакции производится при помощи POST-запроса /transactions/sign.
4. JavaScript SDK отправляет транзакцию в блокчейн при помощи POST-запроса /transactions/broadcast.
5. Приложение получает ответ в виде хэша транзакции на выполненный POST-запрос.

Примеры подписания и отправки различных типов транзакций приведены в разделе «Примеры создания и отправки транзакций».

Криптографические методы ноды, используемые JavaScript SDK

JavaScript SDK доступны три криптографических метода:

- `crypto/encryptCommon` - шифрование данных для всех получателей единым ключом СЕК, который в свою очередь оборачивается уникальными ключами КЕК для каждого получателя;
- `crypto/encryptSeparate` - шифрование текста отдельно для каждого получателя уникальным ключом;
- `crypto/decrypt` - расшифровка данных при условии, если ключ получателя сообщения находится в `keystore` ноды.

Компонент **signature-generator** также поддерживает как криптографию по ГОСТ, так и алгоритмы криптографии Waves.

Смотрите также

[JavaScript SDK](#)

[Описание транзакций](#)

[REST API: реализация методов шифрования](#)

12.1.2 Установка и инициализация JS SDK

Если вы планируете пользоваться JS SDK в среде Node.js, установите пакет Node.js с официального сайта.

Установите пакет **js-sdk** при помощи **npm**:

```
npm install @wavesenterprise/js-sdk --save
```

В выбранной среде разработки импортируйте пакет, содержащий библиотеку JS SDK:

```
import WeSdk from '@wavesenterprise/js-sdk'
```

Помимо импорта пакета, вы можете использовать функцию `require`:

```
const WeSdk = require('@wavesenterprise/js-sdk');
```

Затем инициализируйте библиотеку:

```
const config = {
  ...WeSdk.MAINNET_CONFIG,
  nodeAddress: 'https://hoover.welocal.dev/node-0',
  crypto: 'waves',
  networkByte: 'V'.charCodeAt(0)
}

const Waves = WeSdk.create({
  initialConfiguration: config,
  fetchInstance: window.fetch // Browser feature. For Node.js use node-fetch
});
```

При работе в браузере, в качестве `fetchInstance` используется функция `window.fetch`. Если вы работаете в Node.js, воспользуйтесь модулем `node-fetch`.

После инициализации JavaScript SDK вы можете начать создание и отправку транзакций.

Ниже приведен полный листинг с созданием типовой транзакции:

```
import WeSdk from '@wavesenterprise/js-sdk'

const config = {
  ...WeSdk.MAINNET_CONFIG,
  nodeAddress: 'https://hoover.welocal.dev/node-0',
  crypto: 'waves',
  networkByte: 'V'.charCodeAt(0)
}

const Waves = WeSdk.create({
  initialConfiguration: config,
  fetchInstance: window.fetch
});

// Create a seed phrase from an existing one
const seed = Waves.Seed.fromExistingPhrase('examples seed phrase');

const txBody = {
  recipient: seed.address, // Send tokens to the same address
  assetId: '',
  amount: '10000',
  fee: '1000000',
  attachment: 'Examples transfer attachment',
  timestamp: Date.now()
};

const tx = Waves.API.Transactions.Transfer.V3(txBody);

await tx.broadcast(seed.keyPair)
```

Описание параметров транзакций, а также их примеры доступны в разделе «Создание и отправка транзакций».

Смотрите также

JavaScript SDK

12.1.3 Создание и отправка транзакций при помощи JS SDK

Принципы создания транзакции

Вызов любой транзакции осуществляется при помощи функции `Waves.API.Transactions.<ИМЯ_ТРАНЗАКЦИИ>.<ВЕРСИЯ_ТРАНЗАКЦИИ>`.

Например, так выглядит вызов транзакции для перевода токенов 3 версии:

```
const tx = Waves.API.Transactions.Transfer.V3(txBody);
```

txBody - тело транзакции, содержащее необходимые параметры. К примеру, для вышеуказанной транзакции `Transfer` оно может выглядеть так:

```
const tx = Waves.API.Transactions.Transfer.V3(txBody);

{
  "sender": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "password": "",
  "recipient": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "amount": 40000000000,
  "fee": 100000
}
```

Тело транзакции можно оставить пустым и заполнить необходимые параметры позднее при помощи обращения к переменной, в которую возвращается результат функции вызова транзакции (в примере - переменная `tx`):

```
const tx = Waves.API.Transactions.Transfer.V3({});
tx.recipient = '12afdsdga243134';
tx.amount = 10000;
//...
tx.sender = "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX";
//...
tx.amount = 40000000000;
tx.fee = 10000;
```

Такой способ вызова транзакции позволяет более гибко производить числовые операции в коде и пользоваться отдельными функциями для определения тех или иных параметров.

Транзакции [3](#), [13](#), [14](#) и [112](#) используют текстовое поле `description`, а транзакции [4](#) и [6](#) - текстовое поле `attachment`. Сообщения, отправляемые в этих полях транзакций, перед отправкой необходимо перевести в формат **base58**. Для этого в JS SDK предусмотрены две функции:

- `base58.encode` - перевод текстовой строки в формат `base58`;
- `base58.decode` - обратная рашифровка строки в формате `base58` в текст.

Пример тела транзакции с применением `base58.encode`:

```
const txBody = {
  recipient: seed.address,
  assetId: '',
  amount: 10000,
  fee: minimumFee[4],
  attachment: Waves.tools.base58.encode('Examples transfer attachment'),
  timestamp: Date.now()
}

const tx = Waves.API.Transactions.Transfer.V3(txBody);
```

Внимание: При вызове транзакций при помощи JS SDK вам требуется заполнить все необходимые параметры тела транзакции, кроме `type`, `version`, `id`, `proofs` и `senderPublicKey`. Эти параметры заполняются автоматически при генерации пары ключей (`keyPair`).

Описание параметров, входящих в тело каждой транзакции, см. в разделе [Описание транзакций](#).

Отправка транзакции

Для отправки транзакции в сеть посредством JS SDK используется метод `broadcast`:

```
await tx.broadcast(seed.keyPair);
```

Этот метод вызывается после создания транзакции и заполнения ее параметров. Результат его выполнения может быть присвоен переменной для отображения результата отправки транзакции в сеть (в примере - переменная `result`):

```
try {
  const result = await tx.broadcast(seed.keyPair);
  console.log('Broadcast PolicyCreate result: ', result)
} catch (err) {
  console.log('Broadcast error:', err)
}
```

Ниже приведен полный листинг вызова транзакции перевода токенов и ее отправки:

```
const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
  ↪config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,
    fetchInstance: fetch
  });

  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

  const txBody = {
    recipient: seed.address,
    assetId: '',
    amount: 10000,
    fee: minimumFee[4],
    attachment: Waves.tools.base58.encode('Examples transfer attachment'),
    timestamp: Date.now()
  }

  const tx = Waves.API.Transactions.Transfer.V3(txBody);

  try {
    const result = await tx.broadcast(seed.keyPair);
    console.log('Broadcast transfer result: ', result)
  } catch (err) {
    console.log('Broadcast error:', err)
  }

})();

```

Примеры вызова и отправки других транзакций см. в разделе «Примеры использования» Дополнительные методы, доступные при создании и отправке транзакции

Помимо метода broadcast, для отладки и определения параметров транзакции доступны следующие методы:

- isValid - проверка тела транзакции, возвращает 0 или 1;
- getErrors - возвращает строковый массив, содержащий описание ошибок, допущенных при заполнении полей;
- getSignature - возвращает строку с ключом, которым была подписана транзакция;
- getId - возвращает строку с идентификатором отправляемой транзакции;
- getBytes - внутренний метод, который возвращает массив байт для подписания.

Смотрите также*JavaScript SDK**Описание транзакций**Комиссии в сети «Waves Enterprise Mainnet»***12.1.4 Примеры использования JavaScript SDK****Передача токенов (4)**

```

const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
  ↪config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,
    fetchInstance: fetch
  });

  // Create Seed object from phrase
  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

  // see docs: https://docs.wavesenterprise.com/en/latest/how-the-platform-works/data-
  ↪structures/transactions-structure.html#transfertransaction
  const txBody = {
    recipient: seed.address,
    assetId: '',
    amount: 10000,
    fee: minimumFee[4],
    attachment: Waves.tools.base58.encode('Examples transfer attachment'),
  }

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    timestamp: Date.now()
  }

  const tx = Waves.API.Transactions.Transfer.V3(txBody);

  try {
    const result = await tx.broadcast(seed.keyPair);
    console.log('Broadcast transfer result: ', result)
  } catch (err) {
    console.log('Broadcast error:', err)
  }
}());

```

Создание группы доступа к конфиденциальным данным (112)

```

const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
  ↪config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,
    fetchInstance: fetch
  });

  // Create Seed object from phrase
  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

  // Transaction data
  // https://docs.wavesenterprise.com/en/latest/how-the-platform-works/data-structures/

```

(continues on next page)

(продолжение с предыдущей страницы)

```

↪ transactions-structure.html#createpolicytransaction
const txBody = {
  sender: seed.address,
  policyName: 'Example policy',
  description: 'Description for example policy',
  owners: [seed.address],
  recipients: [],
  fee: minimumFee[112],
  timestamp: Date.now(),
}

const tx = Waves.API.Transactions.CreatePolicy.V3(txBody);

try {
  const result = await tx.broadcast(seed.keyPair);
  console.log('Broadcast PolicyCreate result: ', result)
} catch (err) {
  console.log('Broadcast error:', err)
}

})();

```

Выдача или отзыв роли участника (102)

```

const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
↪ config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,

```

(continues on next page)

(продолжение с предыдущей страницы)

```
    fetchInstance: fetch
  });

  // Create Seed object from phrase
  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);
  const targetSeed = Waves.Seed.create(15);

  // https://docs.wavesenterprise.com/en/latest/how-the-platform-works/data-structures/
  ↪ transactions-structure.html#permittransaction
  const txBody = {
    target: targetSeed.address,
    opType: 'add',
    role: 'issuer',
    fee: minimumFee[102],
    timestamp: Date.now(),
  }

  const permTx = Waves.API.Transactions.Permit.V2(txBody);

  try {
    const result = await permTx.broadcast(seed.keyPair);
    console.log('Broadcast ADD PERMIT: ', result)

    const waitTimeout = 30

    console.log(`Wait ${waitTimeout} seconds while tx is mining...`)

    await new Promise(resolve => {
      setTimeout(resolve, waitTimeout * 1000)
    })

    const removePermitBody = {
      ...txBody,
      opType: 'remove',
      timestamp: Date.now()
    }

    const removePermitTx = Waves.API.Transactions.Permit.V2(removePermitBody);

    const removePermitResult = await removePermitTx.broadcast(seed.keyPair);

    console.log('Broadcast REMOVE PERMIT: ', removePermitResult)
  } catch (err) {
    console.log('Broadcast error:', err)
  }
})();
```

Создание смарт-контракта (103)

```

const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
↪config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,
    fetchInstance: fetch
  });

  // Create Seed object from phrase
  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

  const timestamp = Date.now();

  //body description: https://docs.wavesenterprise.com/en/latest/how-the-platform-
↪works/data-structures/transactions-structure.html#createcontracttransaction
  const txBody = {
    senderPublicKey: seed.keyPair.publicKey,
    image: 'vostok-sc/grpc-contract-example:2.1',
    imageHash: '9fddd69022f6a47f39d692dfb19cf2bdb793d8af7b28b3d03e4d5d81f0aa9058',
    contractName: 'Sample GRPC contract',
    timestamp,
    params: [],
    fee: minimumFee[103]
  };

  const tx = Waves.API.Transactions.CreateContract.V3(txBody)

  try {
    const result = await tx.broadcast(seed.keyPair);
    console.log('Broadcast docker create result: ', result)
  }

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    } catch (err) {
      console.log('Broadcast error:', err)
    }
  })();

```

Вызов смарт-контракта (104)

```

const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
  ↪config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,
    fetchInstance: fetch
  });

  // Create Seed object from phrase
  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

  const timestamp = Date.now()

  //body description: https://docs.wavesenterprise.com/en/latest/how-the-platform-
  ↪works/data-structures/transactions-structure.html#callcontracttransaction
  const txBody = {
    authorPublicKey: seed.keyPair.publicKey,
    contractId: '4pSJowSaYvT8iCSAxUYdc7LwznFexnBGPRoUJX7Lw3sh', // Predefined
    ↪contract
    contractVersion: 1,
    timestamp,

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    params: [],
    fee: minimumFee[104]
  };

  const tx = Waves.API.Transactions.CallContract.V4(txBody)

  try {
    const result = await tx.broadcast(seed.keyPair);
    console.log('Broadcast docker call result: ', result)
  } catch (err) {
    console.log('Broadcast error:', err)
  }
}());

```

Атомарная транзакция (120)

```

const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
  ↪config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,
    fetchInstance: fetch
  });

  // Create Seed object from phrase
  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

  const transfer1Body = {

```

(continues on next page)

(продолжение с предыдущей страницы)

```
    recipient: seed.address,
    amount: 10000,
    fee: minimumFee[4],
    attachment: Waves.tools.base58.encode('Its beautiful!'),
    timestamp: Date.now(),
    atomicBadge: {
      trustedSender: seed.address
    }
  }

const transfer1 = Waves.API.Transactions.Transfer.V3(transfer1Body);

const transfer2Body = {
  recipient: seed.address,
  amount: 100000,
  fee: minimumFee[4],
  attachment: Waves.tools.base58.encode('Its beautiful!'),
  timestamp: Date.now(),
  atomicBadge: {
    trustedSender: seed.address
  }
}

const transfer2 = Waves.API.Transactions.Transfer.V3(transfer2Body);

const dockerCall1Body = {
  authorPublicKey: seed.keyPair.publicKey,
  contractId: '4pSJoWsaYvT8iCSAxUYdc7LwznFexnBGPRoUJX7Lw3sh', // Predefined contract
  contractVersion: 1,
  timestamp: Date.now(),
  params: [],
  fee: minimumFee[104],
  atomicBadge: {
    trustedSender: seed.address
  }
}

const dockerCall1 = Waves.API.Transactions.CallContract.V4(dockerCall1Body);

const dockerCall2Body = {
  authorPublicKey: seed.keyPair.publicKey,
  contractId: '4pSJoWsaYvT8iCSAxUYdc7LwznFexnBGPRoUJX7Lw3sh',
  contractVersion: 1,
  timestamp: Date.now() + 1,
  params: [],
  fee: minimumFee[104],
  atomicBadge: {
    trustedSender: seed.address
  }
}

const dockerCall2 = Waves.API.Transactions.CallContract.V4(dockerCall1Body);
```

(continues on next page)

(продолжение с предыдущей страницы)

```

const policyDataText = `Some random text ${Date.now()}`
const uint8array = Waves.tools.convert.stringToByteArray(policyDataText);
const { base64Text, hash } = Waves.tools.encodePolicyData(uint8array)

const policyDataHashBody = {
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "policyId": "9QUUuQ5XetCe2wEyrSX95NEVzPw2bscfFfAzVZL5ZJN",
  "type": "file",
  "data": base64Text,
  "hash": hash,
  "info": {
    "filename": "test-send1.txt",
    "size": 1,
    "timestamp": Date.now(),
    "author": "temakolodko@gmail.com",
    "comment": ""
  },
  "fee": 5000000,
  "password": "sfgKYBFCF0#$fsdf()*%",
  "timestamp": Date.now(),
  "version": 3,
  "apiKey": 'vostok',
}
const policyDataHashTxBody = {
  ...policyDataHashBody,
  atomicBadge: {
    trustedSender: seed.address
  }
}

const policyDataHashTx = Waves.API.Transactions.PolicyDataHash.
↪ V3(policyDataHashTxBody);

try {
  const transactions = [transfer1, transfer2, policyDataHashTx]
  const broadcast = await Waves.API.Transactions.broadcastAtomic(
    Waves.API.Transactions.Atomic.V1({transactions}),
    seed.keyPair
  );
  console.log('Atomic broadcast successful, tx id:', broadcast.id)
} catch (err) {
  console.log('Create atomic error:', err)
}

})();

```

Выпуск/сжигание токенов (3 / 6)

```

const { create: createApiInstance, MAINNET_CONFIG } = require('..');
const nodeFetch = require('node-fetch');

const nodeAddress = 'https://hoover.welocal.dev/node-0';
const seedPhrase = 'examples seed phrase';

const fetch = (url, options = {}) => {
  const headers = options.headers || {}
  return nodeFetch(url, { ...options, headers: {...headers, 'x-api-key': 'vostok'} });
}

(async () => {
  const { chainId, minimumFee, gostCrypto } = await (await fetch(`${nodeAddress}/node/
  ↪config`)).json();

  const wavesApiConfig = {
    ...MAINNET_CONFIG,
    nodeAddress,
    crypto: gostCrypto ? 'gost' : 'waves',
    networkByte: chainId.charCodeAt(0),
  };

  const Waves = createApiInstance({
    initialConfiguration: wavesApiConfig,
    fetchInstance: fetch
  });

  // Create Seed object from phrase
  const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

  const quantity = 1000000

  //https://docs.wavesenterprise.com/en/latest/how-the-platform-works/data-structures/
  ↪transactions-structure.html#issuetransaction
  const issueBody = {
    name: 'Sample token',
    description: 'The best token ever made',
    quantity,
    decimals: 8,
    reissuable: false,
    chainId: Waves.config.getNetworkByte(),
    fee: minimumFee[3],
    timestamp: Date.now(),
    script: null
  }

  const issueTx = Waves.API.Transactions.Issue.V2(issueBody)
  try {
    const result = await issueTx.broadcast(seed.keyPair);

```

(continues on next page)

(продолжение с предыдущей страницы)

```

console.log('Broadcast ISSUE result: ', result)
const waitTimeout = 30
console.log(`Wait ${waitTimeout} seconds while tx is mining...`)

await new Promise(resolve => {
  setTimeout(resolve, waitTimeout * 1000)
})

const burnBody = {
  assetId: result.assetId,
  amount: quantity,
  fee: minimumFee[6],
  chainId: Waves.config.getNetworkByte(),
  timestamp: Date.now()
}

const burnTx = Waves.API.Transactions.Burn.V2(burnBody)

const burnResult = await burnTx.broadcast(seed.keyPair);
console.log('Broadcast BURN result: ', burnResult)
} catch (err) {
  console.log('Broadcast error:', err)
}
})();

```

Смотрите также*JavaScript SDK***12.1.5 Применение JS SDK в ноде с OAuth-авторизацией**

Если нода использует OAuth-авторизацию, необходимо инициализировать Waves API с заголовками авторизации для вызова.

Для автоматического обновления токенов при разработке приложений с JS SDK мы рекомендуем использовать внешний модуль **api-token-refresher**. Однако вместо него вы можете использовать свое решение.

Для работы с **api-token-refresher** установите зависимости при помощи **npm**:

```

npm i @wavesenterprise/api-token-refresher@3.1.0 --save, axios --save-dev, cross-fetch --
→save-dev, @wavesenterprise/js-sdk@3.1.1 --save

```

Инициализация **api-token-refresher** производится следующим образом:

```

import { init: initRefresher } from '@wavesenterprise/api-token-refresher/dist/fetch'

const { fetch } = initRefresher({
  authorization: {
    access_token,

```

(continues on next page)

(продолжение с предыдущей страницы)

```
    refresh_token
  }
});
```

```
const Waves = WeSdk.create({
  initialConfiguration: config,
  fetchInstance: fetch
});
```

Параметры `access_token` и `refresh_token` приведены в ответе на запрос авторизации в клиент `loginSecure`, который доступен в браузере.

Ниже приведен листинг, содержащий инициализацию библиотеки с последующей проверкой первого блока:

```
const WeSdk = require('@wavesenterprise/js-sdk');
const { ApiTokenRefresher } = require('@wavesenterprise/api-token-refresher');

const apiTokenRefresher = new ApiTokenRefresher({
  authorization: {
    access_token: 'access_token',
    refresh_token: 'refresh_token'
  }
})

const { fetch } = apiTokenRefresher.init()

const Waves = WeSdk.create({
  initialConfiguration: {
    ...WeSdk.MAINNET_CONFIG,
    nodeAddress: 'https://hoover.welocal.dev/node-1',
    crypto: 'waves',
    networkByte: 'V'.charCodeAt(0)
  },
  fetchInstance: fetch
});

const testFirstBlock = async () => {
  const data = await Waves.API.Node.blocks.first()
  console.log('First block:', data)
}

testFirstBlock()
```

Смотрите также

JavaScript SDK

Сервисы авторизации и подготовки данных

12.1.6 Варианты создания seed-фразы и работы с ней в JS SDK

1. Создание новой рандомизированной seed-фразы

```
const seed = Waves.Seed.create();

console.log(seed.phrase); // 'hole law front bottom then mobile fabric under horse drink
↳ other member work twenty boss'
console.log(seed.address); // '3Mr5af3Y7r7gQej3tRtugYbKaPr5qYps2ei'
console.log(seed.keyPair); // { privateKey: 'HkFCbtBHX1ZUF42aNE4av52JvdDPWth2jbP88HPTDyp4
↳ ', publicKey: 'AF9HLq2Rsv2fVfLPtsWxT7Y3S9ZTv6Mw4ZTp8K8LNdEp' }
```

2. Создание seed-фразы из существующей

```
const anotherSeed = Waves.Seed.fromExistingPhrase('a seed which was backed up some time
↳ ago');

console.log(seed.phrase); // 'newly created seed'
console.log(seed.address); // '3N3dy1P8Dccup5WnYsrC6VmaGHF6wMxdLn4'
console.log(seed.keyPair); // { privateKey: '2gSboTPsiQfi1i3zNtFppVJVgjoCA9P4HE9K95y8yCMm
↳ ', publicKey: 'CFr94paUndSTRk8jz6Ep3bzhXb9LKarNmLYXW6gqw6Y3' }
```

3. Шифрование seed-фразы паролем и расшифровка

Пример шифрования seed-фразы паролем:

```
const password = '0123456789';
const encrypted = seed.encrypt(password);

console.log(encrypted); // 'U2FsdGVkX1+5TpaxcK/
↳ eJyjht7bSpjLY1SU8gVXNapU3MG8xgWm3uavW37aPz/
↳ KTcR0K70j0A3dpCLXfZ4YjCV30W2r1CCaUhOMPBCX64QA/iAlgPJNtfMvjLKTHZko/
↳ JDgrxBHgQkz76apORWdKEQ=='
```

Пример расшифровки seed-фразы при помощи пароля:

```
const restoredPhrase = Waves.Seed.decryptSeedPhrase(encrypted, password);

console.log(restoredPhrase); // 'hole law front bottom then mobile fabric under horse
↳ drink other member work twenty boss'
```

Смотрите также

JavaScript SDK

Смотрите также

Криптография

REST API: реализация методов шифрования

Транзакции блокчейн-платформы

Обмен конфиденциальными данными

Блокчейн-платформа Waves Enterprise позволяет ограничить доступ к определенным данным, размещаемым в блокчейне. Для этого пользователи объединяются в группы, получающие доступ к конфиденциальным данным.

13.1 Создание группы доступа

Создать группу доступа к конфиденциальным данным может любой участник сети. Прежде, чем создавать группу доступа, определитесь со списком участников, которые будут в нее входить. Затем подпишите и отправьте транзакцию [112 CreatePolicy](#):

1. В поле `recipients` впишите через запятую адреса участников, которые получают доступ к конфиденциальным данным.
2. В поле `owners` добавьте через запятую адреса участников группы, которым будут предоставлены права администраторов. Администраторы группы доступа, помимо доступа к конфиденциальным данным, смогут изменять состав группы доступа.

При отправке транзакции, вы получите идентификатор созданной группы доступа (`policyId`). Он потребуется при изменении состава ее участников.

После отправки транзакции в блокчейн доступ к отправляемым в сеть конфиденциальным данным получат все участники, зарегистрированные в созданной группе доступа. Как создатель транзакции, вы сможете изменять ее состав, как и участники, добавленные в поле `owners`.

13.2 Изменение группы доступа

Изменять состав группы доступа могут только ее участники, добавленные в поле `owners` при создании группы, а также сам ее создатель - **владелец группы**.

Для этого подпишите и отправьте транзакцию [113 UpdatePolicy](#):

1. В поле `policyId` введите идентификатор изменяемой группы доступа.
2. В поле `opType` введите действие, которое необходимо произвести с группой: `add` - добавить участников; `remove` - удалить участников.
3. Если вы хотите добавить или удалить участников группы доступа, впишите их публичные ключи в поле `recipients`.
4. Для добавления или удаления владельцев группы доступа впишите их публичные ключи в поле `owners`.

Информация о группе доступа обновляется после отправки транзакции в блокчейн.

13.3 Отправка конфиденциальных данных в сеть

Для отправки конфиденциальных данных в сеть предусмотрены методы REST API **POST /privacy/sendData** и **POST /privacy/sendDataV2**. Эти методы требуют авторизации.

С помощью методов **POST /privacy/sendData** и **POST /privacy/sendDataV2** вы можете отправить данные размером до **20 мегабайт**.

При отправке данных, укажите в запросе следующую информацию:

- `sender` - блокчейн-адрес, от которого должны рассылаться данные (соответствуют значению параметра `privacy.owner-address` в конфигурационном файле ноды);
- `password` - пароль для доступа к закрытому ключу `keystore` ноды;
- `policyId` - идентификатор группы, которая будет иметь доступ к отправляемым данным;
- `info` - информация об отправляемых данных;
- `data` - строка, содержащая данные в формате **base64**;
- `hash` - sha256-хеш данных в формате **base58**.

