Техническое описание платформы Waves Enterprise

Выпуск 1.12.0

https://wavesenterprise.com

янв. 30, 2024

Установка и использование платформы

1 Содержание документации **2**

1.1 Системные требования . . . 2
1.2 Лицензии блокчейн-платформы Waves Enterprise . . . 3
1.3 Развертывание платформы в ознакомительном режиме (Sandbox) . . . 5
1.4 Развертывание платформы с подключением к Mainnet . . . 13
1.5 Развертывание платформы в частной сети . . . 19
1.6 Примеры конфигурационных файлов ноды . . . 49
1.7 Инструментарий gRPC . . . 58
1.8 Методы REST API . . . 75
1.9 Разработка и применение смарт-контрактов . . . 161
1.10 JavaScript SDK . . . 186
1.11 Обмен конфиденциальными данными . . . 206
1.12 Управление ролями участников . . . 208
1.13 Подключение и удаление нод . . . 208
1.14 Запуск ноды с созданным снимком данных . . . 209
1.15 Архитектура . . . 210
1.16 Протокол работы блокчейна Waves-NG . . . 213
1.17 Неизменяемость данных в блокчейне . . . 215
1.18 Токены блокчейн-платформы Waves Enterprise . . . 216
1.19 Подключение новой ноды к сети . . . 216
1.20 Активация функциональных возможностей . . . 218
1.21 Анкоринг . . . 220
1.22 Механизм создания снимка данных . . . 223
1.23 Смарт-контракты . . . 225
1.24 Транзакции блокчейн-платформы . . . 238
1.25 Атомарные транзакции . . . 299
1.26 Алгоритмы консенсуса . . . 300
1.27 Криптография . . . 308
1.28 Роли участников . . . 312
1.29 Клиент . . . 313
1.30 Генераторы . . . 327
1.31 Сервисы авторизации и подготовки данных . . . 330
1.32 Различия opensource и коммерческой версий блокчейн-платформы Waves Enterprise . . . 361
1.33 Внешние компоненты платформы . . . 363
1.34 Официальные ресурсы и контакты . . . 364
1.35 Словарь терминов . . . 364

1.36 Что нового в блокчейн-платформе Waves Enterprise 369

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

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 блокчейна.

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.

Для чего предназначена блокчейн-платформа Waves Enterprise?

Блокчейн-платформа Waves Enterprise позволяет решать широкий спектр задач:

- 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.

С частными проектами, реализованными на базе блокчейн-платформы Waves Enterprise, вы можете ознакомиться на нашем официальном сайте.

Глава 1

Содержание документации

1.1 Системные требования

На данный момент блокчейн платформа Waves Enterprise поддерживает операционные системы на базе Unix (например, популярные дистрибутивы Linux или MacOS). Эффективная работа платформы обеспечивается для следующих операционных систем:

- операционная система для серверов:
  - CentOS 6/7 (x64);
  - Debian 8/9/10 (x64);
  - Red Hat Enterprise Linux 6/7 (x86);
  - Ubuntu 18.04 (x64).
- операционные системы для рабочих станций:
  - Ubuntu 18.04 (x64) и выше;
  - macOS Sierra и выше.

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.

1.1.1 Требования к окружению для платформы Waves Enterprise

Важно: Платформа Waves Enterprise распространяется в формате docker-образа, поэтому нет необходимости устанавливать какое-либо ПО кроме Docker и Docker-compose и настраивать окружение. Docker позволяет развернуть из docker-образа docker-контейнер, который уже содержит Java, CryptoPro и другое необходимое ПО. Однако пользователь должен самостоятельно приобрести лицензии на это ПО у его производителя, а затем с помощью переменных окружения передать лицензии ноде.

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

Смотрите также

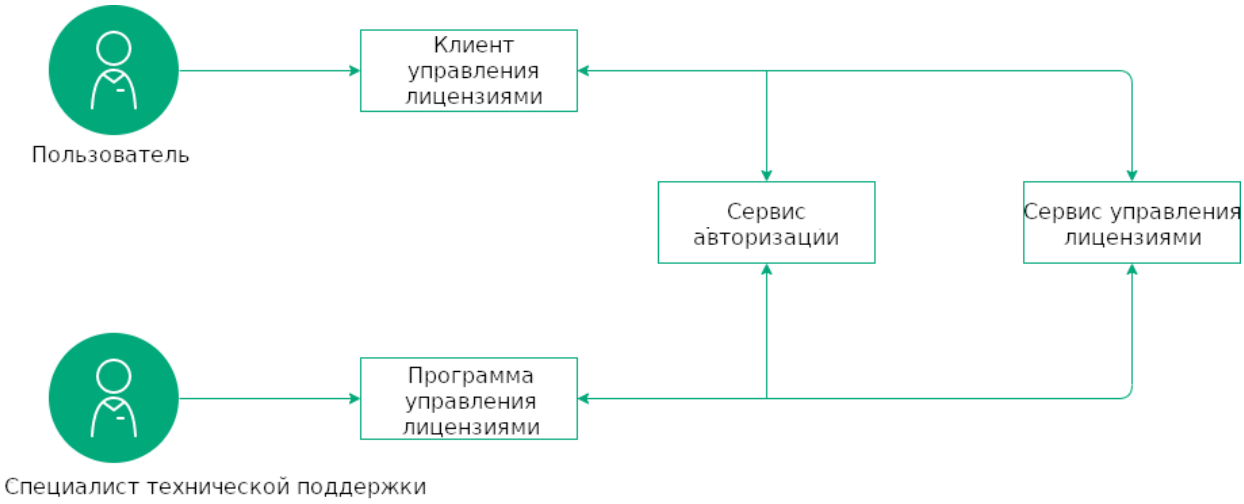
Внешние компоненты платформы

1.2 Лицензии блокчейн-платформы Waves Enterprise

Коммерческая версия блокчейн-платформы Waves Enterprise ориентирована на использование в корпоративном и государственном секторах и распространяется при помощи пользовательских лицензий.

Примечание: *Opensource* версия блокчейн-платформы Waves Enterprise не требует лицензии.

Схема получения лицензии на использование коммерческой версии платформы выглядит следующим образом:



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

1.2.1 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 платформу 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. Лицензия выдаётся бесплатно всем, кто выполнил условия для подключения, на 1 год. По истечении года держатель ноды должен запросить новую лицензию.

Каждый вид лицензии распространяется на одну ноду.

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.

1.2.2 Применение лицензии

После получения файла лицензии выполните следующие действия:

- Если нода не запущена, поместите файл лицензии в папку, путь к которой указан в параметре `license-file` конфигурационного файла ноды.
- Если нода запущена, скопируйте содержимое файла лицензии и передайте его ноде с помощью API метода *POST /licenses/upload*.

1.2.3 Сроки действия лицензий

Срок действия лицензии обговаривается при заключении договора.

Стандартный срок действия пробной лицензии составляет 3 месяца.

Лицензия для работы в сети Mainnet предоставляется на 1 год. По истечении года держатель ноды должен запросить новую лицензию.

Для остальных проектов лицензия выдается на любой срок по согласованию.

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

Смотрите также

Комиссии в сети Mainnet

1.3 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 блоков. 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.

1.3.1 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://raw.githubusercontent.com/waves-enterprise/we-node/release-1.12/node/src/
↪docker/docker-compose.yml
```

2. 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.
↪12.1
```

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:  OEtrVSL9gzjO87jYx-gIoQ
keypair password:   JInWk1kauuZDHGXFJ-rNXQ
API key:            we

node-1
blockchain address: 3Nxz6BYyk6CYrqH4Zudu5UYoHU6w7NXbZMs
public key:         VBkFFQmaHzv3YMiWLhh4qsCn4prUvteWsjgiiHEpWEp
keystore password:  FsUp3xiX_NF-bQ9gw6t0sA
keypair password:   Qf2rBgBT9pnozLPOkO1yYw
API key:            we

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

3. 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
```

1.3.2 Последующие действия

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␣
↪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"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    }

    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
  }
}
```

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

Полный листинг файла docker-compose.yml

5. 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**:

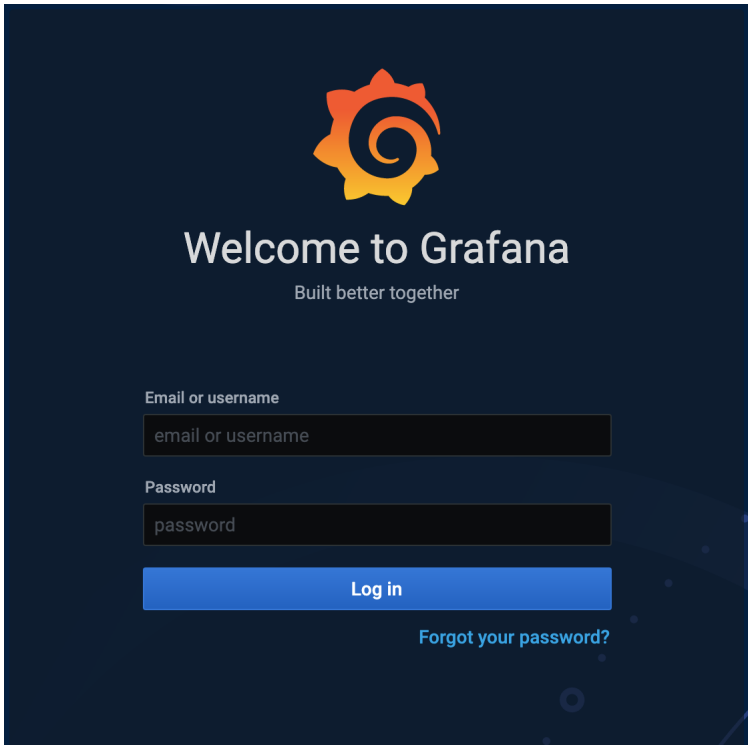
```
services:
  grafana:
    image: grafana/grafana:latest
    hostname: grafana
    container_name: grafana
    environment:
      GF_SECURITY_ADMIN_USER: 'admin'
      GF_SECURITY_ADMIN_PASSWORD: 'pass'
```

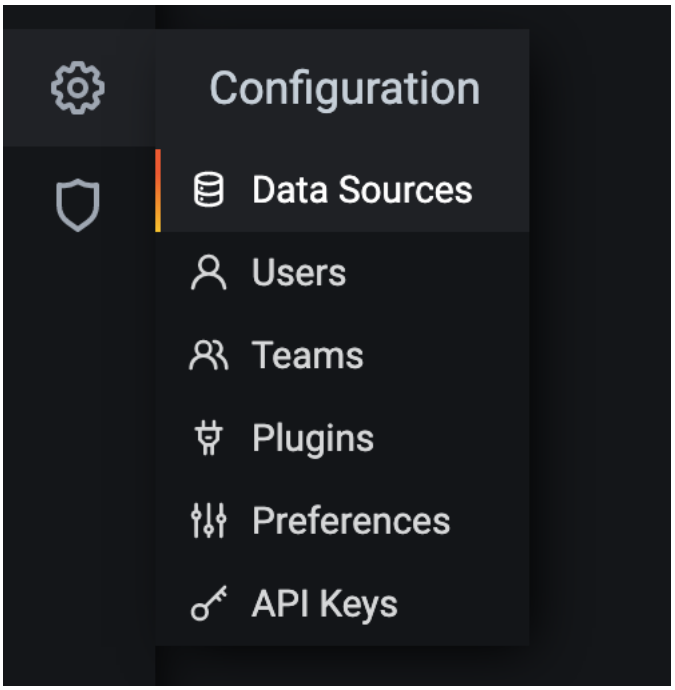
6. Подключите базу данных платформы к Grafana. Для этого нажмите на вкладку **Configuration** (Конфигурация) и перейдите в категорию **Data Sources** (Источники данных):

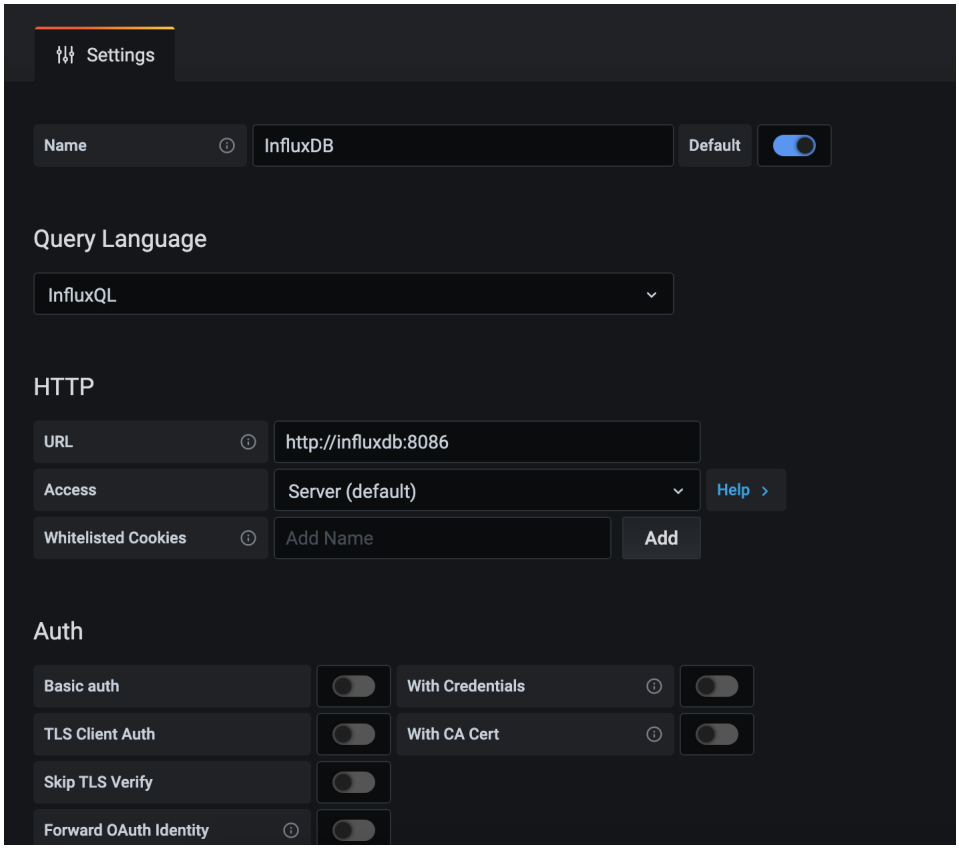
Добавьте новый источник данных, нажав на кнопку **Add Data Source**, и из предложенного списка выберите **InfluxDB**.

Затем настройте базу данных, используемую платформой.

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


Welcome to Grafana
Built better together
Email or username
email or username
Password
password
Log in
Forgot your password?


Configuration
Data Sources
Users
Teams
Plugins
Preferences
API Keys



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**.

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. Перей-

дите 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

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
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.
```

(continues on next page)

(продолжение с предыдущей страницы)

```
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␣
↪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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  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 развертывании 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

1.4 Развертывание платформы с подключением к Mainnet

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

Для подключения к Mainnet вам достаточно установить одну ноду.

Если вам необходимо развернуть сеть из нескольких нод с подключением к Mainnet, обратитесь в службу технической поддержки.

Лицензия для работы в сети Mainnet выдаётся бесплатно на 1 год всем, кто выполнил условия для подключения. По истечении года держатель ноды должен запросить новую лицензию.

1.4.1 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 кнопку **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.

1.4.2 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://raw.githubusercontent.com/waves-enterprise/we-node/release-1.12/node/src/
↪docker/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.
↪12.1
```

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

1.4.3 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.

Комиссии в сети Mainnet

В таблице ниже указаны размеры комиссий, которые взимаются с пользователей за транзакции в сети Waves Enterprise Mainnet.

Номер транзакции	Название транзакции	Комиссия	Описание
1	*Genesis transactio*	отсутствует	Первоначальная привязка баланса к адресам создаваемых при старте блокчейна нод
3	*Issue Transactic*	1 WEST	Выпуск токенов. Комиссия взимается только в WEST
4	*Transfer Transactic*	0.01 WEST	Перевод токенов
5	*Reissue Transactic*	1 WEST	Перевыпуск токенов
6	*Burn Transactic*	0.05 WEST	Сжигание токенов
8	*Lease Transactic*	0.01 WEST	Передача токенов в аренду
9	*Lease Cancel Transactic*	0.01 WEST	Отмена аренды токенов
10	*Create Alias Transactic*	1 WEST	Создание псевдонима
11	*MassTran Transactic*	0.05 WEST	Массовый перевод токенов. Указана минимальная комиссия, размер комиссии зависит от количества адресов в транзакции. Чтобы узнать точный размер комиссии, используйте REST метод *POST /transactions/calculateFee*
12	*Data Transactic*	0.05 WEST за килобайт	Транзакция с данными в виде полей с парой ключ-значение. Комиссия всегда взимается с автора транзакции. Указана минимальная комиссия; размер комиссии зависит от размера данных. Чтобы узнать точный размер комиссии, используйте REST метод *POST /transactions/calculateFee*
13	*SetScript Transactic*	0.5 WEST	Транзакция, привязывающая скрипт с RIDE-контрактом к аккаунту
14	*Sponsorsh Transactic*	1 WEST	Установка или отмена спонсорства
15	*SetAssetS*	1 WEST	Транзакция, привязывающая скрипт с RIDE-контрактом к ассету
101	*Genesis Permissior Transactic*	отсутствует	Назначение первого администратора сети для дальнейшей раздачи прав
102	*Permissior Transactic*	0.01 WEST	Выдача/отзыв прав у аккаунта
103	*CreateCor Transactic*	1 WEST	Создание Docker-контракта
104	*CallContra Transactic*	0.1 WEST	Вызов Docker-контракта
105	*ExecutedC Transactic*	отсутствует	Выполнение Docker-контракта
106	*DisableCo Transactic*	0.01 WEST	Отключение Docker-контракта
107	*UpdateCo Transactic*	1 WEST	Обновление Docker-контракта
110	*GenesisRe Transactic*	отсутствует	Регистрация ноды в генезис-блоке при старте блокчейна
111	*RegisterN Transactic*	0.01 WEST	Регистрация новой ноды в сети
112	*CreatePol Transactic*	1 WEST	Создание группы доступа к конфиденциальным данным
113	*UpdatePo*	0.5	Изменение группы доступа

Смотрите также

GET /licenses

Развертывание платформы с подключением к Mainnet

Обновление ноды в Mainnet

С выходом каждого нового релиза платформы мы рекомендуем обновлять ноды, подключенные к блокчейн-сети Waves Enterprise 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 successfully`, и запуск ноды будет продолжен.

Внимание: Если вы не используете Docker Compose, то для получения инструкций по обновлению ноды свяжитесь со службой технической поддержки.

Смотрите также

Развертывание платформы с подключением к Mainnet

Mainnet: устранение ошибок

Комиссии в сети Mainnet

Mainnet: устранение ошибок

При развертывании платформы с подключением к Mainnet возможно возникновение подобных ошибок на этапе развертывания 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␣
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪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

Генераторы

Лицензии блокчейн-платформы Waves Enterprise

Содержание

- *Развертывание платформы в частной сети*
  - *Создание аккаунта ноды*
  - *Настройка платформы для работы в частной сети*
  - *Получение лицензии для работы в частной сети*
  - *Подписание genesis-блока и запуск сети*
  - *Привязка Клиента к частной сети*

1.5 Развертывание платформы в частной сети

Если ваш проект или решение требует независимого блокчейна, вы можете развернуть собственную блокчейн-сеть на базе платформы Waves Enterprise. Обратитесь в службу технической поддержки, и специалисты компании помогут вам сконфигурировать поставку платформы под нужды вашего 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.

Примечание: Порядок создания аккаунтов нод, подписания genesis-блока и запуска сети в *коммерческой версии платформы* при использовании ГОСТ криптографии с PKI отличается от описанного в этом разделе. Этот порядок представлен в документации к коммерческой версии патформы. За более подробной информацией обратитесь в отдел продаж Waves Enterprise по электронной почте: sales@wavesenterprise.com.

1.5.1 Создание аккаунта ноды

Создайте аккаунты для каждой ноды вашей будущей сети.

Аккаунт ноды включает в себя адрес и ключевую пару – публичный и приватный ключи.

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

Адрес и публичный ключ будут показаны в командной строке во время создания аккаунта при помощи утилиты **generator**. 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 **generator-x.x.x.jar**, где x.x.x – номер релиза блокчейн-платформы. Затем запустите его, введя в качестве аргумента файл `accounts.conf`:

```
java -jar generator-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.

1.5.2 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 хэшу ключевой строки `api-key`. Принципы работы с этим конфигурационным файлом будут рассмотрены при настройке метода авторизации ноды.

Примечание: Параметры конфигурации ноды можно записать в одном файле либо в нескольких файлах, включая один файл в другой, например:

```
include required(file("network.conf"))
include required(file("local.conf"))
```

Таким образом можно вынести в один файл общие для всех нод параметры, а уникальные параметры ноды (например, `owner-address`) задать в отдельном файле для каждой 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 криптографии, консенсуса, исполнения смарт-контрактов Docker и майнинга. Все необходимые для этого параметры располагаются в файле **node.conf**.

Установка и использование платформы

Общая настройка платформы: настройка криптографии

Тип и параметры используемого в блокчейне криптографического алгоритма задаются в разделе crypto конфигурационного файла ноды. Раздел crypto считывается для инициализации криптографии, которая происходит до чтения полного конфигурационного файла ноды.

```
crypto {
  # Possible values: [WAVES, GOST]
  type = WAVES
  pki {
     # Possible values: [OFF, ON, TEST]
     # Could be enabled with GOST crypto type only
     mode = OFF
     required-oids = []
     crl-checks-enabled = false
  }
}
```

- type – тип *криптографии*; доступны значения WAVES для использования алгоритмов криптографии Waves и GOST для ГОСТ-криптографии с PKI. Если в конфигурационном файле присутствует параметр waves-crypto, и он имеет значение yes, то параметру type присваивается значение WAVES; если параметр waves-crypto имеет значение no, то параметру type присваивается значение GOST;
- pki – группа полей *настройки PKI*:
  - mode – допустимые значения: on, off, test; значения on и test допустимы только в случае, если параметр waves-crypto отсутствует или имеет значение no, а параметр type имеет значение GOST. Если параметру mode задано значение on, то выполняется проверка того, что TLS включён на сетевом уровне, то есть параметр node.network.tls имеет значение true.
  - required-oids – для дополнительного разграничения доступа возможно применять OID. Для этого в поле required-oids укажите список значений (whitelist-список идентификаторов OID), наличие которых нода будет проверять в расширении (поле) ExtendedKeyUsage сертификата. Этот список позволяет выделить из множества пользователей, выпустивших сертификат на одном и том же удостоверяющем центре (УЦ), тех пользователей, которым этот УЦ выдал OID специально для работы с блокчейн платформой. Список может быть пустым. Если список не пуст, то он должен представлять собой массив строк, состоящих из цифр, разделенных точками, и соответствовать стандартному формату OID. Например:

    ```
    required-oids = ["1.3.6.1.4.1.8.1.1","1.3.6.1.4.1.9.2.2"]
    ```

    Поле не является обязательным и может отсутствовать в конфигурационном файле ноды.
  - crl-checks-enabled – включение или отключение проверки списка отозванных сертификатов (CRL) при валидации сертификатов. Если параметру задано значение true, то криптопровайдер проверяет в удостоверяющем центре (УЦ), отозван сертификат или нет. Нода, которая синхронизируется с сетью, проверяет весь леджер, чтобы удостовериться в его целостности, то есть в корректности ЭП каждого блока. При проверке сертификатов нода использует списки CRL, валидные на момент подписания блока.

Если нода находилась вне сети какое-то время, или новая нода подключается к сети, то она запрашивает у других нод скачанные ранее CRL.

Важно: Группа полей `pki` используется только с ГОСТ криптографией (то есть когда полю `type` присвоено значение `GOST`). При использовании waves криптографии (то есть когда полю `type` присвоено значение `WAVES`) этой группы полей не должно быть в конфигурационном файле ноды. Если параметры PKI не указаны, то PKI отключен.

Примечание: Поле `node.waves-crypto` со значениями `yes` и `no` по-прежнему поддерживается, но в следующих версиях платформы планируется отказаться от его использования. Вместо него будет использоваться поле `type` в разделе `crypto`.

Смотрите также

Развертывание платформы в частной сети

Криптография

Установка и использование платформы

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*.

Примечание: При использовании ГОСТ криптографии с PKI алгоритмы консенсуса PoS и PoA могут использоваться только в тестовой версии Платформы «Waves Enterprise», то есть когда полю `crypto.type` задано значение `GOST`, а полю `crypto.pki.mode` – значение `TEST`.

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, необходимо только настроить работу майнинга с PoS в блоке `genesis`:

```
consensus {
   type = "pos"
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
...

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 алгоритма консенсуса 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
}
```

По сравнению с PoA для CFT предусмотрены следующие дополнительные параметры конфигурации, необходимые 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, в течение какого 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`: (`round-duration` + `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: `sync-duration` * 2, не может превышать `sync-duration` + `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

Общая настройка платформы: настройка исполнения смарт-контрактов

Для работы со *смарт-контрактами* нода использует два типа соединения, для каждого из которых можно настроить TLS:

1. Соединение с docker-хостом – удалённой машиной, на которой запускаются смарт-контракты. На этой машине используется docker-библиотека, которая обращается на сокет по своим протоколам. Для неё можно включить опцию безопасного соединения, которое в этой документации обозначается как «docker-TLS». Соединение docker-TLS настраивается в секции `node.docker-engine.docker-tls` конфигурационного файла ноды; эта настройка описана ниже в этом разделе;
2. Соединение, которое открывает запущенный смарт-контракт в сторону ноды по протоколу gRPC. Это подключение по API, так как точка подключения смарт-контракта к ноде такая же, как и для любого другого пользователя или приложения. Этот API настраивается в секции `node.api.grpc`, в частности для него можно *настроить TLS*. Пример такой настройки дан в разделе *Примеры конфигурационных файлов ноды*.

Примечание: Протокол TLS недоступен в *opensource* версии платформы.