Примеры запроса и ответа метода **POST /privacy/sendData**:

POST /privacy/sendData:

Запрос:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "info": {
    "filename": "Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "Alvanov@org.com",
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "comment": "some comments"
  },
  "data":
  ↪ "TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpYByZWZzb24sIGJ1dCBieSB0aGlzIHdpbmd1bGFyIHh3c3Np
  ↪ ",
  "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
}

```

Ответ:

```

{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [
    ↪ "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc
    ↪ "
  ],
  "fee": 110000000,
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
}

```

Метод **POST /privacy/sendDataV2** позволяет приложить файл в окне Swagger, не прибегая к его конверсии в формат **base64**. Поле Data в этой версии метода отсутствует.

Примеры запроса и ответа метода **POST /privacy/sendDataV2**:

POST /privacy/sendDataV2:

Запрос:

```

{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "info": {
    "filename": "Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "Alvanov@org.com",
    "comment": "some comments"
  },
  "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
}

```

Ответ:

```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [

    ↪ "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc
    ↪ "
  ],
  "fee": 110000000,
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
}
```

В результате отправки того или иного запроса будет сформирована транзакция *114 PolicyDataHash*, которая отправит хэш конфиденциальных данных в блокчейн.

Смотрите также

Описание транзакций

REST API: обмен конфиденциальными данными и получение информации о группах доступа

Управление ролями участников

Описание всех ролей блокчейн-платформы приведено в статье [Роли участников](#). Роли могут быть произвольно скомбинированы для любого адреса, отдельные роли могут быть отозваны в любой момент.

Для управления ролями участников предусмотрена транзакция *102 Permission Transaction*, которая может быть подписана при помощи метода `sign` REST API ноды и отправлена при помощи соответствующего метода gRPC или REST API. Отправлять транзакцию 102 в блокчейн может только участник с ролью **permissioner**.

Вне зависимости от применяемого метода отправки, транзакция включает следующие поля:

- `type` - тип транзакции для управления полномочиями участников (`type = 102`);
- `sender` - адрес участника с полномочиями на отправку транзакции 102 (ролью **permissioner**);
- `password` - пароль от ключевой пары в `keystore` ноды, опциональное поле;
- `proofs` - подпись транзакции;
- `target` - адрес участника, для которого требуется установить или удалить полномочия;
- `role` - полномочия участника, которые требуется установить или удалить;
- `opType` - тип операции: `add` (добавить роль) или `remove` (удалить роль);
- `dueTimestamp` - дата действия `permission` в формате **Unix Timestamp** (в миллисекундах), опциональное поле.

Полученный ответ метода `sign` передается методу `broadcast` gRPC или REST API ноды.

Смотрите также

Описание транзакций

REST API: информация о ролях участников

Подключение и удаление нод

При работе в Waves Enterprise Mainnet, ноды участников подключаются к сети и удаляются из нее *при помощи специалистов Waves Enterprise*.

В частной сети подключение и удаление новых участников выполняется после ручной конфигурации и старта первой ноды.

15.1 Подключение новой ноды к частной сети

Для подключения новой ноды выполните следующее:

1. Настройте ноду в соответствии с указаниями, приведенными в статье *Развертывание платформы в частной сети*.
2. Передайте публичный ключ новой ноды и ее описание администратору вашей сети.
3. Администратор сети (нода с ролью **connection-manager**) использует полученные публичный ключ и описание ноды при создании транзакции *111 RegisterNode*. Для регистрации ноды в параметре орТуре, определяющем тип совершаемого действия, указывается add (добавление новой ноды).
4. Транзакция 111 попадает в блока затем - в стейты нод участников сети. В дальнейшем каждый участник сети обязательно хранит публичный ключ и адрес новой ноды.
5. При необходимости, администратор сети может добавить новой ноде дополнительные роли при помощи транзакции *102*. Подробнее о назначении ролей участников см. статью *Распределение ролей участников*.
6. Запустите новую ноду.

15.2 Удаление ноды из частной сети

Для удаления ноды из сети администратор сети отправляет в блокчейн транзакцию [111 RegisterNode](#). В ней он указывает публичный ключ удаляемой ноды и параметр "opType": "remove" (удаление ноды из сети).

После публикации транзакции в блокчейн данные ноды удаляются из стейтов всех участников.

Смотрите также

[Описание транзакций](#)

[Управление ролями участников](#)

[Архитектура](#)

Запуск ноды с созданным снимком данных

Для изменения параметров приватного блокчейна без потери сохраненных в нем данных в блокчейн-платформе Waves Enterprise предусмотрен *механизм создания снимка данных*.

Настройка механизма создания снимка данных выполняется в конфигурационном файле ноды (см. раздел *Тонкая настройка платформы: настройка механизма создания снимка данных*).

После создания снимка данных в приватном блокчейне вы, как администратор сети, можете изменить его параметры и перезапустить его с использованием данных, сохраненных в снимке.

Для этого выполните следующие действия:

1. При помощи метода **GET /snapshot/status** убедитесь, что снимок данных был получен вашей нодой и успешно верифицирован;
2. При помощи метода **GET /snapshot/genesis-config** запросите конфигурацию нового genesis-блока и сохраните ее;
3. Методом **POST /snapshot/swap-state** замените текущий стейт сети на снимок данных и дождитесь успешного ответа;
4. Подготовьте конфигурационные файлы ноды для перезапуска:
 - измените параметры генезис-блока на полученные в пункте 2;
 - отключите механизм создания снимка данных (`node.consensual-snapshot.enable = no`);
 - при необходимости, измените параметры секции `blockchain` конфигурационного файла ноды;
5. Перезапустите ноду.

После перезапуска ноды генерируется новый genesis-блок сети. Сеть запускается с обновленными параметрами и данными, записанными в снимке данных.

Смотрите также

REST API: информация о конфигурации и состоянии ноды, остановка ноды

17.1 Устройство платформы

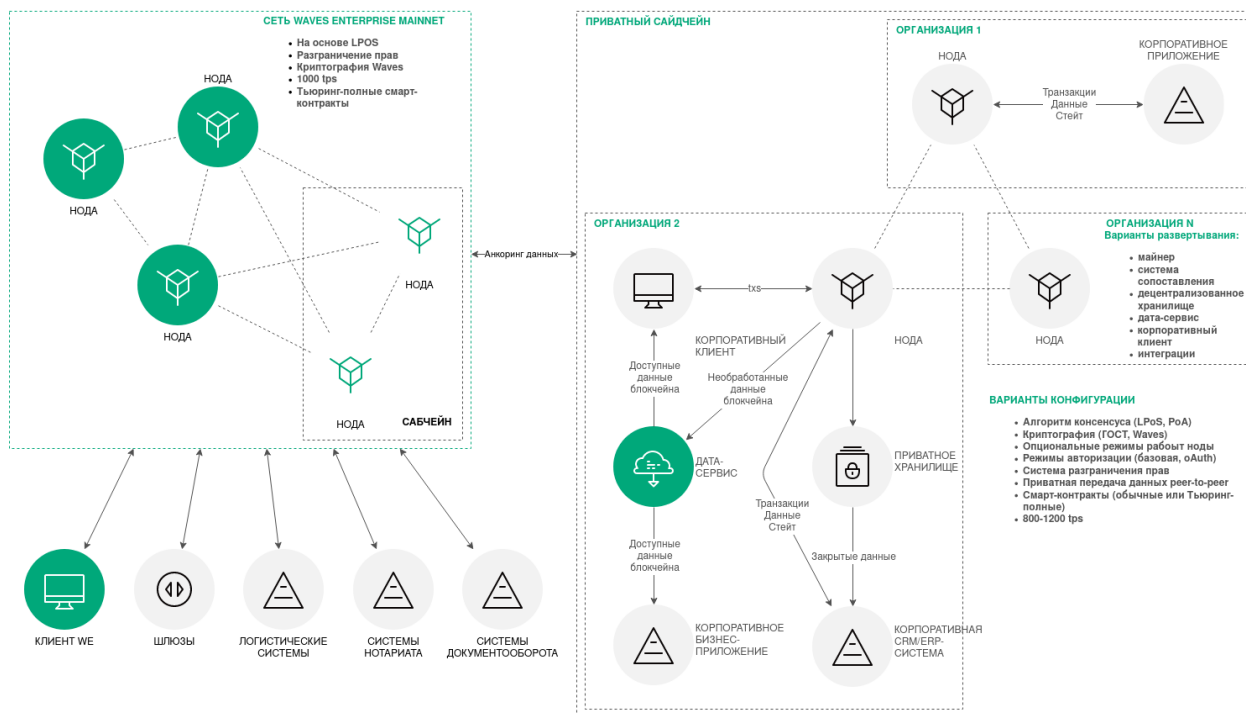
Платформа Waves Enterprise построена на базе технологии распределенного реестра и представляет собой фрактальную сеть, состоящую из двух элементов:

- **мастер-блокчейна** (Waves Enterprise Mainnet), который обеспечивает работу сети в целом и выступает в качестве глобального арбитра как для опорной сети, так и для множества пользовательских;
- отдельных **сайдчейнов**, создаваемых для решения конкретных бизнес-задач.

Взаимодействие между мастер-блокчейном и сайдчейнами обеспечивается при помощи механизма анкоринга сетей, который помещает криптографические доказательства транзакций в основную блокчейн-сеть. Механизм анкоринга позволяет свободно конфигурировать сайдчейны и использовать любой алгоритм консенсуса без потери связи с мастер-блокчейном. Например, мастер-блокчейн Waves Enterprise базируется на алгоритме консенсуса Proof-of-Stake, так как поддерживается независимыми участниками. В то же время корпоративные сайдчейны, в которых нет необходимости стимуляции майнеров за счёт комиссий за транзакции, могут использовать алгоритмы Proof-of-Authority или Crash Fault Tolerance.

Такой двухчастный принцип построения позволяет оптимизировать блокчейн-сеть для высоких вычислительных нагрузок, увеличить скорость передачи информации, а также повысить согласованность и доступность данных. Применение механизма анкоринга повышает доверие к данным в сайдчейнах, поскольку они валидируются в мастер-блокчейне.

Схематичное изображение архитектуры платформы:



17.2 Устройство ноды и дополнительных сервисов

Каждая нода блокчейна - это самостоятельный участник сети, имеющий ПО для работы в ней. Нода состоит из следующих компонентов:

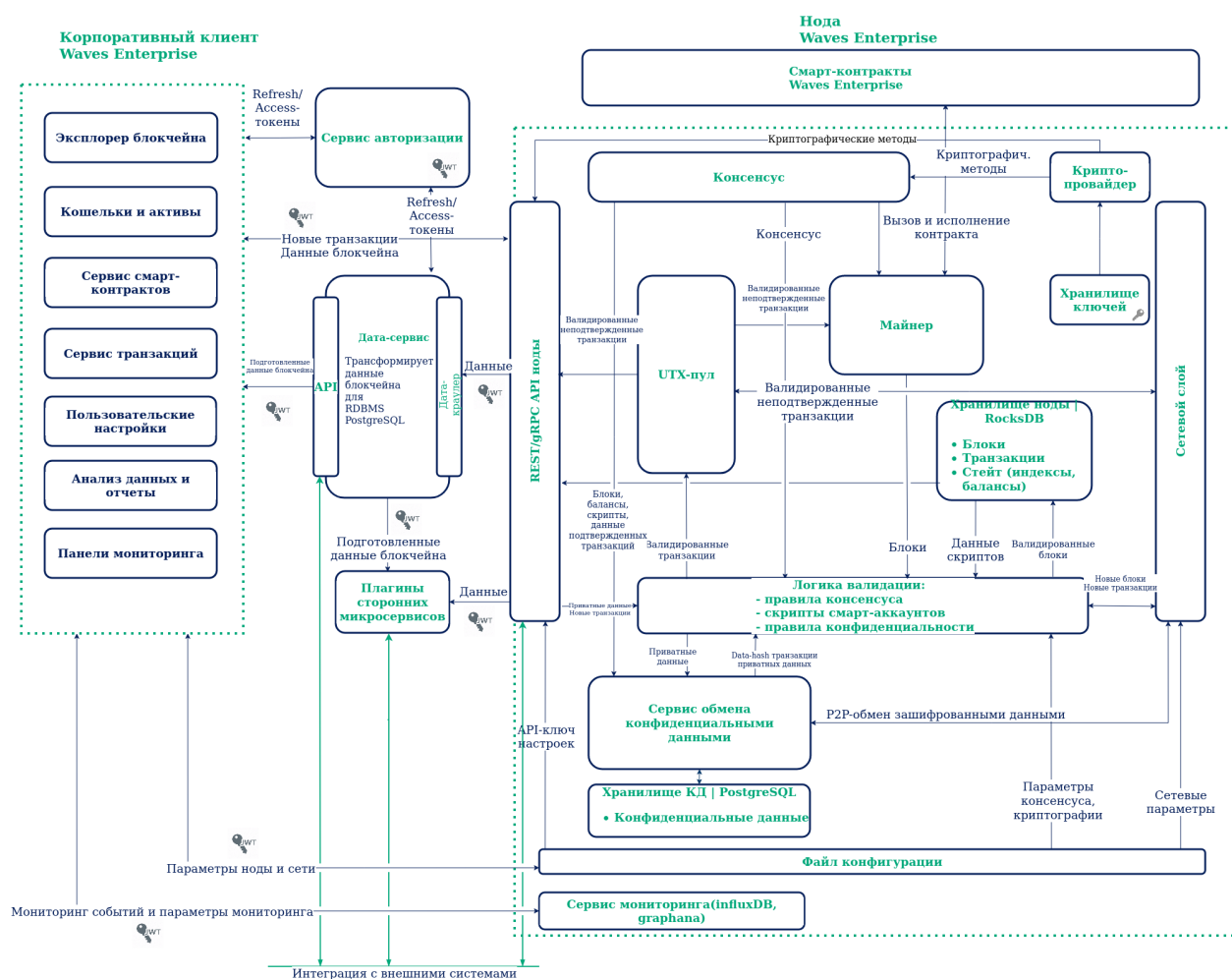
- **Сервисы консенсуса и криптографические библиотеки (Consensus and cryptolibraries)** – компоненты, отвечающие за механизм достижения согласия между узлами, а также за криптографические алгоритмы.
- **API-интерфейсы ноды (Node API)** – интерфейсы gRPC и REST API ноды, позволяющие получать данные из блокчейна, подписывать и отправлять транзакции, отправлять конфиденциальные данные, создавать и выполнять смарт-контракты и др.
- **Пул неподтвержденных транзакций (Unconfirmed transaction pool, UTX pool)** – компонент, обеспечивающий хранение неподтвержденных транзакций до момента их проверки и отправки в блокчейн.
- **Майнер (Miner)** – компонент, отвечающий за формирование блоков транзакций для записи в блокчейн, а также за взаимодействие со смарт-контрактами.
- **Хранилище ключей (Key store)** - хранилище ключевых пар ноды и пользователей. Все ключи защищены паролем.
- **Сетевой слой (Network layer)** – слой логики, обеспечивающий взаимодействие нод на прикладном уровне по сетевому протоколу поверх TCP.
- **Хранилище ноды (Node storage)** – компонент системы на базе RockDB, обеспечивающий хранение пар ключ-значение для полного набора проверенных и подтвержденных транзакций и блоков, а также текущего состояния блокчейна.
- **Логика валидации (Validation logic)** – слой логики, содержащий такие правила проверки транзакций, как базовая проверка подписи и расширенная проверка по сценарию.
- **Конфигурация (Config)** – конфигурационные параметры ноды, задаваемые в файле *node-name.conf*.

Для каждой ноды предусмотрен набор дополнительных сервисов:

- **Сервис авторизации** - сервис обеспечения авторизации для всех компонентов.
- **Дата-краулер** - сервис извлечения данных с ноды и загрузки извлечённых данных в дата-сервис.
- **Генератор** - сервис генерации ключевых пар для новых аккаунтов и создания `api-key-hash`.
- **Сервис мониторинга** - внешний сервис мониторинга, использующий базу данных InfluxDB для хранения временных рядов с данными и метриками приложения.

Установка дополнительных сервисов не обязательна, однако они облегчают взаимодействие пользователя с блокчейн-сетью. Помимо готовых сервисов, в зависимости от поставленных задач, могут разрабатываться интеграционные адаптеры, предназначенные для доставки транзакций от клиентских приложений в сеть, а также обмена данными между нодой и прикладными сервисами заказчика.

Схематичное изображение устройства отдельной ноды и дополнительных сервисов:



Смотрите также

Протокол работы блокчейна Waves-NG

Алгоритмы консенсуса

Криптография

Примеры конфигурационных файлов ноды

Сервисы авторизации и подготовки данных

Генераторы

Протокол работы блокчейна Waves-NG

Waves-NG — протокол, разработанный Waves Enterprise на основе протокола Bitcoin-NG. Основная концепция протокола — непрерывное создание микроблоков вместо одного крупного блока в каждом раунде майнинга. Такой подход позволяет увеличить скорость работы блокчейна, поскольку микроблоки гораздо быстрее валидируются и передаются по сети.

18.1 Описание раунда майнинга

Каждый раунд майнинга состоит из следующих этапов:

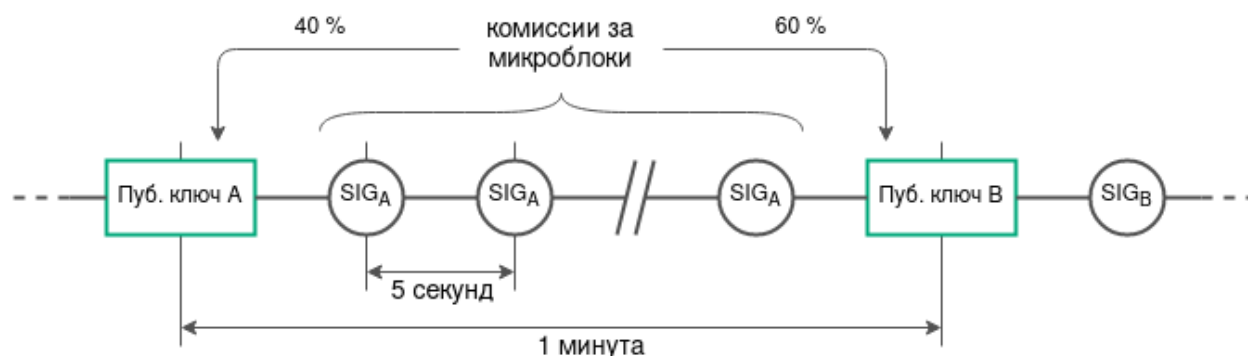
1. Применяемый алгоритм консенсуса определяет майнера раунда и время выпуска **ключевого блока**, не содержащего транзакций.
2. Майнер раунда выпускает ключевой блок, который содержит только служебную информацию:
 - публичный ключ майнера для проверки подписи микроблоков;
 - сумму комиссии майнера за предыдущий блок;
 - подпись майнера;
 - ссылку на предыдущий ключевой блок.
3. После формирования ключевого блока майнер раунда формирует **liquid block**: каждые 5 секунд создает и рассылает по сети микроблоки, содержащие транзакции. На этом этапе микроблоки не валидируются алгоритмом консенсуса, что увеличивает скорость их создания. Первый микроблок ссылается на ключевой блок, каждый последующий — на предыдущий.
4. Процесс формирования микроблоков в составе liquid block продолжается до формирования следующего валидного ключевого блока, который завершает раунд. В момент формирования следующего ключевого блока liquid block со всеми созданными майнером раунда микроблоками оформляется в очередной блок, входящий в блокчейн.

18.2 Механизм вознаграждения майнеров

Протокол Waves-NG предусматривает финансовую мотивацию для майнеров. За каждую транзакцию в блокчейне Waves Enterprise предусмотрена комиссия в WAVES, все комиссии за транзакции внутри микроблоков суммируются в ходе раунда. Вознаграждение распределяется следующим образом:

- **40%** комиссии получает майнер, создавший блок в текущем раунде;
- **60%** комиссии получает майнер следующего раунда.

Транзакция по начислению комиссии происходит каждые 100 блоков для обеспечения доверительного интервала проверок:



18.3 Разрешение конфликтов при создании блоков

Если майнер продолжает уже созданную цепочку, создавая два микроблока с одним и тем же родительским блоком, возникает несогласованность транзакций. Она выявляется какой-либо нодой блокчейна в момент появления очередного микроблока, когда нода применяет полученные изменения к своей копии состояния сети и сверяет с остальными узлами.

Протокол Waves-NG определяет такую ситуацию как мошенничество. Майнер, продолживший чужую цепочку, наказывается лишением дохода от комиссий раунда. Нода, обнаружившая несогласованность, получает награду майнера.

Также факты создания и публикации невалидных блоков в блокчейне выявляются применяемыми алгоритмами консенсуса.

Смотрите также

[Архитектура](#)

[Алгоритмы консенсуса](#)

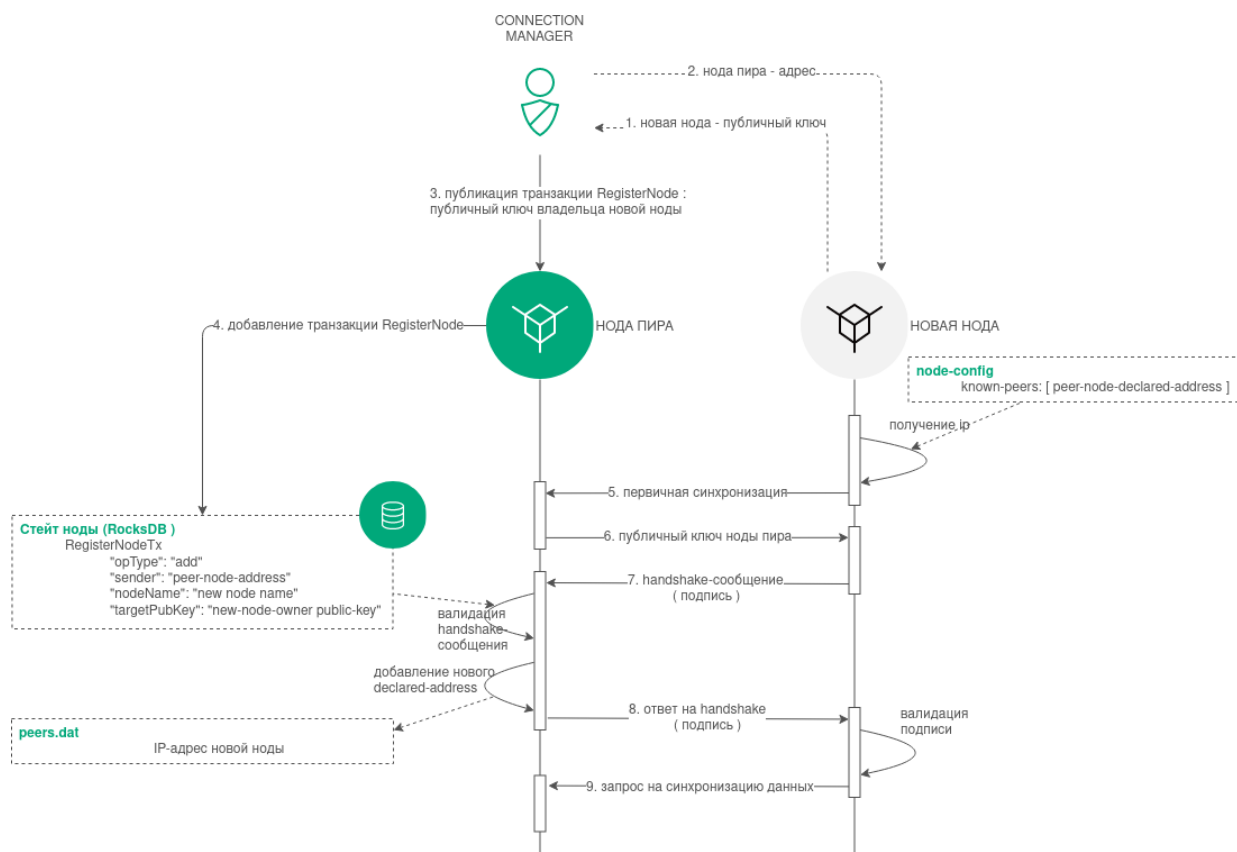
Подключение новой ноды к сети

Блокчейн-платформа Waves Enterprise имеет возможность подключения новых нод к блокчейн-сети в любой момент.

Практические шаги по подключению ноды описаны в статье [Подключение и удаление нод](#).

В общем виде процесс подключения новой ноды к сети представлен на схеме:

1. Пользователь новой ноды передаёт публичный ключ ноды администратору сети (ноде с ролью **connection-manager**).
2. Нода с ролью **connection-manager** использует полученный публичный ключ при создании транзакции [111 RegisterNode](#) с параметром «**opType**»: «**add**».
3. Транзакция 111 попадает в блок.
4. Далее информация из транзакции 111 (адрес отправителя, присвоенное новой ноде имя и ее публичный ключ) передается в стейты нод участников сети.
5. Если ключ новой ноды отсутствует в списке нод, зарегистрированных в genesis-блоке сети (Network Participants), производится процедура первичной синхронизации. Новая нода отправляет всем адресам из списка пиров своего конфигурационного файла сетевое сообщение **PeerIdentityRequest** со своей подписью. Пирей удостоверяются, что нода, отправившая **PeerIdentityRequest**, была зарегистрирована в сети.
6. При успешной проверке, в ответ на **PeerIdentityRequest**, пирей отправляют новой ноде свои публичные ключи. Новая нода сохраняет эти публичные ключи в своем временном хранилище адресов для первичного установления соединения с пирей. После сохранения адресов, новая нода получает возможность валидировать сетевые handshake-сообщения от своих пиров.
7. Новая нода отправляет handshake-сообщение со своим публичным ключом участникам сети из списка пиров своего конфигурационного файла.
8. Пирей сравнивают публичный ключ из handshake-сообщения и ключ новой ноды из транзакции 111, отправленной нодой с ролью **connection-manager**. Если проверка успешна, пирей отправляют новой ноде ответы на handshake-сообщение со своими подписями и рассылают в сеть сообщения **Peers Message**.



9. После успешного подключения новая нода выполняет синхронизацию с сетью и получает таблицу адресов участников сети.

Смотрите также

Архитектура

Подключение и удаление нод

Роли участников

Активация функциональных возможностей

Блокчейн-платформа Waves Enterprise поддерживает возможность активации функциональных возможностей блокчейна путем голосования нод - иными словами, **механизм софт-форка блокчейна**. Активация новых функциональных возможностей - необратимое действие, поскольку блокчейн не поддерживает отката софт-форка.

В голосовании могут участвовать только ноды с ролью `miner`, поскольку голос ноды сохраняется в созданный ей блок.

20.1 Параметры голосования

В блоке `features` секции `node` конфигурационного файла каждой ноды предусмотрен блок `supported`, в который вносятся идентификаторы функциональных возможностей, поддерживаемых нодой:

```
features {  
  supported = [100]  
}
```

Параметры голосования определяются в блоке `functionality` конфигурационного файла ноды:

- `feature-check-blocks-period` - период проведения голосования (в блоках);
- `blocks-for-feature-activation` - количество блоков с идентификатором функциональной возможности, необходимых для ее активации.

По умолчанию каждая нода настроена таким образом, чтобы голосовать за все поддерживаемые ей функциональные возможности.

Внимание: Параметры голосования ноды нельзя менять во время работы блокчейна: для полной синхронизации нод они должны быть унифицированы для всей сети.

20.2 Процедура голосования

1. В своем раунде майнинга нода голосует за функциональные возможности, включенные в блок `features.supported`, если они еще не были активированы в блокчейне: идентификаторы возможностей вносятся в поле `features` блока при его создании. Затем созданные блоки публикуются в блокчейне. Таким образом в течение интервала `feature-check-blocks-period` происходит голосование всех нод, имеющих роль `miner`.
2. По прошествии интервала `feature-check-blocks-period` производится подсчет голосов - идентификаторов каждой функциональной возможности в созданных блоках.
3. Если возможность, вынесенная на голосование, набирает количество голосов, большее или равное параметру `blocks-for-feature-activation`, то она приобретает статус **APPROVED** (утверждена).
4. Утвержденная функциональная возможность активируется по прошествии интервала `feature-check-blocks-period` от текущей высоты блокчейна.

20.3 Использование активированных функциональных возможностей

При активации новой функциональной возможности, она может использоваться всеми нодами блокчейна, которые ее поддерживают. Если какая-либо нода не поддерживает активированную возможность, происходит отключение этой ноды от блокчейна в момент публикации первой транзакции, задействующей неподдерживаемую функциональную возможность.

При включении новой ноды в блокчейн, предусмотрена автоматическая активация возможностей, набравших необходимое число голосов в прошедших периодах голосования. Активация происходит в ходе синхронизации ноды при условии поддержки этих возможностей самой нодой.

20.4 Предварительная активация функциональных возможностей

Все функциональные возможности, за которые предусмотрена возможность голосования, могут быть активированы принудительно при старте нового блокчейна. Для этого предусмотрен блок `pre-activated-features` в секции `blockchain` конфигурационного файла ноды:

```
pre-activated-features = {
  ...
  101 = 0
}
```

После знака равенства напротив каждой функциональной возможности указывается высота, на которой следует активировать ту или иную возможность.

20.5 Список идентификаторов функциональных возможностей

Идентификатор	Название
100	Алгоритм консенсуса LPoS
101	Поддержка gRPC смарт-контрактами Docker
119	Оптимизация производительности для алгоритма консенсуса PoA
120	Поддержка спонсорских транзакций
130	Оптимизация скорости работы с историей банов майнера
140	Поддержка атомарных транзакций
160	Поддержка параллельного создания liquid-block и микроблока

Смотрите также

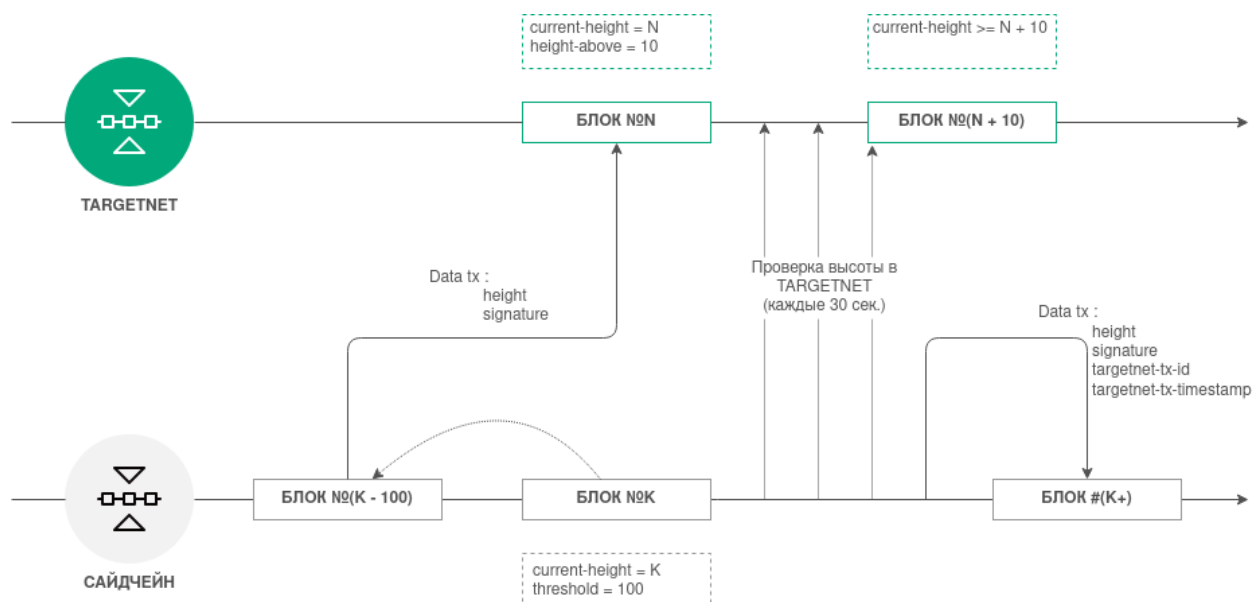
rest-sf

В приватном блокчейне транзакции обрабатываются определенным списком участников, каждый из которых заранее известен. Малое, по сравнению с публичной сетью, количество участников, блоков и транзакций в приватном блокчейне несёт угрозу подмены информации. Что, в свою очередь, создает риск перезаписи цепочки блоков - особенно в случае использования алгоритма консенсуса PoS, не имеющего защиты от таких ситуаций.

Для повышения доверия участников приватного блокчейна к размещенным в нём данным разработан механизм **анкоринга**. Анкоринг позволяет проверить данные на неизменность. Гарантия неизменности достигается публикацией данных из приватного блокчейна в более крупную сеть, где подмена данных менее вероятна из-за большего количества участников и блоков. Из приватной сети публикуются подписи блоков и высота блокчейна. Взаимная связность двух и более сетей повышает их устойчивость, поскольку для подлога или изменения данных в результате **long-range атаки** необходимо атаковать все связанные сети.

21.1 Как работает анкоринг в блокчейне Waves Enterprise

1. Выполняется **настройка анкоринга** в конфигурационном файле ноды приватного блокчейна (установите параметры в соответствии с рекомендациями раздела, чтобы избежать сложностей при работе анкоринга);
2. Через каждый заданный диапазон блоков **height-range** нода фиксирует информацию о блоке на высоте **current-height - threshold** в виде транзакции в Targetnet. В качестве такой транзакции используется **транзакция с данными 12** со списком пар полей «ключ - значение», описание которых приведено в разделе **ниже**;
3. После отправки транзакции нода получает её высоту в Targetnet;
4. Нода выполняет проверку высоты блокчейна в Targetnet каждые 30 секунд, пока высота не достигнет значения **высота созданной транзакции + height-above**.
5. При достижении этой высоты блокчейна Targetnet и подтверждения наличия первой транзакции в блокчейне, Targetnet нода создаёт вторую транзакцию с данными для анкоринга уже в приватном блокчейне.



21.2 Структура транзакции для анкоринга

Транзакция для отправки в Targetnet содержит следующие поля:

- `height` - высота сохраняемого блока из приватного блокчейна;
- `signature` - подпись сохраняемого блока из приватного блокчейна.

Транзакция, создаваемая в приватном блокчейне, содержит следующие поля:

- `height` - высота сохраняемого блока из приватного блокчейна;
- `signature` - подпись сохраняемого блока из приватного блокчейна;
- `targetnet-tx-id` - идентификатор транзакции для анкоринга в Targetnet;
- `targetnet-tx-timestamp` - дата и время создания транзакции для анкоринга в Targetnet.

21.3 Ошибки, возникающие в процессе анкоринга

Ошибки в анкоринге могут возникать на любом этапе. В случае возникновения ошибок в приватном блокчейне, публикуется [транзакция 12](#) с кодом и описанием ошибки. Транзакция об ошибке содержит следующие данные:

- `height` - высота сохраняемого блока из приватного блокчейна;
- `signature` - подпись сохраняемого блока из приватного блокчейна;
- `error-code` - код ошибки;
- `error-message` - описание ошибки.

Таблица 1: Типы ошибок при анкоринге

Код	Сообщение об ошибке	Возможная причина
0	Unknown error	При отправке транзакции в Targetnet произошла неизвестная ошибка
1	Fail to create data transaction for Targetnet	Создание транзакции для отправки в Targetnet завершилось ошибкой
2	Fail send transaction to Targetnet	Публикация транзакции в Targetnet завершилась ошибкой (это может br быть ошибка JSON-запроса)
3	Invalid http status of response from Targetnet br transaction broadcast	В результате публикации транзакции в Targetnet вернулся отличный br от 200 код
4	Fail to parse http body of response from Targetnet br transaction broadcast	В результате отправки транзакции в Targetnet вернулся br нераспознаваемый JSON-запрос
5	Targetnet return transaction with id='\$TargetnetTxId' br but it differ from transaction that we sent br id='\$sentTxId'	В результате отправки транзакции в Targetnet вернулся отличный от первой br транзакции идентификатор
6	Targetnet didn't respond on transaction info request	Targetnet не ответил на запрос об информации о транзакции
7	Fail to get current height in Targetnet	Не удалось получить текущую высоту в Targetnet
8	Anchoring transaction in Targetnet disappeared after br height rise enough	Анкоринг транзакция пропала из Targetnet после увеличения высоты br на значение height-above
9	Fail to create sidechain anchoring transaction	Не удалось опубликовать анкоринг транзакцию в приватном блокчейне
10	Anchored transaction in sidechain was changed during br Targetnet height arise await, looks like br a rollback has happened	Ожидалось подтверждение транзакции в Targetnet произошел откат br приватного блокчейна, идентификатор анкоринг транзакции был изменен

Смотрите также

Тонкая настройка платформы: настройка анкоринга

Механизм создания снимка данных

Механизм создания снимка данных - это вспомогательный механизм блокчейн-платформы, который позволяет сохранить данные работающей блокчейн-сети для последующего изменения параметров конфигурации сети и ее запуска с сохраненными данными.

Механизм создания снимка данных позволяет изменять параметры конфигурации блокчейн-сети без потери данных. Процесс изменения параметров конфигурации сети при помощи снимка данных называется **миграцией**.

Снимок данных включает следующие данные:

- стейты адресов сети: балансы, роли в сети, ключи;
- стейты смарт-контрактов, загруженных в сеть: данные, полученные в результате исполнения смарт-контрактов и прикрепленные к ним при помощи *транзакций 105*;
- данные майнеров прошедших раундов;
- данные *групп доступа к конфиденциальным данным*.

В снимке данных не сохраняется история транзакций, банов и блоков сети.

При выполнении миграции снимок данных становится начальным стейтом блокчейн-сети с новыми параметрами, сама сеть перезапускается с формированием нового генезис-блока.

Механизм создания снимка данных включается и настраивается в секции `node.consensual-snapshot` *конфигурационного файла ноды*.

22.1 Компоненты механизма создания снимка данных

SnapshotBroadcaster – компонент, предназначенный для рассылки сообщений `SnapshotNotification`, обработки запросов на создание снимка данных (`SnapshotRequest`) и последующей отдачи снимка данных. Так как снимки данных могут быть большими по размеру, в один момент компонентом обрабатывается не более 2 запросов.

SnapshotLoader – компонент, предназначенный для регистрации входящих сообщений `SnapshotNotification` на ноду, отправки запросов на получение снимка данных (`SnapshotRequest`) и его загрузки. Если на ноду приходит сообщение `SnapshotNotification`, то адрес, отправивший его, записывается в массив адресов, у которых есть снейпшот. Затем сообщение пересылается другим пирам ноды.

`SnapshotLoader` периодически проверяет массив адресов на наличие адреса со снимком данных. При наличии такого адреса и открытого сетевого канала с ним, адресу отправляется сообщение `SnapshotRequest` на загрузку снимка данных. Время ожидания ответа на сообщение составляет 10 секунд. Если нода, у которой есть снимок данных, не отвечает в течение этого времени, она исключается из массива адресов. В этом случае выбирается следующий доступный владелец снимка данных с отправкой ему сообщения `SnapshotRequest`.

В случае успешного получения снимка данных, он распаковывается, после чего запускается его верификация со стейтом ноды. В случае успешной верификации ноды, получившая снимок данных, рассылает своим пирам сообщения `SnapshotNotification`.

SnapshotApiRoute – контроллер REST API для работы со снимками данных.

22.2 Процесс создания и распространения снимка данных в работающей сети

1. Нода, назначенная для майнинга блока на высоте `snapshot-height`, также назначается создателем снимка данных. На высоте `snapshot-height + 1` стартует создание снимка данных в директорию `snapshot-directory`. На период создания снимка данных поступление новых транзакций в UTX-пул блокируется. После успешного создания снимка нода создает пустой `genesis`-блок с типом консенсуса новой сети (`consensus-type`) и сохраняет его в снимке данных.

2. При достижении высоты блокчейна `snapshot-height + wait-blocks-count` нода, создавшая снимок данных, архивирует его и распространяет своим пирам уведомления о готовности снимка (`SnapshotNotification`).

3. Ноды при получении `SnapshotNotification` иницируют запрос на получение снимка данных (`SnapshotRequest`). В случае истечения таймута по получению снимка данных или ошибки при его загрузке, нода выбирает другого пира и запрашивает снимок у него.

4. Каждая нода, получившая архив со снимком данных, сохраняет его на в `snapshot-directory`, распаковывает и проверяет корректность снимка: сверяет балансы адресов и ключи, проверяет целостность смарт-контрактов, состав и параметры групп доступа к конфиденциальным данным, роли участников. При успешной верификации снимка данных, нода рассылает своим пирам сообщени о наличии снимка (`SnapshotNotification`). После этого пиры ноды могут посылать ей запрос о загрузке снимка данных себе.

В результате, созданный снимок данных поступает всем нодам блокчейна, а верификация на уровне каждой ноды исключает возможность подмены данных в снимке.

После создания снимка вы можете запустить вашу ноду с измененными параметрами и созданным снимком. Подробнее см. статью [Запуск ноды с созданным снимком данных](#).

22.3 Методы REST API для работы со снимками данных

GET /snapshot/status – возвращает актуальный статус снимка данных на ноде:

- **Exists** – снимок данных существует / загружен;
- **NotExists** – снимок данных не существует / еще не загружен;
- **Failed** – ошибка распаковки или верификации снимка данных;
- **Verified** – снимок данных успешно верифицирован.

GET /snapshot/genesis-config – возвращает в ответе конфиг genesis-блока для новой сети;

POST /snapshot/swap-state – приостанавливает работу ноды и подменяет ее стейт на снимок данных. В запросе указывается параметр `backupOldState`, предназначенный для сохранения или удаления текущего стейта:

- **true** - сохранить текущий стейт в директорию ноды `PreSnapshotBackup`;
- **false** - удалить текущий стейт.

22.4 Сетевые сообщения

- **SnapshotNotification(sender)** – сообщение ноды о наличии у нее снимка данных, отправляется с публичным ключом ноды;
- **SnapshotRequest(sender)** – запрос ноды на получение снимка данных, также отправляется с публичным ключом ноды.

Смотрите также

Запуск ноды с созданным снимком данных *Тонкая настройка платформы: настройка механизма создания снимка данных*

Смарт-контракты

Смарт-контракт - это отдельное приложение, которое записывает в блокчейн свои входные данные и результаты исполнения заложенного алгоритма. Блокчейн-платформа Waves Enterprise поддерживает разработку и применение Тьюринг-полных смарт-контрактов для создания высокоуровневых бизнес-приложений.

Смарт-контракт может быть разработан на любом языке программирования и не имеет ограничений на реализацию заложенной логики. Для того, чтобы отделить запуск и исполнение смарт-контракта от самой блокчейн-платформы, смарт-контракт выполняется в контейнере Docker.

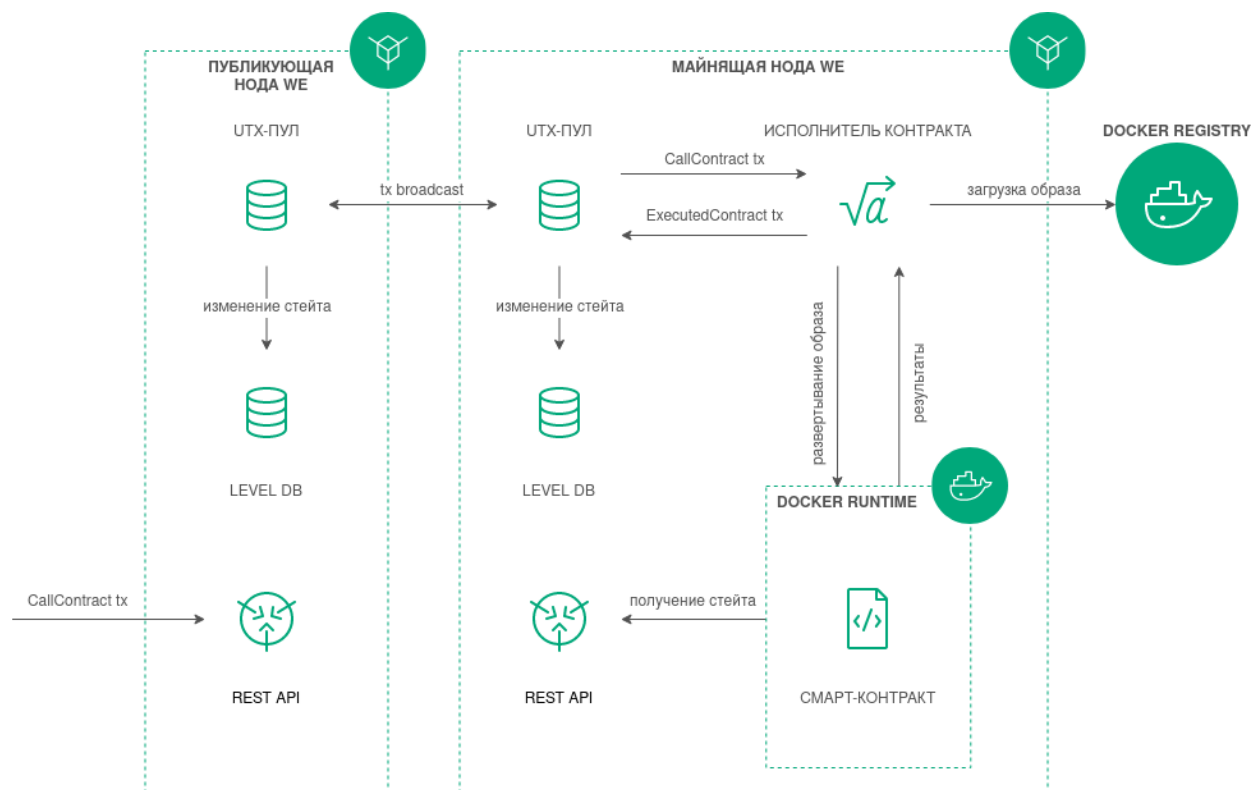
Когда смарт-контракт запускается в блокчейн сети, его код нельзя произвольно изменить, заменить или запретить его выполнение без вмешательства в работу всей сети. Это свойство позволяет обеспечить безопасность работы бизнес-приложений.

Доступ смарт-контракта к стеиту ноды для обмена данными осуществляется посредством gRPC и REST API-интерфейсов.

Создавать и вызывать смарт-контракты может любой участник сети. Разработанный смарт-контракт упаковывается в Docker-образ, который хранится в [открытом репозитории Waves Enterprise](#). Этот репозиторий основан на технологии [Docker Registry](#), к нему имеет доступ любой разработчик смарт-контрактов. Для добавления смарт-контракта в репозиторий свяжитесь со [службой технической поддержки Waves Enterprise](#). После одобрения вашей заявки смарт-контракт будет загружен в репозиторий, и вы сможете вызвать его при помощи клиентского приложения или запроса по REST API к вашей ноде.

Если вы планируете использовать смарт-контракты в собственной частной блокчейн-сети, вам потребуется создать собственный репозиторий для загрузки и вызова смарт-контрактов.

Ниже приведена общая схема работы смарт-контракта:



23.1 Создание и установка смарт-контракта

Практические указания по разработке логики смарт-контрактов, а также пример реализации на Python приведены в статье [Разработка и применение смарт-контрактов Docker](#).

Участник, разрабатывающий смарт-контракт, должен иметь роль **contract_developer** в сети. Участник с ролью разработчика смарт-контрактов получает возможность вызывать смарт-контракты, а также запрещать их исполнение и обновлять их код.

Создание смарт-контракта начинается с подготовки Docker-образа, который содержит готовый код смарт-контракта, сценарный файл *Dockerfile*, а также, в случае использования gRPC-интерфейса для обмена данными с нодой, необходимые **protobuf**-файлы.

Подготовленный образ собирается при помощи утилиты **build**, входящей в состав пакета Docker, после чего отправляется в репозиторий.

Для установки смарт-контракта и работы с ним необходима настройка секции *docker-engine* [конфигурационного файла ноды](#). Если ваша нода работает в Waves Enterprise Mainnet, на ней по умолчанию настроены установка смарт-контрактов из открытого репозитория и установлены рекомендованные параметры для обеспечения оптимального исполнения смарт-контрактов.

Установка смарт-контракта в блокчейне выполняется посредством транзакции [103 CreateContract Transaction](#), в теле которой указывается ссылка на образ смарт-контракта в репозитории. При работе со смарт-контрактами рекомендуется отправлять транзакции [последних версий](#).

При работе в частной сети транзакция 103 предусматривает загрузку Docker-образа контракта не только из репозитория, указанных в секции *docker-engine* конфигурационного файла ноды. Если вам необходимо загрузить смарт-контракт из репозитория, не внесенного в конфигурационный файл, укажите в поле *name* транзакции 103 полный адрес смарт-контракта в созданном вами репозитории. Пример заполнения полей транзакции 103 приведен в ее [описании](#).

После получения транзакции нода скачивает образ по ссылке, указанной в поле ``image``. Затем скачанный образ проверяется нодой и запускается в Docker-контейнере.

23.2 Запуск смарт-контракта и фиксация результатов исполнения

Запуск смарт-контракта инициируется участником сети при помощи транзакции [104 CallContract Transaction](#). В этой транзакции передается ID Docker-контейнера, в котором запускается смарт-контракт, а также его входные и выходные параметры в виде пар «ключ-значение». Контейнер запускается, если не был запущен ранее.

Результаты исполнения смарт-контракта фиксируются в его стеите при помощи транзакции [105 ExecutedContract Transaction](#).

23.3 Запрет запуска смарт-контракта

Для того, чтобы отключить запуск смарт-контракта в блокчейне, отправьте транзакцию [106 DisableContract Transaction](#) с указанием ID Docker-контейнера, в котором запускается смарт-контракт. Отправить эту транзакцию может только участник с ролью **contract_developer**.

После отключения смарт-контракт становится недоступен для запуска. Информация об отключенном смарт-контракте продолжает храниться в блокчейне и доступна для gRPC или REST API-методов.

23.4 Обновление смарт-контракта

Если вы изменили код вашего смарт-контракта, обновите его. Для этого заново загрузите смарт-контракт в репозиторий Waves Enterprise, отправив заявку на обновление смарт-контракта в службу технической поддержки.

Затем отправьте на ноду транзакцию [107 UpdateContract Transaction](#). Обновляемый смарт-контракт не должен быть отключен при помощи транзакции 106.

После обновления смарт-контракта ноды-майнеры блокчейна скачивают его и проверяют корректность исполнения. Затем информация об обновлении смарт-контракта вносится в его стейт при помощи транзакции 105, содержащей тело исполненной транзакции 107.

Подсказка: Изменять смарт-контракт может только участник, создавший транзакцию 103 для этого смарт-контракта и имеющий роль **contract_developer**.

23.5 Параллельное исполнение контрактов

Платформа Waves Enterprise позволяет запускать несколько смарт-контрактов одновременно. Такая опция поддерживается только смарт-контрактами, использующими gRPC-интерфейс для обмена данными.

Принцип параллельного исполнения нескольких смарт-контрактов:

1. Разработчик смарт-контракта указывает параметр `async-factor` в коде контракта. Данный параметр определяет допустимое количество одновременно выполняемых транзакций по смарт-контракту.
2. При старте контракт передает в ноду значение параметра `async-factor`.
3. Когда запускается исполнение контрактов, транзакции на исполнение смарт-контрактов передаются из UTX-пула в буфер для необработанных транзакций смарт-контрактов до его заполнения.
4. Транзакции, поступившие в буфер, разбиваются на группы по идентификаторам смарт-контрактов. В один момент времени допускается выполнение только одной группы, в которой параллельно выполняется `async-factor` транзакций.
5. При переходе очередного смарт-контракта к выполнению освобождается одна ячейка в буфере транзакций. При этом, ячейка блокируется при поступлении транзакции из UTX-пула. Таким образом, операции заполнения буфера и обработки вызовов контракта происходят параллельно, что позволяет избежать пауз при загрузке новых транзакций.

Значение параметра `async-factor` может быть как установлено заранее в диапазоне от 1 до 999, так и вычисляться динамически. Вы можете устанавливать фиксированное значение этого параметра как константу, однако рекомендуется устанавливать вычисляемое значение данного параметра. Например, контракт может запросить количество свободных ядер и передать это число в качестве значения параметра `async-factor`. Это число будет использоваться для параллельной обработки транзакций с контрактом.

Если параметр `async-factor` не будет определён, то по умолчанию все транзакции с контрактом будут обрабатываться последовательно.

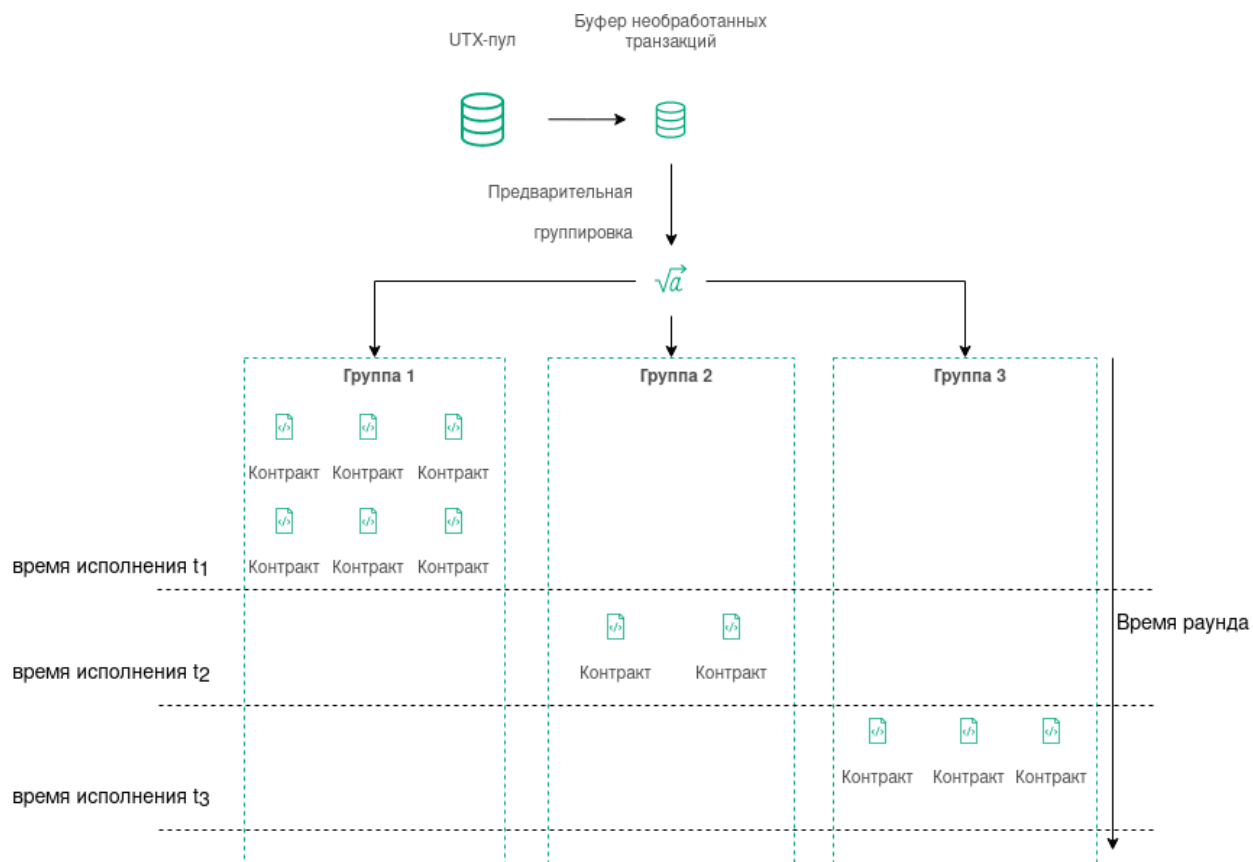
Параллельное исполнение нескольких смарт-контрактов иллюстрируется следующей схемой:

Логика кода смарт-контракта, как и язык программирования, выбранный для его разработки, должны учитывать специфику параллельного исполнения смарт-контрактов. Например, если смарт-контракт с функцией инкремента переменной при каждой транзакции вызова контракта будет исполняться параллельно, то результат получится некорректным, поскольку используется общий ключ авторизации во время каждого вызова контракта.

23.6 API-инструменты, доступные смарт-контракту

Для обмена данными между смарт-контрактом и нодой предусмотрены методы gRPC и REST API. При использовании этих методов вы можете осуществлять широкий спектр операций с блокчейном.

Подробнее:



23.6.1 Сервисы gRPC, используемые смарт-контрактом

Общие указания по применению gRPC при разработке смарт-контрактов рассмотрены в статье [Пример смарт-контракта с использованием gRPC](#).

Смарт-контрактам, использующим gRPC для обмена данными с нодой, доступны сервисы, названия protobuf-файлов которых начинаются с `contract`:

`contract_address_service.proto`

Набор методов, предназначенных для получения адресов участников из keystore ноды и получения данных, записанных на адресе.

`GetAddresses` - метод для получения всех адресов участников, ключевые пары которых хранятся в keystore ноды. В ответе метода массив строк `addresses`.

`GetAddressData` - метод для получения всех данных, записанных на аккаунт адресата при помощи транзакций [12](#). В запросе метода вводятся следующие параметры:

- `address` - адрес, данные которого необходимо вывести;
- `limit` - ограничение количества выводимых блоков данных;
- `offset` - количество блоков данных для пропуска в выводе.

В ответе метода выводится массив `DataEntry`, содержащий записанные данные адреса.

contract_contract_service.proto

Набор методов, предназначенный для работы со смарт-контрактами.

Connect - метод для подключения смарт-контракта к ноду. В запросе метода указываются следующие параметры:

- `connection_id` - идентификатор соединения смарт-контракта (см. [Авторизация смарт-контракта с gRPC](#));
- `async_factor` - максимальное количество одновременно исполняемых транзакций по смарт-контракту (см. [Параллельное исполнения смарт-контрактов](#)).

CommitExecutionSuccess - метод для получения результата успешного исполнения контракта по его идентификатору и отправки результатов на ноду.

CommitExecutionError - метод для получения ошибки, возникшей при исполнении смарт-контракта, по его идентификатору и отправки содержания ошибки на ноду.

GetContractKeys - метод для получения результата исполнения смарт-контракта по его идентификатору. В запросе метода указываются следующие данные:

- `contract_id` - идентификатор смарт-контракта;
- `limit` - ограничение количества выводимых блоков данных;
- `offset` - количество блоков данных для пропуска в выводе;
- `matches` - опциональный параметр для составления регулярного выражения, по которому фильтруются ключи.

В ответе метода выводится массив `Entries`, содержащий результаты исполнения смарт-контракта.

GetContractKey - метод для получения определенного результата исполнения смарт-контракта по его ключу. В запросе метода указываются следующие данные:

- `contract_id` - идентификатор смарт-контракта;
- `key` - запрашиваемый ключ.

В ответе метода выводится массив данных `entry`, содержащий результат исполнения смарт-контракта по запрошенному ключу.

contract_crypto_service.proto

Набор методов для шифрования и дешифровки. Подробнее см. статью [gRPC: реализация методов шифрования](#).

contract_permission_service.proto

Набор методов, предназначенный для получения информации о ролях участников.

GetPermissions - метод для получения списка всех ролей указанного адреса, действительных на момент отправки запроса. В запросе передаются следующие данные:

- `address` - запрашиваемый адрес;
- `timestamp` - временная метка в формате *Unix Timestamp* (в миллисекундах), на момент которой запрашиваются действующие роли.

В ответе метода выводится массив `roles`, содержащий роли запрашиваемого адреса, и указанная временная метка `timestamp`.

`GetPermissionsForAddresses` - метод для получения списка всех ролей нескольких адресов, действительных на момент отправки запроса. В запросе передаются следующие данные:

- `addresses` - массив строк с запрашиваемыми адресами;
- `timestamp` - временная метка в формате *Unix Timestamp* (в миллисекундах), на момент которой запрашиваются действующие роли.

В ответе метода выводится массив `address_to_roles`, содержащий роли для каждого запрашиваемого адреса, и указанная временная метка `timestamp`.

`contract_pki_service.proto`

Набор методов, предназначенный для формирования и проверки электронных подписей для данных. Подробнее см. статью [gRPC: формирование и проверка электронной подписи данных \(PKI\)](#).

`contract_privacy_service.proto`

Набор методов, предназначенный для получения информации о группах для обмена конфиденциальными данными и работы с конфиденциальными данными.

Подробнее об обмене конфиденциальными данными и группах доступа см. статью [Обмен конфиденциальными данными](#).

`GetPolicyRecipients` - метод для получения адресов участников группы доступа к конфиденциальным данным, идентификатор которой передается в запросе как `policy_id`. В ответе метода выводится массив строк `recipients`, содержащий адреса участников группы доступа.

`GetPolicyOwners` - метод для получения адресов владельцев группы для обмена конфиденциальными данными, идентификатор которой передается в запросе как `policy_id`. В ответе метода выводится массив строк `owners`, содержащий адреса владельцев группы доступа.

`GetPolicyItemData` - метод для получения пакета конфиденциальных данных по его идентификационному хэшу. Метод доступен, если вызывающий адрес принадлежит к группе доступа к конфиденциальным данным.

В запросе метода указываются идентификатор группы доступа `policy_id` и идентификационный хэш конфиденциальных данных `item_hash`. В ответе метода возвращается строка `data`, содержащая хэш пакета конфиденциальных данных.

`GetPolicyItemInfo` - метод для получения метаданных для пакета конфиденциальных данных по его идентификационному хэшу. Метод доступен, если вызывающий адрес принадлежит к группе доступа к конфиденциальным данным.

В запросе метода указываются идентификатор группы доступа `policy_id` и идентификационный хэш конфиденциальных данных `item_hash`. В ответе метода возвращаются следующие данные:

- `sender` - адрес отправителя конфиденциальных данных;
- `policy_id` - идентификатор группы доступа;
- `type` - тип конфиденциальных данных (`file`);
- `info` - массив данных о файле: * `filename` - имя файла; * `size` - размер файла; * `timestamp` - временная метка размещения файла в формате *Unix Timestamp* (в миллисекундах); * `author` - автор файла; * `comment` - опциональный комментарий к файлу;
- `hash` - идентификационный хэш конфиденциальных данных.

contract_transaction_service.proto

Набор методов, предназначенный для получения информации о транзакциях, отправленных в блокчейн. Подробнее см. статью *gRPC: получение информации о транзакции по ее ID*.

contract_util_service.proto

Файл содержит метод ``GetNodeTime``, предназначенный для получения текущего времени ноды. Метод возвращает текущее время ноды в двух форматах:

- system - системное время на машине ноды;
- ntp - сетевое время.

Смотрите также

Смарт-контракты

sc-rest

Разработка и применение смарт-контрактов

Общая настройка платформы: настройка исполнения смарт-контрактов

23.6.2 Методы REST API, доступные смарт-контрактам

Общие указания по разработке смарт-контракта с использованием REST API рассмотрены в статье *Пример смарт-контракта с использованием REST API*.

Смарт-контрактам, использующим REST API для обмена данными, доступны следующие методы:

Группа методов ``Addresses``:

- *GET /addresses*
- *GET /addresses/publicKey/{publicKey}*
- *GET /addresses/balance/{address}*
- *GET /addresses/data/{address}*
- *GET /addresses/data/{address}/{key}*

Группа методов ``Crypto``:

- *POST /crypto/encryptSeparate*
- *POST /crypto/encryptCommon*
- *POST /crypto/decrypt*

Группа методов ``Privacy``:

- *GET /privacy/{policy-id}/getData/{policy-item-hash}*
- *GET /privacy/{policy-id}/getInfo/{policy-item-hash}*
- *GET /privacy/{policy-id}/hashes*
- *GET /privacy/{policy-id}/recipients*

Группа методов ``Transactions``:

- *GET /transactions/info/{id}*
- *GET /transactions/address/{address}/limit/{limit}*

Группа методов ``Contracts``:

- *GET /internal/contracts/{contractId}/{key}*
- *GET /internal/contracts/executed-tx-for/{id}*
- *GET /internal/contracts/{contractId}*
- *GET /internal/contracts*

Для улучшения производительности, смарт-контракт может использовать методы группы Contracts по выделенному маршруту `/internal/contracts/`. Эндпоинты этого маршрута идентичны эндпоинтам обычных методов Contracts.

Группа методов ``PKI``:

- *PKI /verify*

Смотрите также

Смарт-контракты

Сервисы gRPC, используемые смарт-контрактом

Разработка и применение смарт-контрактов

Общая настройка платформы: настройка исполнения смарт-контрактов

Смотрите также

Разработка и применение смарт-контрактов

Общая настройка платформы: настройка исполнения смарт-контрактов

Транзакции блокчейн-платформы

Транзакция - это отдельная операция в блокчейне от имени участника, изменяющая стейт сети. Отправляя ту или иную транзакцию, участник отправляет в сеть запрос с набором данных, необходимых для соответствующего изменения стейта.

24.1 Подписание и отправка транзакций

Перед отправкой транзакции участник генерирует для нее цифровую подпись. Для этого он использует закрытый ключ своего аккаунта. Подписание транзакций может осуществляться тремя способами:

- посредством клиента блокчейн-платформы;
- при помощи метода REST API (см. *REST API: работа с транзакциями*);
- при помощи *JavaScript SDK*.

Подпись транзакции записывается в поле `proofs` при отправке транзакции в блокчейн. Как правило, в это поле записывается одна подпись участника, отправившего транзакцию. Однако поле поддерживает до 8 подписей: в случае подписания транзакции смарт-аккаунтом, при заполнении атомарной транзакции или при публикации смарт-контракта.

После подписания транзакция отправляется в блокчейн - это можно сделать как тремя способами, приведенными выше, так и при помощи gRPC-интерфейса (см. *gRPC: отправка транзакций в блокчейн*).

24.2 Обработка транзакций в блокчейне

Получив транзакцию, нода проверяет ее на валидность:

1. Соответствие временной метки (*timestamp*): временная метка транзакции должна отклоняться от временной метки текущего блока на более, чем на 2 часа назад или 1,5 часа вперед.
2. Тип и версия транзакции: активирована ли в блокчейне поддержка транзакций указанного типа и версии (см. [Активация функциональных возможностей](#)).
3. Соответствие полей транзакции заданному типу данных;
4. Проверка баланса отправителя: достаточно ли средств для оплаты комиссии;
5. Проверка подписи транзакции.

Если транзакция не проходит валидацию, нода отклоняет ее. В случае успешного прохождения проверок транзакция добавляется в пул неподтвержденных транзакций (UTX-пул), где ожидает следующего раунда майнинга для передачи в блокчейн. Вместе с передачей транзакции в UTX-пул нода рассылает ее другим нодам в сети.

Поскольку у каждого микроблока есть ограничение на количество поступающих транзакций, отдельная транзакция может попасть из UTX-пула в блокчейн далеко не сразу. Во время нахождения транзакции в UTX-пуле транзакция может стать невалидной. Например, ее временная метка перестала соответствовать параметрам временной метки текущего блока, либо транзакция, попавшая в блокчейн, уменьшила баланс отправителя, сделав его недостаточным для оплаты транзакции. В таком случае транзакция отклоняется и удаляется из UTX-пула.

После добавления в блок транзакция меняет стейт блокчейна. После этого транзакция считается выполненной.

Подробная информация о транзакциях блокчейн-платформы Waves Enterprise:

24.2.1 Описание транзакций

Блокчейн-платформа Waves Enterprise поддерживает 28 типов транзакций. Для каждой из них предусмотрен свой набор данных, отправляемых в блокчейн.

Формат ответов, возвращаемых нодой по gRPC-интерфейсу, определяется в protobuf-файлах (см. [Инструментарий gRPC](#)).

Подсказка: В случае если вы защитили ключевую пару вашей ноды паролем при [генерации аккаунта](#), укажите пароль от вашей ключевой пары в поле `password` транзакции.

1. Genesis Transaction

Первая транзакция нового блокчейна, которая осуществляет первоначальную привязку баланса к адресам созданных нод.

Подписание этой транзакции не требуется, поэтому выполняется только ее публикация.

Структура данных транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (1)
id	Byte	ID транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах)
signature	ByteStr	Подпись генезис-блока. Генерируется при старте блокчейна
recipient	ByteStr	Адрес получателя распределенных токенов
amount	Long	Сумма токенов
height	Int	Высота выполнения транзакции. Для первой транзакции - 1

3. Issue Transaction

Транзакция, инициирующая выпуск токенов в обращение.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (3)
version	Byte	Версия транзакции
name	Array[Byte]	Произвольное имя транзакции
quantity	Long	Количество выпускаемых токенов
description	Array[Byte]	Произвольное описание транзакции (в формате base58)
sender	ByteStr	Адрес отправителя транзакции распределенных токенов
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
decimals	Byte	Количество разрядов после запятой у используемого токена (WEST - 8)
reissuable	Boolean	Возможность довыпуска токенов
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции
version	Byte	Версия транзакции
assetId	Byte	ID выпускаемого токена
name	Array[Byte]	Произвольное имя транзакции
quantity	Long	Количество выпускаемых токенов
reissuable	Boolean	Возможность довыпуска токенов
decimals	Byte	Количество разрядов после запятой у используемого токена (WAVES - 8)
description	Array[Byte]	Произвольное описание транзакции
chainId	Byte	Идентификационный байт сети (Mainnet - 87 или V)
script	Array[Byte]	Скрипт для валидации транзакции - <i>опциональное поле</i>
height	Int	Высота выполнения транзакции

4. Transfer Transaction

Транзакция для перевода токенов с одного адреса на другой.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (4)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
recipient	ByteStr	Адрес получателя токенов
amount	Long	Сумма токенов
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
amount	Long	Сумма токенов
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
type	Byte	Номер транзакции (4)
version	Byte	Версия транзакции
attachment	Byte	Комментарий к транзакции (в формате base58) - <i>опциональное поле</i>
sender	ByteStr	Адрес отправителя транзакции
feeAssetId	Byte	ID токена комиссии - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID токена для перевода - <i>опциональное поле</i>
recipient	ByteStr	Адрес получателя токенов
id	Byte	ID транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>

5. Reissue Transaction

Транзакция для довыпуска токенов.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (5)
version	Byte	Версия транзакции
quantity	Long	Количество токенов для довыпуска
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
assetId	Byte	ID довыпускаемого токена - <i>опциональное поле</i>
reissuable	Boolean	Возможность довыпуска токенов
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
quantity	Long	Количество токенов для довыпуска
sender	ByteStr	Адрес отправителя транзакции
chainId	Byte	Идентификационный байт сети (Mainnet - 87 или V)
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID довыпускаемого токена - <i>опциональное поле</i>
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
id	Byte	ID транзакции
type	Byte	Номер транзакции (5)
version	Byte	Версия транзакции
reissuable	Boolean	Возможность довыпуска токенов
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
height	Int	Высота выполнения транзакции

6. Burn Transaction

Транзакция для сжигания токенов: уменьшает количество токенов на счету отправителя, тем самым снижая общее количество токенов в обращении. Сожженные токены невозможно восстановить.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (6)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
assetId	Byte	ID сжигаемого токена - <i>опциональное поле</i>
quantity	Long	Количество токенов для сжигания
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
attachment	Byte	Комментарий к транзакции (в формате base58) - <i>опциональное поле</i>

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
amount	Long	Количество токенов для сжигания
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID сжигаемого токена - <i>опциональное поле</i>
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
id	Byte	ID транзакции
type	Byte	Номер транзакции (6)
version	Byte	Версия транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
height	Int	Высота выполнения транзакции

8. Lease Transaction

Передача токенов в аренду другому адресу. Средства, переданные в аренду, начинают учитываться в генерирующем балансе получателя через 1000 блоков.

Передача токенов в лизинг может проводиться для повышения вероятности выбора ноды в качестве майнера следующего раунда. Как правило, в обмен на аренду токенов получатель делится вознаграждением, полученным за генерацию блока, с адресом, предоставившим токены в лизинг.

Токены, переданные в лизинг, остаются заблокированными на адресе отправителя. Отмена лизинга производится с помощью транзакции отмены лизинга.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (8)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
recipient	ByteStr	Адрес получателя токенов
amount	Long	Количество токенов для передачи в аренду
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
amount	Long	Количество токенов для передачи в аренду
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
recipient	ByteStr	Адрес получателя токенов
id	Byte	ID транзакции
type	Byte	Номер транзакции (8)
version	Byte	Версия транзакции
height	Int	Высота выполнения транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>

9. LeaseCancel Transaction

Отмена аренды токенов, переданных в транзакции с определенным ID. Структура lease данной транзакции не заполняется: нода автоматически заполняет ее при предоставлении данных о транзакции.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (9)
version	Byte	Версия транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
txId	Byte	ID транзакции аренды токенов

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
leaseId	Byte	ID транзакции аренды токенов
sender	ByteStr	Адрес отправителя транзакции
chainId	Byte	Идентификационный байт сети (Mainnet - 87 или V)
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
id	Byte	ID транзакции отмены аренды токенов
type	Byte	Номер транзакции (9)
version	Byte	Версия транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
height	Int	Высота выполнения транзакции

10. CreateAlias Transaction

Создание псевдонима для адреса отправителя. Псевдоним может использоваться для проведения транзакций в качестве идентификатора получателя.

Структура данных запроса на подписание транзакции:

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (10)
id	Byte	ID транзакции создания псевдонима
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
alias	Byte	Псевдоним
height	Byte	Высота выполнения транзакции

11. MassTransfer Transaction

Перевод токенов нескольким получателям (от 1 до 100 адресов). Комиссия за транзакцию зависит от количества задействованных адресов.

Структура данных запроса на подписание транзакции:

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
type	Byte	Номер транзакции (11)
transferCount	Byte	Количество адресов-получателей
version	Byte	Версия транзакции
totalAmount	Byte	Общая сумма токенов для перевода
attachment	Byte	Комментарий к транзакции (в формате base58) - <i>опциональное поле</i>
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
assetId	Byte	ID токена для перевода - <i>опциональное поле</i>
id	Byte	ID транзакции перевода токенов
transfers	List	Список получателей с полями recipient: ` и ``amount через запятую
transfers.recipient	ByteStr	Адрес получателя токенов
transfers.amount	Long	Количество токенов для передачи адресу
height	Byte	Высота выполнения транзакции

Пример заполнения поля transfers:

```
"transfers":
[
  { "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB", "amount": 100000 },
  { "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc", "amount": 100000 }
]
```

12. Data Transaction

Транзакция для добавления, изменения или удаления записей в хранилище данных адреса. В хранилище данных адреса представлены записи в формате «ключ:значение».

Размер хранилища данных адреса неограничен, однако при помощи одной транзакции данных можно внести до 100 новых пар «ключ:значение». Также байтовое представление транзакции после подписания не должно превышать **150 килобайт**.

Если автор данных (адрес в поле author) совпадает с отправителем транзакции (адрес в поле sender), при подписании транзакции не требуется указывать параметр senderPublicKey.

Структура данных запроса на подписание транзакции:

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublic	PublicKeyAcc	Открытый ключ отправителя транзакции
senderPublic	PublicKeyAcc	Открытый ключ автора данных
data	List	Список данных с полями ``key: `` ``type: `` и ``value: `` через запятую
data.key	Byte	Ключ записи
data.type	Byte	Тип данных записи. Возможные значения: binary bool integer string и null (удаление записи по ее ключу)
data.value	Byte	Значение записи
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
author	Byte	Адрес автора вносимых данных
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
id	Byte	ID транзакции с данными
type	Byte	Номер транзакции (12)
version	Byte	Версия транзакции
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>

Пример заполнения поля data:

```
"data": [
  {
    "key": "objectId",
    "type": "string",
    "value": "obj:123:1234"
  }, {...}
]
```

13. SetScript Transaction

Транзакция для привязки скрипта к аккаунту или удаления скрипта. Аккаунт с привязанным к нему скриптом называется *смарт-аккаунтом*.

Скрипт позволяет верифицировать транзакции, передаваемые от имени аккаунта, без использования механизма верификации транзакций блокчейна.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (13)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
name	Array[Byte]	Имя скрипта
script	Array[Byte]	Скомпилированный скрипт в формате base64 . Если вы оставите это поле пустым (<i>null</i>) - скрипт будет отвязан от аккаунта

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (13)
id	Byte	ID транзакции установки скрипта
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в WE Mainnet
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
chainId	Byte	Идентификационный байт сети (Mainnet - 87 или V)
version	Byte	Версия транзакции
script	Array[Byte]	Скомпилированный скрипт в формате base64 - <i>опциональное поле</i>
name	Array[Byte]	Имя скрипта
description	Byte	Комментарий к транзакции (в формате base58) - <i>опциональное поле</i>
height	Byte	Высота выполнения транзакции

14. Sponsorship Transaction

Транзакция, устанавливающая или отменяющая спонсирование.

Механизм спонсирования позволяет адресам выплачивать комиссии за транзакции вызова скрипта и транзакции перевода в спонсорском ассете, заменяющем WEST.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
assetId	Byte	ID спонсорского ассета (токена) - <i>опциональное поле</i>
fee	Long	Комиссия за транзакцию в WE Mainnet
isEnabled	Bool	Установка спонсирования (true) или его отмена (false)
type	Byte	Номер транзакции (14)
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (14)
id	Byte	ID транзакции спонсирования
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в WE Mainnet
assetId	Byte	ID спонсорского ассета (токена) - <i>опциональное поле</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
chainId	Byte	Идентификационный байт сети (Mainnet - 87 или V)
version	Byte	Версия транзакции
isEnabled	Bool	Установка спонсирования (true) или его отмена (false)
height	Byte	Высота выполнения транзакции

15. SetAssetScript Transaction

Транзакция для установки или удаления скрипта ассета для адреса. Скрипт ассета позволяет верифицировать транзакции с участием того или иного ассета (токена) без использования механизма верификации транзакций блокчейна.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (15)
version	Byte	Версия транзакции скрипта ассета
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
script	Array[Byte]	Скомпилированный скрипт в формате base64 - <i>опциональное поле</i>
assetId	Byte	ID спонсорского ассета (токена) - <i>опциональное поле</i>

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (15)
id	Byte	ID транзакции скрипта ассета
sender	ByteStr	Адрес отправителя транзакции
senderPubli	PublicKeyAc	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
chainId	Byte	Идентификационный байт сети (Mainnet - 87 или V)
assetId	Byte	ID спонсорского ассета (токена) - <i>опциональное поле</i>
script	Array[Byte]	Скомпилированный скрипт в формате base64 . Если вы оставите это поле пустым (<i>null</i>) - скрипт будет отвязан от аккаунта
height	Byte	Высота выполнения транзакции

101. GenesisPermission Transaction

Транзакция для назначения первого администратора сети, который раздает роли другим участникам.

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (101)
id	Byte	ID транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
signature	ByteStr	Подпись транзакции (в формате base58)
target	ByteStr	Адрес назначаемого первого администратора
role	String	Назначаемая роль (для администратора - permissioner)

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (101)
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
target	ByteStr	Адрес назначаемого первого администратора
role	String	Назначаемая роль (для администратора - permissioner)

102. Permission Transaction

Выдача или отзыв роли участника. Отправлять транзакцию 102 в блокчейн может только участник с ролью **permissioner**.

Возможные роли для указания в поле role:

- permissioner
- sender
- blacklister
- miner
- issuer
- contract_developer
- connection_manager
- banned

Описание ролей см. в статье [Роли участников](#).

Структура данных запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (102)
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
senderPublic	PublicKeyAccs	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
target	ByteStr	Адрес участника для назначения роли
opType	String	Тип операции: add - добавить роль; remove - отозвать роль
dueTimestamp	Long	Временная метка срока действия роли в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
senderPublic	PublicKeyAccs	Открытый ключ отправителя транзакции
role	String	Назначаемая роль (для администратора - permissioner)
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
opType	String	Тип операции: add - добавить роль; remove - отозвать роль
id	Byte	ID транзакции назначения или отмены роли
type	Byte	Номер транзакции (102)
dueTimestamp	Long	Временная метка срока действия роли в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
target	ByteStr	Адрес назначаемого первого администратора

103. CreateContract Transaction

Создание смарт-контракта. Байтовое представление этой транзакции после ее подписания не должно превышать **150 килобайт**.

Поле feeAssetId этой транзакции опционально и используется только для смарт-контрактов с поддержкой gRPC. Значение поля version для этого типа смарт-контрактов - **2**.

Подписание транзакции 103 может производиться только пользователем с ролью **contract_developer**.

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
image	Array[Byte]	Имя Docker-образа смарт-контракта
imageHash	Array[Byte]	Хэш Docker-образа смарт-контракта
contract	Array[Byte]	Имя смарт-контракта (при загрузке из предустановленного репозитория) или его полный адрес (если репозиторий смарт-контракта не указан в конфигурационном файле ноды)
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
params	List[DataE]	Входные и выходные данные смарт-контракта. Вносятся при помощи полей type value и key через запятую - <i>опциональное поле</i>
params.l	Byte	Ключ параметра
params.t	Byte	Тип данных параметра. Возможные значения: binary bool integer string
params.v	Byte	Значение параметра
type	Byte	Номер транзакции (103)
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (103)
id	Byte	ID транзакции создания контракта
sender	ByteStr	Адрес отправителя транзакции
senderPub	PublicKey	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List[ByteS]	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
image	Array[Byte]	Имя смарт-контракта (при загрузке из предустановленного репозитория) или его полный адрес (если репозиторий смарт-контракта не указан в конфигурационном файле ноды)
imageHash	Array[Byte]	Хэш Docker-образа смарт-контракта
contractId	Array[Byte]	Имя смарт-контракта
params	List[DataE]	Входные и выходные данные смарт-контракта. Вносятся при помощи полей type value и key через запятую - <i>опциональное поле</i>
params.k	Byte	Ключ параметра
params.t	Byte	Тип данных параметра. Возможные значения: binary bool integer string
params.v	Byte	Значение параметра
height	Byte	Высота выполнения транзакции

При работе в частной сети транзакция 103 предусматривает загрузку Docker-образа контракта не только из репозитория, указанных в секции docker-engine конфигурационного файла ноды. Если вам необходимо загрузить смарт-контракт из репозитория, не внесенного в конфигурационный файл, укажите в поле name транзакции полный адрес смарт-контракта в созданном вами репозитории.

Пример запроса на публикацию смарт-контракта из непредустановленного репозитория:

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTjtNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yecRFZm9iBlyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv",
    ↪ " ],
  "version": 1,
  "image": "customregistry.com:5000/stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "params": [],
  "height": 1619
}
```

104. CallContract Transaction

Вызов смарт-контракта на исполнение. Байтовое представление этой транзакции после ее подписания не должно превышать **150 килобайт**.

Подписание транзакции производится инициатором исполнения контракта.

В поле `contractVersion` транзакции указывается версия контракта:

- 1 - для нового контракта;
- 2 - для обновленного контракта.

Данное поле доступно только для второй версии транзакции: если в поле `version` транзакции создания смарт-контракта указано значение 2. Контракт обновляется при помощи транзакции [107](#). При создании контракта автоматически создается транзакция 104, вызывающая контракт для его проверки.

Если контракт не выполнен или выполнен с ошибкой, то транзакции 103 и 104 удаляются и не попадают в блок.

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
<code>contractId</code>	ByteStr	ID смарт-контракта
<code>fee</code>	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
<code>sender</code>	ByteStr	Адрес отправителя транзакции
<code>password</code>	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
<code>type</code>	Byte	Номер транзакции (104)
<code>params</code>	List[DataEnt	Входные и выходные данные смарт-контракта. Вносятся при помощи полей <code>type</code> <code>value</code> и <code>key</code> через запятую - <i>опциональное поле</i>
<code>params.key</code>	Byte	Ключ параметра
<code>params.type</code>	Byte	Тип данных параметра. Возможные значения: <code>binary</code> <code>bool</code> <code>integer</code> <code>string</code>
<code>params.value</code>	Byte	Значение параметра
<code>version</code>	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (104)
id	Byte	ID транзакции вызова контракта
sender	ByteStr	Адрес отправителя транзакции
senderPubli	PublicKeyAc	Открытый ключ отправителя транзакции
fee	Long	Комиссия за транзакцию в WE Mainnet
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
contractId	ByteStr	ID смарт-контракта
params	List[DataEnt	Входные и выходные данные смарт-контракта. Вносятся при помощи полей type value и key через запятую - <i>опциональное поле</i>
params.key	Byte	Ключ параметра
params.typ	Byte	Тип данных параметра. Возможные значения: binary bool integer string
params.val	Byte	Значение параметра

105. ExecutedContract Transaction

Запись результата исполнения смарт-контракта в его стейт. Байтовое представление этой транзакции после ее подписания не должно превышать **150 килобайт**.