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
  use-node-docker-host = yes
  # 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"
    }
  ]
  check-registry-auth-on-startup = no
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  # default-registry-domain = "registry.wavesenterprise.com"
  contract-execution-messages-cache {
    expire-after = 60m
    max-buffer-size = 10
    max-buffer-time = 100ms
    utx-cleanup-interval = 1m
    contract-error-quorum = 2
  }
  contract-auth-expires-in = 1m
  grpc-server {
    # host = "192.168.97.3"
    port = 6865
    }
  remove-container-on-fail = yes
  docker-tls {
    tls-verify = yes
    cert-path = "/node/certificates"
    }
  contracts-parallelism = 8
}
```

- `enable` – включение обработки транзакций для Docker-контрактов.
- `use-node-docker-host` – задайте параметру значение `yes`, чтобы определить IP-адрес gRPC API, доступного контрактам. При этом IP-адрес будет считан из файла `/etc/hosts` внутри контейнера ноды. Также для того чтобы контракты могли обращаться к ноде, их контейнеры при создании будут присоединены к той же docker сети (docker network), в которой создан контейнер ноды.
- `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-репозиториев и настройки авторизации к ним.
- `check-registry-auth-on-startup` – проверка авторизации для Docker-репозиториев 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` – секция настроек 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.
- `utx-cleanup-interval` – интервал, по прошествии которого невалидные транзакции (со статусом Error) удаляются из UTX-пула ноды, которая не является майнером. Значение по умолчанию – 1m.
- `contract-error-quorum` – минимальное количество полученных от разных нод-майнеров сообщений, в которых статус транзакции по вызову смарт-контракта содержит бизнес-ошибку (Error); когда указанное в параметре количество сообщений получено, транзакция удаляется из UTX-пула ноды, которая не является майнером. Значение по умолчанию – 2.
- `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`.
- `tls-verify` – включение или выключение TLS; если указано значение `yes`, то выполняется поиск сертификатов в директории, указанной в `certs-path`; если указано значение `no`, то поиск сертификатов не выполняется.
- `certs-path` – путь до директории с сертификатами для TLS; по умолчанию параметр имеет значение `{node.directory}/certificates`.
- `contracts-parallelism` – параметр определяет количество *параллельно выполняемых транзакций всех контейнеризированных смарт-контрактов*. По умолчанию параметр имеет значение 8.

Смотрите также

Тонкая настройка платформы: настройка TLS

Развертывание платформы в частной сети

Разработка и применение смарт-контрактов

Общая настройка платформы: настройка консенсуса

Общая настройка платформы: настройка майнинга

Смарт-контракты

Установка и использование платформы

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
  no-quorum-mining-delay = 5s
  micro-block-interval = 5s
  min-micro-block-age = 3s
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  max-transactions-in-micro-block = 500
  max-block-size-in-bytes = 1048576
  min-micro-block-age = 6 s
  minimal-block-generation-offset = 200ms
  pullin-buffer-size = 100
  utx-check-delay = 1s
}
```

- `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).
- `pulling-buffer-size` – размер буфера транзакций. Чем выше значение параметра, тем дольше группируются транзакции.
- `utx-check-delay` – задержка проверки UTX-пула (есть ли в пуле транзакции или он пуст) майнером. По умолчанию используется значение 1 с. Значение параметра должно быть больше либо равно 100 мс.

Настройки майнинга зависят от планируемого в вашей сети размера транзакций.

Настройки майнинга и алгоритм консенсуса

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.

Настройки UTX

В пуле неподтвержденных транзакций (UTX) предусмотрен механизм реброадкастинга, который позволяет сети быстрее восстановиться в случае возникновения каких-либо сбоев — например, при потере сетевой связности между нодами. В таких случаях транзакции, отправленные в одну ноду, могут оказаться не распространёнными. Механизм реброадкастинга решает такие проблемы, периодически проверяя актуальность транзакций, лежащих у ноды в UTX.

Этот механизм через заданный в параметре `interval` промежуток времени проверяет все транзакции в UTX; затем он повторно отправляет своим пирам те транзакции, дата создания которых отличается от текущей более чем на период, заданный в параметре `threshold`.

Параметры UTX задаются в разделе `utx` конфигурационного файла ноды:

```
utx {
   memory-limit=100Mb
   rebroadcast-threshold=5m
   rebroadcast-interval=5m
   }
```

- `memory-limit` – максимальный размер UTX-пула; при подсчёте размера UTX-пула учитывается не итоговый размер транзакций в памяти, а только сериализованный вид;
- `rebroadcast-threshold` – когда после создания транзакции проходит указанное в параметре время, транзакция считается «старой» и подлежит повторной отправке (ребродкастингу); значение параметра по умолчанию – `5m`;
- `rebroadcast-interval` – интервал запуска механизма реброадкастинга «старых» транзакций; значение параметра по умолчанию – `5m`.

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, настройка групп доступа к конфиденциальным данным и так далее. Эти настройки могут потребоваться вам в случае изменения предустановленных параметров для конфигурации вашего оборудования или ПО.

Все необходимые параметры также располагаются 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*.

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 из пакета generator-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 {
  crypto {
    type = GOST
    pki {
        mode = ON
        required-oids = ["1.2.3.4.5.6.7.8.9.10.11"]
    }
  }
  api-key = "some string for api-key"
  file = ${user.home}"/apikeyhash.out"
}
```

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

Параметр `file` позволяет указать имя файла, в который будет сохранён хэш. Параметр является опциональным. Если он не указан, то хэш выводится в консоль.

Примечание: Поле `waves-crypto` со значениями `yes` и `no` по-прежнему поддерживается, но в следующих версиях платформы планируется отказаться от его использования. Вместо него используйте поле `type` в разделе `crypto`.

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

```
java -jar generator-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

Авторизация методов PrivacyEventsService и PrivacyPublicService

Авторизация методов группы Privacy

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

    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

    # Parameters for internal gRPC services. Recommended to be left as is.
    services {
            blockchain-events {
              max-connections = 5
              history-events-buffer {
                enable: false
                size-in-bytes: 50MB
              }
            }

            privacy-events {
              max-connections = 5
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            history-events-buffer {
              enable: false
              size-in-bytes: 50MB
            }
          }

          contract-status-events {
            max-connections = 5
          }
        }
}
```

Блок `rest { }`

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

- `enable` – активация опции REST API на ноде. 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`. Для включения требуется *настройка TLS ноды*.

> **Примечание:** Протокол TLS недоступен в *opensource* версии платформы.

- `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 ноде.
- `bind-address` – сетевой адрес ноды, на котором будет доступен gRPC-интерфейс.
- `port` – порт прослушивания gRPC запросов.
- `tls` – включение/отключение TLS для gRPC запросов. Включение – `yes`, отключение – `no`. Для включения требуется *настройка TLS ноды*.

> **Примечание:** Протокол TLS недоступен в *opensource* версии платформы.

Секция `services{ }` содержит настройки публичных gRPC-сервисов, собирающих данные из компонентов платформы:

- `blockchain-events` – сервис сбора данных о событиях блокчейн-сети;
- `privacy-events` – сервис сбора данных о событиях, связанных с группами доступа к конфиденциальным данным;
- `contract-status-events` – сервис сбора данных о состоянии смарт-контрактов.

В этой секции рекомендуется использовать предустановленные параметры.

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

Тонкая настройка платформы: настройка TLS

Для работы со смарт-контрактами нода использует два типа соединения, для каждого из которых можно настроить TLS: *docker-TLS и подключение по API*.

Примечание: Протокол TLS недоступен в *opensource* версии платформы.

Настроить TLS для gRPC и REST API для каждой ноды можно с помощью параметров работы gRPC и REST API в секции `api` конфигурационного файла ноды. Для настройки TLS используйте параметр `TLS` *в блоке rest* и в блоке *в блоке grpc*.

Для работы с TLS для API необходимо:

1. *включить TLS в секции node.api.grpc конфигурационного файла ноды*;
2. получить артефакты TLS:
   - получить файл keystore с именем `we.jks`;
   - выпустить клиентский сертификат `we.cert`;
   - импортировать клиентский сертификат в хранилище доверенных сертификатов.

   Пример подготовки этих артефактов представлен в следующем разделе:

Пример подготовки артефактов для TLS

Если вы планируете *использовать TLS*, то в рамках настройки инфраструктуры нужно настроить параметры TLS.

Для работы с TLS для API необходимо получить файл keystore. Ниже представлен пример использования для этого 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 конфигурационном файле ноды в секции `keystore-password`;
- `keypass` – пароль от приватного ключа, указывается в конфигурационном файле ноды 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 ноде 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
```

Смотрите также

Тонкая настройка платформы: настройка TLS

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

3. указать относительный путь к файлу keystore `we.jks` в секции `tls` файла конфигурации ноды.

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

Секция `tls` конфигурационного файла ноды

Секция `tls` содержит следующие параметры:

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

Пример подготовки артефактов для TLS

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

Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным

Если вы используете API-методы **privacy** для управления конфиденциальными данными, настройте параметры доступа к этим данным в конфигурационном файле ноды. Для этого предназначена секция `privacy`. Ниже представлен пример настройки с использованием БД PostgreSQL:

```
privacy {

  replier {
    parallelism = 10
    stream-timeout = 1 minute
    stream-chunk-size = 1MiB
  }

  synchronizer {
    request-timeout = 2 minute
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    init-retry-delay = 5 seconds
    inventory-stream-timeout = 15 seconds
    inventory-request-delay = 3 seconds
    inventory-timestamp-threshold = 10 minutes
    crawling-parallelism = 100
    max-attempt-count = 24
    lost-data-processing-delay = 10 minutes
    network-stream-buffer-size = 10
  }

  inventory-handler {
    max-buffer-time = 500ms
    max-buffer-size = 100
    max-cache-size = 100000
    expiration-time = 5m
    replier-parallelism = 10
  }

  cache {
    max-size = 100
    expire-after = 10m
  }

  storage {
    vendor = postgres
    schema = "public"
    migration-dir = "db/migration"
    profile = "slick.jdbc.PostgresProfile$"
    upload-chunk-size = 1MiB
    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
    }
  }

  service {
    request-buffer-size = 10MiB
    meta-data-accumulation-timeout = 3s
  }
}
```

Выбор базы данных

Перед изменением конфигурационного файла ноды выберите базу данных, которую планируете использовать для хранения конфиденциальных данных. Блокчейн-платформа Waves Enterprise поддерживает взаимодействие с БД PostgreSQL и Amazon S3.

PostgreSQL

Во время установки БД под управлением PostgreSQL вы создадите аккаунт для доступа к БД. Заданные при этом логин и пароль затем необходимо будет указать в конфигурационном файле ноды (в полях `user` и `password` блока `storage` секции `privacy`, подробнее см. раздел *vendor = postgres*).

Для использования СУБД PostgreSQL потребуется установка JDBC-интерфейса (Java DataBase Connectivity). При установке JDBC, задайте имя профиля. Это имя затем необходимо будет указать в конфигурационном файле ноды (в поле `profile` блока `storage` секции `privacy`, подробнее см. раздел *vendor = postgres*).

В целях оптимизации подключение к PostgreSQL может осуществляться через инструмент pgBouncer. В этом случае pgBouncer требует особой настройки, которая описана ниже в разделе *storage-pgBouncer*.

Amazon S3

При использовании Amazon S3 информация должна храниться на сервере Minio. В процессе установки сервера Minio вам будет предложено задать логин и пароль для доступа к данным. Эти логин и пароль затем необходимо будет указать в конфигурационном файле ноды (в полях `access-key-id` и `secret-access-key`, подробнее см. раздел *vendor = s3*).

После установки подходящей для вашего проекта СУБД измените блок `storage` секции `privacy` конфигурационного файла ноды, как описано ниже.

Блок `storage`

В блоке `storage` секции `privacy` укажите используемую вами СУБД в параметре `vendor`:

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

Важно: Если вы не используете API-методы **privacy**, в параметре `vendor` укажите значение `none` и закомментируйте или удалите остальные параметры в секции `privacy`.

`vendor = postgres`

При использовании СУБД PostgreSQL блок `storage` секции `privacy` выглядит следующим образом:

```
storage {
  vendor = postgres
  schema = "public"
  migration-dir = "db/migration"
  profile = "slick.jdbc.PostgresProfile$"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  upload-chunk-size = 1MiB
  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
   }
}
```

В блоке должны быть указаны следующие параметры:

- `schema` – используемая схема взаимодействия между элементами в рамках БД; по умолчанию применяется схема `public`; если в вашей БД предусмотрена иная схема, то укажите ее название;
- `migration-dir` – директория для миграции данных;
- `profile` – имя профиля для доступа к JDBC, заданное при установке JDBC (см. раздел *PostgreSQL*);
- `upload-chunk-size` – размер фрагмента данных, загружаемых с помощью REST API метода *POST /privacy/sendLargeData* или gRPC API метода *SendLargeData*;
- `url` – адрес БД PostgreSQL; подробнее см. *Поле url*;
- `driver` – имя драйвера JDBC, позволяющим Java-приложениям взаимодействовать с БД;
- `user` – имя пользователя для доступа к БД; укажите логин созданного вами аккаунта для доступа к БД под управлением *PostgreSQL*;
- `password` – пароль для доступа к БД; укажите пароль созданного вами аккаунта для доступа к БД под управлением *PostgreSQL*;
- `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 запрос `SELECT 1`;
- `queueSize` – размер очереди запросов;
- `numThreads` – количество одновременных подключений к БД.

Поле `url`

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

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

, где

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

Можно 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=7nZL7Jr41qOWUHz5qKdypA&sslmode=require"
    ...
    }
}
```

В этом примере user_privacy_node_0@company – имя пользователя, 7nZL7Jr41qOWUHz5qKdypA – его пароль. Также вы можете использовать команду sslmode=require для требования использования ssl при авторизации.

pgBouncer

Для оптимизации работы с базой данных PostgreSQL используется pgBouncer – инструмент, через который осуществляется подключение к базе данных PostgreSQL. pgBouncer настраивается в отдельном конфигурационном файле данного инструмента – **pgbouncer.ini**. В связи с тем, что pool_mode = transaction режим в настройке pgBouncer не поддерживает подготовленные операторы на стороне сервера, в целях предотвращения потери данных мы рекомендуем использовать pool_mode с session режимом в настройках файла **pgbouncer.ini**. При использовании сессионного режима следует задавать параметр server_reset_query со значением DISCARD ALL.

```
[pgbouncer]
pool_mode = session
server_reset_query = DISCARD ALL
```

Больше информации о работе сессионного режима с подготовленными операторами можно найти в официальной документации к pgBouncer.

vendor = s3

При использовании СУБД Amazon S3, блок storage секции privacy выглядит следующим образом:

```
storage {
  vendor = s3
  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
  upload-chunk-size = 5MiB
}
```

- 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; укажите логин для доступа к данным, который вы задали в процессе установки сервера Minio (см. раздел *Amazon S3*);
- `secret-access-key` – ключ доступа к данным в хранилище S3; укажите пароль для доступа к данным, который вы задали в процессе установки сервера Minio (см. раздел *Amazon S3*);
- `path-style-access-enabled = true` – путь к таблице S3 – неизменяемый параметр;
- `connection-timeout` – период бездействия до разрыва соединения (в секундах);
- `connection-acquisition-timeout` – период бездействия при установлении соединения (в секундах);
- `max-concurrency` – максимальное число параллельных обращений к хранилищу;
- `read-timeout` – период бездействия при чтении данных (в секундах);
- `upload-chunk-size` – размер фрагмента данных, загружаемых с помощью REST API метода *POST /privacy/sendLargeData* или gRPC API метода *SendLargeData*.

Блок `replier`

В блоке `replier` секции `privacy` укажите параметры потоковой передачи конфиденциальных данных:

```
replier {
  parallelism = 10
  stream-timeout = 1 minute
  stream-chunk-size = 1MiB
}
```

В блоке должны быть указаны следующие параметры:

- `parallelism` – максимальное количество параллельных задач обработки запросов конфиденциальных данных;
- `stream-timeout` – максимальное время выполнения операции чтения потока данных (стрима);
- `stream-chunk-size` – размер фрагмента данных при передаче данных в виде потока (стрима).

Блок `inventory-handler`

В блоке `inventory-handler` секции `privacy` укажите параметры сбора инвентаризационной информации (privacy inventory) конфиденциальных данных:

```
inventory-handler {
  max-buffer-time = 500ms
  max-buffer-size = 100
  max-cache-size = 100000
  expiration-time = 5m
  replier-parallelism = 10
}
```

В блоке должны быть указаны следующие параметры:

- `max-buffer-time` – максимальное время накопления данных в буфере; по истечении указанного времени нода пакетно обрабатывает всю инвентаризационную информацию (privacy inventory);

- `max-buffer-size` – максимальное количество инвентаризационной информации в буфере; когда лимит достигнут, нода пакетно обрабатывает всю инвентаризационную информацию;
- `max-cache-size` – максимальный размер кэша инвентаризационной информации; используя этот кэш, нода выбирает только новую инвентаризационную информацию;
- `expiration-time` – время, когда истекает срок действия элементов кэша (инвентаризационной информации);
- `replier-parallelism` – максимальное количество параллельно выполняемых задач обработки запросов инвентаризационной информации.

Блок `cache`

В блоке `cache` секции `privacy` укажите параметры кэша ответов конфиденциальных данных:

```
cache {
  max-size = 100
  expire-after = 10m
}
```

Примечание: Большие файлы (файлы, загружаемые с помощью REST API метода *POST /privacy/sendLargeData* или gRPC API метода *SendLargeData*) не подлежат кешированию.

В блоке должны быть указаны следующие параметры кэша:

- `max-size` – максимальное количество элементов;
- `expire-after` – время, по истечении которого заканчивается срок действия элементов кэша, которые не получили доступ.

Блок `synchronizer`

В блоке `synchronizer` секции `privacy` укажите параметры синхронизации конфиденциальных данных:

```
synchronizer {
  request-timeout = 2 minute
  init-retry-delay = 5 seconds
  inventory-stream-timeout = 15 seconds
  inventory-request-delay = 3 seconds
  inventory-timestamp-threshold = 10 minutes
  crawling-parallelism = 100
  max-attempt-count = 24
  lost-data-processing-delay = 10 minutes
  network-stream-buffer-size = 10
}
```

В блоке должны быть указаны следующие параметры:

- `request-timeout` – максимальное время ожидания ответа после запроса данных; значение по умолчанию – `2 minute`;
- `init-retry-delay` – пауза после неудачной попытки; с каждой попыткой задержка увеличивается на 4/3; значение по умолчанию – `5 seconds`;

- `inventory-stream-timeout` – максимальное время ожидания сетевого сообщения с инвентаризационной информацией (privacy inventory), т.е. подтверждения от конкретной ноды, что у нее есть определенные данные, и она может их предоставить для загрузки. По истечении этого таймаута нода опрашивает всех пиров (рассылает inventory-request), есть ли у них необходимые для загрузки данные; значение по умолчанию – `15 seconds`;
- `inventory-request-delay` – задержка после запроса инвентарных данных у пиров (inventory-request); значение по умолчанию – `3 seconds`;
- `inventory-timestamp-threshold` – параметр используется для принятия решения, отправлять ли PrivacyInventory сообщение при успешной синхронизации (загрузке) данных; значение по умолчанию – `10 minutes`;
- `crawling-parallelism` – максимальное количество параллельно выполняемых задач *краулера* – компонента, который собирает конфиденциальные данные у пиров; значение по умолчанию – 100;
- `max-attempt-count` – количество попыток, которые предпримет *краулер*, прежде чем данные будут помечены как потерянные; значение по умолчанию – 24;
- `lost-data-processing-delay` – задержка между попытками обработки очереди потерянных данных; значение по умолчанию – `10 minutes`;
- `network-stream-buffer-size` – максимальное количество фрагментов данных в буфере; когда указанное количество достигнуто, активируется обратное давление; значение по умолчанию – 10.

Поле `inventory-timestamp-threshold`

Нода отправляет пирам сообщение PrivacyInventory после того, как она загружает в своё приватное хранилище данные по определенному хэшу данных, то есть успешно проводит синхронизацию данных. Для хранения PrivacyInventory используется кэш, ограниченный по количеству объектов и времени их нахождения в кэше. В зависимости от значения параметра `inventory-timestamp-threshold` обработчик событий вставки данных принимает решение, нужно ли отправлять сообщение PrivacyInventory при загрузке данных. Обработчик сравнивает время транзакции (timestamp), которая соответствует данному хэшу данных, и текущее время на ноде. Если разница превышает значение параметра `inventory-timestamp-threshold`, то сообщения PrivacyIntentory не отправляются. Подобрав значение параметра `inventory-timestamp-threshold` можно избежать ситуации, когда нода, которая синхронизирует стейт с сетью, засоряет сеть лишними сообщениями PrivacyInventory.

Блок `service`

В блоке `service` секции `privacy` укажите параметры *gRPC метода SendLargeData* и *REST метода POST /privacy/sendLargeData* для отправки потока конфиденциальных данных.

```
service {
  request-buffer-size = 10MiB
  meta-data-accumulation-timeout = 3s
}
```

В блоке должны быть указаны следующие параметры:

- `request-buffer-size` – максимальный размер буфера запроса; когда указанный размер достигнут, активируется обратное давление;
- `meta-data-accumulation-timeout` – максимальное время, за которое должны быть обработаны метаданные при отправке данных через REST API метод *POST /privacy/sendLargeData*.

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

Обмен конфиденциальными данными

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 смайненный 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 ноде 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

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 – поддиректория `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

Тонкая настройка платформы: настройка ноды в режиме наблюдения

Нода блокчейна может быть настроена для работы в режиме наблюдения.

В этом режиме нода работает следующим образом:

- Нода-наблюдатель не получает и не отправляет неподтвержденные транзакции.
- Нода-наблюдатель не имеет возможности создавать новые блоки.
- Нода-наблюдатель не имеет возможности загружать и запускать смарт-контракты.
- UTX ноды-наблюдателя не синхронизируется с другими нодами.
- Нода-наблюдатель получает микроблоки, блоки и транзакции для обновления своего стейта.

Этот режим позволяет создать ноду, которая может получать актуальное состояние блокчейна, при этом не участвуя в майнинге и не перегружая сеть сообщениями.

Конфигурация

Для переключения ноды в режим наблюдения измените параметр `mode` в разделе `node.network` конфигурационного файла:

```
node {
...
  network {
    # ENUM: default or watcher
    mode = default
    ...
  }
}
```

- `default` - стандартный режим работы ноды;
- `watcher` - режим наблюдения.

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

Полные примеры конфигурационных файлов для настройки каждой ноды приведены *здесь*.

1.5.3 Получение лицензии для работы в частной сети

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

Примечание: Opensource-версия блокчейн-платформы Waves Enterprise не требует лицензии.

Лицензия для запуска ноды привязана к ключу владельца ноды. В самой лицензии прописан адрес ноды, для которого лицензия выпущена.

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

По результатам обсуждения вам будет прислан файл лицензии. Поместите этот файл в папку, путь к которой указан в параметре `license-file` конфигурационного файла ноды.

Перед развертыванием блокчейн-платформы ознакомьтесь с *системными требованиями*.

1.5.4 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 **generator**. В качестве аргумента она использует настроенный вами конфигурационный файл ноды `node.conf`:

```
java -jar generator-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 технической поддержки Waves Enterprise.

1.5.5 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

Лицензии блокчейн-платформы Waves Enterprise

Установка и использование платформы

1.6 Примеры конфигурационных файлов ноды

1.6.1 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;

- используются методы **privacy** с БД PostgreSQL для хранения конфиденциальных данных;
- функция периодического удаления невалидных транзакций из UTX-пула участника блокчейна, который не является майнером, настроена.
- настроена задержка проверки UTX-пула (есть ли в пуле транзакции или он пуст) майнером.

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. The field is deprecated since v1.9.0, use 'node.crypto.type =
↪waves | gost' instead.
  waves-crypto = yes

  crypto {
    # Possible values: [WAVES, GOST]
    type = WAVES
    pki {
      # Possible values: [OFF, ON, TEST]
      # Can be enabled with GOST crypto type only
      mode = OFF
      required-oids = []
    }
  }

  # 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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  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
      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 = [
```

(continues on next page)

(продолжение с предыдущей страницы)

```
          # 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: []},
      ]
    }
  }
}

# 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

  # ENUM: regular or watcher
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  mode = regular

  # 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
  # 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
  utx-check-delay = 100ms
}

# 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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  }

  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/ "
  }
}

#Settings for Privacy Data Exchange
privacy {

  replier {
    parallelism = 10
    stream-timeout = 1 minute
    stream-chunk-size = 1MiB
  }

  # Syncs private data.
  synchronizer {
    request-timeout = 2 minute
    init-retry-delay = 5 seconds
    inventory-stream-timeout = 15 seconds
    inventory-request-delay = 3 seconds
    inventory-timestamp-threshold = 10 minutes
    crawling-parallelism = 100
    max-attempt-count = 24
    lost-data-processing-delay = 10 minutes
    network-stream-buffer-size = 10
  }

  inventory-handler {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    max-buffer-time = 500ms
    max-buffer-size = 100
    max-cache-size = 100000
    expiration-time = 5m
    replier-parallelism = 10
  }

  cache {
    max-size = 100
    expire-after = 10m
  }

  storage {
    vendor = postgres

    # for postgres vendor:
    schema = "public"
    migration-dir = "db/migration"
    profile = "slick.jdbc.PostgresProfile$"
    upload-chunk-size = 1MiB
    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
    # upload-chunk-size = 5MiB
  }

  service {
    request-buffer-size = 10MiB
    meta-data-accumulation-timeout = 3s
  }
}
```

(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
    #The interval after which invalid transactions (with Error status) are removed from␣
↪the UTX pool of a non-miner node
    utx-cleanup-interval = 1m
    #The minimum number of transaction Error statuses received from other nodes, after␣
↪which the transaction is removed from the UTX pool of a non-miner node
    contract-error-quorum = 2
  }
 }
}
```

1.6.2 accounts.conf

В этом примере включено шифрование Waves Crypto, не используется PKI, используется 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 {
  crypto {
    type = WAVES
    pki {
      mode = OFF
      required-oids = []
    }
  }
  chain-id = T
  amount = 5
  wallet = ${user.home}"/node/wallet/wallet1.dat"
  wallet-password = "/FILL/"
  reload-node-wallet {
    enabled = false
    url = "http://localhost:6869/utils/reload-wallet"
  }
}
```

1.6.3 api-key-hash.conf

В этом примере включено ГОСТ шифрование.

api-key-hash.conf:

```
apikeyhash-generator {
  crypto {
    type = GOST
    pki {
        mode = ON
        required-oids = ["1.2.3.4.5.6.7.8.9.10.11"]
    }
  }
  api-key = "some string for api-key"
}
```

1.6.4 Дополнительные примеры

Дополнительные примеры конфигурационных файлов с комментариями приведены в официальном GitHub-репозитории Waves Enterprise.

Смотрите также

Развертывание платформы в частной сети

Генераторы

1.7 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.

Некоторые сервисы представлены в двух вариантах: для внешней интеграции (паблик сервисы) и для смарт-контрактов (контрактные сервисы). Используйте паблик сервисы для интеграции с WE. Контрактные сервисы не предназначены для вызова внешним пользователем, они имеют другую авторизацию и поведение. Контрактные сервисы упакованы в protobuf-файлы, размещенные в директории **contract** и описаны в разделе *Сервисы gRPC, используемые смарт-контрактом*. При использовании в смарт-контрактах эти методы требуют авторизации.

1.7.1 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**.

1.7.2 Для чего предназначен gRPC-интерфейс платформы

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

gRPC: отслеживание событий в блокчейне

gRPC-интерфейс предоставляет 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_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.

Важно: Типы данных полей для запросов и ответов указаны в protobuf-файлах.

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, указанными в виде списка;
- `FilterNot` – выводить события со всеми транзакциями кроме тех, которые указаны в этом параметре в виде списка.

connectionId – опциональный параметр, отправляемый для удобства идентификации запроса в логах ноды.

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` – при авторизации по OAuth, неверный JWT-токен.

Смотрите также

Инструментарий gRPC

gRPC: получение информации о ноде

Для получения параметров конфигурации ноды и данных о её владельце предусмотрен gRPC сервис **NodeInfoService**.

У сервиса **NodeInfoService** есть следующие методы, описанные в protobuf-файле **util_node_info_service.proto**:

- **NodeConfig**;
- **NodeOwner**.

Важно: Типы данных полей для запросов и ответов указаны в protobuf-файле.

---

gRPC: получение параметров конфигурации ноды

Используйте метод `NodeConfig` для получения параметров конфигурации ноды. Метод `NodeConfig` не требует ввода дополнительных параметров запроса. 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);
- `pki_mode` – при использовании ГОСТ криптографии с PKI выводится используемый режим PKI:
  - `ON` – PKI используется,
  - `OFF` – PKI не используется,
  - `TEST` – тестовый режим.
- `required_oids` – при использовании алгоритмов ГОСТ криптографии с PKI выводится список OID-строк пользователей, которым УЦ выдал OID специально для работы с блокчейн платформой. Подробнее об этом параметре см. раздел *Общая настройка платформы: настройка криптографии*.
- `pos_round info`– при использовании алгоритма консенсуса PoS, выводится значение параметра `average_block_delay` (время средней задержки создания блоков, в секундах), которое задано в *конфигурационном файле ноды*;
- `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.
- `crlChecksEnabled` – режим проверки списка отозванных сертификатов (CRL) при валидации сертификатов.

gRPC: получение данных о владельце ноды

Используйте метод `NodeOwner` для получения данных о владельце ноды. Метод `NodeOwner` не требует ввода дополнительных параметров запроса. В ответе выводятся следующие данные ноды, к которой был осуществлен запрос:

- `address` – адрес ноды;
- `public_key` – публичный ключ.

Смотрите также

Инструментарий gRPC

gRPC: получение информации о результатах исполнения вызова смарт-контракта

Для получения информации о результатах вызовов смарт-контрактов служит gRPC сервис **ContractStatusService**.

У сервиса есть два метода, описанных в protobuf-файле **util_contract_status_service.proto**:

- **ContractExecutionStatuses**,
- **ContractsExecutionEvents**.

Важно: Типы данных полей для запросов и ответов указаны в protobuf-файле.

Используйте метод **ContractExecutionStatuses** для получения информации о результатах исполнения вызова отдельного смарт-контракта. Метод принимает запрос **ContractExecutionRequest**, который требует

ввода параметра `tx_id` – идентификатора вызывающей транзакции смарт-контракта, информацию о состоянии которого необходимо получить.

Используйте метод **ContractsExecutionEvents** для подписки на поток (стрим) с результатами исполнения вызова всех смарт-контрактов. Метод не требует ввода входных параметров.

Информация о результатах исполнения вызова смарт-контракта

В ответе на запрос оба метода возвращают следующие данные смарт-контракта:

- `senderAddress` – адрес участника, который отправил смарт-контракт в блокчейн;
- `senderPublicKey` – публичный ключ участника, который отправил смарт-контракт в блокчейн;
- `tx_id` – идентификатор транзакции вызова смарт-контракта;
- `Status` – информация об исполнении смарт-контракта:
  - `0` – успешно исполнен (SUCCESS);
  - `1` – бизнес ошибка, контракт не исполнен, вызов отклонён (ERROR);
  - `2` – системная ошибка в ходе исполнения смарт-контракта (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

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 секунду.

Этот запрос не требует ввода дополнительных параметров и описан 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.

Важно: Типы данных полей для запросов и ответов указаны в protobuf-файлах.

Смотрите также

Инструментарий gRPC

gRPC: получение сертификатов

Для запроса у ноды сертификата из хранилища сертификатов предусмотрена группа методов gRPC сервиса **PkiPublicService**. Методы работы с сертификатом описаны в файле **pki_public_service.proto**.

Примечание: gRPC методы сервиса **PkiPublicService** недоступны в *opensource* версии платформы.

Методы этой группы позволяют получить сертификат по разным полям:

- *GetCertificateByDn(CertByDNRequest)* – по полю `DN` (distinguished name),
- *GetCertificateByDnHash(CertByDNHashRequest)* – по полю `DN Hash`,
- *GetCertificateByPublicKey(CertByPublicKeyRequest)* – по полю `publicKey`,
- *GetCertificateByFingerprint(CertByFingerprintRequest)* – по полю `fingerprint`.

В запросе эти методы принимают значение соответствующего поля сертификата и, опционально, параметр `plainText`, который задаёт формат ответа.

Если сертификат существует, то в ответе каждого из этих методов нода возвращает сертификат в формате DER (как он и записан в хранилище сертификатов ноды). Если в запросе метода параметру `plainText` задано значение `true`, то сертификат возвращается в формате plainText.

Если сертификата не существует, то в ответе каждого из этих методов возвращается ошибка.

Авторизация методов получения сертификатов

В случае API-KEY авторизация не требуется.

В случае OAuth2 авторизации требуется наличие роли `user` в JWT токене.

Получение сертификата по DN

Метод **GetCertificateByDn(CertByDNRequest)** возвращает сертификат по его отличительному имени (distinguished name), записанному в поле `DN`.

Получение сертификата по хэшу DN

Метод **GetCertificateByDnHash(CertByDNHashRequest)** возвращает сертификат по хэшу SHA-1 (Keccak) от поля `DN` сертификата.

Получение сертификата по публичному ключу

Метод **GetCertificateByPublicKey(CertByPublicKeyRequest)** возвращает сертификат по байтам публичного ключа (поле `publicKey`).

Получение сертификата по его отпечатку

Метод **GetCertificateByFingerprint(CertByFingerprintRequest)** возвращает сертификат по его SHA-1 отпечатку (поле `fingerprint`).

Смотрите также

Инструментарий gRPC

REST API: получение сертификатов

gRPC: работа с транзакциями

Для работы с транзакциями предусмотрен gRPC сервис **TransactionPublicService**.

У сервиса **TransactionPublicService** есть следующие методы, описанные в protobuf-файле **transaction_public_service.proto**:

- *Broadcast*;
- *BroadcastWithCerts*;
- *UtxInfo*;
- *TransactionInfo*;
- *UnconfirmedTransactionInfo*.

Важно: Типы данных полей для запросов и ответов указаны в protobuf-файлах.

Отправка транзакций в блокчейн

Выберите подходящий для вашей задачи метод отправки транзакций в блокчейн:

- `BroadcastWithCerts` – для отправки транзакции *RegisterNode*;
- `Broadcast` – для отправки всех остальных транзакций.

Broadcast

Метод требует ввода следующих параметров запроса:

- `version` – версия транзакции;
- `transaction` – название транзакции вместе с предназначенным для нее набором параметров.
- `certificates` – цепочка сертификатов байтами в формате DER; параметр является обязательным при одновременном соблюдении следующих условий:
  - используется PKI или тестовый режим PKI (то есть в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `TEST` или `ON`),
  - новый пользователь, который не является владельцем ноды (node-owner), делает свою первую транзакцию.

  В этом случае необходимо в запросе в поле `certificates` передать цепочку сертификатов пользователя; в других случаях поле `certificates` является необязательным.

Примечание: Поле `certificates` в запросе на публикацию транзакции *RegisterNode* является обязательным при использовании PKI или тестового режима PKI (то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON` или `TEST`. В этом случае поле `certificates` должно содержать цепочку сертификатов, которая соответствует публичному ключу в поле `target` транзакции.

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*.

BroadcastWithCerts

Метод используется для отправки транзакции *RegisterNode* и требует тех же входных параметров, что и метод *Broadcast*.

Поле `certificates` является обязательным и должно содержать цепочку сертификатов, которая соответствует публичному ключу в поле `target` транзакции.

Получение данных транзакции

Используйте метод `TransactionInfo`, чтобы получить данные транзакции.

Метод требует ввода одного параметра запроса:

- `tx_id` – ID транзакции, о которой запрашивается информация.

В ответе метода `TransactionInfo` содержится следующая информация о транзакции:

- `height` – высота блокчейна, на которой была произведена транзакция;
- `transaction` – название транзакции;

а также данные транзакции, аналогичные ответу метода `Broadcast`.

Получение данных транзакции, находящейся в UTX-пуле

Используйте метод `UnconfirmedTransactionInfo`, чтобы получить данные транзакции, находящейся в UTX-пуле. В ответе метода содержатся данные транзакции, аналогичные ответу метода `Broadcast`.

Смотрите также

Инструментарий gRPC

Описание транзакций

Комиссии в сети Mainnet

gRPC: работа с конфиденциальными данными

Для работы с *конфиденциальными данными (privacy)* предусмотрены gRPC сервисы **PrivacyEventsService** и **PrivacyPublicService**.

Важно: Методы для работы с конфиденциальными данными недоступны при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) методы можно использовать.

Авторизация методов PrivacyEventsService и PrivacyPublicService

:opticon:`report, size=24` Авторизация методов PrivacyEventsService и PrivacyPublicService:

Для использования методов gRPC API сервисов **PrivacyEventsService** и **PrivacyPublicService** требуется авторизация по api-key или JWT-токену. Авторизация методов реализована следующим образом:

- в случае api-key авторизации требуется PrivacyApiKey;
- в случае OAuth2 авторизации требуется наличие роли Privacy в JWT токене.

Для каждого из методов необходимо передавать следующие данные:

- Recipients — userAuth;
- Owners — userAuth;
- Hashes — userAuth;
- GetPolicyItemData — privacyAuth;
- GetPolicyItemInfo — privacyAuth;
- SendData — privacyAuth;
- SendLargeData — privacyAuth,
- forceSync — privacyAuth.

  где

- userAuth — api-key пользователя, передаваемый в заголовке „X-Api-Key“ к запросу
  ИЛИ передача JWT токена с ролью user в заголовке „Authorization“;

- privacyAuth — api-key privacy пользователя в заголовке „X-Api-Key“ к запросу ИЛИ передача JWT токена с ролью privacy в заголовке „Authorization“.

Кроме того, авторизация gRPC и REST API настраивается в секции `auth` *конфигурационного файла ноды*.

PrivacyEventsService

У сервиса **PrivacyEventsService** есть один метод **SubscribeOn**, описанный в protobuf-файле **privacy_events_service.proto**. Используйте этот метод для получения потока (стрима) событий по получению или удалению конфиденциальных данных, которые поступают в gRPC-интерфейс ноды. Для этого отправьте запрос `SubscribeOn (SubscribeOnRequest)`, который инициализирует подписку на стрим.

Информация о получении или удалении конфиденциальных данных

После успешной отправки запроса на gRPC-интерфейс будут приходить следующие данные:

- `policy_id` – идентификатор группы доступа к конфиденциальным данным;
- `data_hash` – идентификационный хэш конфиденциальных данных;
- `event_type` – тип события; доступны следующие типы:
  - DATA_ACQUIRED – данные сохранены в БД;
  - DATA_INVALIDATED – данные помечены на удаление в связи с отсутствием активности по ним или при роллбэке (откате).

PrivacyPublicService

У сервиса **PrivacyPublicService** есть следующие методы, описанные в protobuf-файле **privacy_public_service.proto**:

- *GetPolicyItemData*;
- *GetDataLarge*;
- *GetPolicyItemInfo*;
- *PolicyItemDataExists*;
- *SendData*;
- *SendLargeData*;
- *Recipients*;
- *Owners*;
- *Hashes*;
- *forceSync*.

Важно: Типы данных полей для запросов и ответов указаны в protobuf-файле.

Получение хэш-суммы конфиденциальных данных

Используйте метод **GetPolicyItemData** для получения пакета конфиденциальных данных группы доступа по идентификационному хэшу. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс возвращается хэш-сумма конфиденциальных данных.

Скачивание из ноды больших данных

Используйте метод **GetDataLarge** для скачивания из ноды больших данных, которые были отправлены с помощью метода *SendLargeData*. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс возвращается поток PolicyItemDataResponse с данными.

Получение метаданных для пакета конфиденциальных данных

Используйте метод **GetPolicyItemInfo** для получения метаданных для пакета конфиденциальных данных группы по идентификационному хэшу. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс 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.

Проверка существования пакета конфиденциальных данных

Используйте метод **PolicyItemDataExists** для получения информации о наличии пакета конфиденциальных данных группы доступа по идентификационному хэшу. Метод требует ввода параметров запроса `policy_id` (идентификатор группы доступа) и `data_hash` (идентификационный хэш). После успешной отправки запроса на gRPC-интерфейс возвращается true, если данные в наличии, или false, если данные отсутствуют.

Отправка в блокчейн конфиденциальных данных

Используйте метод **SendData** для отправки в блокчейн *конфиденциальных данных*, доступных только для участников группы доступа, определенной для этих данных.

Примечание: Для отправки данных размером более 20 МБ используйте метод *SendLargeData*.

Примечание: Для отправки в блокчейн потока конфиденциальных данных используйте метод *SendLargeData*.

Важно: Метод SendData недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. Метод можно использовать в тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`).

Метод требует ввода следующих параметров запроса:

- `sender_address` – блокчейн-адрес, от которого должны рассылаться данные (соответствуют значению параметра `privacy.owner-address` в конфигурационном файле ноды);
- `policy_id` – идентификатор группы доступа к конфиденциальным данным, которая будет иметь доступ к отправляемым данным;
- `data_hash` – идентификационный sha256-хэш конфиденциальных данных в формате base58;
- `info` – информация об отправляемых данных:
  - `filename` – имя файла данных,
  - `size` – размер файла данных,
  - `timestamp` – временная метка,
  - `author` – электронный адрес автора отправляемых данных,
  - `comment` – произвольный комментарий.
- `fee` – комиссия за транзакции;
- `fee_asset_id` – поле опционально и используется только для смарт-контрактов с поддержкой gRPC;
- `atomic_badge` – поле-метка, указывающая, что транзакция поддерживается атомарной транзакцией;
- `password` – пароль для доступа к закрытому ключу keystore ноды;
- `broadcast_tx` – если передается значение `true`, то созданная *PolicyDataHash* транзакция отправляется в блокчейн, если `false`, то транзакция и сообщение о наличии данных (Privacy Inventory) не отправляется; подробнее см. *ниже*;
- `data` – строка, содержащая данные в формате base64.
- `certificates` – цепочка сертификатов байтами в формате DER; параметр является обязательным при одновременном соблюдении следующих условий:
  - используется тестовый режим PKI (то есть в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `TEST`),
  - новый пользователь, который не является владельцем ноды (node-owner), делает свою первую транзакцию.

В этом случае необходимо в запросе в поле `certificates` передать цепочку сертификатов пользователя; в других случаях поле `certificates` является необязательным.

Примечание: При отправке файлов через Amazon S3/Minio в полях `comment`, `author`, `filename` должны быть ascii символы. Это ограничение Java SDK AWS.

После успешной отправки запроса на gRPC-интерфейс будут приходить следующие данные:

- `tx_version` – версия транзакции;
- `tx` – созданная PolicyDataHash транзакция.

Параметр `broadcast_tx`

Для снижения вероятности ошибок доставки данных рекомендуется установить для параметра `broadcast_tx` значение `false`, если после отправки данных с помощью API метода **SendData** отправляется атомарная транзакция, которая содержит транзакцию *CreatePolicy* и транзакцию *PolicyDataHash*.

Отправка в блокчейн потока конфиденциальных данных

Используйте метод **SendLargeData** для отправки в блокчейн потока конфиденциальных данных. Данные будут доступны только для участников группы доступа, определенной для этих данных.

Примечание: Для отправки данных размером менее 20 МБ используйте метод *SendData*.

Важно: Метод SendLargeData недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. Метод можно использовать в тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`).

Метод принимает в запросе поток данных в следующем формате:

- `metadata` – метаданные для пакета конфиденциальных данных, аналогичные входным данным метода *SendData*;
- `content` – массив байт, представляющих собой пакет конфиденциальных данных.

После успешной отправки запроса на gRPC-интерфейс будут приходить те же данные, что и для метода *SendData*.

Получение адресов всех участников группы доступа к конфиденциальным данным

Используйте метод **Recipients** для получения адресов всех участников группы доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` – идентификатор группы доступа. В ответе метод возвращает массив строк с адресами участников группы доступа.

Получение адресов владельцев группы доступа к конфиденциальным данным

Используйте метод **Owners** для получения адресов владельцев группы доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` (идентификатор группы доступа). В ответе метод возвращает массив строк с адресами владельцев группы доступа.

Получение массива идентификационных хэшей данных

Используйте метод **Hashes** для получения массива идентификационных хэшей данных, которые привязаны к группе доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` (идентификатор группы доступа). В ответе метод возвращает массив строк с идентификационными хэшами данных группы доступа.

Синхронизация данных по указанной группе доступа к конфиденциальным данным

Используйте метод **forceSync** для синхронизации данных по указанной группе доступа к конфиденциальным данным. Метод требует ввода параметра запроса `policy_id` (идентификатор группы доступа).

В результате выполнения метода нода запускает процесс синхронизации и возвращает размер конфиденциальных данных в Мб. Если синхронизацию не удалось запустить, нода возвращает и описание ошибки.

Смотрите также

Инструментарий gRPC

Тонкая настройка платформы: настройка авторизации для gRPC и REST API

gRPC: получение вспомогательной информации

Для получения вспомогательной информации предусмотрен gRPC сервис **UtilPublicService**.

Получение текущего времени ноды

У сервиса **UtilPublicService** есть один метод **GetNodeTime**, описанный в protobuf-файле **util_public_service.proto**. Используйте этот метод для получения текущего времени ноды. Метод не требует ввода дополнительных параметров запроса.

Важно: Типы данных полей для ответов указаны в protobuf-файлах.

Метод возвращает текущее время ноды в двух форматах:

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

Смотрите также

Инструментарий gRPC Вспомогательные запросы

gRPC: получение информации об участниках сети

Для получения информации об участниках сети предусмотрены gRPC сервисы **AddressPublicService** и **AliasPublicService**.

gRPC: получение информации об адресах участников сети

Для получения информации об адресах участников сети предусмотрен gRPC сервис **AddressPublicService**.

У сервиса **AddressPublicService** есть следующие методы, описанные в protobuf-файле **address_public_service.proto**:

- **GetAddresses**;
- **GetAddressData**;
- **GetAddressDataByKey**.

Важно: Типы данных полей для запросов и ответов указаны в protobuf-файле.

Получение всех адресов участников

Используйте метод **GetAddresses** для получения всех адресов участников, ключевые пары которых хранятся в keystore ноды. Метод не требует ввода дополнительных параметров запроса.

Метод возвращает массив адресов участников.

Получение данных с указанного адреса

Используйте метод **GetAddressData** для получения данных, записанных на указанном адресе при помощи *транзакций 12*. Метод требует ввода следующих параметров запроса:

- `address` – адрес ноды;
- `limit` – максимальное количество записей, которые вернет метод;
- `offset` – количество первых записей по данному адресу, которые метод пропустит.

Метод возвращает данные, записанные на указанном адресе.

Получение данных с указанного адреса по ключу

Используйте метод **GetAddressDataByKey** для получения данных, записанных на указанном адресе с ключом при помощи *транзакций 12*. Этот ключ указывается в транзакции 12 в поле `data.key`. Метод требует ввода следующих параметров запроса:

- `address` – адрес ноды;
- `key` – ключ.

Метод возвращает данные, записанные на указанном адресе с ключом `key`.

gRPC: получение информации об участниках сети по псевдониму

Для получения информации об участниках сети по псевдониму предусмотрен gRPC сервис **AliasPublicService**.

У сервиса **AliasPublicService** есть следующие методы, описанные в protobuf-файле **alias_public_service.proto**:

- **AddressByAlias**;
- **AliasesByAddress**.

Получение адреса по псевдониму

Используйте метод **AddressByAlias** для получения адреса по псевдониму. Метод требует ввода одного параметра запроса:

- `alias` – псевдоним участника сети.

Метод возвращает адрес участника сети.

Получение псевдонима по адресу

Используйте метод **AliasesByAddress** для получения псевдонима по адресу. Метод требует ввода в запросе адреса участника сети.

Метод возвращает все псевдонимы участника сети.

Смотрите также

Инструментарий gRPC

GET /addresses

Группа alias:

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, ссылки на которые приведены выше.

Авторизация gRPC-методов ноды настраивается в секции `auth` *конфигурационного файла ноды*.

Смотрите также

Сервисы gRPC, используемые смарт-контрактом

1.8 Методы REST API

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.

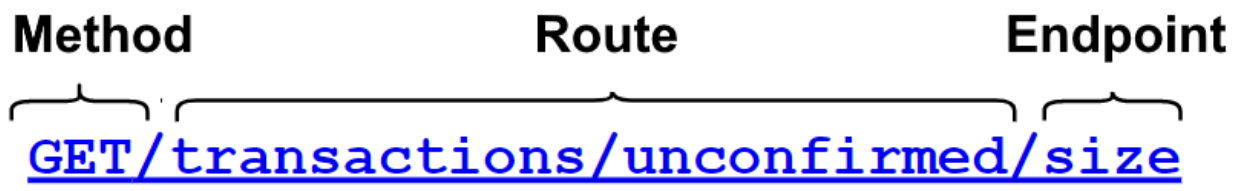
1.8.1 Использование REST API

Все вызовы методов REST API — это HTTP-запросы GET, POST или DELETE к URL `https://yournetwork.com/node-N`, содержащие соответствующие наборы параметров.

Платформа также предоставляет доступ к интерфейсу Swagger `https://yournetwork.com/node-N/api-docs/index.html`, который позволяет составлять и отправлять HTTP-запросы в ноду через веб-интерфейс. Нужные группы запросов 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:



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.

1.8.2 Для чего предназначен REST API платформы

Вы можете использовать интерфейс REST API для выполнения следующих задач:

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*.

POST /transactions/sign

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`.

Важно: Метод /transactions/sign недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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": [
↪"NqZGcbcQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFAknTMEGuFrEndCjXBtrueLWaqbJhpeiG
↪" ],
    "version": 2,
    "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "name": "Test Asset 1",
    "quantity": 10000,
    "reissuable": true,
    "decimals": 8,
    "description": "Some description",
    "chainId": 84,
    "script": "base64:AQa3b8tH",
    "height": 60719
}
```

POST /transactions/broadcast

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*.

Когда новый пользователь, который не является владельцем ноды (node-owner), делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным.

Примечание: Поле `certificates` в запросе на публикацию транзакции *RegisterNode* является обязательным при использовании PKI или тестового режима PKI (то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON` или `TEST`. В этом случае поле `certificates` должно содержать цепочку сертификатов, которая соответствует публичному ключу в поле `target` транзакции.

Пример запроса метода POST /transactions/broadcast

POST /transactions/broadcast:

```
{
    "type": 3,
    "id": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 100000000,
    "timestamp": 1549378509516,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "proofs": [
↪"NqZGcbcQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFAknTMEGuFrEndCjXBtrueLWaqbJhpeiG
↪" ],
    "version": 2,
    "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "name": "Test Asset 1",
    "quantity": 10000,
    "reissuable": true,
    "decimals": 8,
    "description": "Some description",
    "chainId": 84,
    "script": "base64:AQa3b8tH",
    "height": 60719
    "certificates": ["a", "b", ...]
}
```

В случае успешной публикации транзакции метод возвращает json с транзакцией и сообщение 200OK.

POST /transactions/signAndBroadcast

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.

Важно: Метод /transactions/signAndBroadcast недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

Когда новый пользователь, который не является владельцем ноды (node-owner), делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным.

Примечание: Поле `certificates` в запросе на публикацию транзакции *RegisterNode* является обязательным при использовании тестового режима PKI (то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `TEST`. В этом случае поле `certificates` должно содержать цепочку сертификатов, которая соответствует публичному ключу в поле `target` транзакции.

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",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 112
  "certificates": ["a", "b", ...]
}
```

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"
```

(continues on next page)

(продолжение с предыдущей страницы)


```
  ],
  "id": "EyymzQcM2LrsgGDFFxeGn8DhahJbFYmorcBrEh8phv5S",
  "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:

GET /transactions/info/{id}:


```
{
  "type": 4,
  "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
  "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
  "recipient": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
  "amount": 100000,
  "fee": 100000,
  "timestamp": 1549365736923,
  "attachment": "string",
  "signature":
↪"GkkccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm
↪",
  "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":
↪"4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CdddnyBW
↪",
    "fee": 100000,
    "timestamp": 1549365736923,
    "signature":
↪"4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLNkFQjWp95CdddnyBW
↪",
    "sender": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
    "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": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMhM3Uki7pLw",
    "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":
↪"GkнccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtsUm
↪",
  "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

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

Для сетей, работающих с использованием ГОСТ-криптографии, REST API-интерфейс предоставляет возможность формирования отсоединенной электронной подписи для передаваемых данных, а также ее проверки. Для формирования и проверки электронных подписей предусмотрена группа методов `pki`: `POST /pki/sign` и `POST /pki/verify`.

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.

Важно: Метод /pki/sign недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

Запрос метода POST /pki/sign 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" :
↪"c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQjsndjfvnksdnjfn="
}
```

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

REST API: получение сертификатов

Для запроса у ноды сертификата из хранилища сертификатов предусмотрена группа методов `/pki/certificate`. Методы этой группы позволяют получить сертификат по разным полям:

- */pki/certificate/by-dn/%percent-encoded-DN%* – по полю `DN` (distinguished name),
- */pki/certificate/by-dn-hash/%DN-hash-string%* – по полю `DN Hash`,
- */pki/certificate/by-public-key/%public-key-base58%* – по полю `publicKey`,
- */pki/certificate/by-fingerprint/%fingerprint-base64%* – по полю `fingerprint`.

В запросе эти методы принимают значение соответствующего поля сертификата и, опционально, параметр `plainText`, который задаёт формат ответа.

Если сертификат существует, то в ответе каждого из этих методов нода возвращает сертификат в формате DER (как он и записан в хранилище сертификатов ноды), байты кодируются в формат Base64. Если в запросе метода параметру `plainText` задано значение `true`, то сертификат возвращается в формате plainText.

Если сертификата не существует, то в ответе каждого из этих методов возвращается ошибка с кодом `404 Not Found`.

Авторизация методов группы /pki/certificate

В случае API-KEY авторизация не требуется.

В случае OAuth2 авторизации требуется наличие роли `user` в JWT токене.

GET /pki/certificate/by-dn/%percent-encoded-DN%

Метод возвращает сертификат по его отличительному имени (distinguished name), записанному в поле `DN`.

Пример запроса метода GET /pki/certificate/by-dn/%percent-encoded-DN%:

GET /pki/certificate/by-dn/%percent-encoded-DN%:

Запрос:

```
{
  "DN": "CN=Steve Kille,O=Isode Limited,C=GB",
  "plainText": false
}
```

GET /pki/certificate/by-dn-hash/%DN-hash-string%

Метод возвращает сертификат по хэшу SHA-1 (Keccak) от поля `DN` сертификата.

GET /pki/certificate/by-public-key/%public-key-base58%

Метод возвращает сертификат по байтам публичного ключа (поле `publicKey`).

GET /pki/certificate/by-fingerprint/%fingerprint-base64%

Метод возвращает сертификат по его SHA-1 отпечатку (поле `fingerprint`).

Смотрите также

Методы REST API

Проверка УКЭП

gRPC: получение сертификатов

REST API: реализация методов шифрования

REST API-интерфейс ноды предусматривает возможность зашифровать произвольные данные при помощи алгоритмов шифрования, применяемых в блокчейн-платформе Waves Enterprise, а также дешифровать их. Для этого предусмотрены методы REST API группы `crypto`:

- `EncryptSeparate` – шифрование данных уникальными ключами CEK отдельно для каждого получателя, каждый CEK шифруется (оборачивается) отдельным ключом KEK;
- `EncryptCommon` – шифрование данных единым ключом CEK для всех получателей, каждый ключ CEK шифруется (оборачивается) отдельным ключом KEK для каждого получателя;
- `Decrypt` – дешифровка данных.

Важно: Методы crypto/encryptCommon, crypto/encryptSeparate, crypto/decrypt недоступны при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) методы можно использовать.

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:
  - `aes` – AES
- `gost-3412-2015-k` – ГОСТ 34.12-2015

Примечание: Начиная с версии 1.8 алгоритм шифрования ГОСТ 28147-89 (значение `gost-28147`) не поддерживается.

Если в вашей сети используется шифрование по ГОСТ, вам будет доступен только алгоритм `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/jmlTizVxD6Nik=",
 "publicKey":
↪"5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRL352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeiZz5i2qXc
↪",
 "wrappedKey":
↪"uWVoxJAzruwTDDSbphDS31TjSQX6CSWXivp3x34uE3XtnMqqK9swoaZ3LyAgFDR7o6CfkgzFkWmTen4qAZewPfBbwR
↪"
},
```

POST /crypto/encryptCommon

Шифрование данных, переданных в запросе, единым ключом CEK для всех получателей, каждый ключ CEK шифруется (*оборачивается*) отдельным ключом KEK 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
↪",

↪"9LopMj2GqWxBYgnZ2gxaNxwXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S
↪":

↪"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:
  - `aes` – AES

- `gost-3412-2015-k` – ГОСТ 34.12-2015

Примечание: Начиная с версии 1.8 алгоритм шифрования ГОСТ 28147-89 (значение `gost-28147`) не поддерживается.

Если в вашей сети используется шифрование по ГОСТ, для дешифровки будет доступен только алгоритм `gost-3412-2015-k`. При отключенном шифровании по ГОСТ доступен только алгоритм шифрования `aes`.

Пример запроса:

POST /crypto/decrypt:

```
{
  "recipient": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "password": "12345qwert",
  "encryptedText":
↪"t859AE7idnjPpn3lUiorfzSGwcGPMVdOhQe1HAhoI0MOXOQPBc8TUhn+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",
}
```

Смотрите также

Методы REST API

Криптография

REST API: обмен конфиденциальными данными и получение информации о группах доступа

Подробнее об обмене конфиденциальными данными и группах доступа см. статью *Обмен конфиденциальными данными*.

Для реализации этих функций при помощи REST API предусмотрен набор методов группы `Privacy`.

Важно: Методы группы `Privacy` недоступны при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) методы можно использовать.

Авторизация методов группы `Privacy`

:opticon:`report, size=24` Авторизация методов группы `Privacy`:

Для использования методов REST API группы `Privacy` требуется авторизация по api-key или JWT-токену. Авторизация методов реализована следующим образом:

- в случае api-key авторизации требуется PrivacyApiKey;
- в случае OAuth2 авторизации требуется наличие роли Privacy в JWT токене.

Для каждого из методов необходимо передавать следующие данные:

- policyRecipients — contractAuth | userAuth;
- policyOwners — userAuth;
- policyHashes — contractAuth | userAuth;
- policyItemData — contractAuth | privacyAuth;
- policyItemLargeData — contractAuth | privacyAuth;
- policyItemInfo — contractAuth | privacyAuth;
- policyItemsInfo — contractAuth | privacyAuth;
- sendData — privacyAuth;
- sendDataV2 — privacyAuth;
- forceSync — privacyAuth;
- forceSyncByPolicyId — privacyAuth;
- sendLargeData — privacyAuth,

где

- contractAuth — токен авторизации смарт контракта, передаваемый в заголовке „X-Contract-Api-Token“ к запросу;
- userAuth — api-key пользователя, передаваемый в заголовке „X-Api-Key“ к запросу ИЛИ передача JWT токена с ролью user в заголовке „Authorization“;
- privacyAuth — api-key privacy пользователя в заголовке „X-Api-Key“ к запросу ИЛИ передача JWT токена с ролью privacy в заголовке „Authorization“.

Кроме того, авторизация gRPC и REST API настраивается в секции `auth` *конфигурационного файла ноды*.

POST /privacy/sendData

Метод предназначен для отправки в блокчейн *конфиденциальных данных*, доступных только для участников группы доступа, определенной для этих данных.

Примечание: Для отправки данных размером более 20 МБ используйте метод *POST /privacy/sendLargeData*.

Важно: Метод /privacy/sendData недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI

(node.crypto.pki.mode = TEST) или при отключенном PKI (node.crypto.pki.mode = OFF) метод можно использовать.

Запрос метода POST /privacy/sendData должен содержать 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:
  - `filename` – имя файла данных,
  - `size` – размер файла данных,
  - `timestamp` – временная метка,
  - `author` – электронный адрес автора отправляемых данных,
  - `comment` – произвольный комментарий.
- `data` – строка, содержащая данные в формате **base64**;
- `hash` – sha256-хэш данных в формате **base58**;
- `broadcast` – если передается значение `true`, то созданная *PolicyDataHash* транзакция отправляется в блокчейн, если `false`, то транзакция и сообщение о наличии данных (Privacy Inventory) не отправляется; подробнее см. *ниже*.
- `certificates` – цепочка сертификатов байтами в формате DER; параметр является обязательным при одновременном соблюдении следующих условий:
  - используется тестовый режим PKI (то есть в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `TEST`),
  - новый пользователь, который не является владельцем ноды (node-owner), делает свою первую транзакцию.

  В этом случае необходимо в запросе в поле `certificates` передать цепочку сертификатов пользователя; в других случаях поле `certificates` является необязательным.

Примечание: При отправке файлов через Amazon S3/Minio в полях `comment`, `author`, `filename` должны быть ascii символы. Это ограничение Java SDK AWS.

В результате отправки запроса будет сформирована транзакция *114 PolicyDataHash*, которая отправит хэш конфиденциальных данных в блокчейн.

Параметр `broadcast`

Для снижения вероятности ошибок доставки данных рекомендуется установить для параметра `broadcast` значение `false`, если после отправки данных с помощью API метода **sendData** отправляется атомарная транзакция, которая содержит транзакцию *CreatePolicy* и транзакцию *PolicyDataHash*.

Примеры запроса и ответа:

POST /privacy/sendData:

Запрос:

```
{
"sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
"password": "apgJP9atQccdBPA",
"policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
"info": {
  "filename":"Service contract #100/5.doc",
  "size": 2048,
  "timestamp": 1000000000,
  "author": "AIvanov@org.com",
  "comment": "some comments"
 },
 "data":
↪"TWFuIGlzIGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWFzb24sIGJ1dCBieSBOaGlzIHNpbmd1bGFyIHBhc3Np
↪",
 "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
 "broadcast": false
 }
```

Ответ:

```
{
"senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
"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**. Метод предоставляет возможность потоковой передачи данных. Поле `Data` в этой версии метода отсутствует.

Примечание: Для отправки данных размером более 20 МБ используйте метод *POST /privacy/sendLargeData*.

Примечание: При отправке файлов через Amazon S3/Minio в полях `comment`, `author`, `filename` должны быть ascii символы. Это ограничение Java SDK AWS.

Примечание: Когда новый пользователь, который не является владельцем ноды (node-owner), в тестовом режиме PKI (*параметру node.crypto.pki.mode* присвоено значение `TEST`) делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным.

Важно: Метод /privacy/sendDataV2 недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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": "AIvanov@org.com",
  "comment": "some comments"
 },
 "hash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
 "broadcast": false
 }
```

Ответ:

```
{
"senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
```

(continues on next page)

(продолжение с предыдущей страницы)


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

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


POST /privacy/sendLargeData

Метод **POST /privacy/sendLargeData** аналогичен методу *POST /privacy/sendDataV2*, но используется для отправки данных размером не менее 20 МБ.

Примечание: Для отправки данных размером менее 20 МБ используйте методы *POST /privacy/sendData* и *POST /privacy/sendDataV2*.

В конфигурационном файле ноды в секции *node.privacy.service* можно настроить обратное давление на входящие фрагменты данных: задать максимальный размер для буфера в памяти (по умолчанию – 100 МБ

Примечание: При отправке файлов через Amazon S3/Minio в полях `comment`, `author`, `filename` должны быть ascii символы. Это ограничение Java SDK AWS.

Примечание: Когда новый пользователь, который не является владельцем ноды (node-owner), в тестовом режиме PKI (*параметру node.crypto.pki.mode* присвоено значение `TEST`) делает свою первую транзакцию, ему необходимо в запросе в поле `certificates` приложить цепочку своих сертификатов. В других случаях поле `certificates` является необязательным.

Важно: Метод /privacy/sendLargeData недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

Примеры запроса и ответа:

POST /privacy/sendLargeData:

Запрос:

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

Ответ:

```
{
"senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrgsLk5VY",
"policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
"sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
"dataHash": "FRog42mnzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
"proofs": [

↪"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"
]
```

Важно: Метод GET /privacy/{policy-id}/recipients недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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:

```
[
  "3GCFaCWtvLDnC9yX29YftMbn75gwfdwGsBn",
  "3GGxcmNyq8ZAHzK7or14Ma84khwW8peBohJ",
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZyNh84pn",
  "3GKpShRQRTddF1yYhQ58ZnKMTnp2xdEzKqW"
]
```

Важно: Метод GET /privacy/{policy-id}/owners недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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"
]
```

Важно: Метод GET /privacy/{policy-id}/hashes недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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=
```

Важно: Метод GET /privacy/{policyId}/getData/{policyItemHash} недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

GET /privacy/{policyId}/getLargeData/{policyItemHash}

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

Метод возвращает стрим, что позволяет пользователю скачать файл с данными неограниченного объёма.

Важно: Метод GET /privacy/{policyId}/getLargeData/{policyItemHash} недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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"
}
```


Важно: Метод GET /privacy/{policyId}/getInfo/{policyItemHash} недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение ON. В тестовом режиме PKI (node.crypto.pki.mode = TEST) или при отключенном PKI (node.crypto.pki.mode = OFF) метод можно использовать.

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. Метод синхронизирует данные по всем группам доступа к конфиденциальным данным.

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 значение `success`, конфиденциальные данные записываются в хранилище ноды.

В случае возникновения ошибки возвращается значение `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'"
}
```

Важно: Метод POST /privacy/forceSync недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

GET /privacy/forceSync/{policyId}

Метод аналогичен методу POST /privacy/forceSync с той разницей, что синхронизирует данные по указанной группе доступа к конфиденциальным данным (*policyId*).

Важно: Метод GET /privacy/forceSync/{policyId} недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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":[
```

(continues on next page)

(продолжение с предыдущей страницы)


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


Важно: Метод POST /privacy/getInfos недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

Смотрите также

Методы REST API

Обмен конфиденциальными данными

Тонкая настройка платформы: настройка авторизации для gRPC и REST API

REST API: работа с лицензиями ноды

Для работы с лицензиями блокчейн-платформы Waves Enterprise предусмотрена группа методов `licenses`.

Примечание: *Opensource-версия* блокчейн-платформы Waves Enterprise не включает группу методов `licenses`.

GET /licenses

Метод возвращает информацию о всех загруженных лицензиях.

В ответе для каждой лицензии поступает набор данных `license`, в котором содержатся параметры, указанные в файле лицензии, полученном от Waves Enterprise.

Пример ответа для одной лицензии:

Ответ GET /licenses:

```
[
  {
    "license": {
      "version": 1,
      "id": "a3d0d17e-eb05-45ac-906c-da847a9d726d",
      "issued_at": "2021-01-28T15:39:59.456Z",
      "node_owner_address": "3JNFkQ2cVu7ndEHLCs9A5HT63jSi1TV3mWK",
      "valid_after": "2021-01-29",
      "valid_before": "2022-11-20",
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key":
↪"p9HrAcGytSBxixJnQXQ87SNXPoXTdnwRzo4FMFvvbNSPzCToqdpJrcgFP6wxmsG23wBfYzcth",
    "signature": "jNjwCXdMPxmdaibXtjYSd8WocFinXKNsrTdPkbWrPTkQstswBp9SHFe",
    "signer_id": "2WDmdaibXtjYSd8WocFinX"
  }
]
```

GET /licenses/status

Метод возвращает статус активации лицензии ноды.

В ответе метода поступают следующие данные:

- `status` – статус активации лицензии:
  - `TRIAL` – активна пробная лицензия (максимальная высота блокчейна - 30000 блоков), по завершении пробного периода валидных лицензий нет;
  - `TRIAL_EXPIRED` – пробная лицензия истекла, валидных лицензий нет;
  - `ACTIVE` – валидная лицензия активна на момент запроса;
  - `PENDING` – на момент запроса активной лицензии нет, есть валидная лицензия, начинающаяся с более поздней даты: этот статус поступает по окончании пробного периода при наличии валидной лицензии с более поздней датой начала;
  - `EXPIRED` – валидная лицензия на момент запроса истекла, валидных лицензий с более поздней датой начала нет.
- `description` – краткое описание статуса, оставшееся количество блоков или дата истечения активной лицензии.

Пример ответа:

Ответ GET /licenses/status:

```
{
  "status" : "TRIAL",
  "description" : "Trial period is active. Blocks before expiration: 23412"
}
```

POST /licenses/upload

Метод добавляет новую лицензию для ноды. Параметры, которые передаются в JSON-формате в запросе, указаны в файле, предоставляемом специалистами Waves Enterprise при оформлении лицензии.

Пример запроса:

Запрос POST /licenses/upload:

```
{
  "license": {
    "version": 1,
    "id": "a3d0d17e-eb05-45ac-906c-da847a9d726d",
    "issued_at": "2021-01-28T15:39:59.456Z",
    "node_owner_address": "3JNFkQ2cVu7ndEHLCs9A5HT63jSi1TV3mWK",
    "valid_after": "2021-01-29",
    "valid_before": "2022-11-20",
    "features": [
      "all_inclusive"
    ]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  },
  "signer_public_key":
↪"p9HrAcGytSBxixJnQXQ87SNXPoXTdnwRzo4FMFvvbNSPzCToqdpJrcgFP6wxmsG23wBfYzcth",
  "signature": "jNjwCXdMPxmdaibXtjYSd8WocFinXKNsrTdPkbWrPTkQstswBp9SHFe",
  "signer_id": "2WDmdaibXtjYSd8WocFinX"
}
```

Пример ответа:

Ответ POST /licenses/upload:

```
{
    "message": "License upload successfully"
}
```

DELETE /licenses/{license_id}

Метод удаляет загруженную лицензию по ее идентификатору {license_id}. Идентификатор лицензии указан в файле лицензии, который вы получите от специалистов Waves Enterprise, а также в ответе метода **GET /licenses**.

Пример ответа:

Ответ DELETE /licenses/{license_id}:

```
{
  "message": "License removed successfully"
}
```

Смотрите также

Методы REST API

GET /licenses

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:1nvAliDAl1ass99911%^&$$$ "
  ]
}
```

Ответ:

```
{
  validations: [
    {
      addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
      valid: true
    },
    {
      addressOrAlias: "alias:T:asdfghjk",
      valid: true
    },
    {
      addressOrAlias: "alias:T:1nvAliDAl1ass99911%^&$$$ ",
      valid: false,
      reason: "GenericError(Alias should contain only following characters: -.
↪0123456789@_abcdefghijklmnopqrstuvwxyz)"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    }
  ]
}
```

Смотрите также

Методы REST API

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**.

Важно: Метод addresses/sign недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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":
↪"5kwwE9sDZzssoNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts
↪"
}
```

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.

Важно: Метод addresses/signText недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение ON. В тестовом режиме PKI (node.crypto.pki.mode = TEST) или при отключенном PKI (node.crypto.pki.mode = OFF) метод можно использовать.

Примеры запроса и ответа:

POST /addresses/signText/{address}:

Запрос:

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

Ответ:

```
{
  "message": "mytext",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪"5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7ZjnrzsQQYRWo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u
↪"
}
```

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":
↪"5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjwHK7ZjnrzsQQYRWo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u
↪"
}
```

Ответ:

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

Смотрите также

Методы REST API

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": "1.9.0-Dev3-213-66e7eb5",
  "gostCrypto": true,
  "cryptoType": "gost",
  "chainId": "T",
  "consensus": "POA",
  "minimumFee": {
    "3": 100000000,
    "4": 1000000,
    "5": 100000000,
    "6": 5000000,
    "7": 500000,
    "8": 1000000,
    "9": 1000000,
    "10": 100000000,
    "11": 5000000,
    "12": 5000000,
    "13": 50000000,
    "14": 100000000,
    "15": 100000000,
    "102": 1000000,
    "103": 100000000,
    "104": 10000000,
    "106": 1000000,
    "107": 100000000,
    "111": 1000000,
    "112": 100000000,
    "113": 50000000,
    "114": 5000000,
    "120": 0
  },
  "additionalFee": {
    "11": 1000000,
    "12": 1000000
  },
  "maxTransactionsInMicroBlock": 500,
  "minMicroBlockAge": 0,
  "microBlockInterval": 1500,
  "pkiMode": "TEST",
  "requiredOids": [
    "1.1.1.1"
```


(continues on next page)

(продолжение с предыдущей страницы)

```
  ],
  "crlChecksEnabled": false,
  "blockTiming": {
    "roundDuration": 7000,
    "syncDuration": 1000
  }
}
```

Начиная с версии 1.9 поле `gostCrypto` не будет поддерживаться. Вместо него будет использоваться поле `cryptoType`.

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,
    "updatedDate": "2018-12-13T13:58:21.138Z"
    "lastCheckTimestamp": 1543719501123,
}
```

Также, при возникновении ошибок с использованием ГОСТ-криптографии на ноде, 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
```

GET /node/healthcheck

Метод производит проверку доступности внешнего сервиса, указанного в запросе. В запросе должен быть указан параметр `service`, который может принимать одно из следующих значений:

- `docker`;
- `privacy-storage`;
- `anchoring-auth`.

По умолчанию используется значение `docker`.

Метод возвращает значение `200 OK` и пустой ответ, если проверка прошла успешно, иначе – `503 Service Unavailable` и описание ошибки. Если один из внешних сервисов не настроен (на ноде отключена *функциональность докер смарт контрактов*, отключена настройка *групп доступа к конфиденциальным данным*, отключен *анкоринг*), метод возвращает ошибку `404 Not Found` с сообщением о том, что определенная настройка отключена.

GET /node/healthcheck:

```
{
  "error": 199,
  "message": "Docker host is not available"
}
```

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.

Важно: Метод POST /node/stop недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

Группа `anchoring`:

GET /anchoring/config

Метод выводит секцию `anchoring` конфигурационного файла ноды.

Пример ответа:

GET /anchoring/config:

```
{
 "enabled": true,
 "currentChainOwnerAddress": "3FWwx4o1177A4oeHAEW5EQ6Bkn4Lv48quYz",
 "targetnetnetNodeAddress": "https://clinton-pool.wavesenterpriseservices.com:443",
 "targetnetnetSchemeByte": "L",
 "targetnetnetRecipientAddress": "3JzVWCSV6v4ucSxtGSjZsvdiCT1FAzwpqrP",
 "targetnetnetFee": 8000000,
 "currentChainFee": 666666,
 "heightRange": 5,
 "heightAbove": 3,
 "threshold": 10
}
```

Смотрите также

Методы REST API

Примеры конфигурационных файлов ноды

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": "3M4Bxh2VfzKFXqiQB8bDgRfVnPWrZUQ2MEF",
    "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 >= {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": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "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",
]
```

Группа `leasing`:

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": "DW9NKLYeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVwRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
↪"25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPFyegP82sNSfM2bXNX3K7p6D3HD
↪",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```

Смотрите также

Методы REST API

REST API: информация об активации новых функциональных возможностей платформы

GET /activation/status

Метод возвращает статус активации новых функциональных возможностей.

Подробнее о процессе активации см. статью *Активация функциональных возможностей*.

Ответ метода содержит следующие общие поля:

- `height` – текущая высота блокчейна;
- `votingInterval` – интервал проведения голосования за активацию;
- `votingThreshold`
- `nextCheck`

Далее выводится массив `features`, содержащий информацию по каждой отдельной функциональной возможности:

- `id` – идентификатор функциональной возможности;
- `description` – описание функциональной возможности;
- `blockchainStatus` – статус функциональной возможности в блокчейне:
  - `UNDEFINED` – функциональная возможность не активирована, голосование за нее не проводилось;
  - `APPROVED` – голосование за функциональную возможность проведено, активация будет произведена на установленной высоте блокчейна;
  - `ACTIVATED` – функциональная возможность активирована;
- `nodeStatus` – статус функциональной возможности на ноде участника:
  - `VOTED` – нода проголосовала за активацию функциональной возможности;
  - `NOT IMPLEMENTED` – функциональная возможность не запущена на ноде;
  - `IMPLEMENTED` – функциональная возможность запущена;
- `activationHeight` – высота блокчейна, на которой активируется функциональная возможность.

Пример ответа:

GET /activation/status:


```
{"height": 47041,
 "votingInterval": 1,
 "votingThreshold": 1,
 "nextCheck": 47041,
 "features": [
    {"id": 1,
     "description": "Minimum Generating Balance of 1000 WEST",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 2,
     "description": "NG Protocol",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 3,
     "description": "Mass Transfer Transaction",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 4,
     "description": "Smart Accounts",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 5,
     "description": "Data Transaction",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 6,
     "description": "Burn Any Tokens",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 7,
     "description": "Fee Sponsorship",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 8,
     "description": "Fair PoS",
     "blockchainStatus": "ACTIVATED",
     "nodeStatus": "IMPLEMENTED",
     "activationHeight": 0 },
    {"id": 9,
     "description": "Smart Assets",
     "blockchainStatus": "VOTING",
     "nodeStatus": "IMPLEMENTED",
     "supportingBlocks": 0 },
    {"id": 10,
```


(continues on next page)

(продолжение с предыдущей страницы)


```
    "description": "Smart Account Trading",
    "blockchainStatus": "ACTIVATED",
    "nodeStatus": "IMPLEMENTED",
    "activationHeight": 0 } ]
}
```


Смотрите также

Методы REST API

Активация функциональных возможностей

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": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
    "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": [
        "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
        "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
        "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, переведенные 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

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


GET /contracts/status/{id}

Метод возвращает статус исполняемой транзакции *103* (создания смарт-контракта) или другой транзакции вызова контракта (*Call*, *Update*) 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":
↪"4gXy7qtzkaHHH6NkksnZ5pnv8juF65MvjQ9JgVztpgNwLNwuyyr27Db3gCh5YyADqZeBH72EyAkBouUoKvwJ3RQJ
```


(continues on next page)

(продолжение с предыдущей страницы)

```
↪"
 }
```

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 стейта 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",
```

(continues on next page)

(продолжение с предыдущей страницы)


```
    "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": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
↪"4BoG6wQnYyZWyUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs
↪"
  ],
  "version": 1,
  "tx": {
      "type": 103,
      "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
      "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
      "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
      "fee": 500000,
      "timestamp": 1550591678479,
      "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv
↪" ],
      "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

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
↪",
    "blocksize": 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,
```


(continues on next page)

(продолжение с предыдущей страницы)


```
        "id":
↪"5VC2LoFTbrfLkd48bjQkp8CmTyqXJSkJh723qxo9v5pz38tBUjRW9tHLuvwajSvkzQNFxrCc6Yjkgx5R2YR3x5VC
↪",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature":
↪"5VC2LoFTbrfLkd48bjQkp8CmTyqXJSkJh723qxo9v5pz38tBUjRW9tHLuvwajSvkzQNFxrCc6Yjkgx5R2YR3x5VC
↪",
        "recipient": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
        "amount": 300000000000000
      },
      {
        "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": 1000000000000000
      },
      {
        "type": 110,
        "id":
↪"3HewQJtzuaumzX4TvmN7fxVCgnsWTTaLeQjYBVDDuYoEW2ijWd7JME8h1gtsqepv5SDhHPvoMesVNm96br8WRgF8
↪",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature":
↪"3HewQJtzuaumzX4TvmN7fxVCgnsWTTaLeQjYBVDDuYoEW2ijWd7JME8h1gtsqepv5SDhHPvoMesVNm96br8WRgF8
↪",
        "targetPublicKey":
↪"56rV5kcR9SBsxQ9LtNrmp6V72S4BDkZUJaA6ujZswDneDmCTmeSG6UE2FQP1rPXdfpWQNunRw4aijGXxoK3o4puj
↪",
        "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP"
      },
```


(continues on next page)

(продолжение с предыдущей страницы)


```
    {
      "type": 101,
      "id":
↪"5r4uLWn3rwmqbBygNj29iR4YsiV82dYWFeCbepAHhKGXqnn27vE6i811U9H2UZgX8zNQYZciyw3PR6nAdwjSPSp5
↪",
      "fee": 0,
      "timestamp": 1606211535609,
      "signature":
↪"5r4uLWn3rwmqbBygNj29iR4YsiV82dYWFeCbepAHhKGXqnn27vE6i811U9H2UZgX8zNQYZciyw3PR6nAdwjSPSp5
↪",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "permissioner"
    },
    {
      "type": 101,
      "id":
↪"4pBwjviNLtSPEBY5YB7ZdUXVSFnEk4rgscW8r9QQKxdxQZzjwjdq1ZnruMxQo7tomQVJf1Ni6SyVxSHrQZhBJaFN
↪",
      "fee": 0,
      "timestamp": 1606211535608,
      "signature":
↪"4pBwjviNLtSPEBY5YB7ZdUXVSFnEk4rgscW8r9QQKxdxQZzjwjdq1ZnruMxQo7tomQVJf1Ni6SyVxSHrQZhBJaFN
↪",
      "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"
    },
```


(continues on next page)