Транзакция 105 содержит все поля (тело) транзакции 103 или 104 смарт-контракта, результат исполнения которого необходимо записать в его стейт (поле tx). Результат исполнения смарт-контракта вносится в его стейт из соответствующих параметров поля params транзакции 103 или 104.

Подписание транзакции производится нодой, формирующей блок после отправки запроса на публикацию транзакции.

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (105)
id	Byte	ID транзакции исполнения контракта
sender	ByteStr	Адрес отправителя транзакции
senderPublicK	PublicKeyAccou	Открытый ключ отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
fee	Long	Комиссия за транзакцию в WE Mainnet
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
tx	Array	Тело транзакции 103 или 104 исполняемого смарт-контракта
results	List[DataEntry[_	Список возможных результатов исполнения смарт-контракта
height	Byte	Высота выполнения транзакции

106. DisableContract Transaction

Отключение смарт-контракта. Байтовое представление этой транзакции после ее подписания не должно превышать **150 килобайт**.

Подписание транзакции 106 может производиться только пользователем с ролью **contract_developer**.

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
contractId	ByteStr	ID смарт-контракта
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
type	Byte	Номер транзакции (106)
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (106)
id	Byte	ID транзакции отключения контракта
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
contractId	ByteStr	ID смарт-контракта
height	Byte	Высота выполнения транзакции

107. UpdateContract Transaction

Обновление кода смарт-контракта. Байтовое представление этой транзакции после ее подписания не должно превышать **150 килобайт**.

Подписание транзакции 107, как и обновление смарт-контракта, может производиться только пользователем с ролью **contract_developer**.

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
image	Array[Byte]	Имя Docker-образа смарт-контракта
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
contractId	ByteStr	ID смарт-контракта
imageHash	Array[Byte]	Хэш Docker-образа смарт-контракта
type	Byte	Номер транзакции (107)
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

110. GenesisRegisterNode Transaction

Регистрация ноды в генезис-блоке при старте блокчейна.

Данная транзакция не требует подписания.

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (110)
id	Byte	ID транзакции регистрации ноды в генезис-блоке
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
signature	ByteStr	Подпись транзакции (в формате base58)
version	Byte	Версия транзакции
targetPubKey	Byte	Публичный ключ регистрируемой ноды
height	Byte	Высота выполнения транзакции

111. RegisterNode Transaction

Регистрация новой ноды в блокчейне или ее удаление.

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (111)
opType	String	Тип операции: add - добавить ноду; remove - удалить ноду
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
targetPubKey	Byte	Публичный ключ регистрируемой или удаляемой ноды
nodeName	Byte	Имя ноды
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (111)
id	Byte	ID транзакции регистрации ноды
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
version	Byte	Версия транзакции
targetPubKey	Byte	Публичный ключ регистрируемой или удаляемой ноды
nodeName	Byte	Имя ноды
opType	String	Тип операции: add - добавить ноду; remove - удалить ноду
height	Byte	Высота выполнения транзакции
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>

112. CreatePolicy Transaction

Создание группы доступа к конфиденциальным данным из указанных адресов.

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
policyName	String	Имя создаваемой группы доступа
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
recipients	Array[Byte]	Массив адресов участников группы доступа к конфиденциальным данным через запятую
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
descriptive	Array[Byte]	Произвольное описание транзакции (в формате base58)
owners	Array[Byte]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
type	Byte	Номер транзакции (112)
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (112)
id	Byte	ID транзакции создания группы доступа
sender	ByteStr	Адрес отправителя транзакции
senderPublic	PublicKeyAsc	Открытый ключ отправителя транзакции
policyName	String	Имя создаваемой группы доступа
recipients	Array[Byte]	Массив адресов участников группы доступа к конфиденциальным данным через запятую
owners	Array[Byte]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
fee	Long	Комиссия за транзакцию в WE Mainnet
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
description	Array[byte]	Произвольное описание транзакции (в формате base58)
version	Byte	Версия транзакции

113. UpdatePolicy Transaction

Изменение группы доступа к конфиденциальным данным.

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
policyId	String	Идентификатор создаваемой группы доступа
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
sender	ByteStr	Адрес отправителя транзакции
recipients	Array[Byte]	Массив адресов участников группы доступа к конфиденциальным данным через запятую
fee	Long	Комиссия за транзакцию в WE Mainnet
opType	String	Тип операции: add - добавить участников; remove - удалить участников
owners	Array[Byte]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
type	Byte	Номер транзакции (113)
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (113)
id	Byte	ID транзакции изменения группы доступа
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAcc	Открытый ключ отправителя транзакции
policyId	String	Идентификатор создаваемой группы доступа
recipients	Array[Byte]	Массив адресов для добавления или удаления участников группы доступа к конфиденциальным данным через запятую
owners	Array[Byte]	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
opType	String	Тип операции: add - добавить роль; remove - отозвать роль
description	Array[Byte]	Произвольное описание транзакции (в формате base58)
version	Byte	Версия транзакции

114. PolicyDataHash Transaction

Отправка хэша конфиденциальных данных в сеть. Эта транзакция создается автоматически при отправке в сеть конфиденциальных данных при помощи REST API метода POST /privacy/sendData.

Данная транзакция не требует подписания.

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (114)
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
policyId	String	Имя создаваемой группы доступа
dataHash	String	Хэш конфиденциальных данных для отправки
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
version	Byte	Версия транзакции

120. AtomicTransaction

Атомарная транзакция: помещает в контейнер другие транзакции для их атомарного выполнения. Транзакция этого типа выполняется полностью (ни одна из включенных транзакций не отклоняется) или не выполняется в принципе.

Поддерживается включение 2 и более транзакций следующих типов:

- *4. Transfer Transaction*, версия 3
- *102. Permission Transaction*, версия 2
- *103. CreateContract Transaction*, версия 3
- *104. CallContract Transaction*, версия 4
- *105. ExecutedContract Transaction*, версии 1 и 2
- *106. DisableContract Transaction*, версия 3
- *107. UpdateContract Transaction*, версия 3
- *112. CreatePolicy Transaction*, версия 3
- *113. UpdatePolicy Transaction*, версия 3
- *114. PolicyDataHash Transaction*, версия 3

Атомарная транзакция сама по себе не требует комиссии: общая сумма складывается из комиссий за транзакции, помещенные в атомарную транзакцию.

Подробнее об атомарных транзакциях: [Атомарные транзакции](#)

Структура данных для запроса на подписание транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (120)
sender	ByteStr	Адрес отправителя транзакции
transactions	Array	Полные тела включаемых транзакций
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
version	Byte	Версия транзакции

Структура данных для запроса на публикацию транзакции:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (114)
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
fee	Long	<i>Комиссия за транзакцию в WE Mainnet</i>
timestamp	Long	Временная метка в формате Unix Timestamp (в миллисекундах) - <i>опциональное поле</i>
proofs	List(ByteStr)	Массив подтверждений транзакции (в формате base58)
height	Byte	Высота выполнения транзакции
transactions	Array	Полные тела включаемых транзакций
miner	String	Публичный ключ майнера блока; заполняется в ходе раунда майнинга
password	String	Пароль от ключевой пары в keystore ноды - <i>опциональное поле</i>
version	Byte	Версия транзакции

Смотрите также*Описание транзакций**Комиссии в сети «Waves Enterprise Mainnet»***24.2.2 Актуальные версии транзакций**

При отправке транзакций в Waves Enterprise Mainnet или частную сеть рекомендуется использовать актуальные версии транзакций. Версия транзакции указывается в поле `version` при подписании и отправке.

Номер транзакции	Название транзакции	Актуальная версия
1	<i>Genesis Transaction</i>	Без версии
3	<i>Issue Transaction</i>	2
4	<i>Transfer Transaction</i>	3
5	<i>Reissue Transaction</i>	2
6	<i>Burn Transaction</i>	2
8	<i>Lease Transaction</i>	2
9	<i>Lease Cancel Transaction</i>	2
10	<i>Create Alias Transaction</i>	3
11	<i>Mass Transfer Transaction</i>	2
12	<i>Data Transaction</i>	2
13	<i>Set Script Transaction</i>	1
14	<i>Sponsorship Transaction</i>	1
15	<i>Set Asset Script Transaction</i>	1
101	<i>Genesis Permission Transaction</i>	Без версии
102	<i>Permission Transaction</i>	2
103	<i>Create Contract Transaction</i>	3
104	<i>Call Contract Transaction</i>	4
105	<i>Executed Contract Transaction</i>	2
106	<i>Disable Contract Transaction</i>	3
107	<i>Update Contract Transaction</i>	3
110	<i>Genesis Register Node Transaction</i>	1
111	<i>Register Node Transaction</i>	1
112	<i>Create Policy Transaction</i>	3
113	<i>Update Policy Transaction</i>	3
114	<i>Policy Data Hash Transaction</i>	3
120	<i>Atomic Transaction</i>	1

Смотрите также*Транзакции блокчейн-платформы**Описание транзакций**Комиссии в сети «Waves Enterprise Mainnet»*

Смотрите также

Описание транзакций

Актуальные версии транзакций

Комиссии в сети «Waves Enterprise Mainnet»

Атомарные транзакции

Платформа Waves Enterprise поддерживает выполнение атомарных операций. Атомарные операции состоят из нескольких действий, при невыполнении одного из действий все остальные также не выполняются. Для этого в системе существует транзакция [120 Atomic Transaction](#), представляющая собой контейнер, в который помещаются две и более подписанные транзакции.

Поддерживается включение 2 и более транзакций следующих типов:

- [4. Transfer Transaction](#), версия 3
- [102. Permission Transaction](#), версия 2
- [103. CreateContract Transaction](#), версия 3
- [104. CallContract Transaction](#), версия 4
- [105. ExecutedContract Transaction](#), версии 1 и 2
- [106. DisableContract Transaction](#), версия 3
- [107. UpdateContract Transaction](#), версия 3
- [112. CreatePolicy Transaction](#), версия 3
- [113. UpdatePolicy Transaction](#), версия 3
- [114. PolicyDataHash Transaction](#), версия 3

Ключевым отличием версий транзакций, которые поддерживаются атомарной транзакцией, является наличие поля-метки *atomicBadge*. Это поле содержит доверенный адрес отправителя транзакции *trustedSender* для добавления в контейнер транзакции [120](#). Если адрес отправителя не указывается, отправителем становится адрес, с которого в блокчейн отправляется транзакция [120](#).

25.1 Обработка атомарной транзакции

Атомарная транзакция имеет две подписи. Первым транзакцию подписывает отправитель для её успешной отправки в сеть. Вторая подпись формируется майнером и необходима для добавления транзакции в блокчейн. При добавлении атомарной транзакции в UTX-пул, проверяется её подпись, а также подписи всех транзакций, входящих в контейнер.

Валидация таких транзакций выполняется по следующим правилам:

- Количество транзакций должно быть больше одной.
- Все транзакции должны иметь разные идентификаторы.
- Список транзакций должен содержать только поддерживаемые типы транзакций.

Вкладывать одну атомарную транзакцию в другую не допускается.

Внутри атомарной транзакции, отправляемой в UTX пул, не должно быть исполненных (*executed*) транзакций, и поле `miner` должно быть пустым. Это поле заполняется при передаче атомарной транзакции в блок.

Внутри атомарной транзакции, попавшей в блок, не должно быть исполняемых (*executable*) транзакций.

После исполнения атомарной транзакции в блок попадает её «копия», сформированная по следующим правилам:

- Поле `miner` не участвует в формировании подписи транзакции и заполняется публичным ключом майнера блока.
- Майнером блока формируется массив `proofs`, источником которого служат идентификаторы транзакций, входящих в атомарную транзакцию. При включении в блок, атомарная транзакция имеет 2 подписи – подпись исходной транзакции и подпись майнера.
- Если в списке присутствуют *executable* транзакции, они заменяются на *executed* транзакции. При валидации атомарной транзакции в составе блока проверяются обе подписи.

25.2 Создание атомарной транзакции

Для создания атомарной транзакции необходим доступ к [REST API](#) ноды.

1. Пользователь подбирает из списка поддерживаемых транзакций те транзакции, которые должны выполняться как атомарная операция.
2. Затем корректно заполняет поля всех транзакций и подписывает их.
3. Далее пользователь заполняет поле `transactions` атомарной транзакции данными подписанных, но не отправленных в блокчейн транзакций.
4. После внесения всех данных о транзакциях пользователь подписывает и отправляет в блокчейн готовую атомарную транзакцию.

Структуры данных для подписания и отправки атомарной транзакции приведены в [списке транзакций](#).

Внимание: Если вы создаёте атомарную транзакцию с включением [114](#) транзакции, то при её подписании установите значение параметра `broadcast = false`.

Смотрите также

Описание транзакций

Комиссии в сети «Waves Enterprise Mainnet»

 Алгоритмы консенсуса

Блокчейн — это децентрализованная система, в которой нет единого регулятора процессов. Децентрализация исключает возможность коррупции внутри системы, однако создает сложности с итоговым принятием решений и организацией работы.

Эти задачи решает **консенсус** - алгоритм, согласующий работу участников блокчейна путем того или иного метода голосования. Голосование в блокчейне всегда происходит в пользу большинства - интересы меньшинства не учитываются, а принятое решение становится обязательным к исполнению для всех участников. Однако, несмотря на это, голосование гарантирует достижение соглашения, которое принесет пользу всей сети.

Блокчейн-платформа Waves Enterprise поддерживает три алгоритма консенсуса:

26.1 Алгоритм консенсуса LPoS

Алгоритм консенсуса основан на доказательстве доли владения (**Proof of Stake**) с правом аренды (**Leased Proof of Stake**). При использовании создание блока не требует энергозатратных вычислений, задача майнера — создание цифровой подписи блока.

26.1.1 Proof of Stake

Механизм распределения прав создания блоков основан на количестве токенов на счету пользователя. Чем больше у пользователя токенов, тем выше вероятность, что он сможет создать блок.

В консенсусе PoS право выпуска блока определяется псевдослучайным образом: следующий майнер вычисляется на основе данных предыдущего майнера и балансов всех пользователей сети. Это возможно, благодаря детерминированному вычислению генерирующей подписи блока, которая получается путем хэширования генерирующей подписи текущего блока и публичного ключа аккаунта. Первые 8 байт полученного хеша преобразуются в число X_n , которое указывает на следующего майнера. Время генерации блока для аккаунта i , рассчитывается следующим образом:

$$T_i = T_{min} + C_1 \log\left(1 - C_2 \frac{\log \frac{X_n}{X_{max}}}{b_i A_n}\right)$$

где:

- b_i - доля баланса участника от общего баланса сети;
- A_n - BaseTarget, адаптивный коэффициент, регулирующий среднее время выпуска блока;
- X_n - указатель на следующего майнера;
- T_{min} - константа, определяющая минимальный временной интервал между блоками (**5 секунд**);
- C_1 - константа, корректирующая форму распределения интервала между блоками (**70**);
- C_2 - константа, равная и предназначенная для регулировки значения BaseTarget (**5E17**).

Из приведенной формулы легко убедиться, что вероятность выбора участника зависит от доли активов участника в системе: больше доля — выше шанс. Минимальное количество токенов на балансе для майнинга — **50 000 WEST**.

BaseTarget — параметр, удерживающий время генерации блоков в заданном диапазоне. Этот параметр может быть определен как сложность вычислений, и рассчитывается следующим образом:

$$(S > R_{max} \rightarrow T_b = T_p + \max(1, \frac{T_p}{100})) \wedge (S < R_{min} \wedge \wedge T_b > 1 \rightarrow T_b = T_p - \max(1, \frac{T_p}{100}))$$

где

- R_{max} = максимальное уменьшение сложности, когда время генерации блока в сети превышает 40 секунд (**90**);
- R_{min} = минимальное увеличение сложности, когда время генерации блока в сети составляет менее 40 секунд (**30**);
- S - среднее время генерации как минимум для трех последних блоков;
- T_p - предыдущее значение baseTarget;
- T_b - вычисленное значение baseTarget.

Подробное описание технических особенностей и доработок классического алгоритма PoS для блокчейн-платформы Waves Enterprise приведено в [этой статье](#).

Преимущества перед PoW

Отсутствие сложных вычислений позволяет сетям на основе PoS снизить требования к аппаратному обеспечению участников системы, что снижает стоимость разворачивания частных сетей. Также в таких сетях дополнительная эмиссия, которая в системах на основе алгоритма консенсуса PoW (Proof of Work) используется для вознаграждения майнеров за нахождение нового блока. В PoS-системах майнер получает вознаграждение в виде комиссий за транзакции, которые попали в его блок.

26.1.2 Leased Proof of Stake

Для пользователя, который обладает балансом, недостаточным для эффективного майнинга, есть возможность передать свой баланс в аренду другим участникам и получать долю дохода от майнинга. Так вы можете увеличить вероятность выбора майнера и получать часть комиссии за транзакции, которые этот майнер поместил в свои блоки. Лизинг является полностью безопасной операцией. Токены не покидают ваш счет, вы передаете право учитывать свой баланс при розыгрыше права майнинга другому участнику сети.

Смотрите также*Общая настройка платформы: настройка консенсуса**Алгоритмы консенсуса**Алгоритм консенсуса PoA**Алгоритм консенсуса CFT*

26.2 Алгоритм консенсуса PoA

В приватном блокчейне не всегда нужны токены - например, блокчейн может быть использован для хранения хэшей документов, которыми обмениваются организации. В таком случае, при отсутствии токенов и комиссий с транзакций, решение на базе алгоритма консенсуса PoS является избыточным. Для реализации таких решений в блокчейн-платформе Waves Enterprise предусмотрен альтернативный алгоритм консенсуса — PoA (Proof of Authority). Разрешение на майнинг в алгоритме PoA выдаётся централизованно. Это упрощает принятие решений по сравнению с алгоритмом PoS. Модель Proof of Authority основана на ограниченном количестве валидаторов блока, что делает её масштабируемой. Блоки и транзакции проверяются заранее утвержденными участниками, которые выступают в качестве модераторов системы.

26.2.1 Описание алгоритма

На базе приведенных ниже параметров формируется алгоритм определения майнера текущего блока. Параметры консенсуса указываются в блоке consensus конфигурационного файла ноды.

- t - длительность раунда в секундах (параметр конфигурационного файла ноды: round-duration).
- t_s - длительность периода синхронизации, вычисляется как $t \cdot 0,1$, но не более 30 секунд (параметр конфигурационного файла ноды: sync-duration).
- N_{ban} - количество пропущенных подряд раундов для выдачи бана майнеру (параметр конфигурационного файла ноды: warnings-for-ban);
- P_{ban} - доля максимального количества забаненных майнеров, в процентах от 0 до 100 (параметр конфигурационного файла ноды: max-bans-percentage);
- t_{ban} - продолжительность бана майнера в блоках (параметр конфигурационного файла ноды: ban-duration-blocks).
- T_0 - unix time создания genesis блока.
- T_H - unix time создания блока H — ключевой блок для NG.
- r - номер раунда, вычисляется как $(T_{\text{Current}} - T_0) \div (t + t_s)$.
- A_r - лидер раунда r , имеющий право на создание ключевых блоков и микроблоков для NG в раунде r .
- H — высота цепочки, на которой создается ключевой блок и микроблоки для NG. Право на выпуск блока на высоте H имеет лидер раунда A_r .
- M_H - майнер, выпустивший блок на высоте H .
- Q_H - очередь активных на высоте H майнеров.

Очередь Q_H формируется из адресов, имеющих роль майнера. При этом учитывается, что роль майнера у выбираемых адресов не должна быть отозвана до высоты H , и не истекает до момента времени T_H .

Очередь сортируется по временной метке транзакции предоставления прав на майнинг – узел, которому права были предоставлены раньше, помещается ближе к началу очереди. Для согласованной сети эта очередь будет одинакова на каждой ноде.

Новый блок создается в течение каждого раунда r . Раунд длится t секунд. После каждого раунда отводится t_s секунд на синхронизацию данных в сети. В период синхронизации микроблоки и ключевые блоки не формируются. Для каждого раунда существует единственный лидер A_r , который имеет право создать блок в этом раунде. Определение лидера может производиться на каждом узле сети с одинаковым результатом.

Определение лидера раунда осуществляется следующим образом:

1. Определяется майнер M_{H-1} , который создал предыдущий ключевой блок на высоте $H-1$.
2. Вычисляется очередь Q_H активных майнеров.
3. Из очереди исключаются неактивные майнеры (подробнее в пункте [Исключение неактивных майнеров](#)).
4. Если майнер блока $H-1$ (M_{H-1}) есть в очереди Q_H , лидером A_r становится следующий по очереди майнер.
5. Если майнера блока $H-1$ (M_{H-1}), нет в очереди Q_H , лидером A_r становится майнер, идущий в очереди за майнером блока $H-2$ (M_{H-2}), и так далее.
6. Если ни одного из майнеров блоков ($H-1..1$) нет в очереди, лидером становится первый майнер очереди.

Данный алгоритм позволяет детерминировано вычислить и проверить майнера, который должен был создать каждый блок цепочки, за счет возможности вычислить список авторизованных майнеров на каждый момент времени. Если блок не был создан назначенным лидером в отведенное время, блоки в текущем раунде не создаются (производится пропуск раунда). Лидеры, пропускающие создание блоков, временно исключаются из очереди по алгоритму, описанному в пункте [Исключение неактивных майнеров](#).

Валидным считается блок, выпущенный лидером A_r с временем блока T_H из полуинтервала $(T_0 + (r-1)*(t+t_s); T_0 + (r-1)*(t+t_s) + t]$. Блок, созданный майнером не в свою очередь или с превышением отводимого времени, не считается валидным. После раунда длительностью t сеть синхронизирует данные в течение t_s . Лидер раунда A_r получает время t_s для того, чтобы распространить валидный блок по сети. Если каким-либо узлом сети за время t_s не был получен блок от лидера A_r , этот узел признает раунд пропущенным и ожидает новый блок H в следующем раунде $r+1$, от следующего лидера A_{r+1} .

Параметры консенсуса t и t_s задаются в [конфигурационном файле ноды](#). При этом, параметр t должен совпадать у всех участников сети, иначе произойдет форк сети.

26.2.2 Синхронизация времени между узлами сети

Каждый узел сети должен синхронизировать время приложения с доверенным NTP-сервером в начале каждого раунда. Адрес и порт сервера указывается в конфигурационном файле ноды. Сервер должен быть доступен каждой ноде сети.

26.2.3 Исключение неактивных майнеров

Если каким-либо майнером N_{ban} раз подряд было пропущено создание блока, этот майнер исключается из очереди на t_{ban} последующих блоков (параметр `ban-duration-blocks` в конфигурационном файле ноды). Исключение выполняется каждым узлом самостоятельно на основании вычисляемой очереди Q_H и информации о блоке H и майнере M_H . С помощью параметра P_{ban} задается максимально допустимая доля исключенных майнеров в сети относительно всех активных майнеров в любой момент времени. Если при достижении N_{ban} пропусков раунда известно, что максимальная доля исключенных майнеров P_{ban} достигнута, то исключение очередного майнера не производится.

26.2.4 Мониторинг

Мониторинг консенсуса PoA помогает выявлять факты создания и распространения невалидных блоков, а также пропуски очереди майнерами. Дальнейшие действия по выявлению и устранению неисправностей, а также блокировке вредоносных узлов выполняются администраторами сети.

В целях мониторинга процесса формирования блоков для алгоритма PoA в InfluxDB размещаются следующие данные:

- Активный список майнеров, отсортированный в порядке предоставления прав на майнинг.
- Плановая временная метка раунда.
- Фактическая временная метка раунда.
- Текущий майнер.

26.2.5 Изменение параметров консенсуса

Изменение параметров консенсуса (время раунда и периода синхронизации) выполняется на основании данных конфигурационного файла ноды на высоте `from-height`. Если какая-либо из нод сети не укажет новые параметры, произойдет форк блокчейна.

Пример конфигурации:

```
// specifying inside of the blockchain parameter
consensus {
  type = poa
  sync-duration = 10s
  round-duration = 60s
  ban-duration-blocks = 100
  changes = [
    {
      from-height = 18345
      sync-duration = 5s
      round-duration = 60s
    },
    {
      from-height = 25000
      sync-duration = 10s
      round-duration = 30s
    }
  ]
}
```

Смотрите также

Общая настройка платформы: настройка консенсуса

Алгоритмы консенсуса

Алгоритм консенсуса LPoS

Алгоритм консенсуса CFT

26.3 Алгоритм консенсуса CFT

При интенсивном обмене информацией в корпоративном блокчейне важна согласованность действий между элементами сети, формирующими единый блокчейн. И чем больше участников обмена - тем больше вероятность возникновения какой-либо ошибки: отказ оборудования одного из участников, проблемы с сетью, и так далее. Это может привести к возникновению форков основного блокчейна и, как следствие, откату блока, который, казалось бы, уже сформирован и включен в блокчейн. В такой ситуации откаченные блоки начинают майниться заново и на некоторое время становятся недоступны в блокчейне - а это, в свою очередь, может повлиять на использующие блокчейн бизнес-процессы. Алгоритм консенсуса CFT (Crash Fault Tolerance) исключает возникновение таких ситуаций.

26.3.1 Описание алгоритма

В основе реализации CFT лежит алгоритм консенсуса *PoA* с добавленной фазой голосования **валидаторов раунда майнинга** - участников сети, автоматически назначаемых алгоритмом консенсуса. Такой подход гарантирует следующее:

- блок известен более чем половине участников сети и завалидирован ими;
- блок не будет откачен и попадет в цепочку;
- в блокчейне не произойдет образования параллельной цепочки.

Все это достигается посредством финализации выпущенного блока. Сама финализация блока опирается на консенсус большинства валидаторов раунда (50% + 1), в соответствии с которым и принимается решение о добавлении блока в сеть. В случае отсутствия такого большинства майнинг останавливается до восстановления связности сети.

Консенсус CFT, также как и PoA, зависит от текущего времени, а время начала и окончания каждого раунда рассчитывается на основе временной метки genesis-блока. Основные параметры, на основе которых формируется алгоритм для определения майнера текущего блока, также идентичны параметрам алгоритма PoA (см. раздел Описание алгоритма). Для валидации блоков в блок consensus конфигурационного файла ноды были добавлены три новых параметра:

- **max-validators** – лимит валидаторов, участвующих в голосовании в конкретном раунде.
- **finalization-timeout** – время, в течение которого майнер ждет финализации последнего блока в цепочке. По прошествии этого времени майнер вернет транзакции обратно в UTX-пул и начнет майнить раунд заново.
- **full-vote-set-timeout** - опциональный параметр, определяющий, сколько времени после окончания раунда (параметр конфигурационного файла ноды: round-duration) майнер ожидает полный набор голосов от всех валидаторов.