(продолжение с предыдущей страницы)


```
    {
      "type": 101,
      "id":
↪"2sNwzGbwDL2Es53P8XY5wA9T9wwu3eXJbJUrtXJ9wg49urPjuBejWbidat2z3yZ8JrTpkWWFEsrerCtnC38XuRTJ
↪",
      "fee": 0,
      "timestamp": 1606211535605,
      "signature":
↪"2sNwzGbwDL2Es53P8XY5wA9T9wwu3eXJbJUrtXJ9wg49urPjuBejWbidat2z3yZ8JrTpkWWFEsrerCtnC38XuRTJ
↪",
      "target": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
      "role": "issuer"
    },
    {
      "type": 110,
      "id":
↪"4hLep3GngPEBH2xEmuUZ323muT8BstFdT552e42z6ZXCKGnF1PABGGjEiCkHfr6hMuyvRJ7axD9qoGeWQCU5yaCk
↪",
      "fee": 0,
      "timestamp": 1606211535610,
      "signature":
↪"4hLep3GngPEBH2xEmuUZ323muT8BstFdT552e42z6ZXCKGnF1PABGGjEiCkHfr6hMuyvRJ7axD9qoGeWQCU5yaCk
↪",
      "targetPublicKey":
↪"5nGi8XoiGjjyjbPmjLNy1k2bus4yXLaeuA3Hb7BikwD9tboFwFXJYUmto5Joox76c3pp2Mr1LjgodUJuxryCJofQ
↪",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c"
    },
    {
      "type": 101,
      "id":
↪"nj9Xfqm3pPLmuLsWfDZx4htKaNKAyvhen7tF95T9YwdmK1pqkiCjtaV9AxCwzEceViyo5rHPapigxPyCZdBWvRn
↪",
      "fee": 0,
      "timestamp": 1606211535604,
      "signature":
↪"nj9Xfqm3pPLmuLsWfDZx4htKaNKAyvhen7tF95T9YwdmK1pqkiCjtaV9AxCwzEceViyo5rHPapigxPyCZdBWvRn
↪",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
      "role": "permissioner"
    },
    {
      "type": 101,
      "id":
↪"24AmxdGyH3afYRxPXn5zqvU1Fro1MwVQPDqwkdjCKLddSEiKVhyeMHTAVrRpHu83ZDPMyQkf3ty161PrujmGYtef
↪",
      "fee": 0,
      "timestamp": 1606211535603,
      "signature":
↪"24AmxdGyH3afYRxPXn5zqvU1Fro1MwVQPDqwkdjCKLddSEiKVhyeMHTAVrRpHu83ZDPMyQkf3ty161PrujmGYtef
↪",
      "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
```


(continues on next page)

(продолжение с предыдущей страницы)


```
        "role": "miner"
      },
      {
        "type": 101,
        "id":
↪"4xsEQoh6Z4wDW6jT9UP3SqA1Yv5trbaGfF4uHajWxayBU8hrw2ZAYmtAWwDFytTdc6yqDepj6GwzxZuFYTq6638v
↪",
        "fee": 0,
        "timestamp": 1606211535602,
        "signature":
↪"4xsEQoh6Z4wDW6jT9UP3SqA1Yv5trbaGfF4uHajWxayBU8hrw2ZAYmtAWwDFytTdc6yqDepj6GwzxZuFYTq6638v
↪",
        "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
        "role": "connection_manager"
      },
      {
        "type": 101,
        "id":
↪"FSNaHMC11W3VskpGYfgxt3fqAMvt6gUmgy61CX8mm93QykuRp2E9Z8BtQc8w22Awc6W8CpXGJn6VcpkcBdAx4Tj
↪",
        "fee": 0,
        "timestamp": 1606211535601,
        "signature":
↪"FSNaHMC11W3VskpGYfgxt3fqAMvt6gUmgy61CX8mm93QykuRp2E9Z8BtQc8w22Awc6W8CpXGJn6VcpkcBdAx4Tj
↪",
        "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
        "role": "contract_developer"
      },
      {
        "type": 101,
        "id":
↪"4rfDMTGjbHENy3uiACLmfAHFJWyouhridZHGpynfV8S6aX3XmZHjUSfCvadn3KSzb8eHRq1kmzEaLMxvbqWkUKBY
↪",
        "fee": 0,
        "timestamp": 1606211535600,
        "signature":
↪"4rfDMTGjbHENy3uiACLmfAHFJWyouhridZHGpynfV8S6aX3XmZHjUSfCvadn3KSzb8eHRq1kmzEaLMxvbqWkUKBY
↪",
        "target": "3Mv79dyPX2cvLtRXn1MDDWiCZMBrkw9d97c",
        "role": "issuer"
      },
      {
        "type": 110,
        "id":
↪"4q5iXHv8jZ1qw5FptfBCz1cic14u1M4zCzE1i5qqEA4z6TQmeVFaqhZRpepFpdyGiSyKH4s6XqKPTgxuEJ8Sp4QQ
↪",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature":
↪"4q5iXHv8jZ1qw5FptfBCz1cic14u1M4zCzE1i5qqEA4z6TQmeVFaqhZRpepFpdyGiSyKH4s6XqKPTgxuEJ8Sp4QQ
↪",
        "targetPublicKey":
```


(continues on next page)

(продолжение с предыдущей страницы)


```
↪"25GXtqKBAHTCrHuDoXvwQGXnHKBdeVcjdLvSmQ7SVFq4FDoMWzV78oRkgoS32AFDQ23DvfGFX6QpRkQRShQ4zMJy
↪",
        "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ"
      },
      {
        "type": 101,
        "id":
↪"2gjzK3qSp89ywXCjEpvCHKSeyqoBYR2XCKegZ1ngGrQF8cDGXjA19HN8eYTgw8DRoXy62MM138EXXiZyV7oCaZrt
↪",
        "fee": 0,
        "timestamp": 1606211535599,
        "signature":
↪"2gjzK3qSp89ywXCjEpvCHKSeyqoBYR2XCKegZ1ngGrQF8cDGXjA19HN8eYTgw8DRoXy62MM138EXXiZyV7oCaZrt
↪",
        "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
```


(continues on next page)

(продолжение с предыдущей страницы)


```
↪",
        "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
        "role": "contract_developer"
      },
      {
        "type": 101,
        "id":
↪"FwNbJyr2Est9DFi5uch1ZfkQjDg13asqSsAdm37381aMWMrdaxcjqXMpKus1rxDcxZd5YnD4MNkz1ZpPgZ8nupn
↪",
        "fee": 0,
        "timestamp": 1606211535595,
        "signature":
↪"FwNbJyr2Est9DFi5uch1ZfkQjDg13asqSsAdm37381aMWMrdaxcjqXMpKus1rxDcxZd5YnD4MNkz1ZpPgZ8nupn
↪",
        "target": "3N9nNFySk1zVSVf9DUWR9DiBA1jEmmDDpaJ",
        "role": "issuer"
      },
      {
        "type": 110,
        "id":
↪"ps5vGHxv4DfTFnTXsqeS22hXQQm8uBf1mwnc7gtDvGxGGfEhDq8DvnCjtKukYmuEW6adz5NQGLbaqbMJK7ChYdA
↪",
        "fee": 0,
        "timestamp": 1606211535610,
        "signature":
↪"ps5vGHxv4DfTFnTXsqeS22hXQQm8uBf1mwnc7gtDvGxGGfEhDq8DvnCjtKukYmuEW6adz5NQGLbaqbMJK7ChYdA
↪",
        "targetPublicKey":
↪"5fbBNmkW9LJBUFNJW6vsjnmBzGf2AMwdqgHNvne2iYPMNW2wtDJGmF4PGnqyzTYJyYN3kWNWd4cFf9xBZ8Qi9Hki
↪",
        "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn"
      },
      {
        "type": 101,
        "id":
↪"5BG3AhFnGbDcSDJ88KmXViU2tCxs4VNhXGjgocn2ZCcvCjTbxGjso4DKPkcajUNJBhPZHqgMmEKugVxqBMjNf2YY
↪",
        "fee": 0,
        "timestamp": 1606211535594,
        "signature":
↪"5BG3AhFnGbDcSDJ88KmXViU2tCxs4VNhXGjgocn2ZCcvCjTbxGjso4DKPkcajUNJBhPZHqgMmEKugVxqBMjNf2YY
↪",
        "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
        "role": "permissioner"
      },
      {
        "type": 101,
        "id":
↪"HYoFXRgsyHGTa9JTnCDpJtBu6hr61LTYTA2zGPkUAVaTn6mhHfSKoVJbn91DN2gtqZxNreQnrV4GGnMR4cFikAE
↪",
        "fee": 0,
        "timestamp": 1606211535593,
```


(continues on next page)

(продолжение с предыдущей страницы)


```
        "signature":
↪"HYoFXRgsyHGTa9JTnCDpJtBu6hr61LTYTA2zGPkUAVaTn6mhHfSKoVJbn91DN2gtqZxNreQnrV4GGnMR4cFikAE
↪",
        "target": "3N3jgxvmSsBBV4oz9BcKhT8War1em2sKoJn",
        "role": "contract_developer"
      },
      {
        "type": 101,
        "id":
↪"4snBMYD3dDw9pivJM2YFSJBPPtK4K43YGL8Qjw4APadgZCtqsR4yoo3CZC4bgf5ZffwVWQQzVmfSjxpzsiwCjNju
↪",
        "fee": 0,
        "timestamp": 1606211535592,
        "signature":
↪"4snBMYD3dDw9pivJM2YFSJBPPtK4K43YGL8Qjw4APadgZCtqsR4yoo3CZC4bgf5ZffwVWQQzVmfSjxpzsiwCjNju
↪",
        "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
  ],
```

(continues on next page)

(продолжение с предыдущей страницы)


```
  "signature":
↪"4U4Hmg4mDYrvxaZ3JVzL1Z1piPDZ1PJ61vd1PeS7ESZFkHsUCUqeeAZoszTVr43Z4NV44dqbLv9WdrLytDL6gHuv
↪",
  "fee": 5000000,
  "generator": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "pos-consensus": {
    "base-target": 249912231,
    "generation-signature": "LM83w6eWQHnLJF2D9RQNdNcHAdnZLCLWrn5bfcoqcZy"
  },
  "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

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", "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7"
    ],
    "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": "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
        "roles": [
            {
              "role": "miner"
            }
        ]
        }
    ],
    "timestamp": 1544703449430
}
```

Смотрите также

Методы REST API

Роли участников

Управление ролями участников

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.

Примечание: Для получения информации об ассете рекомендуется использовать метод *GET /assets/details/{assetId}*.

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;
  - `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,
    },
    {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      "assetId": "49KfHPJcKvSAvNKwM7CTofjKHzL87SaSx8eyADBjv5Wi",
      "balance": 10,
      "quantity": 10000000000,
      "reissuable": false,
    }
  ]
}
```

GET /assets/balance-v2/{address}

Метод возвращает баланс всех ассетов адреса, в том числе баланс ассетов, выпущенных смарт-контрактом.

В ответе возвращаются следующие параметры:

- `address` – адрес участника;
- `balances` – объект с балансами участника:
  - `name` – имя ассета;
  - `assetId` – ID ассета;
  - `balance` – баланс ассета;
  - `reissuable` – флаг, который указывает на перевыпускаемость ассета;
  - `sponsorshipIsEnabled` – флаг, который принимает значение `true` или `false`, и который в соответствии со значением позволяет или не позволяет платить комиссию в несистемном (не WEST) токене;
  - `sponsorBalance` – средства, выделенные для оплаты транзакций по спонсируемым ассетам;
  - `quantity` – общее количество выпущенных токенов ассета;
  - `decimals` – максимальное количество знаков после запятой для конкретного ассета;
  - `description` – описание ассета, заданное участником, который его выпустил;
  - `timestamp` – время выпуска ассета;
  - `issueHeight` – высота, на которой был выпущен ассет;
  - `issuer` – адрес участника, который выпустил ассет.

Пример ответа:

GET /assets/balance-v2/{address}:

```
{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "balances": [
    {
      "name": "WBTC",
      "assetId": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
      "balance": 100000000,
      "reissuable": true,
      "sponsorshipIsEnabled": true,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      "sponsorBalance": 0,
      "quantity": 10000000,
      "decimals": 8,
      "description": "Wrapped BTC token",
      "timestamp": 100,
      "issueHeight": 100,
      "issuer": {}
    }
  ]
}
```

POST /assets/balance

Метод возвращает набор пар `assetid – balance` для каждого адреса из переданных при вызове метода в поле `addresses`.

В ответе возвращаются следующие параметры:

- `assetid` – ID ассета;
- `balance` – баланс ассета.

Пример ответа для одного адреса:

POST /assets/balance:

```
[{"3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB": {"assedId": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB
↪", "balance": 1}]
```

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

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 ноде.

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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    }
  ]
}
```

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":
↪"6RwMUQcwrxtKDgM4ANes9Amu5EJgyfF9Bo6nTpXyD89ZKMAcpCM97igbWf2MmLXLdqNxdsUc68fd5TyRBEB6nqf
↪"
}
```

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 получает блокчейн адрес (owner-address) и, если среди пиров такой адрес есть, то возвращает соответствующее ему имя хоста (hostname) и IP-адрес ноды.

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"
    },
    {
```

(continues on next page)

(продолжение с предыдущей страницы)


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


Смотрите также

Методы REST API

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.

Важно: Метод POST /utils/hash/fast недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

Входящая строка преобразуется в байты по кодировке UTF-8, от этих байтов вычисляется хэш. Для Waves-криптографии используется алгоритм Blake2b256. Для ГОСТ-криптографии используется алгоритм ГОСТ 34.11-2012 (256). Результат преобразуется в формат Base58.

Пример ответа:

POST /utils/hash/fast:


```
{
  "message": "ridethewaves!",
  "hash": "DJ35ymschUFDmqCnDJewjcnVExVkWgX7mJDXhFy9X8oQ"
}
```

POST /utils/hash/secure

Метод возвращает двойную хэш-сумму строки, переданной в запросе. При этом применяется алгоритм Blake2b256, если в системе используется WAVES криптография (то есть в конфигурационном файле ноды *параметру node.crypto.type* присвоено значение `WAVES`) или ГОСТ 34.11-2012 (256), если используется ГОСТ криптография (то есть в конфигурационном файле ноды *параметру node.crypto.type* присвоено значение `GOST`).

Важно: Метод POST /utils/hash/secure недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

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

REST API: Отладка блокчейна

Для отладки блокчейн-сети предусмотрены методы группы debug:

Важно: Все методы группы debug недоступны при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) методы можно использовать.

GET /debug/blocks/{howMany}

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

Пример ответа:

GET /debug/blocks/{howMany}:


```
[
    {
        "226": "7CkZxrAjU8bnat8CjVAPagobNYazyv1HASubmp7YYqGe"
    },
    {
        "226": "GS3y9fUHAKCamq52TPsjizDVir8J7iGoe8P2XZLasxsC"
    },
    {
        "226": "B9LmhGGDdvcfUA9JEWvyVrT9sazZE6gibpAN13xUN7KV"
    },
    {
        "226": "Byb9MHtwYf3MFyi2tbhQ3GTdCct5phKq9REkbjQTzdne"
    },
    {
        "226": "HSxSHbiV4tZc8RaN6jxdhgtkAhjxuLn76uHxerMRUefA"
    }
]
```


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
↪",
"QSuBkEtVe9nik5T5S33ogeCbgKy7ihBkS2pwYayK23m4ANier83ThpajEzvpbyPy9pPWZc5St8mYUKxXDscKuRC
↪",
"4udpNnz3e1M1GbVZxtwfg8gpF6EbiKxRCRBwi6iRMyLsvh5J2Ec9Wqyu2sq2KYL75o12yiP8TszworeUfuxNmJ5g
↪",
"5BZYZ4RZAJjM5KKCaHpyUsXnb4uunnM5kcfTojc5QzQo3vyP2w3YD4qrALizkkQQR4ziS77BoAGb56QCecUtHFFN
↪",
"5JwfLaF1oGxRXVCdDbFuKpxrvxgLCGU3kCFwxUhLL8G3xV211MrKBuAuQ4MaC5uN574uV9U8M6HfHTMERnfr5jGJ
↪",
"4bysMhz14E1rC7dLYScfVVqPmHqzi8jdhcnkruJmCNL86TwV2cbF7G9YVchvTrv9qbQZ7JQownV59gRRcD26zm16
↪"
],
"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,
    "3JJDpPDqSPokKp5jEmzwMzmaPUyopLZjW1C": 800000000,
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t": 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,
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t": 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

1.9 Разработка и применение смарт-контрактов

Определение и общее описание работы смарт-контрактов блокчейн-платформы Waves Enterprise приведено в статье *Смарт-контракты*.

1.9.1 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 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 Mainnet в конфигурационном файле предустановлен открытый репозиторий Waves Enterprise.

1.9.2 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 используемыми **protobuf**-файлами (при использовании gRPC) или параметрами авторизации смарт-контракта (при использовании REST API).

Важно: Начиная с релиза 1.12 (после *активации фичи* 1120) создание и вызов REST контрактов невозможны. В последующих релизах REST контракты невозможно будет исполнить. Рекомендуется использовать gRPC контракты вместо REST контрактов.

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:

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 каждом вызове 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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
CallContractTransactionType = 104

AUTH_METADATA_KEY = "authorization"

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

    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,
        r   esults=[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)]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        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,
            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-protobuf-archive-x.x.x.zip, который размещен 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. Если вы работаете в сети Waves Enterprise Mainnet, то чтобы поместить ваш смарт-контракт в открытый репозиторий, свяжитесь со службой технической поддержки Waves Enterprise.

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

Пример смарт-контракта с использованием REST API

Важно: Начиная с релиза 1.12 (после *активации фичи* 1120) создание и вызов REST контрактов невозможны. В последующих релизах REST контракты невозможно будет исполнить. Рекомендуется использовать gRPC контракты вместо REST контрактов.

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

```

(continues on next page)

(продолжение с предыдущей страницы)

```
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")
    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",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        "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 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

Для разработки, тестирования и развертывания смарт-контрактов в публичных блокчейн сетях Waves Enterprise вы можете использовать инструментарии JS Contract SDK Toolkit или Java/Kotlin Contract SDK Toolkit. Они описаны в следующих разделах:

Создание смарт-контрактов с помощью JS Contract SDK

В этом разделе описан **JS Contract SDK Toolkit** – инструментарий для разработки, тестирования и развертывания смарт-контрактов в публичных блокчейн сетях Waves Enterprise. Этот инструментарий позволяет быстро освоить экосистему Waves Enterprise, используя такие языки программирования, как JavaScript или TypeScript, поскольку смарт-контракт разворачивается в Docker-контейнере.

Контракт можно развернуть в различных средах и сетях. Например, для локальной разработки смарт-контрактов и их тестирования вы можете локально развернуть свою сеть (создать локальную среду) на основе ноды в ознакомительном режиме (Sandbox) и развернуть контракты в этой сети.

Для *развёртывания контракта* в различных средах используйте инструмент **WE Contract Command line interface (CLI)**.

Системные требования

Перед началом работы убедитесь, что на вашей машине установлено следующее ПО:

- Docker
- Node.js (LTS)

Быстрый старт

Для создания вашего нового проекта выполните в командной строке следующую команду:

С помощью npm npx

```
npx create-we-contract YourContractName -t path-to-contract -n package-name
```

или

```
npm create we-contract YourContractName -t path-to-contract -n package-name
```

или с помощью yarn

```
yarn create we-contract YourContractName -t path-to-contract -n package-name
```

Таким образом будет создан ваш первый смарт-контракт, готовый к разработке и внедрению в блокчейн Waves Enterprise. Затем выполните следующую команду для инициализации зависимостей и начала разработки проекта:

```
npm i // or yarn
```

Конфигурация

Файл конфигурации используется для того, чтобы задать имя образа и имя контракта, которые будут отображаться в проводнике. Также в файле конфигурации можно задать тег образа (свойство `name`), который будет использоваться для отправки контракта в реестр.

Добавьте конфигурационный файл `contract.config.js` в корневую директорию вашего проекта для инициализации конфигурации контракта.

Если вы создали проект с помощью команды `create-we-contract` (как описано выше в разделе *Быстрый старт*), то конфигурация настраивается по умолчанию.

Конфигурация по умолчанию

Ниже приведён пример конфигурации по умолчанию:

```
module.exports = {
  image: "my-contract",
  name: 'My Contract Name',
  version: '1.0.1',
  networks: {
  /// ...
  }
}
```

Конфигурация сети

В разделе `networks` задайте конфигурацию для вашей сети:

```
module.exports = {
  networks: {
    "sandbox": {
        seed: "#your secret seed phrase" // or get it from env process.env.MY_SECRET_SEED

        // also you can provide
        registry: 'localhost:5000',
        nodeAddress: 'http://localhost:6862',
        params: {
          init: () => ({
            paramName: 'paramValue'
          })
        }
    }
  }
}
```

- seed – если вы хотите развернуть контракт в сети в ознакомительном режиме (Sandbox), укажите seed-фразу инициатора контракта;

- `registry` – если вы использовали определенный реестр Docker, укажите имя этого реестра;
- `nodeAddress` – укажите конкретный адрес ноды для развертывания.
- `params.init` – чтобы задать параметры инициализации, задайте функцию.

Осторожно: Не публикуйте свои секретные фразы в открытых хранилищах.

Развертывание контракта

Смарт-контракты выполняются, как только они развёрнуты в блокчейне. Для развертывания контракта используйте команду `deploy` в WE Contract CLI:

```
we-toolkit deploy -n testnet
```

где testnet – название сети, указанное в конфигурационном файле. Например, для развертывания контракта в сети в ознакомительном режиме (Sandbox), выполните следующую команду:

```
we-toolkit deploy -n sandbox
```

Набор инструментов для разработки смарт контрактов Contract SDK Toolkit

Основные понятия

Для создания класса контракта в Contract SDK Toolkit необходимо указать аннотации к методам. Следующие аннотации являются наиболее важными:

- `Contract` – регистрация класса как контракта;
- `Action` – регистрация обработчика действия контракта;
- `State` – декоратор свойства класса для доступа к состоянию контракта;
- `Param` – декоратор, который отображает параметры транзакции на параметры действия класса контракта.

SDK предоставляет шаблоны контрактов, в которые вы можете добавить свою бизнес-логику:

```
@Contract
export class ExampleContract {
    @State state: ContractState;

    @Action
    greeting(@Param('name') name: string) {
        this.state.set('Greeting', `Hello, ${name}`);
    }
}
```

Методы

Методы управления состоянием смарт контракта

Класс ContractState предоставляет методы для записи в состояние контракта. В документации ноды описаны доступные на данный момент типы данных в состоянии контракта. Contract SDK поддерживает все доступные на данный момент типы данных в состоянии контракта.

Запись

Самый простой способ записать состояние – использовать метод set. Этот метод автоматически приводит тип данных.

```
this.state.set('key', 'value')
```

Для явного приведения типов используйте методы, указанные ниже:

```
// for binary
this.state.setBinary('binary', Buffer.from('example', 'base64'));

// for boolean
this.state.setBool('boolean', true);

// for integer
this.state.setInt('integer', 102);

// for string
this.state.setString('string', 'example');
```

Считывание

Чтение состояния в настоящее время является асинхронным и зависит от конфигурации контракта.

```
@Contract
export class ExampleContract {
    @State state: ContractState;

    @Action
    async exampleAction(@Param('name') name: string) {
        const stateValue: string = await this.state.get('value', 'default-value');
    }
}
```

Осторожно: У метода state.get нет информации о типе внутреннего состояния во время выполнения. Для явного приведения типов используйте методы getBinary, getString, getBool, getNum.

Write Actions

Ключевыми декораторами являются `Action` и `Param`.

Init Actions

Для описания действия создания контракта задайте параметру `onInit` декоратора действия значение `true`.

```
@Contract
export class ExampleContract {
    @State state: ContractState;

    @Action({onInit: true})
    exampleAction(@Param('name') name: string) {

        this.state.set('state-initial-value', 'initialized')
    }
}
```

По умолчанию используется имя метода контракта `action`. Для того, чтобы задать другое имя действия, присвойте его параметру `name` декоратору.

```
@Contract
export class ExampleContract {
    @State state: ContractState;

    @Action({name: 'specificActionName'})
    exampleAction() {
        // Your code
    }
}
```

Обновление версии контракта

Для обновления версии контракта используйте метод `update`. Метод обновляет последний развернутый контракт. Если ни один контракт не был развёрнут, метод ничего не обновляет.

```
we-cli update -n, --network <char>
```

Смотрите также

Разработка и применение смарт-контрактов

Создание смарт-контрактов с помощью Java/Kotlin Contract SDK

Смарт-контракты

Создание смарт-контрактов с помощью Java/Kotlin Contract SDK

В этом разделе описан **Java/Kotlin Contract SDK Toolkit** – инструментарий для разработки, тестирования и развертывания Docker смарт-контрактов в публичных блокчейн сетях Waves Enterprise. Этот инструментарий позволяет быстро освоить экосистему Waves Enterprise, используя любой из языков программирования JVM, поскольку смарт-контракт разворачивается в Docker-контейнере. Вы можете создать смарт-контракт с помощью любого из JVM языков, например Java.

Контракт можно развернуть в различных средах и сетях. Например, для локальной разработки смарт-контрактов и их тестирования вы можете локально развернуть свою сеть (создать локальную среду) на основе ноды в ознакомительном режиме (Sandbox) и развернуть контракты в этой сети.

Вся обработка транзакций осуществляется с помощью методов одного класса, помеченных аннотацией `@ContractHandler`. Методы, реализующие логику обработки, помечены `@ContractInit` (для CreateContractTx) и `@ContractAction` (для CallContractTx).

Для развёртывания контракта необходимо выпустить транзакции *103* и *104*.

Системные требования

Перед началом разработки смарт-контрактов убедитесь, что на вашей машине установлено следующее ПО:

- Docker
- JDK версии 8 и выше

Для запуска смарт-контрактов необходимо следующее ПО:

- Docker
- JRE версии 8 и выше

Зависимости

Быстрый старт

Для создания вашего нового контракта выполните следующие шаги.

Примечание: Все примеры, приведённые ниже, доступны в разделе Samples GitHub-репозитория Waves Enterprise.

---

1. Создайте обработчик контрактов

```
@ContractHandler
public class SampleContractHandler {

    private final ContractState contractState;
    private final ContractTransaction tx;

    private final Mapping<List<MySampleContractDto>> mapping;

    public SampleContractHandler(ContractState contractState, ContractTransaction tx) {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        this.contractState = contractState;
        mapping = contractState.getMapping(
          new TypeReference<List<MySampleContractDto>>() {
          }, "SOME_PREFIX");
        this.tx = tx;
    }
}
```

2. Добавьте методы обработки транзакций контракта @ContractInit и @ContractAction

```
public class SampleContractHandler {

  // ...

  @ContractInit
  public void createContract(String initialParam) {
    contractState.put("INITIAL_PARAM", initialParam);
  }

  @ContractAction
  public void doSomeAction(String dtoId) {
      contractState.put("INITIAL_PARAM", Instant.ofEpochMilli(tx.getTimestamp().
↪getUtcTimestampMillis()));

      if (mapping.has(dtoId)) {
          throw new IllegalArgumentException("Already has " + dtoId + " on state");
      }
      mapping.put(dtoId,
              Arrays.asList(
                      new MySampleContractDto("john", 18),
                      new MySampleContractDto("harry", 54)
              ));
  }
}
```

3. Отправьте контракт с указанным обработчиком контракта и настройками

```
public class MainDispatch {
  public static void main(String[] args) {
      ContractDispatcher contractDispatcher = GrpcJacksonContractDispatcherBuilder.
↪builder()
              .contractHandlerType(SampleContractHandler.class)
              .objectMapper(getObjectMapper())
              .build();

      contractDispatcher.dispatch();
  }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  private static ObjectMapper getObjectMapper() {
      ObjectMapper objectMapper = new ObjectMapper();
      objectMapper.registerModule(new JavaTimeModule());
      return objectMapper;
  }
}
```

4. Создайте Docker-образ

```
FROM openjdk:8-alpine
MAINTAINER Waves Enterprise <>

ENV JAVA_MEM="-Xmx256M"
ENV JAVA_OPTS=""

ADD build/libs/*-all.jar app.jar

RUN chmod +x app.jar
RUN eval $SET_ENV_CMD
CMD ["/bin/sh", "-c", "eval ${SET_ENV_CMD} ; java $JAVA_MEM $JAVA_OPTS -jar app.jar"]
```

5. Отправьте образ в Docker-репозиторий, используемый нодой WE, которая майнит транзакции по контрактам

Опубликуйте образ в репозиторий, используемый нодой блокчейн сети Waves Enterprise. Для удобства вы можете использовать bash-скрипт build_and_push_to_docker.sh, который соберёт образ вашего смарт-контракта, опубликует его в указанный реестр и выведет image и imageHash на экран.

```
./build_and_push_to_docker.sh my.registry.com/contracts/my-awesome-docker-contract:1.0.0
```

6. Подпишите и отправьте в блокчейн транзакции создания и вызова опубликованного смарт-контракта

Для создания контракта вам понадобятся image и imageHash опубликованного контракта.