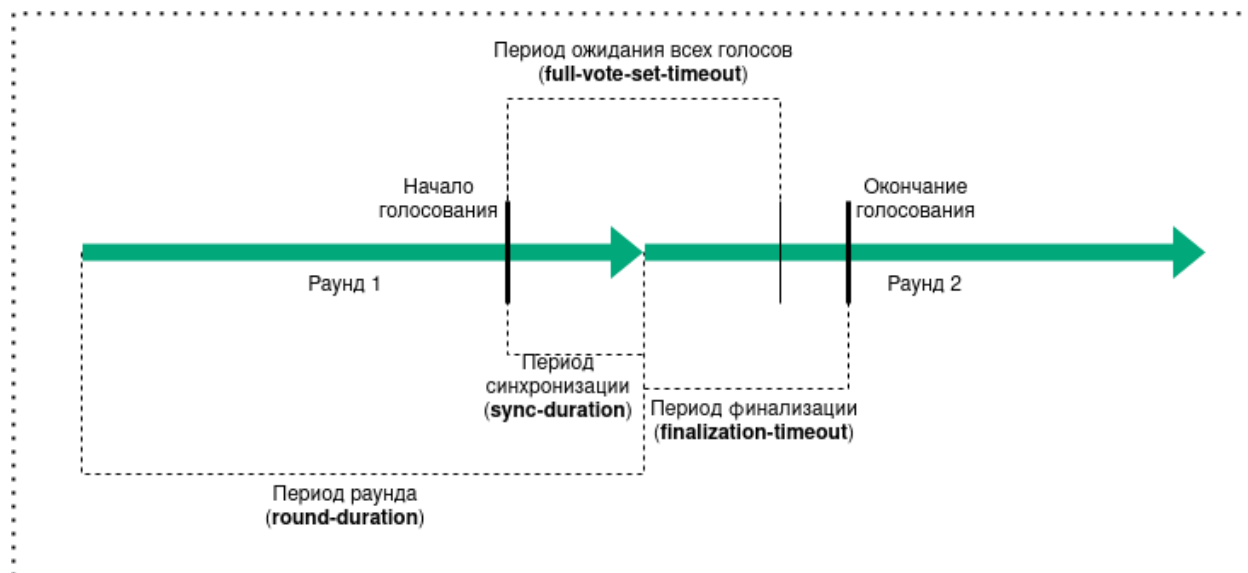
Для приведенного ниже описания функциональности CFT используются следующие обозначения:

- **t** - длительность раунда в секундах (параметр конфигурационного файла ноды: round-duration).

- t_{start} - время начала раунда.
- t_{sync} - время синхронизации блокчейна ($t_{start} + t$).
- t_{end} - время окончания раунда.
- t_{fin} - время ожидания финализации последнего блока майнером (параметр конфигурационного файла ноды: `finalization-timeout`).
- V_{max} - лимит валидаторов, участвующих в голосовании (параметр конфигурационного файла ноды: `max-validator`).

26.3.2 Голосование

Общая схема раунда при использовании CFT выглядит следующим образом:



Голосование проводится каждый раунд, в нем могут участвовать ноды с ролью майнера. Голосование начинается при наступлении t_{sync} и заканчивается при достижении $t_{end} + t_{fin}$. В рамках каждого временного интервала, выделенного для голосования, проводится *голосование валидаторов* и *голосование майнера текущего раунда*. Каждый валидатор раунда может отправить несколько голосов, в то время как майнер - единожды проголосовать за свой последний микроблок.

Для голосования используется сущность голоса, которая включает следующие параметры:

- **senderPublicKey** – публичный ключ валидатора, который сформировал голос;
- **blockVotingHash** – хэш жидкого блока с голосами, который подтвердил валидатор;
- **signature** – подпись голоса, сформированная валидатором.

Определение валидаторов раунда и их голосование

Для определения валидаторов, которые могут голосовать в конкретный раунд, используется настраиваемый параметр ноды `max-validators` (V_{\max}). Если число активных майнеров за вычетом майнера текущего раунда не превышает V_{\max} , то в голосовании может участвовать каждый из них. В противном случае для определения валидаторов применяется алгоритм псевдослучайного выбора, который позволяет исключить влияние конкретного майнера на выборку голосующих.

Голосование валидатора запускается при двух условиях:

- очередная попытка голосования попадает во временной интервал, необходимый для голосования;
- адрес текущей ноды является одним из определенных для голосования валидаторов раунда.

После окончания голосования валидаторов раунда запускается голосование майнера.

Голосование майнера текущего раунда

Голосование майнера запускается при двух условиях:

- очередная попытка голосования попадает во временной интервал, необходимый для голосования;
- адрес текущей ноды является майнером раунда.

Голос считается валидным в случае, если его выпустил адрес, который входит в число валидаторов текущего раунда и при этом имеет корректную подпись. Как только майнер набирает необходимое число голосов, выполняется проверка временного интервала голосования. Затем выпускается финализирующий микроблок с набранными голосами. Блок, имеющий голоса, считается финализированным.

26.3.3 Особенности майнинга

Основные правила майнинга в рамках консенсуса CFT идентичны правилам консенсуса PoA. При этом был введен дополнительный механизм, обеспечивающий отказоустойчивость консенсуса.

При использовании консенсуса CFT очередная попытка майнинга считается неудачной, если последний полученный блок не был финализирован - иными словами, к стеиту не применен микроблок с набранными валидными голосами. При этом, если попытки майнинга выходят за временные рамки $t_{\text{start}} + t_{\text{fin}}$, нода принимает решение вернуть все транзакции из последнего блока обратно в UTX-пул, после чего раунд начинает майниться заново.

Чтобы избежать возможного возврата транзакций в UTX-пул, рекомендуется работать не с последним (жидким) блоком блокчейна, а с финализированным - подтвержденным валидаторами сети.

26.3.4 Выбор канала для синхронизации

Для алгоритмов консенсуса PoS и PoA используется модуль, выбирающий для синхронизации наиболее сильную цепочку на основе сравнения данных задействованных нод. В CFT применяется иной механизм выбора, также увеличивающий отказоустойчивость системы: выбирается случайный канал из активных на момент синхронизации. Перечень активных каналов постоянно обновляется в ходе работы системы, а для равномерного распределения нагрузки на сеть время синхронизации с конкретным каналом ограничено.

26.3.5 Изменение параметров консенсуса

Как и в случае с алгоритмами консенсуса PoS и PoA, параметры консенсуса настраиваются на основе конфигурационного файла ноды. Ниже приведен пример конфигурации:

```
consensus {  
  type: cft  
  warnings-for-ban: 3  
  ban-duration-blocks: 15  
  max-bans-percentage: 33  
  round-duration: 7s  
  sync-duration: 2s  
  max-validators: 7  
  finalization-timeout: 4s  
  full-vote-set-timeout: 4s  
}
```

Рекомендации по конфигурации CFT см. в разделе [Общая настройка платформы: настройка консенсуса](#).

Смотрите также

[Общая настройка платформы: настройка консенсуса](#)

[Алгоритмы консенсуса](#)

[Алгоритм консенсуса LPoS](#)

[Алгоритм консенсуса PoA](#)

Сеть Waves Enterprise Mainnet применяет алгоритм **Leased Proof of Stake** для принятия решений. Для реализации алгоритма предусмотрен технический токен **WEST**, который служит не только доказательством права ноды на майнинг, но и финансовой мотивацией участников.

Сайдчейны и частные сети на основе блокчейн-платформы Waves Enterprise могут применять любой из трех алгоритмов консенсуса, в зависимости от потребностей проекта. Алгоритм консенсуса частной сети настраивается в [конфигурационном файле ноды](#).

Смотрите также

[Общая настройка платформы: настройка консенсуса](#)

Платформа Waves Enterprise предоставляет возможность выбора используемого криптографического алгоритма, в зависимости от особенностей проекта.

27.1 Хеширование

Операции хеширования выполняются функциями **Blake2b256** и **Кеccak256** последовательно, либо функцией «**Стрибог**» в соответствии с **ГОСТ Р 34.11-2012** «Информационная технология. Криптографическая защита информации. Функция хеширования». Размер блока выходных данных: **256 бит**.

27.2 Электронная подпись

Алгоритмы генерации ключей, формирования и проверки электронной подписи реализованы на базе эллиптической кривой **Curve25519** (ED25519 с ключами X25519), либо в соответствии с **ГОСТ Р 34.10-2012** «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи».

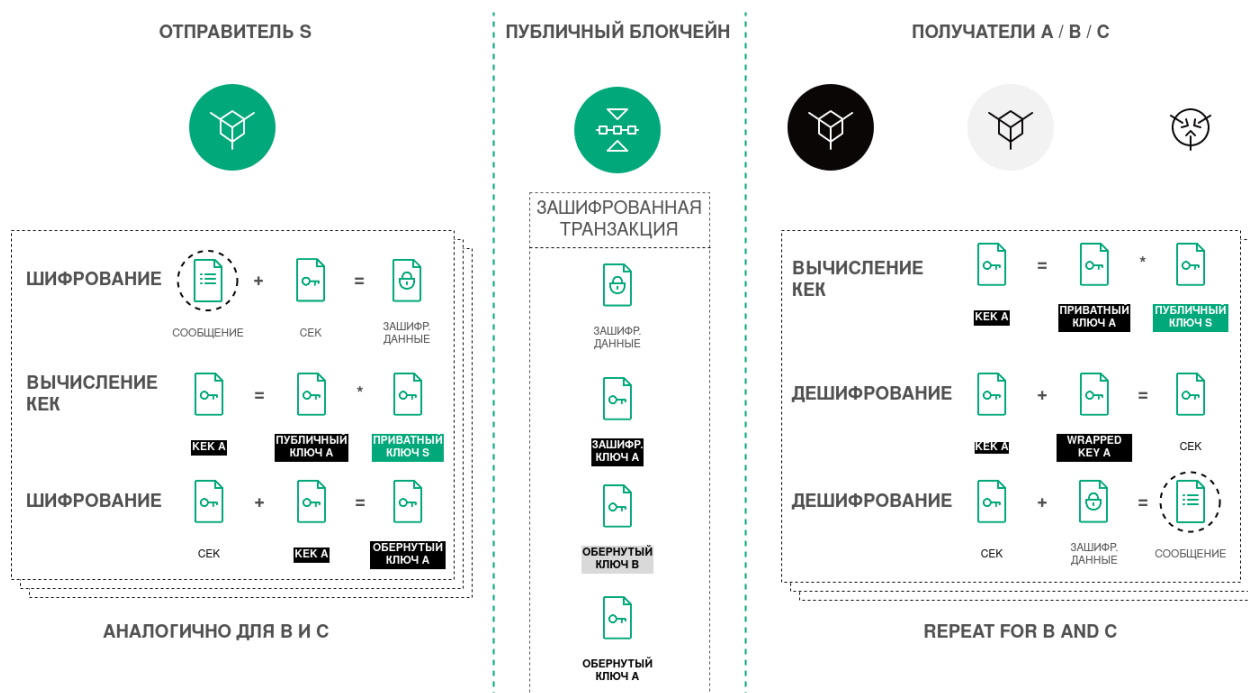
Подробнее о генерации и проверке электронной подписи с использованием *gRPC* и *REST API*-методов.

27.3 Шифрование данных

Блокчейн-платформа поддерживает шифрование данных при помощи сессионных ключей на базе **протокола Диффи-Хелмана**. Операция применяется для шифрования любого вида текстовой информации, например, данных смарт-контрактов, которые не должны быть доступны для других участников блокчейна. Шифрование может выполняться как индивидуально для каждого получателя, с формированием уникального экземпляра шифротекста, так и с формированием единого шифротекста для группы получателей.

Используемые алгоритмы для симметричного шифрования соответствуют стандарту **AES**, либо **ГОСТ Р 34.12-2015** «Информационная технология. Криптографическая защита информации. Блочные шифры».

Ниже приведено схематичное описание процедуры шифрования текстовых данных на базе протокола Диффи-Хелмана:



Для операций шифрования/расшифрования данных применяются симметричные ключи СЕК и КЕК.

СЕК (Content Encryption Key, ключ шифрования данных) используется для шифрования текстовых данных. **КЕК (Key Encryption Key, ключ шифрования ключа)** используется для шифрования СЕК.

Ключ СЕК формируется блокчейн-узлом случайным образом с применением соответствующих алгоритмов хеширования. Ключ КЕК формируется нодой на базе алгоритма Диффи-Хелмана с использованием публичных и частных ключей отправителя и получателей и применяется для шифрования ключа СЕК.

Симметричный ключ СЕК недоступен для прочтения и не отображается в процессе шифрования. От отправителя к получателю он передается в зашифрованном виде (**wrappedKey**) по открытым каналам связи вместе с зашифрованным сообщением. Примеры таких каналов в блокчейн-сети - транзакция [112](#) для записи данных в блокчейн или стейт смарт-контракта. Ключ КЕК от отправителя к получателю не передается: он восстанавливается получателем на основе своего закрытого ключа и известного публичного ключа отправителя (алгоритм **Diffie-Hellman key exchange**).

Подробнее о [применении шифрования](#) с использованием gRPC-методов.

Для шифрования сетевых каналов используется алгоритм **TLS v. 1.2 (AES-256 CBC SHA)**. Также при выборе ГОСТ-шифрования протокол устанавливает шифрованное TLS-like соединение с применением алгоритма шифрования «Кузнечик».

Смотрите также

gRPC: реализация методов шифрования

REST API: реализация методов шифрования

Роли участников

Блокчейн-платформа Waves Enterprise реализует закрытую (permissioned) модель блокчейна, доступ к которому имеют только авторизованные участники.

Также в платформе реализована ролевая модель, которая позволяет разграничить полномочия участников сети. Управление ролями осуществляется посредством транзакции *102 Permission Transaction*.

28.1 Описание ролей

permissioner

Участник с ролью `permissioner` является администратором сети и имеет право назначать или удалять любые роли участников сети. Первый участник с ролью `permissioner` назначается при запуске блокчейн-сети.

sender

Участник с ролью `sender` имеет право отправлять транзакции в сеть.

Использование этой роли включается и отключается при помощи параметра `sender-role-enabled`, который находится в блоке `genesis` *конфигурационного файла ноды*.

blacklister

Участник с ролью `blacklister` имеет право временно или постоянно ограничивать действия других участников сети, присваивая им роль `banned`. Для этого `blacklister` отправляет транзакцию 102 с соответствующими параметрами.

miner

Участник с ролью `miner` может быть выбран в качестве майнера очередного раунда и имеет право формировать блоки.

issuer

Участник с ролью `issuer` имеет право на выпуск, перевыпуск и сжигание токенов.

contract_developer

Участник с ролью `contract_developer` имеет право на установку смарт-контрактов в блокчейне.

Подробнее о смарт-контрактах и применении этой роли: [Смарт-контракты](#).

connection-manager

Участник с ролью `connection-manager` имеет право на подключение или отключение нод от сети. Как правило, роль `connection-manager` присваивается администратору сети.

Подробнее о подключении и отключении нод: [Подключение и удаление нод](#).

banned

Участник с ролью `banned` временно или постоянно ограничивается в действиях в сети. Адрес с ролью `banned` попадает в черный список нод (**blacklist**).

28.2 Управление ролями

Изменить список полномочий может только нода с ролью `permissioner`. Для добавления или удаления ролей используется транзакция 102 Permission Transaction.

Процесс назначения и удаления ролей описан в статье [Управление ролями участников](#).

При отправке транзакции 102 нода выполняет следующие проверки:

1. Отправитель транзакции 102 не находится в списке **blacklist**.
2. У адреса отправителя есть роль `permissioner`.
3. Роль `permissioner` у адреса отправителя активна в момент отправки транзакции.
4. Роль, указанная в транзакции 102, неактивна в случае её добавления адресу, и активна в случае её удаления у адреса.

Удаление или назначение ролей участникам производится при попадании соответствующих транзакций 102 в блокчейн. Роли могут быть произвольно скомбинированы для любого адреса, отдельные роли могут быть отозваны в любой момент.

Смотрите также

[REST API: информация о ролях участников](#)

[Описание транзакций](#)

Клиент [Waves Enterprise](#) — это веб-приложение для управления блокчейном Waves Enterprise, предназначенное для работы в [публичной сети](#) Waves Enterprise.

Клиент состоит из следующих разделов:

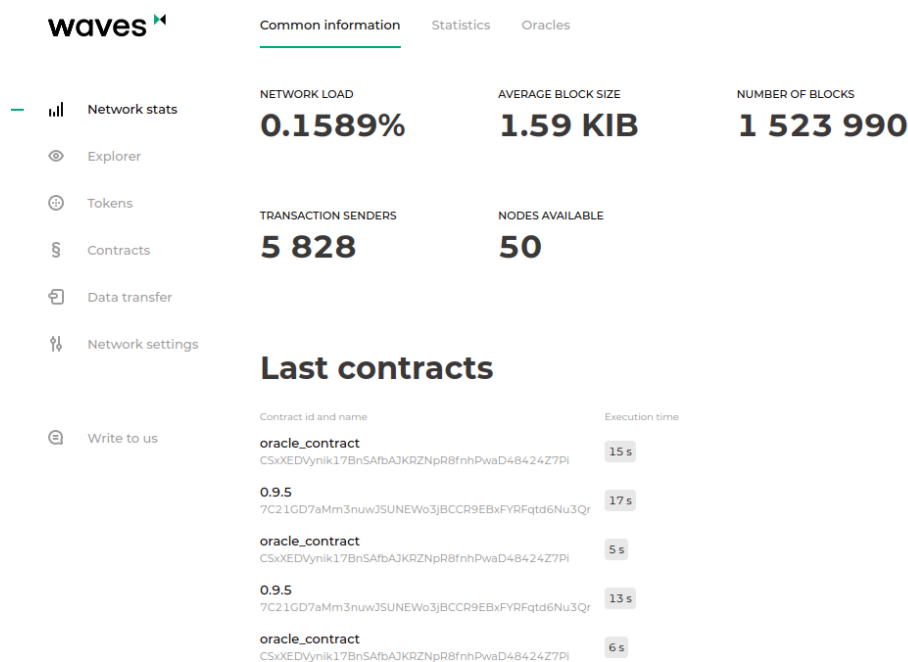
- **Статистика сети** - общая информация о текущем состоянии Waves Enterprise Mainnet, статистические данные сети и [оракулов](#);
- **Транзакции** - информация о транзакциях, отправленных в сеть;
- **Токены** - выпуск, перевод и передача токенов в аренду;
- **Контракты** - публикация смарт-контрактов;
- **Передача данных** - отправка транзакций с данными и файлов, работа с группами доступа к конфиденциальным данным;
- **Настройки сети** - информация о нодах сети, регистрация новой ноды и расчет лизинга;
- **Написать нам** - форма обратной связи со службой технической поддержки Waves Enterprise.

Настройки вашего профиля вы можете найти в верхнем правом углу интерфейса, нажав на иконку с электронным адресом.

При нажатии на кнопку [Адрес](#) в правом верхнем углу вы увидите форму выбора адреса ноды или создания нового блокчейн-адреса для привязки профиля к нему. После выбора адреса вам будет доступна информация о вашем аккаунте (публичный и приватный ключи, seed-фраза, текущий баланс).

Также в окне «Адрес» вы можете управлять разрешениями для других участников, при наличии у вашего адреса роли [permissioner](#).

29.1 Статистика сети



На вкладке **Общая информация** представлено текущее состояние Waves Enterprise Mainnet:

- нагрузка на сеть;
- средний размер одного блока;
- общее количество блоков в сети;
- количество нод и отправителей транзакций;
- последние вызванные смарт-контракты.

На вкладке **Статистика** приведены основные метрики блокчейна:

- Количество транзакций в сети;
- Количество транзакций вызова смарт-контрактов;
- Количество транзакций, предназначенных для операций с токенами;
- Количество всех остальных транзакций;
- Список последних вызванных смарт-контрактов;
- Список используемых образов смарт-контрактов;
- Количество активных адресов;
- Топ-10 адресов по количеству отправленных транзакций;
- Топ-10 нод-майнеров;
- Статистика оборота токенов.

На вкладке **Оракулы** приведены данные, полученные из внешних источников.

Относительный график отображает зависимость колебаний стоимости WEST и традиционных активов по следующим парам:

- WEST - USDN;
- BTC - USD;
- BRENT - USD;
- Золото - USD;

График стоимости WEST отображает стоимость WEST в других криптовалютах:

- WEST - USDN;
- WEST - WAVES;
- WEST - BTC.

29.2 Транзакции

The screenshot displays the Waves Explorer interface. On the left is a sidebar with navigation links: Network stats, Explorer, Tokens, Contracts, Data transfer, Network settings, and Write to us. The main content area features a search bar with the placeholder text 'Enter your query, search will be performed in filtered list'. Below the search bar, it indicates 'Total records: 1000+'. A table lists transactions, all of which are 'Call: oracle_contract' transactions from the author '3Ng3g5ZSpbZQZa2P87AUzW...'. Each transaction is of type 'Docker' and occurred on '23 April 2021, 14:33'.

Icon	Call: oracle_contract	Author	Type	Timestamp
</>	Call: oracle_contract	3Ng3g5ZSpbZQZa2P87AUzW...	Docker	23 April 2021, 14:33
</>	Call: oracle_contract	3Ng3g5ZSpbZQZa2P87AUzW...	Docker	23 April 2021, 14:33
</>	Call: oracle_contract	3Ng3g5ZSpbZQZa2P87AUzW...	Docker	23 April 2021, 14:33
</>	Call: oracle_contract	3Ng3g5ZSpbZQZa2P87AUzW...	Docker	23 April 2021, 14:33
</>	Call: oracle_contract	3Ng3g5ZSpbZQZa2P87AUzW...	Docker	23 April 2021, 14:33
</>	Call: oracle_contract	3Ng3g5ZSpbZQZa2P87AUzW...	Docker	23 April 2021, 14:33

Раздел «Транзакции» содержит информацию о транзакциях в блокчейне. Для поиска доступна фильтрация по периоду публикации, а также по следующим категориям:

- по участникам;
- по транзакциям с данными;
- по идентификаторам транзакций;
- по именам смарт-контрактов;

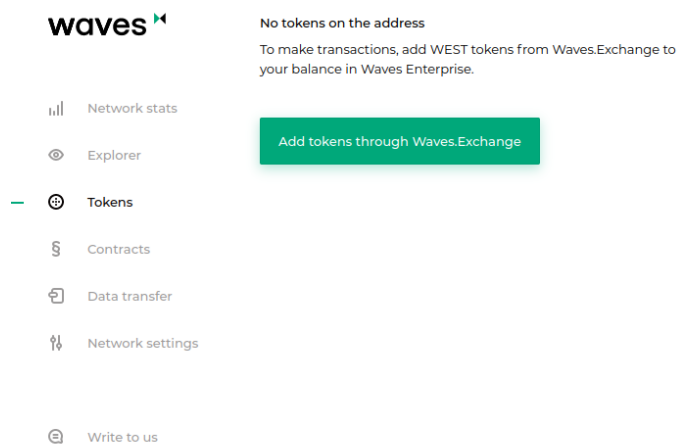
- по подписям транзакций;
- по номеру блока, содержащего транзакции.

Также доступны дополнительные фильтры, отображающие только транзакции выбранной категории:

- *Токены* - операции с токенами;
- *Контракты* - операции со смарт-контрактами;
- *Транзакции с данными*;
- *Разрешения* - управление ролями участников;
- *Группы* - управление группами доступа к конфиденциальным данным;
- *Неподтвержденные транзакции* - содержимое UTX-пула.

Строка **Пользователи**, расположенная в конце списка фильтров, перенаправит вас на список пользователей сети с доступным фильтром по выданным ролям.

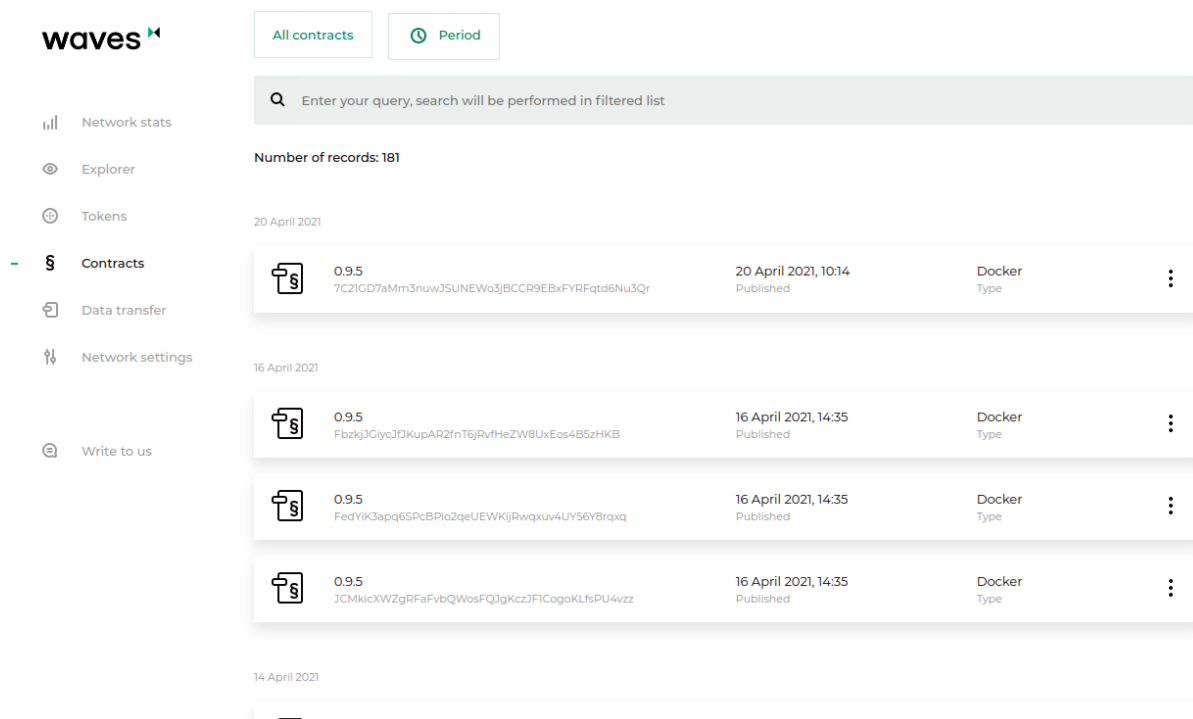
29.3 Токены



При отсутствии токенов на вашем адресе, на вкладке отображается кнопка, перенаправляющая на биржу Waves Exchange.

При наличии токенов на адресе, на вкладке отображается текущий баланс, а также кнопки для перевода токенов другим участникам сети, передачи токенов в аренду и выпуска токенов. Выпуск токенов требует роли *issuer*.

29.4 Контракты



Раздел «Контракты» содержит информацию о существующих контрактах в блокчейне, а также позволяет запускать выбранные контракты. Для поиска смарт-контрактов доступна фильтрация в поисковой строке по параметрам транзакций:

- по авторам и отправителям транзакций;
- по подписям;
- по идентификатору смарт-контракта;
- по имени смарт-контракта;
- по имени образа.

Также доступны дополнительные фильтры, отображающие смарт-контракты выбранной категории:

- *Мои контракты* - смарт-контракты, разработанные и загруженные в блокчейн вами;
- *Все контракты* - значение по умолчанию;
- *Отключенные контракты* - смарт-контракты, запуск которых был запрещен их разработчиками при помощи транзакции [106](#).

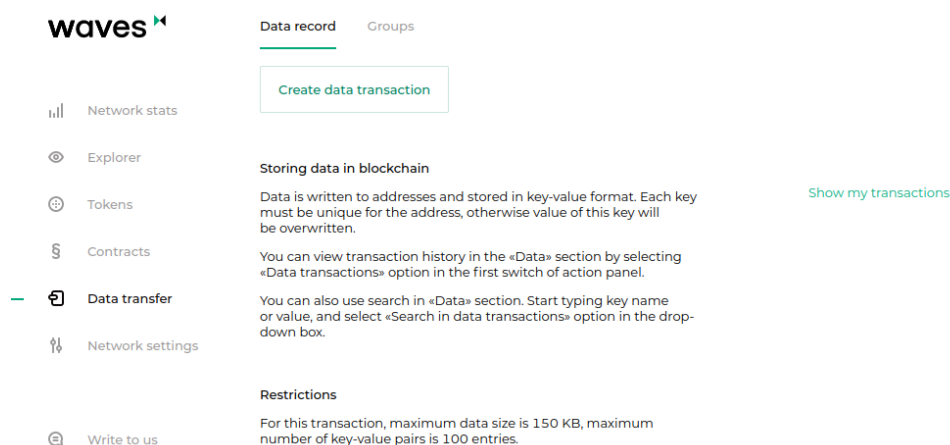
В карточке каждого смарт-контракта вы увидите четыре вкладки:

- **Информация** - адрес автора, имя образа, контрольная сумма, версия и дата создания смарт-контракта;
- **Данные** - результат последнего вызова смарт-контракта;
- **Вызов** - на этой вкладке вы можете вызвать смарт-контракт при достаточном балансе на адресе;

- **История версий** - таблица с именами образов, датами создания и контрольными суммами для каждой версии смарт-контракта.

Подробнее о смарт-контрактах блокчейн-платформы Waves Enterprise см. статью [Смарт-контракты](#).

29.5 Передача данных



Раздел «Передача данных» позволяет подписывать и отправлять в блокчейн транзакции с данными. Также в этом разделе вы сможете создавать группы доступа к конфиденциальным данным и отправлять в них транзакции с конфиденциальными данными.

Подробнее об обмене конфиденциальными данными см. статью [Обмен конфиденциальными данными](#).

На вкладке **Запись** вы можете создать и отправить транзакцию с данными. Для этого заполните поля для необходимых пар «ключ-значение» и выберите адрес получателя.

На вкладке **Группы** вы можете создавать и редактировать группы доступа к конфиденциальным данным и отправлять в них транзакции с данными. Также на вкладке отображается информация о группах доступа, в которых вы состоите.

29.6 Настройки сети

waves

Nodes Calculation of lease payouts

Create request All nodes

Enter address or public key

Total records: 56

Node ID	Public Key	Status / Date
node-1190421	3NiIN5UvMB3WpgHPwtBTi4L3BT7EpVg7XDM	Active 19 April, 2021, 08:44
node-1310321	3P15gvVT3K7grgNC1EM2zvQsdjPSPQB3an	Active 31 March, 2021, 12:44
ETWYQQskAYEhxbqtmAWXC6cKt7xR8jJo	3Nym9N8TcTEU4wt3SzBf3wE75VNvtgF7RA	Active 14 December, 2020, 09:22
yar_test_node	3P1igxLg8jKf85LSMSMR5uKzffajzMGtIU	Active 11 December, 2020, 13:30
node-1081220	3NhwpgkjLThZiVGTmMm93UwgNWHv2s9C1	Active 08 December, 2020, 13:09
node-2071220	3NuvaLo7XXFat5x7wbvJFw9w93hWWcrKsr	Active 07 December, 2020, 15:41

Раздел «Настройки сети» предназначен для просмотра информации о нодах, зарегистрированных в сети, а также расчета выплат лизинга.

На вкладке **Ноды** вам доступна информация о каждой ноде блокчейн-сети:

- Публичный ключ;
- Адрес;
- Статус;
- Адрес отправителя последней транзакции, изменившей стейт ноды;
- Дата последнего изменения стейта;
- Наличие ролей **miner** или **banned**;
- Участие ноды в группах по обмену конфиденциальными данными с информацией о них.

Доступен поиск и фильтрация нод по следующим параметрам:

- Название;
- Адрес;
- Публичный ключ;
- Активность в сети.

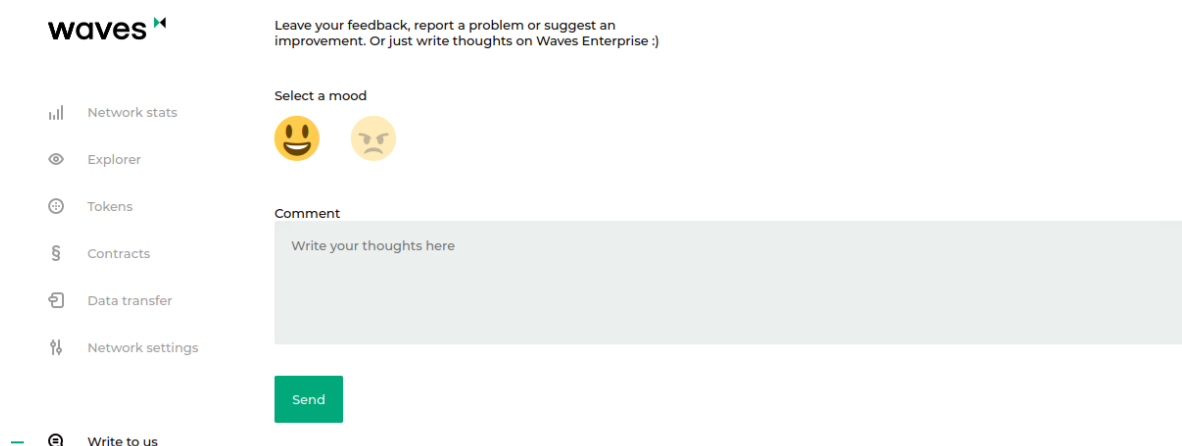
Также вы можете оставить заявку на подключение новой ноды к сети, нажав на кнопку Создать заявку.

На вкладке **Расчет выплат лизинга** приведена форма для проведения расчета.

Алгоритм расчёта суммы лизинга следующий:

1. На начало периода запрашивается генерирующий баланс с ноды, адрес которой был указан в качестве лизингового пула;
2. Выполняется расчёт суммы лизинга с учётом прибыли майнера (майнер должен получить 40% за свой блок и 60% за предыдущий блок);
3. Сумма делится на каждого участника пула пропорционально сумме средств в лизинге и генерирующего баланса ноды на указанной высоте;
4. Рассчитанная сумма лизинга умножается на процент прибыли;
5. Пересчитывается генерирующий баланс ноды для новой высоты с учётом новых и отменённых лизингов.

29.7 Написать нам



The screenshot shows a web interface for providing feedback. On the left is a sidebar with the 'waves' logo and a list of navigation items: 'Network stats', 'Explorer', 'Tokens', 'Contracts', 'Data transfer', 'Network settings', and 'Write to us' (which is highlighted with a green line). The main content area has the heading 'Leave your feedback, report a problem or suggest an improvement. Or just write thoughts on Waves Enterprise :)'. Below this is a 'Select a mood' section with two emoji buttons: a happy face and an angry face. Underneath is a 'Comment' section with a large text input area containing the placeholder text 'Write your thoughts here'. At the bottom of the form is a green 'Send' button.

В разделе вы можете оставить любой комментарий или сообщение для службы технической поддержки Waves Enterprise.

Смотрите также

Привязка блокчейн-платформы к клиенту

Генераторы - это набор утилит, входящий в комплект поставки блокчейн-платформы Waves Enterprise. Генераторы выполнены в виде пакетного файла **generator-x.x.x.jar**, где x.x.x - номер релиза блокчейн-платформы.

Генераторы для каждой версии доступны в [официальном репозитории Waves Enterprise в GitHub](#).

Для работы с генераторами вам следует установить [Java Runtime Environment](#) для вашей операционной системы. Все утилиты пакета запускаются из терминала или командной строки с аргументами, соответствующими названию генераторов.

В набор генераторов входят следующие утилиты:

- **AccountsGeneratorApp** - утилита для создания аккаунта на ноде;
- **GenesisBlockGenerator** - утилита для подписания genesis-блока;
- **ApiKeyHash** - утилита для настройки авторизации API-методов ноды;

30.1 AccountsGeneratorApp

Утилита **AccountsGeneratorApp** применяется при конфигурировании аккаунта ноды в частной сети - набора данных об участнике блокчейн-сети. Для генерации аккаунта требуется настроить файл **accounts.conf**, расположенный в директории ноды.

Запуск **AccountsGeneratorApp**:

```
java -jar generators-x.x.x.jar AccountsGeneratorApp YourNode/accounts.conf
```

Генератор создает публичный ключ для ноды (аккаунт) и записывает его в файл **keystore.dat**, который будет расположен в директории вашей ноды. При необходимости, вы можете задать пароль для доступа к ключевой паре.

Подсказка: В случае, если вы задали пароль, вам следует указывать его в поле password при формировании запросов и транзакций.

Подробнее о создании аккаунта для ноды см. раздел [Создание аккаунта ноды](#).

30.2 GenesisBlockGenerator

Утилита **GenesisBlockGenerator** применяется для подписания genesis-блока частной сети - первого блока сети, содержащего транзакции, определяющие первоначальный баланс и разрешения ноды. Для подписания genesis-блока утилита использует блок blockchain.genesis конфигурационного файла ноды **node.conf**.

Запуск **GenesisBlockGenerator**:

```
java -jar generators-x.x.x.jar GenesisBlockGenerator YourNode/node.conf
```

Утилита заполняет поля genesis-public-key-base-58 (открытый ключ genesis-блока) и signature (подпись genesis-блока) конфигурационного файла ноды.

Подробнее о подписании genesis-блока см. раздел [Подписание genesis-блока и запуск сети](#).

30.3 ApiKeyHash

Утилита **ApiKeyHash** применяется для настройки авторизации API-методов ноды (gRPC и REST API-интерфейсов для обмена данными). Для генерации JWT-токена (при авторизации по OAuth) или токена на основе хэша ключевой строки api-key утилита использует данные конфигурационного файла **api-key-hash.conf**, который расположен в директории ноды.

Запуск **ApiKeyHash**:

```
java -jar generators-x.x.x.jar ApiKeyHash YourNode/api-key-hash.conf
```

Утилита генерирует JWT-токен или хэш заданной ключевой строки api-key, которые затем указываются в секции auth конфигурационного файла ноды.

Подробнее об авторизации gRPC и REST API см. раздел [Тонкая настройка платформы: настройка авторизации для gRPC и REST API](#).

Смотрите также

[Архитектура](#)

Сервисы авторизации и подготовки данных

Блокчейн-платформа Waves Enterprise включает два внешних интеграционных сервиса:

- **Сервис авторизации**, обеспечивающий авторизацию всех компонентов блокчейн-сети;
- **Сервис подготовки данных**, собирающий данные блокчейна в БД и предоставляющий API для доступа к этим данным.

31.1 Сервис авторизации

Сервис обеспечивает авторизацию всех компонентов блокчейн-сети на базе протокола **OAuth 2.0**. OAuth 2.0 - это открытый фреймворк авторизации, который позволяет предоставлять третьей стороне ограниченный доступ к защищенным ресурсам пользователя без раскрытия логина и пароля.

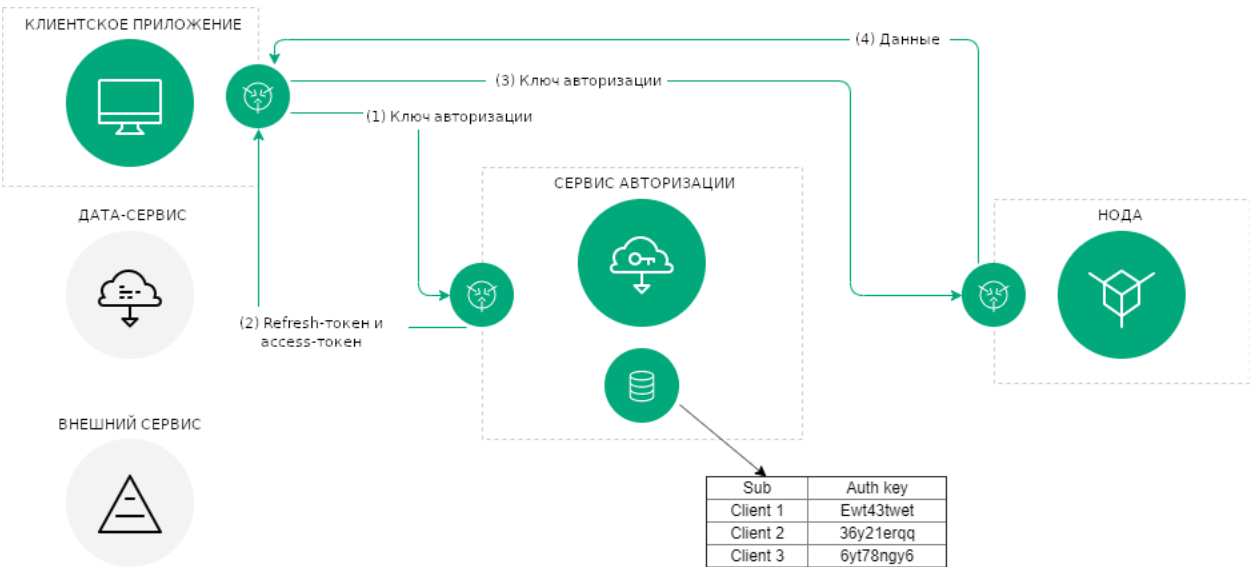
Общая схема обмена данными при авторизации по протоколу OAuth 2.0:

Средством авторизации является **JSON Web Token (JWT)**. Токены используются для авторизации каждого запроса от клиента к серверу и имеют ограниченное время жизни. Клиент получает два токена - **access** и **refresh**. Access-токен используется для авторизации запросов на доступ к защищенным ресурсам и для хранения дополнительной информации о пользователе. Refresh-токен используется для получения нового access-токена и обновления refresh-токена.

Ниже представлена схема авторизации для сетей на основе блокчейн-платформы Waves Enterprise:

Общий порядок авторизации выглядит следующим образом:

1. Клиент (компонент блокчейн-сети: корпоративный клиент, сервис обмена данными или стороннее приложение) единоразово предоставляет свои аутентификационные данные сервису авторизации;
2. В случае успешного прохождения процедуры первичной аутентификации сервис авторизации сохраняет аутентификационные данные клиента в хранилище данных, генерирует и отправляет клиенту подписанные access и refresh-токены. В токенах указываются время жизни токена и основные данные клиента: идентификатор и роль. Аутентификационные данные клиентов хранятся в конфигурационном файле сервиса авторизации. Каждый раз перед отправкой запроса стороннему сервису клиент проверяет время жизни access-токена и, в случае истечения срока жизни токена, обращается к сервису



авторизации для получения нового access токена. Для запросов к сервису авторизации используется refresh-токен;

3. Используя актуальный access-токен, клиент отправляет запрос на получение данных стороннего сервиса;
4. Сторонний сервис проверяет время жизни access-токена, его целостность, а также сравнивает полученный ранее публичный ключ сервиса авторизации с ключом, содержащимся в подписи access-токена. В случае успешной проверки сторонний сервис предоставляет клиенту запрашиваемые данные.

Описание способов авторизации приведено в статье [Сервис авторизации: варианты авторизации](#).

31.2 Сервис подготовки данных

Сервис подготовки данных предназначен для сбора данных из блокчейна в реляционную БД. Для получения доступа к собранным данным сервис имеет собственный API.

В Waves Enterprise Mainnet сервис работает в автономном режиме, доступ к его API ограничен. Для развертывания в частной сети сервис настраивается специалистами Waves Enterprise в зависимости от особенностей проекта. Также вы можете изменить параметры работы сервиса самостоятельно при помощи переменных окружения, описание которых приведено в разделе [Сервис подготовки данных: ручная настройка](#).

31.3 API-методы интеграционных сервисов

Для обмена данными интеграционным сервисам доступны отдельные методы REST API:

31.3.1 REST API: методы сервиса авторизации

GET /status

Метод предназначен для получения статуса сервиса авторизации.

Пример ответа:

GET /status:

```
{
  "status": "string",
  "version": "string",
  "commit": "string"
}
```


POST /v1/user

Метод предназначен для регистрации нового пользователя через сервис авторизации.

Для запроса передаются следующие параметры:

- login - логин пользователя (адрес электронной почты);
- password - пароль для доступа к аккаунту;
- locale - выбор языка, на котором пользователю будет предоставляться информация на почту (возможные варианты: *en* и *ru*);
- source - тип пользователя: *license* - владелец лицензии на использование блокчейн-платформы, *voting* - пользователь [сервиса голосования Waves Enterprise Voting](#).

Если регистрация прошла успешно, в качестве ответа метод возвращает код 201. В случае иного ответа, пользователь не был зарегистрирован.

GET /v1/user/profile

Метод предназначен для получения данных пользователя.

Пример ответа:

GET /v1/user/profile:

```
{
  "id": "string",
  "name": "string",
  "locale": "en",
  "addresses": [
    "string"
  ],
  "roles": [
    "string"
  ]
}
```

POST /v1/user/address

Метод предназначен для получения идентификатора адреса пользователя. В запросе метода передаются следующие данные:

- address - адрес пользователя в блокчейне;
- name - имя пользователя.

Пример ответа:

POST /v1/user/address: :animate: fade-in-slide-down

```
{
  "addressId": "string"
}
```

GET /v1/user/address/exists

Метод предназначен для проверки адреса электронной почты пользователя. В качестве параметра на вход метод принимает электронный адрес пользователя.

Пример ответа:

GET /v1/user/address/exists: :animate: fade-in-slide-down

```
{
  "exist": true
}
```

POST /v1/user/password/restore

Метод предназначен для восстановления пароля доступа к аккаунту пользователя.

В запросе метода указываются следующие данные:

- email - электронный адрес пользователя;
- source - тип пользователя: * license - владелец лицензии на использование блокчейн-платформы;
* voting - пользователь сервиса голосования [Waves Enterprise Voting](#).

Пример ответа:

POST /v1/user/password/restore: :animate: fade-in-slide-down

```
{
  "email": "string"
}
```

POST /v1/user/password/reset

Метод предназначен для сброса пароля пользователя.

В запросе указываются следующие данные:

- token - токен авторизации пользователя;
- password - текущий пароль пользователя.

Пример ответа:

POST /v1/user/password/reset: :animate: fade-in-slide-down

```
{
  "userId": "string"
}
```

GET /v1/user/confirm/{code}

Метод предназначен для передачи кода подтверждения для восстановления пароля для доступа к аккаунту пользователя. В запросе передаётся значение кода подтверждения.

POST /v1/user/resendEmail

Метод предназначен для повторной отправке кода восстановления пароля на указанный электронный адрес.

В запросе метода передаются следующие данные:

- email - электронный адрес пользователя;
- source - тип пользователя: * license - владелец лицензии на использование блокчейн-платформы;
* voting - пользователь сервиса голосования [Waves Enterprise Voting](#).

В ответе метод возвращает электронный адрес пользователя, на который был отправлен код восстановления.

Пример ответа:

POST /v1/user/resendEmail:

```
{
  "email": "string"
}
```

POST /v1/auth/login

Метод предназначен для получения нового токена авторизации для пользователя.

Для запроса передаются следующие параметры:

- username - имя пользователя;
- password - пароль для доступа к аккаунту;
- locale - выбор языка, на котором пользователю будет предоставляться информация на почту (возможные варианты: *en* и *ru*);
- source - тип пользователя: * license - владелец лицензии на использование блокчейн-платформы;
* voting - пользователь сервиса голосования [Waves Enterprise Voting](#).

Пример ответа:

POST /v1/auth/login:

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

POST /v1/auth/token

Метод предназначен для получения токенов авторизации для внешних сервисов и приложений. Не требует параметров запроса.

Пример ответа:**POST /v1/auth/token:**

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

POST /v1/auth/refresh

Метод предназначен для получения нового **refresh**-токена. В запросе метода передается значение текущего **refresh**-токена.

Пример ответа:**POST /v1/auth/refresh:**

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

GET /v1/auth/publicKey

Метод предназначен для получения публичного ключа сервиса авторизации. Не требует параметров запроса.

Пример ответа:

POST /v1/auth/refresh:

```

-----BEGIN PUBLIC KEY-----
MIICIJANBgqhkiG9w0BAQEFAA0CAg8AMIICGgKCAgEA7d90j/ZQTkkjf4UuMfUu
QIFDTYxYf6QBKMVJnq/wXyPYYkV8HVFYFizCaEciv3CXmBH77sXnuTlrEtvK7zHB
KvV870HmZuazjlgZVSk0n0Y7F8UUVNXnlzVD1dPs0GJ6orM41DnC1W65mCrP3bjn
fV4RbmykN/lk7McA6EsMcLEGbKkFhmeq2Nk4hn2CQvoTkupJU0CP1dh04bq1lQ7
Ffj9K/FJq73wSXD0H+qqdRG9sfrtgrhtJHerruhv3456e0zyAcD08+sJUQFKY80B
SZMEndVzFS2ub9Q8e7BfcNxTmQPM4PhH05wuTqL32qt3uJBx20I4lu30ND44ZrDJ
BbVog73oPjRYXj+kTbwUZI66SP4aLcQ8sypQyLwqKk5DtLRozSN00IrupJJ/pwZs
9zPEggL91T0rirbEhGlf5U8/6XN8GVXX4iMk2fD8FHLFJuXCD70j4JC2iWfFDC6a
uUkwUfqfjJB8BzIHkncoq0ZbpideE2lTWl+svuEu/wyP5rNlyMiE/e/fZQqM2+o0
ch5Qow6HH35BrloCSZciutUcd1U7YPqESJ5tryy1xn9bsMb+On1ocZTtvec/ow4M
RmnJwm0j1nd+cc190KLG5/boeA+2zqWu0jCbWR9c0oCmgbhuqZCHaHTBEAKDWcsC
VRz5qD6FPpePpTQDb6ss3bkCAwEAAQ==
-----END PUBLIC KEY-----

```

Смотрите также*Сервисы авторизации и подготовки данных*

data-sv-conf

auth-sv-var

*REST API: методы сервиса подготовки данных***31.3.2 REST API: методы сервиса подготовки данных**

Сервису подготовки данных доступны следующие группы методов:

Группа методов Assets

Методы группы **Assets** предназначены для получения данных о наборах токенов (аскетах).

GET /assets

Метод предназначен для получения списка доступных в блокчейне ассетах. Список выводится в виде транзакций об эмиссии соответствующих ассетов.

Пример ответа:

GET /assets:

```
[
  {
    "index": 0,
    "id": "string",
    "name": "string",
    "description": "string",
    "reissuable": true,
    "quantity": 0,
    "decimals": 0
  }
]
```

POST /assets/count

Метод возвращает количество доступных в блокчейне ассетов.

Пример ответа:

POST /assets/count:

```
{
  "count": 0
}
```

GET /assets/{id}

Метод возвращает информацию о доступном наборе токенов по его {id}.

В ответе метода выводятся следующие данные:

- index - порядковый номер ассета;
- id - идентификатор ассета;
- name - имя ассета;
- description - описание ассета;
- reissuable - перевыпускаемость ассета;
- quantity - количество токенов в ассете;
- decimals - количество разрядов после запятой у используемого токена (WEST - 8)

Пример ответа:

GET /assets/{id}:

```
{
  "index": 14,
  "id": "12nx0qnhjd83",
  "name": "Demo asset",
  "description": "Demo asset",
  "reissuable": true,
  "quantity": 400,
  "decimals": 8
}
```

Группа методов Blocks**GET /blocks/at/{height}**

Метод возвращает содержимое блока на высоте height.

В ответе метода возвращаются следующие параметры:

- reference - хэш-сумма блока;
- blocksize - размер блока;
- features - *функциональные возможности*, запущенные на момент создания блока;
- signature - подпись блока;
- fee - комиссия за транзакции, включенные в блок;
- generator - адрес создателя блока;
- transactionCount - количество транзакций, включенных в блок;
- transactions - массив с телами транзакций, включенных в блок;
- version - версия блока;
- poa-consensus.overall-skipped-rounds - количество пропущенных раундов майнинга, при использовании алгоритма консенсуса *PoA*;
- timestamp - временная метка создания блока в формате **Unix Timestamp** (в миллисекундах);
- height - высота создания блока.

Пример ответа:**GET /blocks/at/{height}:**

```
{
  "reference":
  ↪ "hT5RcPT4jDVoNspFzKnhKqfGuMbrizjpG4vmPecVfWgWaGMoAn5hgPBjPc9696TL8wGDKJzkwewiqe8m26C4aPd
  ↪ ",
  "blocksize": 226,
  "features": [],
  "signature":
  ↪ "5GAM7jfQScw4g3g7PCNNtz5xG3JzjJnW4Ap2soThirSx1AmUQHQMjz8VMtkFEzK7L447ouKHfj2gMvZyP5u94Rps
```

(continues on next page)

(продолжение с предыдущей страницы)

```

↪ ",
  "fee": 0,
  "generator": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkW9d97c",
  "transactionCount": 0,
  "transactions": [],
  "version": 3,
  "poa-consensus": {
    "overall-skipped-rounds": 1065423
  },
  "timestamp": 1615816767694,
  "height": 1826
}

```

Группа методов **Contracts**

Методы группы **Contracts** предназначены для получения информации о смарт-контрактах блокчейна.

GET /contracts

Метод возвращает информацию по всем смарт-контрактам, загруженным в сеть. Для каждого смарт-контракта в ответе возвращаются следующие параметры:

- `contractId` - идентификатор смарт-контракта;
- `image` - имя Docker-образа смарт-контракта, либо его абсолютный путь в репозитории;
- `imageHash` - хэш-сумма смарт-контракта;
- `version` - версия смарт-контракта;
- `active` - статус смарт-контракта на момент отправки запроса: `true` - запущен, `false` - не запущен.

Пример ответа для одного смарт-контракта:

GET /contracts:

```

[
  {
    "contractId": "dmLT1ippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "image": "registry.wvservices.com/wv-sc/may14_1:latest",
    "imageHash": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957",
    "version": 1,
    "active": false
  }
]

```


GET /contracts/count

Метод возвращает количество смарт-контрактов в блокчейне, соответствующих заданным условиям и фильтрам.

Пример ответа:

GET /contracts/count:

```
{
  "count": 19
}
```

GET /contracts/id/{id}

Метод возвращает информацию о смарт-контракте по его идентификатору {id}.

Пример ответа:

GET /contracts/id/{id}:

```
{
  "id": "string",
  "type": 0,
  "height": 0,
  "fee": 0,
  "sender": "string",
  "senderPublicKey": "string",
  "signature": "string",
  "timestamp": 0,
  "version": 0
}
```

GET /contracts/id/{id}/versions

Метод возвращает историю версий смарт-контракта с заданным {id}.

Пример ответа для одной версии:

GET /contracts/id/{id}/versions:

```
[
  {
    "version": 0,
    "image": "string",
    "imageHash": "string",
    "timestamp": "string"
  }
]
```

GET /contacts/history/{id}/key/{key}

Возвращает историю изменений ключа {key} смарт-контракта по его {id}.

Пример ответа для одного ключа:

GET /contacts/history/{id}/key/{key}:

```
{
  "total": 777,
  "data": [
    {
      "key": "some_key",
      "type": "integer",
      "value": "777",
      "timestamp": 1559347200000,
      "height": 14024
    }
  ]
}
```

GET /contracts/senders-count

Метод возвращает количество уникальных участников, отправляющих транзакции [104](#) на вызов смарт-контрактов.

Пример ответа:

GET /contracts/senders-count:

```
{
  "count": 777
}
```

GET /contracts/calls

Возвращает список транзакций [104](#) на вызов смарт-контрактов с их параметрами и результатами.