Пример CreateContractTx:

```
{
  "image": "my.registry.com/contracts/my-awesome-docker-contract:1.0.0",
  "fee": 0,
  "imageHash": "d17f6c1823176aa56e0e8184f9c45bc852ee9b076b06a586e40c23abde4d7dfa",
  "type": 103,
  "params": [
      {
        "type": "string",
        "value": "createContract",
        "key": "action"
      },
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      {
        "type": "string",
        "value": "initialValue",
        "key": "createContract"
      }
  ],
  "version": 2,
  "sender": "3M3ybNZvLG7o7rnM4F7ViRPnDTfVggdfmRX",
  "feeAssetId": null,
  "contractName": "myAwesomeContract"
}
```

Для вызова контракта вам понадобится `contractId = CreateContractTx.id`.

Пример CallContractTx:

```
{
   "contractId": "7sVc6ybnqZr523xWK5Sg7xADsX597qga8iQNAS9f1D3c",
   "fee": 0,
   "type": 104,
   "params": [
     {
     "type": "string",
     "value": "doSomeAction",
     "key": "action"
     },
     {
       "type": "string",
       "value": "someValue",
       "key": "createContract"
     }
   ],
   "version": 2,
   "sender": "3M3ybNZvLG7o7rnM4F7ViRPnDTfVggdfmRX",
   "feeAssetId": null,
   "contractVersion": 1
}
```

Примечания по использованию

Использование с Java 11 и выше

Библиотека протестирована с Java 8, 11 и 17. При использовании с Java версии 11 и выше необходимо указать дополнительные опции Java для io.grpc, чтобы включить оптимизацию:

```
--add-opens java.base/jdk.internal.misc=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-
↪UNNAMED -Dio.netty.tryReflectionSetAccessible=true
```

Полный пример можно найти в Dockerfile для Java 17.

Смотрите также

Разработка и применение смарт-контрактов

Создание смарт-контрактов с помощью JS Contract SDK

Смарт-контракты

Клиент для WE contract SDK (Java/Kotlin Contract SDK)

В этом разделе описан **Клиент для WE contract SDK**. Клиент для контрактов используется для взаимодействия с контрактами из бэкенд-кода Java/Kotlin-приложений.

Основные абстракции

- ContractBlockingClientFactory – фабрика для создания клиента для контракта;
- NodeBlockingServiceFactory – фабрика, которая создает сервисы для взаимодействия с нодой;
- TxService – интерфейс для работы с транзакциями на ноде;
- TxSigner – интерфейс для подписания транзакций на ноде;
- ConverterFactory – фабрика для создания сервисов для преобразования значений при работе с состоянием;
- ContractToDataValueConverter – интерфейс для преобразования значений в объекты DataValue;
- ContractFromDataEntryConverter – интерфейс для преобразования значений Data Entry из состояния;
- ContractClientParams – класс для настроек создаваемого клиента;
- ContractSignRequestBuilder – конструктор SignRequest(transaction); создает объект создания контракта (103-я транзакция) или объект вызова контракта (104-я транзакция).

Быстрый старт

Для создания клиента для WE contract SDK выполните следующие шаги.

Примечание: Все примеры, приведённые ниже, доступны в GitHub-репозитории Waves Enterprise. Помимо этого в GitHub-репозитории Waves Enterprise представлены примеры

---

1. Создайте и настройте службы для работы с нодой:

```
val objectMapper = ObjectMapper()
 .configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false)
 .configure(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS, false)
 .registerModule(JavaTimeModule())
 .registerModule(
     KotlinModule.Builder()
         .configure(KotlinFeature.NullIsSameAsDefault, true)
         .build()
)
```

(continues on next page)

(продолжение с предыдущей страницы)

```
val converterFactory = JacksonConverterFactory(objectMapper)
val feignNodeClientParams = FeignNodeClientParams(
    url = "{node.url}",
    decode404 = true,
    connectTimeout = 5000L,
    readTimeout = 3000L,
    loggerLevel = Logger.Level.FULL,
)
val feignTxService = FeignTxService(
    weTxApiFeign = FeignWeApiFactory.createClient(
        clientClass = WeTxApiFeign::class.java,
        feignProperties = feignNodeClientParams,
    )
)
val feignNodeServiceFactory = FeignNodeServiceFactory(
    params = feignNodeClientParams
)
val contractProperties = ContractProperties(
    senderAddress = "",
    fee = 0L,
    contractId = "contractId",
    contractVersion = 1,
    version = 1,
    image = "image",
    imageHash = "imageHash",
    contractName = "contractName",
)
val contractClientParams = ContractClientParams(localValidationEnabled = true)
val contractSignRequestBuilder = ContractSignRequestBuilder()
    .senderAddress(Address.fromBase58(contractProperties.senderAddress))
    .fee(Fee(0L))
    .contractId(ContractId.fromBase58(contractProperties.contractId))
    .contractVersion(ContractVersion(contractProperties.contractVersion))
    .version(TxVersion(contractProperties.version))
    .image(ContractImage(contractProperties.image))
    .imageHash(Hash.fromHexString(contractProperties.imageHash))
    .contractName(ContractName(contractProperties.contractName))
val contractClientParams = ContractClientParams(localValidationEnabled = true)
```

2. Сформируйте данные транзакции:

```
 val contractSignRequestBuilder = ContractSignRequestBuilder()
.senderAddress(Address.fromBase58(contractProperties.senderAddress))
.fee(Fee(0L))
.contractId(ContractId.fromBase58(contractProperties.contractId))
.contractVersion(ContractVersion(contractProperties.contractVersion))
.version(TxVersion(contractProperties.version))
.image(ContractImage(contractProperties.image))
.imageHash(Hash.fromHexString(contractProperties.imageHash))
.contractName(ContractName(contractProperties.contractName))
```

3. Создайте фабрику клиента для контракта и настройте ее:

```
val contractFactory = ContractBlockingClientFactory(
   contractClass = TestContractImpl::class.java,
   contractInterface = TestContract::class.java,
   converterFactory = converterFactory,
   contractClientProperties = contractClientParams,
   contractSignRequestBuilder = contractSignRequestBuilder,
   nodeBlockingServiceFactory = nodeBlockingServiceFactory,
)
```

4. Создайте TxSigner

```
val txServiceTxSigner = TxServiceTxSignerFactory(
  txService = feignTxService,
)
```

5. Создайте и вызовите методы клиента

```
val executionContext: ExecutionContext = contractFactory.executeContract(
txSigner = txSigner) { contract ->
    contract.create()
}
```

Смотрите также

Создание смарт-контрактов с помощью Java/Kotlin Contract SDK

Разработка и применение смарт-контрактов

Создание смарт-контрактов с помощью JS Contract SDK

Смарт-контракты

1.9.3 Загрузка смарт-контракта в репозиторий

Если вы работаете в блокчейн-сети Waves Enterprise Mainnet, то чтобы поместить ваш смарт-контракт в открытый репозиторий, свяжитесь со службой технической поддержки 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": "",
"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.

1.9.4 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": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUV2", \
      "password": "", \
      "params": [], \
      "type": 103, \
      "version": 1 \
  }' 'http://my-node:6862/transactions/sign'
```

Ответ метода `sign`, который передается методу `broadcast`:

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
  "fee": 100000000,
  "timestamp": 1550591678479,
  "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv
↪" ],
  "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 '{ \
```

(continues on next page)

(продолжение с предыдущей страницы)

```
{
    "type": 103, \
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M", \
    "fee": 500000, \
    "timestamp": 1550591678479, \
    "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv
↪" ], \
    "version": 1, \
    "image": "my-contract:latest", \
    "imageHash":
↪"7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "my-contract", \
    "params": [], \
    "height": 1619 \
}
}' 'http://my-node:6862/transactions/broadcast'
```

1.9.5 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": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
"fee": 10, \
"sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
"password": "", \
"type": 104, \
"version": 1, \
"params": [ \
    { \
        "type": "integer", \
        "key": "a", \
        "value": 1 \
    } \
```

(continues on next page)

(продолжение с предыдущей страницы)

```
] \
}' 'http://my-node:6862/transactions/sign'
```

Ответ метода `sign`, который передается методу `broadcast`:

```
{
"type": 104,
"id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
"sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
"senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
"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": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq", \
"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

1.10 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.

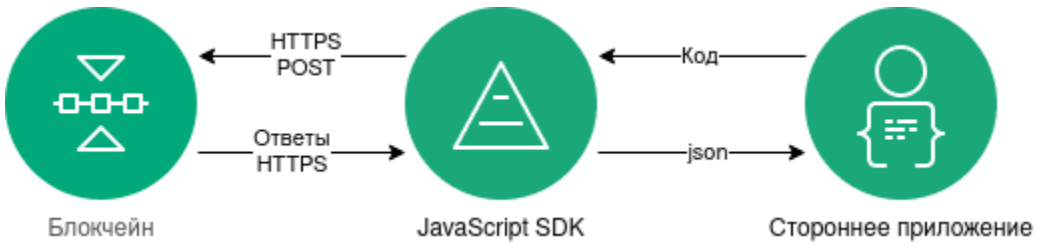
JavaScript SDK поддерживает:

- работу как в браузере, так и в среде Node.js;
- 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`;
- стандарты шифрования ГОСТ.

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.

Подробнее установка и работа с JavaScript SDK описана в следующих разделах:

1.10.1 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-авторизацией*.

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 *Создание и отправка транзакций при помощи JS SDK*.

Криптографические методы ноды, используемые JavaScript SDK

JavaScript SDK доступны три криптографических метода:

- `crypto/encryptCommon` – шифрование данных для всех получателей единым ключом CEK, который в свою очередь оборачивается уникальными ключами KEK 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

1.10.2 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

1.10.3 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 расшифровка 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': 'wavesenterprise
↪'} });
}
```

(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 Mainnet

1.10.4 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': 'wavesenterprise
↪'} });
}

(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],
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    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)
  }

})();
```

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':
↪'wavesenterprise'} });
}

(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);
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    // Transaction data
    // https://docs.wavesenterprise.com/en/latest/how-the-platform-works/data-structures/
↪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':
↪'wavesenterprise'} });
}

(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),
    };
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    const Waves = createApiInstance({
        initialConfiguration: wavesApiConfig,
        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':
↪'wavesenterprise'} });
}

(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: 'we-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);
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        console.log('Broadcast docker create result: ', result)
    } 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':
↪'wavesenterprise'} });
}

(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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        contractVersion: 1,
        timestamp,
        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': 'wavesenterprise
↪'} });
}

(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
```

(continues on next page)

(продолжение с предыдущей страницы)

```
const seed = Waves.Seed.fromExistingPhrase(seedPhrase);

const transfer1Body = {
  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
  }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  }

  const dockerCall2 = Waves.API.Transactions.CallContract.V4(dockerCall1Body);

  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": "9QUUuQ5XetCe2wEyrSX95NEVzPw2bscfcFfAzVZL5ZJN",
    "type": "file",
    "data": base64Text,
    "hash": hash,
    "info": {
      "filename":"test-send1.txt",
      "size":1,
      "timestamp": Date.now(),
      "author":"temakolodko@gmail.com",
      "comment":""
    },
    "fee": 5000000,
    "password": "sfgKYBFCF@#$fsdf()*%",
    "timestamp": Date.now(),
    "version": 3,
    "apiKey": 'wavesenterprise',
  }
  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':
↪'wavesenterprise'} });
}

(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 {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        const result = await issueTx.broadcast(seed.keyPair);

        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

1.10.5 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: {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    access_token,
    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

1.10.6 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/
↪eJyjht7bSpjLYlSU8gVXNapU3MG8xgWm3uavW37aPz/
↪KTcROK7OjOA3dpCLXfZ4YjCV3OW2r1CCaUhOMPBCX64QA/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

1.11 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.

Важно: Если вы передаёте конфиденциальные данные в своей приватной блокчейн сети, то для перехода с версий более старых, чем 1.7.2, необходимо сначала перейти на версию 1.7.2, а затем – на версию 1.8 и выше. Это связано с изменением протокола передачи конфиденциальных данных.

1.11.1 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 получат 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`.

1.11.2 Изменение группы доступа

Изменять состав группы доступа могут только ее участники, добавленные в поле `owners` при создании группы, а также сам ее создатель – **владельцы группы доступа к конфиденциальным данным**.

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.

1.11.3 Отправка конфиденциальных данных в сеть

Для отправки конфиденциальных данных в сеть предусмотрены

- gRPC методы
  - *SendData*,
  - *SendLargeData*.
- REST API методы
  - *POST /privacy/sendData*,
  - *POST /privacy/sendDataV2*,
  - *POST /privacy/sendLargeData*.

    С помощью методов **POST /privacy/sendData** и **POST /privacy/sendDataV2** вы можете отправить данные размером до **20 мегабайт**, с помощью метода **POST /privacy/sendLargeData** – данные размером не менее **20 мегабайт**.

Важно: Методы для отправки конфиденциальных данных в сеть недоступны при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) методы можно использовать.

Эти методы требуют авторизации.

Смотрите также

Описание транзакций

PrivacyPublicService

REST API: обмен конфиденциальными данными и получение информации о группах доступа

Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным

1.12 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

1.13 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.

1.13.1 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 opType, определяющем тип совершаемого действия, указывается `add` (добавление новой ноды).
4. Транзакция 111 попадает в блок, а 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.

1.13.2 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

1.14 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

1.15 Архитектура

1.15.1 Устройство платформы

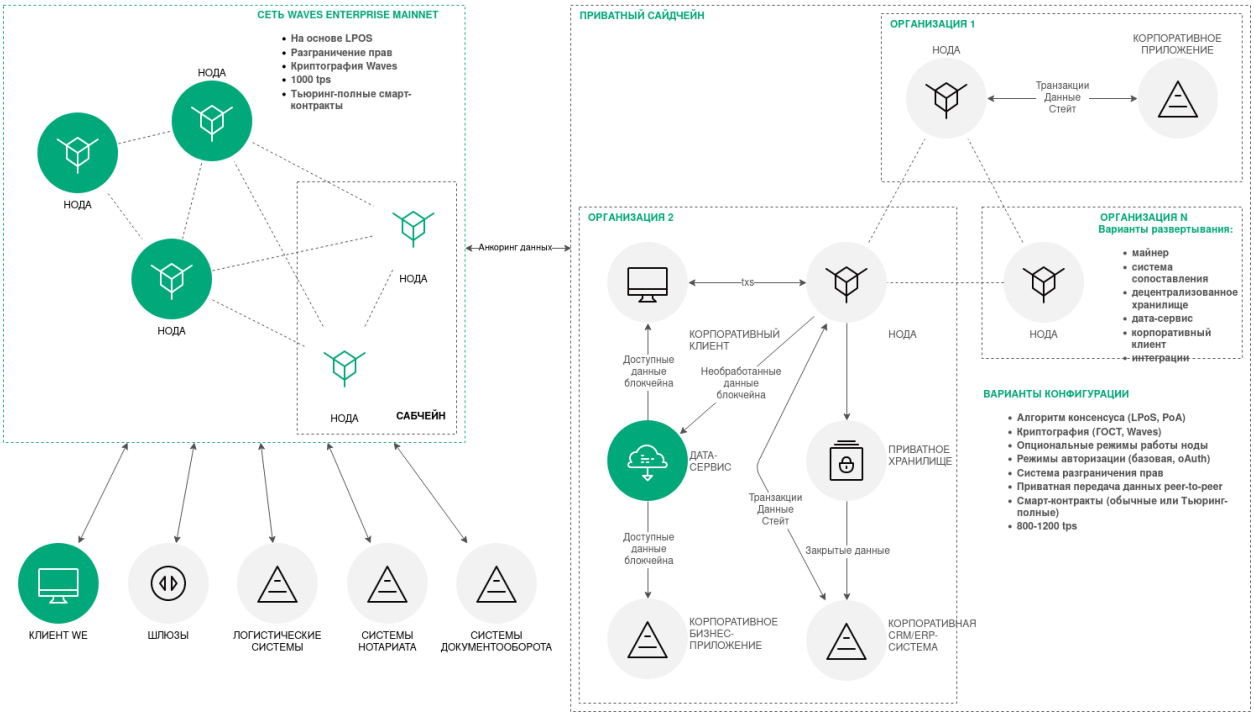
Платформа Waves Enterprise построена на базе технологии распределенного реестра и представляет собой фрактальную сеть, состоящую из двух элементов:

- **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:



1.15.2 Устройство ноды и дополнительных сервисов

Каждая нода блокчейна – это самостоятельный участник сети, имеющий ПО для работы в ней. Нода состоит из следующих компонентов:

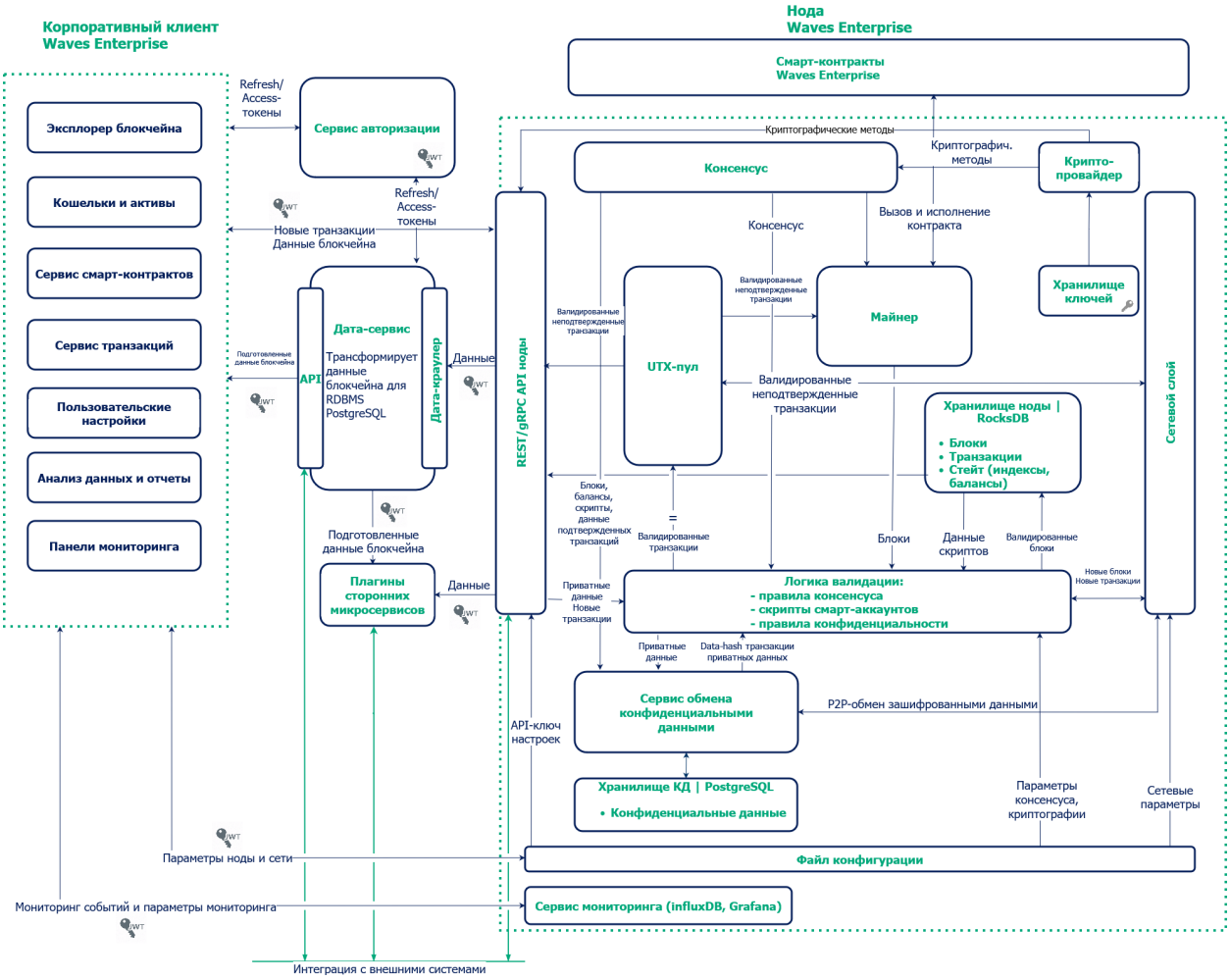
- **Сервисы консенсуса и криптографические библиотеки (Consensus and cryptolibraries)** – компоненты, отвечающие за механизм достижения согласия между узлами, а также за криптографические алгоритмы.
- **API-интерфейсы ноды (Node API)** – интерфейсы gRPC и REST API ноды, позволяющие получать данные из 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 подтверждённых 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

1.16 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.

1.16.1 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.

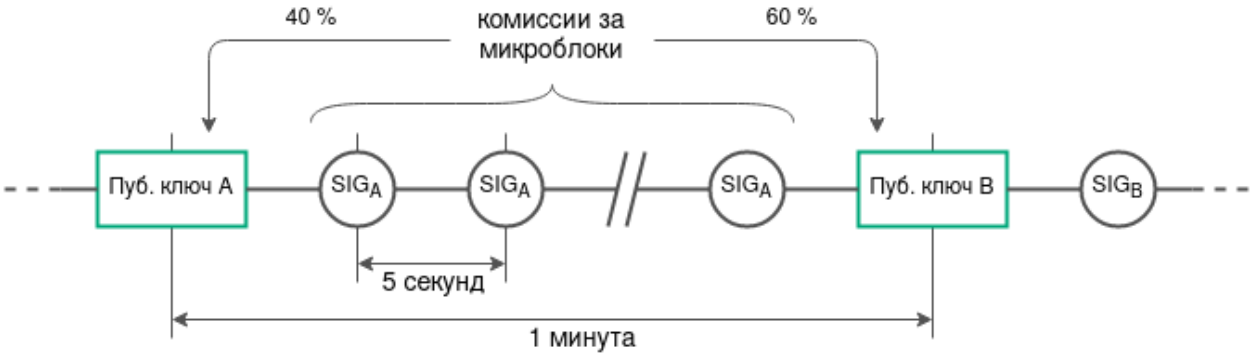
3. После формирования ключевого блока майнер раунда формирует **liquid block**: каждые 5 секунд создает и рассылает по сети микроблоки, содержащие транзакции. На этом этапе микроблоки не валидируются алгоритмом консенсуса, что увеличивает скорость их создания. Первый микроблок ссылается на ключевой блок, 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.

1.16.2 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:



1.16.3 Механизм вознаграждения валидаторов смарт-контрактов

Протокол Waves-NG предусматривает финансовую мотивацию для валидаторов смарт-контрактов. За каждую транзакцию исполнения смарт контракта, который *требует валидации* (т.е. использует политики валидации `Majority` или `MajorityWithOneOf`) в блокчейне Waves Enterprise предусмотрена комиссия в WAVES. Вознаграждение распределяется между майнерами и валидаторами следующим образом:

- **25%** от комиссии за транзакцию исполнения смарт контракта получают валидаторы. Вознаграждение распределяется между валидаторами в равных долях.
- **75%** от комиссии за транзакцию исполнения смарт контракта получают майнеры. Вознаграждение распределяется между майнерами следующим образом:
  - 40% от 75%, то есть **30%** комиссии получает майнер, создавший блок в текущем раунде;
  - 60% от 75%, то есть **45%** комиссии получает майнер следующего раунда.

1.16.4 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

1.17 Неизменяемость данных в блокчейне

Процесс построения цепочки блоков гарантирует невозможность удаления данных из блокчейна.

Пользователь формирует транзакцию. Перед отправкой транзакции он генерирует для нее цифровую подпись, используя закрытый ключ своего аккаунта. Этот ключ известен только пользователю. После подписания у транзакции появляется поле `proofs` с электронной подписью. Теперь «тело» транзакции заверено, ее неизменность и принадлежность автору (открытый ключ, public key) подтверждены.

Пользователь с помощью запросов POST /transactions/broadcast и POST /transactions/signAndbroadcast отправляет подписанную транзакцию в API ноды (узла), к которой у него есть доступ.

Нода проверяет подпись, структуру транзакции, наличие контракта и т.д.. Если все проверки проходят корректно, нода принимает (валидирует) транзакцию.

Провалидированная транзакция попадает в UTX-пул ноды. Эта нода дальше будет рассылать информацию об этой транзакции всем другим нодам, с которыми имеет соединение. Таким образом каждая нода сети будет иметь эту транзакцию.

Для транзакции в UTX-пуле есть два варианта развития событий:

1. транзакция будет добавлена в блок в процессе майнинга, либо
2. транзакция будет удалена из UTX-пула и не попадёт в блок.

Каждая нода в блокчейне знает параметры консенсуса, согласно которому она должна выпускать блоки. Та нода, которая определена лидером (майнером раунда), отбирает те транзакции из UTX пула, которые она готова выпустить в блоке, еще раз их проверяет и выпускает блок.

Выпуская блок нода связывает предыдущий блок, который хранится в её базе данных, и новый блок, включая содержащиеся в нем транзакций. Для этого нода указывает в теле нового выпускаемого блока подписи предыдущего блока. Таким образом подпись нового блока вычисляется из данных, содержащих все транзакции текущего блока и подписи предыдущего блока.

Если злоумышленник попытается удалить или модифицировать данные любой транзакции, то подпись блока, в который она входит, изменится. При синхронизации нод блок будет разослан другим участникам сети, не пройдет проверку и будет отвергнут, как некорректный.

Смотрите также

Архитектура

Подключение и удаление нод

1.18 Токены блокчейн-платформы Waves Enterprise

При использовании платформы Waves Enterprise *с подключением к сети Waves Enterprise Mainnet* применяется системный токен WEST:

1. За каждую транзакцию в блокчейн-сети Waves Enterprise Mainnet взимается *комиссия в WEST*.
2. Майнеры и валидаторы смарт-контрактов получают *вознаграждение в WEST* за создание блока и транзакцию исполнения смарт контракта соответственно.