Пример ответа для одной транзакции:

GET /contracts/calls:

```
[
  {
    "id": "string",
    "type": 0,
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0,
    "contract_id": "string",
    "contract_name": "string",
    "contract_version": "string",
    "image": "string",
    "fee_asset": "string",
    "finished": "string",
    "params": [
      {
        "tx_id": "string",
        "param_key": "string",
        "param_type": "string",
        "param_value_integer": 0,
        "param_value_boolean": true,
        "param_value_binary": "string",
        "param_value_string": "string",
        "position_in_tx": 0,
        "contract_id": "string",
        "sender": "string"
      }
    ],
    "results": [
      {
        "tx_id": "string",
        "result_key": "string",
        "result_type": "string",
        "result_value_integer": 0,
        "result_value_boolean": true,
        "result_value_binary": "string",
        "result_value_string": "string",
        "position_in_tx": 0,
        "contract_id": "string",
        "time_stamp": "string"
      }
    ]
  }
]
```

Группа методов Privacy

Методы группы **Privacy** предназначены для получения информации о группах доступа к конфиденциальным данным.

GET /privacy/groups

Метод возвращает список групп доступа в блокчейне.

Пример ответа для одной группы доступа:

GET /privacy/groups:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/groups/count

Метод возвращает количество групп доступа в блокчейне.

Пример ответа:

GET /privacy/groups/count:

```
{
  "count": 2
}
```

GET /privacy/groups/{address}

Метод возвращает список групп доступа, в которые входит заданный адрес {address}.

Пример ответа для одной группы доступа:

GET /privacy/groups/{address}:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/groups/by-recipient/{address}

Метод возвращает список групп доступа, в которых заданный {address} фигурирует как получатель данных.

Пример ответа для одной группы доступа:

GET /privacy/groups/by-recipient/{address}:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/groups/{address}/count

Метод возвращает количество групп доступа, в которые входит заданный {address}.

Пример ответа:

GET /privacy/groups/{address}/count:

```
{
  "count": 1
}
```

GET /privacy/groups/id/{id}

Метод возвращает информацию о группе доступа по ее идентификатору {id}.

Пример ответа:

GET /privacy/groups/id/{id}:

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

GET /privacy/groups/id/{id}/history

Метод возвращает историю изменений группы доступа по ее {id}. История изменений возвращается в виде списка отправленных *транзакций 112-114* с их описанием.

Пример ответа для одной транзакции:

GET /privacy/groups/id/{id}/history:

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

GET /privacy/groups/id/{id}/history/count

Метод возвращает количество отправленных транзакций 112-114 для внесения изменений в группу доступа с указанным {id}.

Пример ответа:

GET /privacy/groups/id/{id}/history/count:

```
{
  "count": 0
}
```

GET /privacy/nodes

Метод возвращает список доступных нод в блокчейне.

Пример ответа для одной ноды:

GET /privacy/nodes:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/nodes/count

Метод возвращает количество доступных нод в блокчейне.

Пример ответа:

GET /privacy/nodes/count:

```
{
  "count": 0
}
```

GET /privacy/nodes/publicKey/{targetPublicKey}

Метод возвращает информацию о ноде по ее публичному ключу {targetPublicKey}.

Пример ответа:

GET /privacy/nodes/publicKey/{targetPublicKey}:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

GET /privacy/nodes/address/{address}

Метод возвращает информацию о ноде по ее адресу {address}.

Пример ответа:

GET /privacy/nodes/address/{address}:

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

Группа методов Transactions

Методы группы **Transactions** предназначены для получения информации о транзакциях в блокчейне.

GET /transactions

Метод возвращает список транзакций, соответствующий условиям поискового запроса и применённым фильтрам.

Важно: За один запрос через метод **GET /transactions** возвращается не более 500 транзакций.

Пример ответа для одной транзакции:

GET /transactions:

```
[
  {
    "id": "string",
    "type": 0,
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0
  }
]
```


GET /transactions/count

Метод возвращает количество транзакций, соответствующих условиям поискового запроса и применённым фильтрам.

Пример ответа:

GET /transactions/count:

```
{  
  "count": "116"  
}
```

GET /transactions/{id}

Метод возвращает транзакцию по идентификатору {id}.

Пример ответа:

GET /transactions/{id}:

```
{  
  "id": "string",  
  "type": 0,  
  "height": 0,  
  "fee": 0,  
  "sender": "string",  
  "senderPublicKey": "string",  
  "signature": "string",  
  "timestamp": 0,  
  "version": 0  
}
```

Группа методов Users

Методы группы **Users** предназначены для получения информации об участниках сети.

GET /users

Возвращает список участников, соответствующий условиям поискового запроса и применённым фильтрам.

Пример ответа для одного участника:

GET /users:

```
[
  {
    "address": "string",
    "aliases": [
      "string"
    ],
    "registration_date": "string",
    "permissions": [
      "string"
    ]
  }
]
```

GET /users/count

Метод возвращает количество участников, удовлетворяющих установленным в запросе фильтрам.

Пример ответа для одного участника:

GET /users/count:

```
{
  "count": 1198
}
```

GET /users/{userAddressOrAlias}

Метод возвращает информацию об участнике по его адресу или алиасу.

Пример ответа:

GET /users/{userAddressOrAlias}:

```
{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ]
}
```

GET /users/contract-id/{contractId}

Метод возвращает список участников, когда-либо вызывавших смарт-контракт с указанным {contractId}.

Пример ответа для одного участника:

GET /users/contract-id/{contractId}:

```
{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ]
}
```

POST /users/by-addresses

Метод возвращает список участников для заданного набора адресов.

Пример ответа для одного участника:

POST /users/by-addresses:

```
{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ]
}
```

Методы для получения информации о транзакциях с данными (12)

Данная группа методов вызывается по маршруту /api/v1/txIds/.

GET /api/v1/txIds/{key}

Метод возвращает список идентификаторов транзакций с данными, содержащих указанный ключ {key}.

Пример ответа для одной транзакции:

GET /api/v1/txIds/{key}:

```
[
  {
    "id": "string"
  }
]
```

GET /api/v1/txIds/{key}/{value}

Метод возвращает список идентификаторов транзакций с данными, содержащих указанный ключ {key} и значение {value}.

Пример ответа для одной транзакции:

GET /api/v1/txIds/{key}/{value}:

```
[
  {
    "id": "string"
  }
]
```

GET /api/v1/txData/{key}

Метод возвращает тела транзакций с данными, содержащих указанный ключ {key}.

Пример ответа для одной транзакции:

GET /api/v1/txData/{key}:

```
[
  {
    "id": "string",
    "type": "string",
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0,
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "key": "string",
    "value": "string",
    "position_in_tx": 0
  }
]

```

GET /api/v1/txData/{key}/{value}

Метд возвращает тела транзакций с данными, содержащих указанные ключ {key} и значение {value}.

Пример ответа для одной транзакции:

GET /api/v1/txData/{key}/{value}:

```

[
  {
    "id": "string",
    "type": "string",
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0,
    "key": "string",
    "value": "string",
    "position_in_tx": 0
  }
]

```

Группа методов Leasing**GET /leasing/calc**

Метод возвращает сумму выплат за лизинг токенов в указанном интервале высот блоков.

Пример ответа:

GET /leasing/calc:

```

{
  "payouts": [
    {
      "leaser": "3P1EiJnPxFxGyhN9sucXfB2rhQ1ws4cmuS5",
      "payout": 234689
    }
  ],

```

(continues on next page)

(продолжение с предыдущей страницы)

```

"totalSum": 4400000,
"totalBlocks": 1600
}

```

Группа методов Stats

Методы группы **Stats** предназначены для получения статистических данных и мониторинга блокчейна.

GET /stats/transactions

Метод возвращает информацию о проведенных транзакциях за указанный временной промежуток.

Пример ответа:

GET /stats/transactions:

```

{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "transactions": [
        {
          "type": 104,
          "count": 100
        }
      ]
    }
  ]
}

```

GET /stats/contracts

Метод возвращает информацию о транзакциях *104* за указанный временной промежуток.

Пример ответа:

GET /stats/contracts:

```

{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "transactions": [
        {
          "type": 104,

```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "count": 100
  }
]
}
```

GET /stats/tokens

Метод возвращает информацию об обороте токенов в блокчейне за указанный временной промежуток.

Пример ответа:

GET /stats/tokens:

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "sum": "12000.001"
    }
  ]
}
```

GET /stats/addresses-active

Метод возвращает адреса, которые были активными в указанный временной промежуток.

Пример ответа:

GET /stats/addresses-active:

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "senders": "12",
      "recipients": "12"
    }
  ]
}
```

GET /stats/addresses-top

Метод возвращает адреса, которые были наиболее активными отправителями или получателями в указанный временной промежуток.

Пример ответа:

GET /stats/addresses-top:

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "senders": "12",
      "recipients": "12"
    }
  ]
}
```

GET /stats/nodes-top

Метод возвращает адреса нод, которые создали наибольшее количество блоков в указанный временной промежуток.

Пример ответа:

GET /stats/nodes-top:

```
{
  "limit": "10",
  "data": [
    {
      "generator": "3NdPsjaFC7NeioGVF6X4J5A8FVaxdtKvAba",
      "count": "120",
      "node_name": "Genesis Node #5"
    }
  ]
}
```

GET /stats/contract-calls

Метод возвращает список смарт-контрактов, вызванных наибольшее количество раз в указанный временной промежуток.

Пример ответа:

GET /stats/contract-calls:

```
{
  "limit": "5",
  "data": [
    {
      "contract_id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "count": "120",
      "contract_name": "oracle_contract",
      "last_call": "60.321"
    }
  ]
}
```

GET /stats/contract-last-calls

Метод возвращает список последних вызовов смарт-контрактов по их id и названию.

Пример ответа:**GET /stats/contract-last-calls:**

```
{
  "limit": "5",
  "data": [
    {
      "contract_id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "contract_name": "oracle_contract",
      "last_call": "60.321"
    }
  ]
}
```

GET /stats/contract-types

Метод возвращает список смарт-контрактов блокчейна по именам образов и их хэшам.

Пример ответа:**GET /stats/contract-types:**

```
{
  "limit": "5",
  "data": [
    {
      "id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "image": "registry.wvservices.com/waves-enterprise-public/oracle-contract:v0.1",
      "image_hash": "936f10207dee466d051fe09669d5688e817d7cdd81990a7e99f71c1f2546a660",
      "count": "60",

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "sum": "6000"
  }
]
}

```

GET /stats/monitoring

Метод возвращает общую информацию о сети.

Пример ответа:

GET /stats/monitoring:

```

{
  "tps": "5",
  "blockAvgSize": "341.391",
  "senders": "50",
  "nodes": "50",
  "blocks": "500000"
}

```

Группа методов Anchoring

Методы группы **Anchoring** предназначены для получения информации о раундах анкоринга.

GET /anchoring/rounds

Метод возвращает список транзакций, отправленных в раундах анкоринга в соответствии с заданными условиями и фильтрами.

Пример ответа:

GET /anchoring/rounds:

```

[
  {
    "height": 0,
    "sideChainTxIds": [
      "string"
    ],
    "mainNetTxIds": [
      "string"
    ],
    "status": "string",
    "errorCode": 0
  }
]

```

GET /anchoring/round/at/{height}

Метод возвращает информацию о раунде анкоринга на указанной высоте блоков {height}.

Пример ответа:

GET /anchoring/round/at/{height}:

```
{
  "height": 0,
  "sideChainTxIds": [
    "string"
  ],
  "mainNetTxIds": [
    "string"
  ],
  "status": "string",
  "errorCode": 0
}
```

GET /anchoring/info

Метод возвращает информацию об анкоринге в блокчейне.

Пример ответа:

GET /anchoring/info:

```
{
  "height": 0,
  "sideChainTxIds": [
    "string"
  ],
  "mainNetTxIds": [
    "string"
  ],
  "status": "string",
  "errorCode": 0
}
```

Вспомогательные методы сервиса подготовки данных**GET /info**

Метод возвращает информацию об используемом сервисе подготовки данных.

Пример ответа:

GET /info:

```
{
  "version": "string",
  "buildId": "string",
  "gitCommit": "string"
}
```

GET /status

Метод возвращает информацию о состоянии сервиса подготовки данных.

Пример ответа:**GET /status:**

```
{
  "status": "string"
}
```

Смотрите также

Сервисы авторизации и подготовки данных

data-sv-conf

REST API: методы сервиса авторизации

auth-sv-var

Смотрите также

auth-sv-var

data-sv-conf

REST API: методы сервиса авторизации

REST API: методы сервиса подготовки данных

Официальные ресурсы и контакты

32.1 Официальные ресурсы блокчейн-платформы

- Официальный сайт блокчейн-платформы [Waves Enterprise](#)
- Страница проекта в [Github](#)
- Официальный сайт блокчейн-платформы [Waves](#)

32.2 Как с нами связаться

- Служба технической поддержки [Waves Enterprise](#)
- Форма обратной связи в клиенте блокчейн-платформы
- Официальный Telegram-чат на английском: [Waves Enterprise Group](#)
- Официальный Telegram-чат на русском: [Waves Enterprise](#)

Авторизация

Предоставление участнику прав на выполнение тех или иных операций в блокчейне (в частности, на применение API-методов)

Адрес

Идентификатор участника сети, полученный из его публичного ключа. Каждый адрес имеет собственный баланс и стейт

Аккаунт

Набор данных об участнике сети, использующийся для его идентификации

Алиас (псевдоним)

Условное имя участника сети, связанное с его адресом. Алиас присваивается участнику при помощи транзакции *10* и может указываться в транзакциях вместо адреса конкретного участника

Анкоринг

Алгоритм проверки данных в приватном блокчейне на неизменность путем их валидации в более крупной сети

Атомарная транзакция

Транзакция-контейнер, состоящая из нескольких других транзакций. Если одна из транзакций, помещенных в атомарную, не выполняется, также не выполняются и все остальные

Баланс

Количество токенов, которыми владеет адрес в блокчейне

Блок

Зафиксированный в блокчейне набор транзакций, подписанный майнером и содержащий ссылку на подпись предыдущего блока. Размер блока ограничен 1 Мб или 6000 транзакциями

Блокчейн

Децентрализованный, распределённый и общедоступный цифровой реестр, записывающий информацию таким образом, что любая отдельная запись не может быть изменена после ее внесения без изменения всех последующих блоков

Валидация

Подтверждение неизменности (целостности) данных

Генератор

Вспомогательная утилита, позволяющая создавать ключевые пары или ключевые строки

Генерирующий баланс

Минимальный баланс, дающий адресу право на майнинг

Группа доступа

Список адресов, имеющих доступ к конфиденциальным данным, размещенным в блокчейне

Дата-краулер

Сервис извлечения данных из ноды и их загрузки в сервис подготовки данных

Исполнение смарт-контракта

Исполнение программного кода, заложенного в смарт-контракт, в блокчейне

Ключевой блок

Начальный блок раунда майнинга, содержащий служебную информацию:

- публичный ключ майнера для проверки подписи микроблоков;
- сумму комиссии майнера за предыдущий блок;
- подпись майнера;
- ссылку на предыдущий ключевой блок.

Комиссия

Сумма токенов, которую уплачивает адрес за отправленные им транзакции в блокчейн

Консенсус

Алгоритм согласования информации, записываемой в блокчейн, между его участниками

Лицензия

Документ, дающий право использования блокчейн-платформы Waves Enterprise

Лизинг

Предоставление участником токенов, находящихся на его балансе, в аренду другим участникам. Лизинг используется для создания генерирующего баланса у участника, берущего токены в лизинг, а также повышения вероятности выбора участника майнером следующего раунда при использовании алгоритма консенсуса LPoS

Майнер

Нода, имеющая право создания новых блоков блокчейна

Майнинг

Процесс создания новых блоков блокчейна

Миграция

Процесс изменения ключевых параметров блокчейна

Микроблок

Набор транзакций, применяемых к стеиту блокчейна. Микроблоки формируют блок сети, количество транзакций в микроблоке ограничено 500 единицами

Нода (узел)

Компьютер участника сети с установленным ПО блокчейн-платформы Waves Enterprise и присвоенным адресом в сети

Обновление ноды

Обновление ПО блокчейн-платформы Waves Enterprise, установленного на компьютере участника сети

Образ

Шаблон смарт-контракта, содержащий его код и использующийся для создания Docker-контейнера, в котором исполняется смарт-контракт

Откат

Отправка уже созданного блока на повторный майнинг вследствие неполадок, возникающих на нодах блокчейна

Пир

Сетевой адрес ноды

Подписание транзакции

Добавление в тело транзакции публичного ключа ее создателя, используется для подтверждения целостности транзакции в блокчейне

Приватная (частная) сеть, сайдчейн

Блокчейн-сеть, отделенная от Waves Enterprise Mainnet и имеющая собственных зарегистрированных участников

Приватный ключ

Строковая комбинация символов для подписания транзакций и доступа к токенам, доступ к которой имеет только ее владелец. Приватный ключ неразрывно связан с публичным ключом

Публикация транзакции

Запись транзакции в блок блокчейна в ходе раунда майнинга

Публичная сеть

Крупная блокчейн-сеть, каждый участник которой заранее известен и зарегистрирован (к примеру, Waves Enterprise Mainnet)

Публичный ключ

Строковая комбинация символов, неразрывно связанная с приватным ключом. Публичный ключ прикладывается к транзакциями для подтверждения корректности подписи пользователя сделанной на закрытом ключе

Пул неподтвержденных транзакций (UTX-пул)

Компонент блокчейн-платформы Waves Enterprise, обеспечивающий хранение неподтвержденных транзакций до момента их проверки и отправки в блокчейн

Раунд

Процесс майнинга блока участником блокчейн-сети

Репозиторий

Хранилище образов смарт-контрактов, разворачиваемое на основе ПО Docker Registry

Роль

Разрешение или запрет на выполнение тех или иных операций в блокчейне

Сетевое сообщение

Информация о сетевом событии, отправляемая нодой другим нодам блокчейна

Смарт-контракт

Приложение, которое записывает в блокчейн свои входные данные и результаты исполнения заложенного алгоритма

Снимок данных

Сохраняемый набор данных блокчейна:

- стейты адресов сети;
- стейты смарт-контрактов, загруженных в сет;
- данные майнеров прошедших раундов;

- данные групп доступа к конфиденциальным данным.

Создание смарт-контракта

Загрузка нового смарт-контракта в блокчейн при помощи транзакции [103](#)

Софт-форк

Механизм активации предварительно заложенных функциональных возможностей блокчейна

Стейт

История транзакций блокчейна, хранящаяся в БД каждой ноды

Стейт адреса

Набор данных отдельного адреса: балансы, информация об отправленных транзакциях с данными, результаты исполнения вызванных адресом смарт-контрактов

Стейт смарт-контракта

Текущие данные о результатах исполнения смарт-контракта, записываемые и обновляемые при помощи транзакции [104](#)

Токен

1. Расчетная единица блокчейна, используемая для мотивации участников к майнингу в сети
2. Объект, используемый для авторизации участника блокчейна

Транзакция

Отдельная операция в блокчейне от имени участника, изменяющая стейт сети. Отправляя ту или иную транзакцию, участник отправляет в сеть запрос с набором данных, необходимых для соответствующего изменения стейта

Участник

Пользователь ПО блокчейн-платформы Waves Enterprise, отправляющий транзакции в блокчейн

Форк

Образование новой ветки блокчейна

Хранилище ключей (keystore)

Закрытый репозиторий, в котором хранятся ключевые пары нод блокчейна

Хэш

Уникальный набор символов, генерируемый из исходных данных при помощи заданного алгоритма. Хэш позволяет однозначно идентифицировать исходные данные

Хэш ключевой строки

Набор символов, генерируемых из заданной участником ключевой строки и используемый для его авторизации в блокчейне

API-метод

Отдельная процедура, вызываемая участником при помощи API-интерфейса блокчейн-платформы (gRPC или REST API) и предназначенная для выполнения определенной операции в блокчейне

Crash Fault Tolerance (CFT)

Алгоритм консенсуса на основе PoA, исключающий возникновение форков блокчейна при какой-либо неполадке со стороны одного или нескольких участников

Genesis-блок

Начальный блок блокчейн-сети, содержащий служебные транзакции для распределения первичных ролей и балансов участников

Leased Proof of Stake (LPoS)

Алгоритм консенсуса PoS, предоставляющий участнику возможность передавать токены в лизинг другим участникам

Liquid block

Состояние блока в ходе раунда майнинга от формирования его ключевого блока до формирования следующего ключевого блока

JWT-токен (JSON Web Token)

Объект в формате JSON, применяющийся для авторизации участника блокчейна по протоколу OAuth

Proof of Authority (PoA)

Алгоритм консенсуса, при котором возможность проверки транзакций и создание новых блоков отводится более авторитетным узлам

Proof of Stake (PoS)

Алгоритм консенсуса, при котором нода, проверяющая транзакции и осуществляющая майнинг в следующем раунде, выбирается на основе ее текущего баланса

Sandbox

Режим проверки возможностей блокчейн-платформы

Seed-фраза

Набор из 24 произвольно заданных слов для восстановления доступа к балансу адреса

Targetnet

Блокчейн-сеть, в которую осуществляется анкоринг данных из приватной сети

Что нового в Waves Enterprise

34.1 1.6.0

Версия 1.6.0 является последней выпущенной версией и в этой справочной системе имеет тег **latest**.

Полностью переработана структура и содержание документации, добавлена титульная страница с поиском и быстрым доступом к основным разделам.

Добавлены следующие разделы, описывающие разработанный в версии 1.6.0 механизм создания снимка данных:

- *Механизм создания снимка данных*
- *Запуск ноды с созданным снимком данных*
- *Тонкая настройка платформы: настройка механизма создания снимка данных*

34.2 1.5.2

Внесены изменения в раздел *Алгоритм консенсуса CFT*.

Версия 1.5.2 содержит важные исправления, подробнее см. [описание релиза](#).

34.3 1.5.0

Добавлены следующие разделы:

- *Алгоритм консенсуса CFT*
- Подготовка к работе
- gRPC методы ноды
- Отслеживание событий в блокчейне посредством gRPC интерфейса

Изменены следующие разделы:

- Криптография
- Управление полномочиями
- Транзакции
- Подготовка конфигурационных файлов
- Изменения в конфигурационном файле ноды
- Описание основных параметров и секций конфигурационного файла ноды
- Настройка консенсуса
- API-инструменты ноды
- JavaScript SDK
- *Словарь терминов*
- Содержимое раздела Настройка Docker перенесено в новый раздел Подготовка к работе
- Раздел Смарт-контракты Docker с использованием REST API ноды убран из индекса

34.4 1.4.0

Добавлены следующие разделы:

- Атомарные транзакции
- Работа в веб-клиенте
- JavaScript SDK

Изменены следующие разделы:

- *Архитектура*
- Транзакции
- Настройка авторизации и REST API и gRPC интерфейсов ноды
- API-инструменты ноды
- Обновление ноды

34.5 1.3.1

Добавлены следующие разделы:

- Параллельное исполнение контрактов

Изменены следующие разделы:

- Создание смарт-контракта
- Настройка Docker

34.6 1.3.0

Изменены следующие разделы:

- Клиент
- Разделы «Ролевая модель» и «Управление доступом» преобразованы в раздел Управление полномочиями
- Описание основных параметров и секций конфигурационного файла ноды
- Настройка групп доступа к конфиденциальным данным
- Настройка Docker
- Методы REST API Addresses
- Методы REST API Node
- Методы REST API Contracts
- Методы REST API Privacy
- *Системные требования*

34.7 1.2.3

Изменены следующие разделы:

- Смарт-контракты Docker
- Описание основных параметров и секций конфигурационного файла ноды
- Настройка групп доступа к конфиденциальным данным

34.8 1.2.2

Добавились следующие разделы:

- Методы REST API Debug
- Полное описание REST API на странице [Документация API](#)

Изменены следующие разделы:

- Установка и запуск платформы Waves Enterprise

34.9 1.2.0

Добавлены следующие разделы:

- Новый раздел справки Интеграционные сервисы, в который вошли Сервис авторизации и Сервис подготовки данных
- Добавлена инструкция по получению лицензии
- Добавлен новый метод REST API ноды Licenses
- Добавлен новый раздел Смарт-контракты Docker с использованием gRPC
- Добавлен новый раздел Сервисы gRPC, используемые смарт-контрактом

Изменены следующие разделы:

- Установка и запуск платформы Waves Enterprise
- Обновлен раздел Криптография. Часть информации была перенесена в Операции шифрования данных
- Изменения в конфигурационном файле ноды
- Транзакции

34.10 1.1.2

Изменены следующие разделы:

- Демо-версия
- Изменения в конфигурационном файле ноды
- Раздел Установка ноды преобразован в раздел «Установка и запуск платформы Waves Enterprise»
- Подключение участников к сети
- Настройка анкоринга
- Настройка типа авторизации для доступа к REST API ноды
- Подключение ноды в сеть «Waves Enterprise Partnernet»
- Подключение ноды в сеть «Waves Enterprise Mainnet»
- *Системные требования*

34.11 1.1.0

Добавились следующие разделы:

- Методы API, доступные смарт-контракту
- Демо-версия
- Изменения в конфигурационном файле ноды

Изменены следующие разделы:

- Смарт-контракты Docker

- Пример запуска контракта Docker
- Установка ноды
- Конфигурация и запуск дополнительных сервисов

34.12 1.0.0

Добавились следующие разделы:

- Сервис авторизации

Переработаны следующие разделы:

- Конфигурация ноды
- Подключение к Mainnet и Partnernet
- REST API
- Установка ноды

Изменения в конфигурационном файле ноды node.conf

- Добавлена секция NTP-сервер
- Добавлена секция auth выбора типа авторизации в REST API секции

A

API-метод, [294](#)

C

Crash Fault Tolerance (CFT), [294](#)

G

Genesis-блок, [294](#)

J

JWT-токен (*JSON Web Token*), [295](#)

L

Leased Proof of Stake (LPoS), [294](#)

Liquid block, [295](#)

P

Proof of Authority (PoA), [295](#)

Proof of Stake (PoS), [295](#)

S

Sandbox, [295](#)

Seed-фраза, [295](#)

T

Targetnet, [295](#)

Авторизация, [291](#)

Адрес, [291](#)

Аккаунт, [291](#)

Алиас (*псевдоним*), [291](#)

Анкоринг, [291](#)

Атомарная транзакция, [291](#)

Баланс, [291](#)

Блок, [291](#)

Блокчейн, [291](#)

Валидация, [291](#)

Генератор, [292](#)

Генерирующий баланс, [292](#)

Группа доступа, [292](#)

Дата-краулер, [292](#)

Исполнение смарт-контракта, [292](#)

Ключевой блок, [292](#)

Комиссия, [292](#)

Консенсус, [292](#)

Лизинг, [292](#)

Лицензия, [292](#)

Майнер, [292](#)

Майнинг, [292](#)

Миграция, [292](#)

Микроблок, [292](#)

Нода (*узел*), [292](#)

Обновление ноды, [292](#)

Образ, [293](#)

Откат, [293](#)

Пир, [293](#)

Подписание транзакции, [293](#)

Приватная (частная) сеть, сайдчейн, [293](#)

Приватный ключ, [293](#)

Публикация транзакции, [293](#)

Публичная сеть, [293](#)

Публичный ключ, [293](#)

Пул неподтвержденных транзакций (*UTX-пул*), [293](#)

Раунд, [293](#)

Репозиторий, [293](#)

Роль, [293](#)

Сетевое сообщение, [293](#)

Смарт-контракт, [293](#)

Снимок данных, [293](#)

Создание смарт-контракта, [294](#)

Софт-форк, [294](#)

Стейт, [294](#)

Стейт адреса, [294](#)

Стейт смарт-контракта, [294](#)

Токен, [294](#)

Транзакция, [294](#)

Участник, [294](#)

Форк, [294](#)

Хранилище ключей (*keystore*), [294](#)

Хэш, [294](#)

Хэш ключевой строки, [294](#)