Помимо системного токена вы можете создать и использовать другие токены – так называемые нативные токены.

В отличие от блокчейн платформ, где для создания нового токена необходимо публиковать смарт-контакт стандарта *ERC-20*, блокчейн-платформа Waves Enterprise предоставляет нативную возможность выпуска токенов при помощи *транзакции выпуска токена*.

После того как транзакция выпуска токена принята сетью, выпущенный токен можно *перевести* другому участнику сети или в рамках одной транзакции выполнить *массовый трансфер* нескольким участникам сети.

Кроме того, нативные токены можно *довыпустить* после создания, если при их выпуске параметру `reissuable` было присвоено значение `true`, а также *сжечь*, что невозможно сделать с системным токеном WEST.

Нативный токен можно *спонсировать*, то есть обеспечить системным токеном. Это позволяет платить комиссии за транзакции в сети в нативных токенах, например, в маркетинговых целях для привлечения новых пользователей.

Управлять токенами могут не только пользователи, но и *смарт-контракты*.

Смотрите также

Описание транзакций

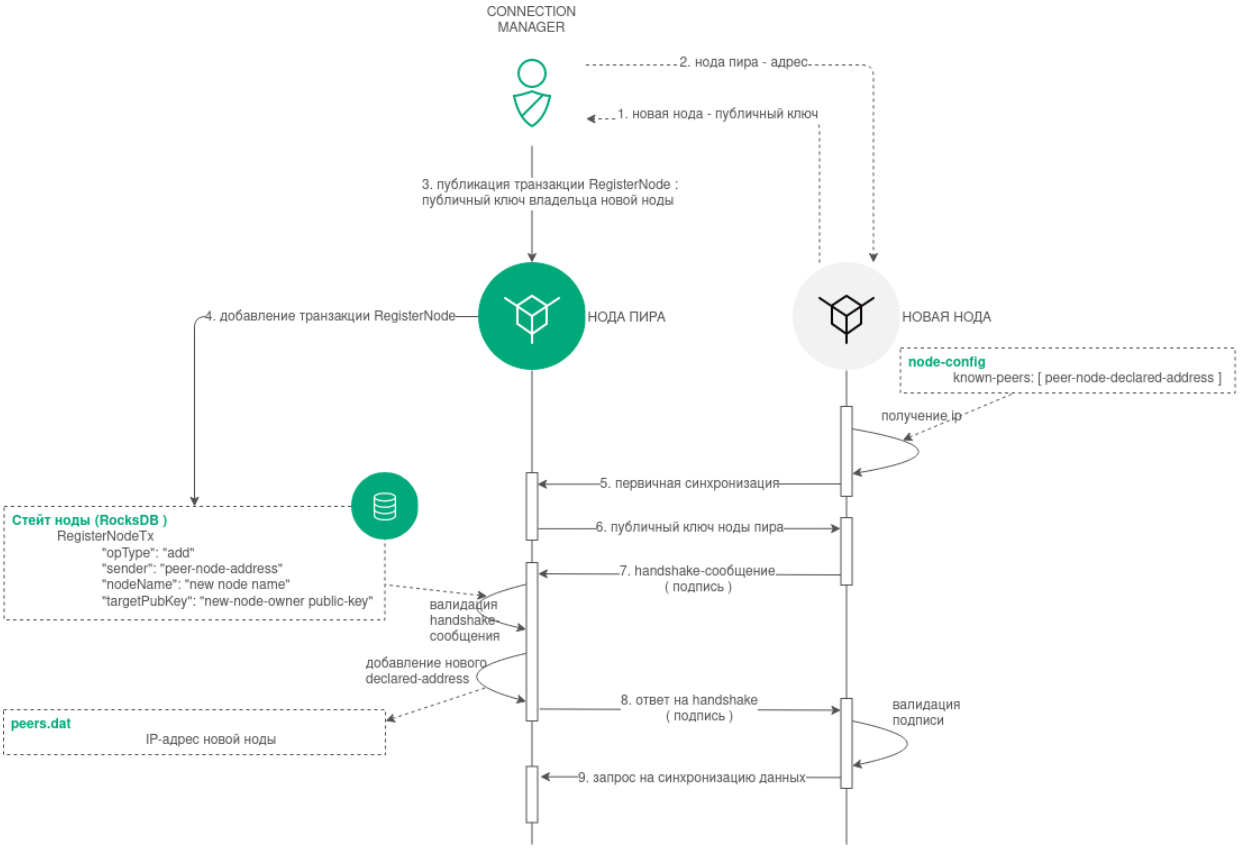
1.19 Подключение новой ноды к сети

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


CONNECTION MANAGER
2. нода пира - адрес
1. новая нода - публичный ключ
3. публикация транзакции RegisterNode : публичный ключ владельца новой ноды
4. добавление транзакции RegisterNode
НОДА ПИРА
НОВАЯ НОДА
node-config
known-peers: [peer-node-declared-address]
получение ip
5. первичная синхронизация
6. публичный ключ ноды пира
7. handshake-сообщение (подпись)
Стейт ноды (RocksDB)
RegisterNodeTx
"opType": "add"
"sender": "peer-node-address"
"nodeName": "new node name"
"targetPubKey": "new-node-owner public-key"
валидация handshake сообщения
добавление нового declared-address
peers.dat
IP-адрес новой ноды
8. ответ на handshake (подпись)
валидация подписи
9. запрос на синхронизацию данных

своей подписью. Пиры удостоверяются, что нода, отправившая **PeerIdentityRequest**, была зарегистрирована в сети.

6. При успешной проверке, в ответ на **PeerIdentityRequest**, пиры отправляют новой ноде свои публичные ключи. Новая нода сохраняет эти публичные ключи в своем временном хранилище адресов для первичного установления соединения с пирами. 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. Пиры сравнивают публичный ключ из handshake-сообщения и ключ новой ноды из транзакции 111, отправленной нодой с ролью **connection-manager**. Если проверка успешна, пиры 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

1.20 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.

1.20.1 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.

1.20.2 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.

1.20.3 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.

1.20.4 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.

1.20.5 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
162	Валидация смарт-контрактов в блокчейне
173	Поддержка сбора инвентаризационной информации о микроблоках (версия 2)
180	Поддержка передачи больших файлов в подсистеме конфиденциальных данных
190	Поддержка PKI (версия 1)
1120	Поддержка *операций с токенами для смарт-контрактов*, поддержка PKI v1, прекращение поддержки смарт-контрактов на базе REST

Смотрите также

REST API: информация об активации новых функциональных возможностей платформы

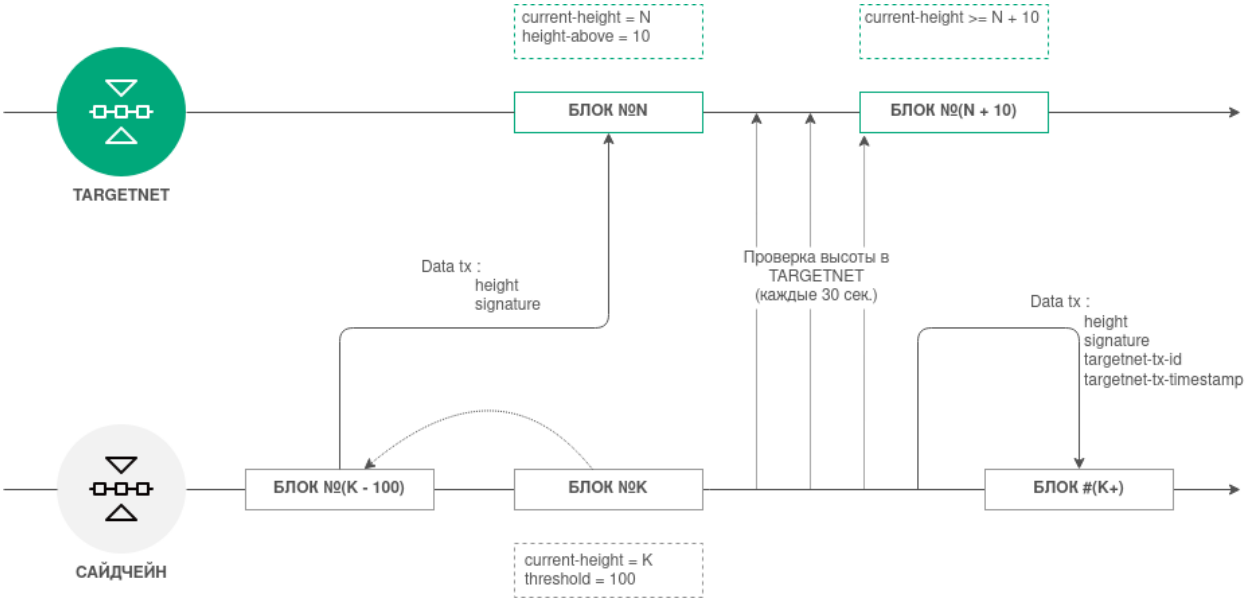
1.21 Анкоринг

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.

1.21.1 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.

1.21.2 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.

1.21.3 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	`failed to create a data transaction for targetnet`	Создание транзакции для отправки в Targetnet завершилась ошибкой
2	`failed to send the transaction to targetnet`	Публикация транзакции в Targetnet завершилась ошибкой (это может быть ошибка JSON-запроса)
3	`invalid http status of response from targetnet transaction broadcast: $responseStatus`	В результате публикации транзакции в Targetnet вернулся отличный от 200 код
4	`failed to parse http body of response from targetnet transaction broadcast`	В результате отправки транзакции в Targetnet вернулся нераспознаваемый JSON-запрос
5	`targetnet returned transaction with id='$targetnetTxId', but it differs from the transaction that was sent(id='$sentTxId')`	В результате отправки транзакции в Targetnet вернулся отличный от первой транзакции идентификатор
6	`targetnet didn't respond to the transaction info request`	Targetnet не ответил на запрос об информации о транзакции
7	`failed to get current height in targetnet`	Не удалось получить текущую высоту в Targetnet
8	`anchoring transaction in targetnet disappeared after the height rose enough`	Анкоринг транзакция пропала из Targetnet после увеличения высоты на значение `height-above`
9	`failed to create sidechain anchoring transaction`	Не удалось опубликовать анкоринг транзакцию в приватном блокчейне
10	`anchored block in sidechain was changed while waiting for targetnet height rise, looks like a rollback has happened`	Пока ожидалось подтверждение транзакции в Targetnet, произошел откат приватного блокчейна, идентификатор анкоринг транзакции был изменен

Смотрите также

Тонкая настройка платформы: настройка анкоринга

1.22 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*.

1.22.1 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 ноде, 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 снепшот (снимок данных). 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 нода, получившая снимок данных, рассылает своим пирам сообщения `SnapshotNotification`.

SnapshotApiRoute – контроллер REST API для работы со снимками данных.

1.22.2 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` инициируют запрос на получение снимка данных (`SnapshotRequest`). В случае истечения таймаута 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 в директорию `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 сообщение 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*.

Если вы подключаете к сети, запущенной со снепшота, ноду с пустым стейтом (новую ноду), процесс получения снимка данных производится автоматически: нода самостоятельно связывается с пирами для получения снимка данных и валидации собственного конфига. Описание процесса подключения новой ноды к сети см. в разделе *Подключение новой ноды к сети*.

1.22.3 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.

1.22.4 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

1.23 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.

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.

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 исполняется в контейнере Docker.

Доступ смарт-контракта к стейту ноды для обмена данными осуществляется посредством *gRPC* API-интерфейса.

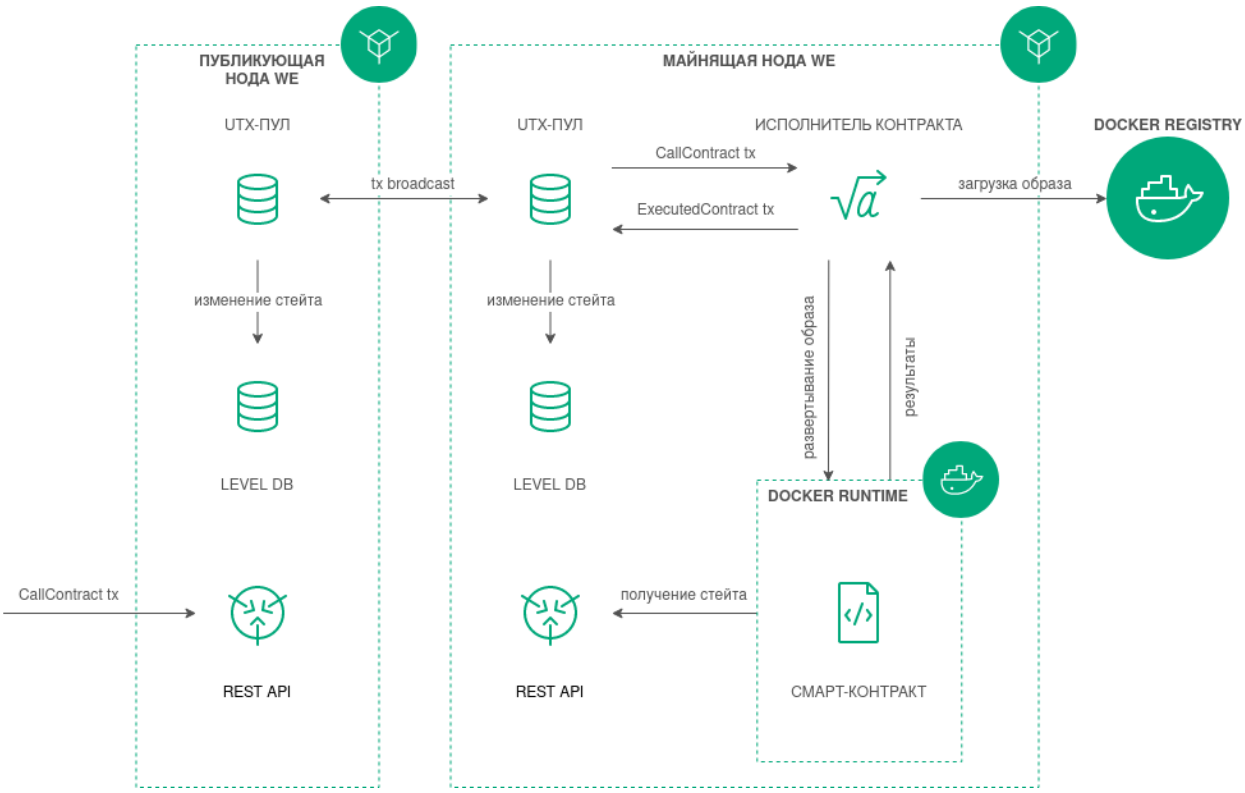
У каждого смарт-контракта есть собственный баланс, на котором могут храниться как системные токены WEST, так и любые другие *токены*. Подробнее об управлении токенами из смарт-контракта *см. ниже*.

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. 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.

В ноду внедрен механизм MVCC (Multiversion concurrency control) – *управление параллельным доступом к состоянию смарт-контрактов посредством многоверсионности*. Благодаря этому нода позволяет параллельно выполнять несколько транзакций любых смарт-контрактов. При этом гарантируется согласованность данных.

Ниже приведена общая схема работы смарт-контракта:



1.23.1 Управление токенами из смарт-контракта

Начиная с версии 1.12 после *активации функциональной возможности 1120* у смарт-контрактов блокчейн-платформы Waves Enterprise появляется собственный баланс, на котором могут храниться как системные токены WEST, так и любые другие *токены*. При этом для существовавших ранее смарт-контрактов баланс системных токенов WEST устанавливается равным нулю.

Также смарт-контрактам становятся доступны базовые функции работы с токенами:

- выпуск токенов,
- довыпуск токенов,
- сжигание токенов, находящихся на балансе смарт-контракта,
- перевод токенов с баланса смарт-контракта на баланс по адресу пользователя или пользователей.

Эти функции реализует метод *CommitExecutionSuccess*.

При помощи этой функциональности смарт-контракты имеют возможность изменять стейты ассетов и пользователей (их балансы). Пользователи также могут отправлять токены на баланс смарт-контракта.

1.23.2 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 данных 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 сети 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 репозиториев, 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.

1.23.3 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 стейте при помощи транзакции *105 ExecutedContract Transaction*.

1.23.4 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.

1.23.5 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**.

1.23.6 Валидация смарт-контрактов

Блокчейн-платформа поддерживает три варианта политик валидации смарт-контракта для обеспечения дополнительного контроля его целостности. Эта возможность доступна при выполнении следующих условий:

- в сети активирована *функциональная возможность 162*;
- в сети присутствует хотя бы один участник с активной ролью **contract_validator**;
- для загрузки и обновления смарт-контрактов используются транзакции *103* и *107* версии **4**.

Политика валидации настраивается при помощи строкового поля `validationPolicy.type` соответствующей транзакции.

Доступные политики валидации:

- `any` – сохраняется действующая в сети общая политика валидации: для майнинга обновляемого смарт-контракта майнер подписывает соответствующую транзакцию *105*. Также этот параметр устанавливается, если в сети нет ни одного зарегистрированного валидатора.
- `majority` – транзакция считается валидной, если она подтверждена большинством валидаторов: **2/3** от общего числа зарегистрированных адресов с ролью **contract_validator**.
- `majorityWithOneOf(List[Address])` – транзакция считается валидной, если собрано большинство валидаторов, среди которых присутствует хотя бы один из адресов, включенных в список параметра. Адреса, включаемые в список, должны иметь действующую роль **contract_validator**.

Предупреждение: При выборе политики валидации `majorityWithOneOf(List[Address])`, список адресов должен содержать хотя бы один адрес, передача пустого списка запрещена.

1.23.7 Параллельное исполнение контрактов

На платформе Waves Enterprise можно запускать несколько смарт-контрактов одновременно. Для этого на ноде реализован механизм MVCC (Multiversion concurrency control) – управление параллельным доступом посредством многоверсионности. Механизм позволяет параллельно выполнять несколько транзакций контейнеризированных смарт-контрактов и сохранять согласованность данных.

Все транзакции делятся на две группы:

1. non-executable транзакции – *атомарные контейнеры* и все *классические транзакции*: transfer transaction, data transaction и т. п.;
2. executable транзакции – транзакции всех контейнеризированных смарт-контрактов.

Транзакции первой группы всегда выполняются последовательно (уровень параллелизма равен единице). Для второй группы транзакций параллелизм исполнения определяется значением параметра `node.docker-engine.contracts-parallelism` в *конфигурации ноды*:

```
node.docker-engine.contracts-parallelism = 8
```

По умолчанию используется значение 8. Таким образом все смарт-контракты выполняются параллельно, независимо от Docker-образа.

Примечание: Между двумя группами транзакций присутствует конкуренция: если в UTX-пуле накапливаются разнородные транзакции, то параллельность может снижаться. Такое поведение можно сгладить, увеличив размер pulling буфера, но полностью исключить нельзя.

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.

1.23.8 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.

Примечание: Начиная с версии 1.12 после *активации функциональной возможности 1120* прекращается поддержка смарт-контрактов на базе REST.

Подробнее:

Сервисы gRPC, используемые смарт-контрактом

Описанные в этом разделе контрактные gRPC сервисы предназначены для обмена данными между смарт-контрактом и нодой. Эти сервисы доступны только смарт-контрактам. Внешний пользователь не сможет вызвать контрактные сервисы и использовать их функции.

Общие принципы применения gRPC при разработке смарт-контрактов рассмотрены в статье *Пример смарт-контракта с использованием gRPC*.

Версии API смарт-контрактов

gRPC-методы, используемые смарт-контрактами, формируют API, заданное protobuf-файлами. Для четкого определения новых методов и внесения изменений в уже существующие предусмотрено версионирование API. Благодаря присвоенному номеру версии нода при исполнении смарт-контракта определяет соответствующий набор методов для использования.

Актуальная версия gRPC API для версии блокчейн-платформы содержится в файле **api_version.proto**. Смарт-контракты, которые требуют версию API выше, чем у майнящей ноды, игнорируются при майнинге.

Для создания и обновления смарт-контрактов предусмотрены поля `apiVersion` в транзакциях *103 CreateContract Transaction* и *107 UpdateContract Transaction* версии 4. Эти поля указывают майнящей ноде на версию API, используемую смарт-контрактом.

В таблице ниже приведены версии API, соответствующие версиям блокчейн-платформы:

Версия API	Версия платформы
1.0	1.6.0 и старше
1.1	1.6.2
1.4	1.7.0

Protobuf-файлы методов

Смарт-контрактам, использующим gRPC для обмена данными с нодой, доступны сервисы, названия protobuf-файлов которых начинаются с `contract`:

protobuf	Методы
contract_address_servi	GetAddresses GetAddressData GetAssetBalance
contract_block_service	GetBlockHeader
contract_contract_serv	Connect CommitExecutionSuccess CommitExecutionError GetContractKeys GetContractKey GetContractBalances CalculateAssetId
contract_permission_se	GetPermissions GetPermissionsForAddresses
contract_privacy_servic	GetPolicyRecipientss GetPolicyOwners
contract_transaction_s	TransactionExists TransactionInfo
contract_util_service.pr	GetNodeTime
contract_pki_service.pr	Verify

Некоторые методы доступны только в корпоративной версии платформы, и не могут быть использованы в *opensource* версии платформы:

- *Verify*

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. Метод возвращает 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.

Метод возвращает массив `DataEntry`, содержащий записанные данные адреса.

GetAssetBalance – метод для получения текущего баланса определенного ассета для определенного пользователя. В запросе метода вводятся следующие параметры:

- `address` – адрес, баланс которого необходимо вывести;
- `assetId` – идентификатор ассета. Для WEST параметр остается пустым.

contract_block_service.proto

Набор методов, позволяющих контрактам запрашивать у ноды информацию о блоке.

GetBlockHeader – метод для получения заголовка блока по подписи (идентификатору блока) или по высоте.

В запросе метода вводится один из следующих параметров:

- `signature` – подпись запрашиваемого блока в виде строки с кодировкой Base58;
- `height` — высота запрашиваемого блока.

Метод возвращает следующую информацию о заголовке блока:

- `version` — версия блока;
- `height` — высота блока;
- `block_signature` — подпись блока (она же идентификатор) в виде строки с кодировкой Base58;
- `reference` — подпись предыдущего блока, на который ссылается текущий, в виде строки с кодировкой Base58;
- `miner_address` — адрес майнера в виде строки с кодировкой Base58;
- `tx_count` — количество транзакций в блоке;
- `timestamp` — время блока.

Если блок не найден, метод возвращает ошибку `BlockDoesNotExist`.

contract_contract_service.proto

Набор методов, предназначенный для работы со смарт-контрактами: служебные методы для исполнения контракта, а также методы для чтения информации о состоянии смарт-контрактов и для действий с ассетами.

Connect – метод для подключения смарт-контракта к ноде.

В запросе метода указываются следующие параметры:

- `connection_id` – идентификатор соединения смарт-контракта (см. раздел *Авторизация смарт-контракта с gRPC*);
- `async_factor` – максимальное количество одновременно исполняемых транзакций по смарт-контракту (см. раздел *Параллельное исполнения смарт-контрактов*).

Метод возвращает следующую информацию о транзакции и блоке:

- `transaction` — транзакция вызова контракта;
- `auth_token` — авторизационный токен;
- `current_block_info` — информация о текущем блоке:
  - `height` — текущая высота;
  - `timestamp` — время блока;
  - `miner_address` — адрес майнера в формате строки в кодировке Base58;
  - `reference` — подпись (идентификатор) предыдущего блока, на который ссылается текущий жидкий блок; в формате строки Base58.

CommitExecutionSuccess – метод для отправки результатов успешного исполнения смарт-контракта на ноду. С помощью этого метода *смарт-контракт может отправлять последовательность операций над ассетами*.

В запросе метода указываются следующие данные:

- `tx_id` — идентификатор транзакции вызова контракта, на которую смарт-контракт даёт результат;
- `results` — массив `key-value` значений, которые смарт-контракт в качестве результата исполнения запишет в свой стейт. Если возвращается ключ, который уже присутствует в стейте, то его значение будет перезаписано;
- `asset_operations` — массив действий смарт-контракта с доступными ему ассетами, в том числе выпуск нового ассета, перевыпуск ассета, сжигание ассета или перевод доступного контракту ассета другому пользователю (`issue`, `reissue`, `burn`, `transfer`).

Ответ метода не предусмотрен.

CommitExecutionError – метод для отправки ошибки исполнения смарт-контракта на ноду.

GetContractKeys – метод для запроса значений из состояния смарт-контракта по переданному фильтру ключей.

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.

Метод возвращает массив `DataEntry`, содержащий запрашиваемые ключи со значениями из текущего состояния смарт-контракта.

GetContractKey – метод для получения значения определённого ключа из состояния смарт-контракта.

В запросе метода указываются следующие данные:

- `contract_id` – идентификатор смарт-контракта;
- `key` – запрашиваемый ключ.

Метод возвращает `DataEntry` из текущего состояния смарт-контракта, который соответствует переданному ключу.

GetContractBalances – метод для получения текущего баланса(ов) (*системный токен и другие токены*) смарт-контракта.

В запросе передаётся список идентификаторов ассета (`assets_ids`); для получения баланса системного токена WEST следует передать в списке пустую строку.

Метод возвращает список с балансами для каждого из запрошенных ассетов.

CalculateAssetId – метод для вычисления assetId при выпуске нового токена смарт-контрактом по переданному параметру:

- `nonce` — одноразовый код типа byte, не равный 0, диапазон допустимых значений: от -128 до 127. В рамках одного вызова контракта не может быть выпущено несколько ассетов с одинаковым `nonce`.

contract_permission_service.proto

Набор методов, предназначенный для получения информации о ролях участников.

GetPermissions – метод для получения списка всех ролей участника, чей адрес указан, действительных на указанный момент времени. В запросе передаются следующие данные:

- `address` – адрес участника;
- `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 участников, чьи адреса указаны, действительных на указанный момент времени. В запросе передаются следующие данные:

- `addresses` – массив строк с адресами участников;
- `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

В protobuf-файле **contract_pki_service.proto** описан контрактный метод **Verify**, предназначенный для проверки отсоединенной электронной подписи для передаваемых данных в сетях, работающих с использованием ГОСТ-криптографии.

Важно: Метод **Verify** недоступен при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. В тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`) метод можно использовать.

Примечание: gRPC метод **Verify** недоступен в *opensource* версии платформы.

Типы данных полей для запросов и ответов указаны в protobuf-файле.

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 **Verify** содержит поле `status` с булевым типом данных:

- `true` – подпись действительна,
- `false` – подпись скомпрометирована.

Проверка УКЭП

Метод **Verify** предоставляет 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 произвольное 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
```

contract_privacy_service.proto

Набор методов, предназначенный для получения информации о группах для обмена конфиденциальными данными и работы с конфиденциальными данными.

Важно: Описанные ниже методы для получения информации о группах для обмена конфиденциальными данными и работы с конфиденциальными данными недоступны при использовании PKI, то есть когда в конфигурационном файле ноды *параметру node.crypto.pki.mode* присвоено значение `ON`. Методы можно использовать в тестовом режиме PKI (node.crypto.pki.mode = `TEST`) или при отключенном PKI (node.crypto.pki.mode = `OFF`).

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.

contract_transaction_service.proto

Набор методов, предназначенный для получения информации о транзакциях, отправленных в блокчейн. Аналогичные gRPC методы, доступные внешнему пользователю, описаны в разделе *gRPC: работа с транзакциями*.

В отличии от методов *TransactionExists и TransactionInfo*, доступных для интеграции извне, контрактные методы возвращают информацию не только о транзакциях, которые уже записаны в блок, но и о транзакциях, которые только готовятся к упаковке в блок.

TransactionExists – метод для проверки существования транзакции с указанным идентификатором. Метод возвращает `true`, если транзакция с указанным идентификатором существует, `false` – если не существует.

TransactionInfo – метод для получения данных о транзакции с указанным идентификатором: название транзакции, версия транзакции, высота блокчейна, на которой была произведена данная транзакция, другие данные о транзакции в зависимости от типа этой транзакции.

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

Управление токенами из смарт-контракта

Методы REST API, доступные смарт-контрактам

Разработка и применение смарт-контрактов

Общая настройка платформы: настройка исполнения смарт-контрактов

Методы REST API, доступные смарт-контрактам

Примечание: Начиная с версии 1.12 после *активации функциональной возможности 1120* прекращается поддержка смарт-контрактов на базе REST.

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

1.24 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.

1.24.1 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*).

1.24.2 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 не 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:

Описание транзакций

Блокчейн-платформа Waves Enterprise поддерживает 28 типов транзакций. Для каждой из них предусмотрен свой набор данных, отправляемых в блокчейн.

Запросы и ответы, передаваемые в рамках каждой транзакции по *REST API*-интерфейсу ноды, имеют формат JSON. Формат запросов и ответов, передающихся по *gRPC*-интерфейсу ноды, определяется соответствующими proto-схемами. JSON и protobuf-представления запросов и ответов каждой транзакции приведены ниже.

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. Транзакция не версионируется.

Структура данных транзакции

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**1**)
id	Byte	ID транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
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.

Структуры данных транзакции

Подписание:

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 – *опциональное поле*
decimals	Byte	Количество разрядов после запятой у используемого токена (WEST - **8**)
reissuable	Boolean	Возможность довыпуска токенов
fee	Long	*Комиссия за транзакцию в WE Mainnet*

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicK	PublicKeyAccc	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
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 – *опциональное поле*
height	Int	Высота выполнения транзакции

JSON-представление:

Version 2

Подписание:

```
{
   "type": 3,
   "version":2,
   "name": "Test Asset 1",
   "quantity": 100000000000,
   "description": "Some description",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
   "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
   "password": "",
   "decimals": 8,
   "reissuable": true,
   "fee": 100000000
}
```

Публикация:

```
{
    "type": 3,
    "id": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 100000000,
    "timestamp": 1549378509516,
    "proofs": [
↪"NqZGcbcQ82FZrPh6aCEjuo9nNnkPTvyhrNq329YWydaYcZTywXUwDxFAknTMEGuFrEndCjXBtrueLWaqbJhpeiG
↪" ],
    "version": 2,
    "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "name": "Token Name",
    "quantity": 10000,
    "reissuable": true,
    "decimals": 2,
    "description": "SmarToken",
    "chainId": 84,
    "script": "base64:AQa3b8tH",
    "height": 60719
},
```

4. Transfer Transaction

Транзакция для перевода *токенов* с одного адреса на другой.

Структуры данных транзакции

Подписание:

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 - *опциональное поле*
recipient	ByteStr	Адрес получателя токенов
amount	Long	Сумма токенов
fee	Long	*Комиссия за транзакцию в WE Mainnet*

Публикация:

Поле	Тип данных	Описание
senderPublic	PublicKeyAcc	Открытый ключ отправителя транзакции
amount	Long	Сумма токенов
fee	Long	*Комиссия за транзакцию в WE Mainnet*
type	Byte	Номер транзакции (**4**)
version	Byte	Версия транзакции
attachment	Byte	Комментарий к транзакции (**в формате base58**) - *опциональное поле*
sender	ByteStr	Адрес отправителя транзакции
feeAssetId	Byte	ID токена комиссии – *опциональное поле*
proofs	List(ByteStr)	Массив подтверждений транзакции (**в формате base58**)
assetId	Byte	ID токена для перевода – *опциональное поле*
recipient	ByteStr	Адрес получателя токенов
id	Byte	ID транзакции
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*

JSON-представление:

Version 2

Подписание:

```
{
  "type": 4,
  "version": 2,
  "sender": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "password": "",
  "recipient": "3M6dRZXaJY9oMA3fJKhMALyYKt13D1aimZX",
  "amount": 40000000000,
  "fee": 100000
}
```

Публикация:

```
{
    "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
    "amount": 200000000,
    "fee": 100000,
    "type": 4,
    "version": 2,
    "attachment": "3uaRTtZ3taQtRSmquqeC1DniK3Dv",
    "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
    "feeAssetId": null,
    "proofs": [

↪"2hRxJ2876CdJ498UCpErNfDSYdt2mTK4XUnmZNgZiq63RupJs5WTrAqR46c4rLQdq4toBZk2tSYCeAQWEQyi72U6
```

(continues on next page)

(продолжение с предыдущей страницы)


```
↪"
    ],
    "assetId": null,
    "recipient": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
    "id": "757aQzJiQZRfVRuJNnP3L1d369H2oTjUEazwtYxGngCd",
    "timestamp": 1558952680800
}
```


Version 3

Подписание:


```
{
  "type": 4,
  "version": 3,
  "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "password": "",
  "recipient": "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
  "amount": 40000000000,
  "fee": 10000000
}
```


Публикация:


```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "amount" : 10,
  "fee" : 10000000,
  "type" : 4,
  "version" : 3,
  "atomicBadge" : {
    "trustedSender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  },
  "attachment" : "",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [
↪"2vbAJmwzQw2FCtozcewxJVfxoHxf97BTNdGuaeSATV4vEHZ3XYA4Z7nXGsSnf18aesnAWTKWCfzwM5yGpWEyGM7f
↪" ],
  "assetId" : null,
  "recipient" : "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
  "id" : "2wCEMREFbgk318hFFaNGsgFzyjZHuCrtwSnpK35qhiw4",
  "timestamp" : 1619186861204,
  "height" : 861644
}
```

5. Reissue Transaction

Транзакция для довыпуска *токенов*.

Структуры данных транзакции

Подписание:

Таблица 2: :header: «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 – *опциональное поле*
assetId	Byte	ID довыпускаемого токена – *опциональное поле*
reissuable	Boolean	Возможность довыпуска токенов
fee	Long	*Комиссия за транзакцию в WE Mainnet*

Публикация:

Поле	Тип данных	Описание
senderPublic	PublicKeyAcc	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 – *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
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) - *опциональное поле*
height	Int	Высота выполнения транзакции

JSON-представление:

Version 2

Подписание:

```
{
    "type": 5,
    "version":2,
    "quantity": 556105,
    "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "password": "",
    "assetId": "6UAMZA6RshxyPvt9W7aoWiUiB6N73yLQMMfiRQYXdWZh",
    "reissuable": true,
    "fee": 100000000
}
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "quantity" : 556105,
  "fee" : 100000000,
  "type" : 5,
  "version" : 2,
  "reissuable" : true,
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "chainId" : 86,
  "proofs" : [
↪"5ahD78wciu8YTsLoxo1XRghJWAGG7At7ePiBWTNzdkvX7cViRCKRLjjjPTGCoAH2mdGQK9i1JiY1wh18eh4h7pGy
↪" ],
  "assetId" : "6UAMZA6RshxyPvt9W7aoWiUiB6N73yLQMMfiRQYXdWZh",
  "id" : "8T9jJUusN5KBexxDUX1XBjoDydXGP34zWH7Qvp5mnmES",
  "timestamp" : 1619187184206,
  "height" : 861645
}
```

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.

Структуры данных транзакции

Подписание:

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 – *опциональное поле*
assetId	Byte	ID сжигаемого токена – *опциональное поле*
quantity	Long	Количество токенов для сжигания
fee	Long	*Комиссия за транзакцию в WE Mainnet*
attachmen	Byte	Комментарий к транзакции (**в формате base58**) - *опциональное поле*

Публикация:

Поле	Тип данных	Описание
senderPubli	PublicKeyAc	Открытый ключ отправителя транзакции
amount	Long	Количество токенов для сжигания
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды – *опциональное поле*
proofs	List(ByteStr)	Массив подтверждений транзакции (**в формате base58**)
assetId	Byte	ID сжигаемого токена – *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
id	Byte	ID транзакции
type	Byte	Номер транзакции (**6**)
version	Byte	Версия транзакции
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
height	Int	Высота выполнения транзакции

JSON-представление:

Version 2

Подписание:

```
{
    "type": 6,
    "version": 2,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "quantity": 1000,
    "fee": 100000,
    "attachment": "string"
}
```

Публикация:

```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "amount": 1000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "chainId": 84,
    "proofs": [
↪"kzTwsNXjJkzk6dpFFZZXyeimYo6iLTVbCnCXBD4xBtyrNjysPqZfGKk9NdJUTP3xeAPhtEgU9hsdwzRVo1hKMgS
↪" ],
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
    "fee": 100000,
    "id": "3yd2HZq7sgun7GakisLH88UeKcpYMUEL4sy57aprAN5E",
    "type": 6,
    "version": 2,
    "timestamp": 1551448489758,
    "height": 1190
}
```

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.

Структуры данных транзакции

Подписание:

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 – *опциональное поле*
recipient	ByteStr	Адрес получателя токенов
amount	Long	Количество токенов для передачи в аренду
fee	Long	*Комиссия за транзакцию в WE Mainnet*

Публикация:

Поле	Тип данных	Описание
senderPublic	PublicKeyAcc	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	*Комиссия за транзакцию в WE Mainnet*
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) – *опциональное поле*

JSON-представление:

Version 2

Подписание:

```
{
    "type": 8,
    "version": 2,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
    "amount": 1000,
    "fee": 100000
}
```

Публикация:

```
{
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "amount": 1000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "proofs": [
↪"5jvmWKmU89HnxXFXNAd9X41zmiB5fSGoXMirsaJ9tNeyiCAJmjm7MR48g789VucckQw2UExaVXfhsdEBuUrchvrq
↪" ],
    "fee": 100000,
    "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
    "id": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
    "type": 8,
    "version": 2,
    "timestamp": 1551449299545,
    "height": 1190
}
```

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.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**9**)
version	Byte	Версия транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды – *опциональное поле*
txId	Byte	ID транзакции аренды токенов

Публикация:

Поле	Тип данных	Описание
senderPubli	PublicKeyAc	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	*Комиссия за транзакцию в WE Mainnet*
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) – *опциональное поле*
height	Int	Высота выполнения транзакции

JSON-представление:

Version 2

Подписание:

```
{
    "type": 9,
    "version": 2,
    "fee": 100000,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "txId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp"
}
```

Публикация:

```
{
   "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "leaseId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "chainId": 84,
    "proofs": [
↪"2Gns72hraH5yay3eiWeyHQEA1wTqiiAztaLjHinEYX91FEv62HFW38Hq89GnsEJFHUvo9KHYtBBrb8hgTA9wN7DM
↪" ],
    "fee": 100000,
    "id": "9vhxB2ZDQcqiumhQbCPnAoPBLuir727qgJhFeBNmPwmu",
    "type": 9,
    "version": 2,
    "timestamp": 1551449835205,
    "height": 1190
}
```

10. CreateAlias Transaction

Создание псевдонима для адреса отправителя. Псевдоним может использоваться для проведения транзакций в качестве идентификатора получателя.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**10**)
version	Byte	Версия транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды – *опциональное поле*
alias	Byte	Псевдоним

Публикация:

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
senderPublicK	PublicKeyAccc	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
proofs	List(ByteStr)	Массив подтверждений транзакции (**в формате base58**)
version	Byte	Версия транзакции
alias	Byte	Псевдоним
height	Byte	Высота выполнения транзакции

JSON-представление:

Version 2

Подписание:

```
{
    "type": 10,
    "version": 2,
    "fee": 100000000,
    "sender": "3NwTvbW7TMckBc785XjtGTUfHmcesaWBe1A",
    "password": "",
    "alias": "1@k1_kv29"
}
```

Публикация:

```
{
  "senderPublicKey" : "C4eRfdUFaZMRkfUp91bYr7uMgdBRnUfAxuAjetxmK7KY",
  "sender" : "3NwTvbW7TMckBc785XjtGTUfHmcesaWBe1A",
  "proofs" : [
↪"3fhJztBNnTDjppmqgi4GugAYo1aS1mzZhVhPdnNsqYqCEyLLHfzgb75psRPntHD4uBZgk8jByFP9mwwx2Ezsdg59
↪" ],
  "fee" : 100000000,
  "alias" : "1@k1_kv29",
  "id" : "AavgVzV7avPMpERro6YqikwFESAgG2wViprtPJUtXP6F",
  "type" : 10,
  "version" : 2,
  "timestamp" : 1608737444468,
  "height" : 595942
}
```

11. MassTransfer Transaction

Перевод *токенов* нескольким получателям (от 1 до 100 адресов). Комиссия за транзакцию зависит от количества задействованных адресов.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**11**)
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды – *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
version	Byte	Версия транзакции
transfers	List	Список получателей с полями `recipient` и `amount` через запятую
recipient	ByteStr	Адрес получателя токенов
amount	Long	Количество токенов для передачи адресу

Публикация:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccou	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
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**) – *опциональное поле*
sender	ByteStr	Адрес отправителя транзакции
proofs	List(ByteStr)	Массив подтверждений транзакции (**в формате base58**)
assetId	Byte	ID токена для перевода – *опциональное поле*
id	Byte	ID транзакции перевода токенов
transfers	List	Список получателей с полями `recipient` и `amount` через запятую
transfers.recipier	ByteStr	Адрес получателя токенов
transfers.amoun	Long	Количество токенов для передачи адресу
height	Byte	Высота выполнения транзакции

Пример заполнения поля `transfers`:

```
"transfers":
[
    { "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB", "amount": 100000 },
    { "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc", "amount": 100000 }
]
```

JSON-представление:

Version 2

Подписание:

```
{
    "type": 11,
    "sender": "3NydXoTq3UgUW5rxsNwEMs1iwbbvVEwxoHU",
    "password": "",
    "fee": 30000000,
    "version": 2,
    "transfers":
    [
        { "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB", "amount": 100000␣
↪},
        { "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc", "amount": 100000␣
↪}
    ]
}
```

Публикация:

```
{
  "senderPublicKey" : "AMhAY8RMy5QsPqj58xeMY3fJxTZKx71QztsjDzqWprHo",
  "fee" : 30000000,
  "type" : 11,
  "transferCount" : 4,
  "version" : 2,
  "totalAmount" : 400000000,
  "attachment" : "",
  "sender" : "3NydXoTq3UgUW5rxsNwEMs1iwbbvVEwxoHU",
  "feeAssetId" : "8bec1mhqTiveMeRTHgYr6az12XdqBBtpeV3ZpXMRHfSB",
  "proofs" : [
↪"21hhAMmwze6nLLQ9K6AoU6scek9Sk5KabR4VggGfdTVFHonfMGwVTse6qL2f8zR8DRm7RckMaikiYRt5XxWEKWcA
↪" ],
  "assetId" : "8bec1mhqTiveMeRTHgYr6az12XdqBBtpeV3ZpXMRHfSB",
  "transfers" : [ {
    "recipient" : "3NqEjAkFVzem9CGa3bEPhakQc1Sm2G8gAFU",
    "amount" : 100000000
  }, {
    "recipient" : "3NzkzibVRkKUzaRzjUxndpTPvoBzQ3iLng3",
    "amount" : 100000000
  }, {
    "recipient" : "3Nnx8cX3UiyfQeC3YQKVRqVr2ewSxrvaDyB",
    "amount" : 100000000
  }, {
    "recipient" : "3NzC4Ex91VBQKfJHPiGhuPEomLg48NMi2ZF",
    "amount" : 100000000
  } ],
  "id" : "EvnxFxdYhYxHgQSMhkyLaqgyUDZdnBknfAWEXyqEHt97",
  "timestamp" : 1627643861044,
  "height" : 1076874
}
```

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:

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**12**)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды – *опциональное поле*
senderPubli	PublicKeyAc	Открытый ключ отправителя транзакции
author	Byte	Адрес автора вносимых данных
data	List	Список данных, в который вносятся поля `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
fee	Long	*Комиссия за транзакцию в WE Mainnet*

Публикация:

Поле	Тип данных	Описание
senderPubli	PublicKeyAc	Открытый ключ отправителя транзакции
senderPubli	PublicKeyAc	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	*Комиссия за транзакцию в WE Mainnet*
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) – *опциональное поле*

Пример заполнения поля `data`:

```
"data": [
      {
       "key": "objectId",
       "type": "string",
       "value": "obj:123:1234"
      }, {...}
     ]
```

JSON-представление:

Version 2

Подписание:

```
{
    "type": 12,
    "version": 2,
    "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "password": "",
    "senderPublicKey": "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
    "author": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "data": [
      ...
    ],
    "fee": 150000000
}
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "data" : [
    ...
  ],
  "author" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "fee" : 150000000,
  "type" : 12,
  "version" : 2,
  "authorPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [
↪"4wFNmn32NZqGwP4D4aAxCMyigGEVZLWftqi919pHAK7mCj3sFw7Ekf76g2rr51PZuk5sLwzjkKiZArQvWY8uEGqk
↪" ],
  "id" : "GcDy84oTFf5NQzDtixkfUqiFNZwMaN2vfXqxsbGxumfo",
  "timestamp" : 1619187166499,
  "height" : 861644
}
```


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.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**13**)
version	Byte	Версия транзакции
sender	ByteStr	Адрес отправителя транзакции
passwo	String	Пароль от ключевой пары в keystore ноды – *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
name	Array[By	Имя скрипта
script	Array[By	Скомпилированный скрипт в формате **base64**. Если вы оставите это поле пустым (*null*) – скрипт будет отвязан от аккаунта

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**13**)
id	Byte	ID транзакции установки скрипта
sender	ByteStr	Адрес отправителя транзакции
senderPublicK	PublicKeyAccc	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
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** – *опциональное поле*
name	Array[Byte]	Имя скрипта
description	Byte	Комментарий к транзакции (**в формате base58**) – *опциональное поле*
height	Byte	Высота выполнения транзакции

JSON-представление:

Version 1

Подписание:

```
{
    "type": 13,
    "version": 1,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "fee": 1000000,
    "name": "faucet",
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ=="
}
```

Публикация:

```
{
    "type": 13,
    "id": "HPDypnQJHJskN8kwszF8rck3E5tQiuiM1fEN42w6PLmt",
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
    "fee": 1000000,
    "timestamp": 1545986757233,
    "proofs": [
↪"2QiGYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzojmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFgG
↪" ],
    "chainId": 84,
    "version": 1,
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
    "name": "faucet",
    "description": "",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "height": 3805
}
```

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.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
assetId	Byte	ID спонсорского ассета (токена) – *опциональное поле*
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 – *опциональное поле*
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**14**)
id	Byte	ID транзакции спонсирования
sender	ByteStr	Адрес отправителя транзакции
senderPublicK	PublicKeyAccc	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
assetId	Byte	ID спонсорского ассета (токена) – *опциональное поле*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
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

JSON-представление:

Version 1

Подписание:

```
{
    "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
    "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
    "fee": 100000000,
    "isEnabled": false,
    "type": 14,
    "password": "1234",
    "version": 1
}
```

Публикация:

```
{
    "type": 14,
    "id": "Ht6kpnQJHJskN8kwszF8rck3E5tQiuiM1fEN42wGfdk7",
    "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
    "senderPublicKey": "Gt55fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUophy89",
    "fee": 100000000,
    "assetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
    "timestamp": 1545986757233,
    "proofs": [
↪"5TfgYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzojmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFh7
↪" ],
    "chainId": 84,
    "version": 1,
    "isEnabled": false,
    "height": 3865
}
```

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.

Структуры данных транзакции

Подписание:

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 – *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
script	Array[Byte]	Скомпилированный скрипт в формате **base64** – *опциональное поле*
assetId	Byte	ID спонсорского ассета (токена) – *опциональное поле*

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**15**)
id	Byte	ID транзакции скрипта ассета
sender	ByteStr	Адрес отправителя транзакции
senderPub	PublicKeyA	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
proofs	List(ByteSt	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) – *опциональное поле*
script	Array[Byte]	Скомпилированный скрипт в формате **base64**. Если вы оставите это поле пустым (*null*) – скрипт будет отвязан от аккаунта
height	Byte	Высота выполнения транзакции

JSON-представление:

Version 1

Подписание:

```
{
    "type": 15,
    "version": 1,
    "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
    "password": "",
    "fee": 100000000,
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
    "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg"
}
```

Публикация:

```
{
    "type": 15,
    "id": "CQpEM9AEDvgxKfgWLH2HxE82iAzpXrtqsDDcgZGPAF9J",
    "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
    "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "fee": 100000000,
    "timestamp": 1549448710502,
    "proofs": [
↪"64eodpuXQjaKQQ4GJBaBrqiBtmkjSxseKC97gn6EwB5kZtMr18mAUHPRkZaHJeJxaDyLzGEZKqhYoUknWfNhXnkf
↪" ],
    "version": 1,
    "chainId": 84,
    "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
    "script": "base64:AQQAAAAHJG1hdGNoMAUAAAACdHgG+RXSzQ==",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    "height": 61895
}
```

101. GenesisPermission Transaction

Транзакция для назначения первого администратора сети, который раздает роли другим участникам.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**101**)
id	Byte	ID транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
signature	ByteStr	Подпись транзакции (**в формате base58**)
target	ByteStr	Адрес назначаемого первого администратора
role	String	Назначаемая роль (для администратора – `permissioner`)

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**101**)
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
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
- contract_validator
- banned

Описание ролей см. в статье *Роли участников*.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**102**)
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды – *опциональное поле*
senderPubli	PublicKeyAc	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
target	ByteStr	Адрес участника для назначения роли
opType	String	Тип операции: `add` – добавить роль; `remove` – отозвать роль
dueTimesta	Long	Временная метка срока действия роли в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
senderPubli	PublicKeyAc	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	*Комиссия за транзакцию в WE Mainnet*
opType	String	Тип операции: `add` – добавить роль; `remove` – отозвать роль
id	Byte	ID транзакции назначения или отмены роли
type	Byte	Номер транзакции (**102**)
dueTimesta	Long	Временная метка срока действия роли в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) – *опциональное поле*
target	ByteStr	Адрес назначаемого первого администратора

JSON-представление:

Version 1

Подписание:


```
{
   "type": 102,
   "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
   "password": "",
   "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
   "fee": 0,
   "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
   "opType": "add",
   "role": "contract_developer",
   "dueTimestamp": null,
   "version": 1
 }
```


Публикация:


```
{
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "role": "contract_developer",
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "proofs": [

↪"5ABJCRTKGo6jmDZCRWcLQc257CCeczmcjmtfJmbBE7TP3KsVkwvisH9kEkfYPckVCzEMKZTCd3LKAPcN8o4Git3j
↪"
  ],
  "fee": 0,
  "opType": "add",
  "id": "8zVUH7nsDCcpwyfxiq8DCTgqL7Q23FW1KWepB9EZcFG6",
  "type": 102,
  "dueTimestamp": null,
  "timestamp": 1559048837487,
  "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL"
}
```


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**.

Важно: Начиная с релиза 1.12 (после *активации фичи* 1120) создание *REST контрактов* невозможно. Рекомендуется использовать gRPC контракты вместо REST контрактов.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
fee	Long	*Комиссия за транзакцию в WE Mainnet*
image	Array[By	Имя Docker-образа смарт-контракта
imageHas	Array[By	Хэш Docker-образа смарт-контракта
contractN	Array[By	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 - *опциональное поле*
params	List[Data	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type` `value` и `key` через запятую - *опциональное поле*
params.ke	Byte	Ключ параметра
params.ty	Byte	Тип данных параметра. Возможные значения: `binary` `bool` `integer` `string`
params.va	Byte	Значение параметра
type	Byte	Номер транзакции (**103**)
version	Byte	Версия транзакции
apiVersion	Byte	Версия API для gRPC-методов смарт-контракта (см. *Сервисы gRPC используемые смарт-контрактом*).
validation	String	Тип политики валидации смарт-контрактов.

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**103**)
id	Byte	ID транзакции создания контракта
sender	ByteSt	Адрес отправителя транзакции
senderP	Publick	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestar	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
proofs	List(By	Массив подтверждений транзакции (**в формате base58**)
version	Byte	Версия транзакции
image	Array[E	Имя смарт-контракта (при загрузке из предустановленного репозитория) или его полный адрес (если репозиторий смарт-контракта не указан в конфигурационном файле ноды)
imageH	Array[E	Хэш Docker-образа смарт-контракта
contract	Array[E	Имя смарт-контракта
params	List[Da	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type` `value` и `key` через запятую - *опциональное поле*
params.	Byte	Ключ параметра
params.	Byte	Тип данных параметра. Возможные значения: `binary` `bool` `integer` `string`
params.	Byte	Значение параметра
height	Byte	Высота выполнения транзакции
apiVersi	Byte	Версия API для gRPC-методов смарт-контракта (см. *Сервисы gRPC используемые смарт-контрактом*).
validatic	String	Тип политики валидации смарт-контрактов.
paymen		Целое число. которое определяет количество передаваемых контракту ассетов; в поле `amount` младшие разряды соответствуют дробным частям количества передаваемого ассета. если его `decimals` не нулевой. Поле является опциональным.
paymen		Идентификатор передаваемого контракту ассета; для передачи системного токена WEST поле assetId должно быть пустым. Поле является опциональным.

JSON-представление:

Version 2

Подписание:

```
{
  "type": 103,
  "version": 2,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "password": "signing-key-password",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "contractName": "Your contract name",
  "imageHash":
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null
}
```

Публикация:

```
{
  "id": "4WVhw3QdiinpE5QXDG7QfqLiLanM7ewBw4ChX4qyGjs2",
  "type": 103,
  "version": 2,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "senderPublicKey": "YNpp7chAaudMqEtSZZPyN4GYLJ5ZTXdjCXrQdszzuRp",
  "contractName": "Your contract name",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "imageHash":
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "proofs": [

↪"4vqLnpJRFpcDgM5vgi78DpZnVfqztsARHNb7Hbmq3mQBjS3SRnzFAiYjRvPazEVMhBM9cE4Rcp6H5K29kk75Uxyh
↪"
  ]
}
```

Version 3

Подписание:

```
{
  "type": 103,
  "version": 3,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "password": "signing-key-password",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "contractName": "Your contract name",
  "imageHash":
```

(continues on next page)

(продолжение с предыдущей страницы)

```
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null
}
```

Публикация:

```
{
  "id": "4WVhw3QdiinpE5QXDG7QfqLiLanM7ewBw4ChX4qyGjs2",
  "type": 103,
  "version": 3,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "senderPublicKey": "YNpp7chAaudMqEtSZZPyN4GYLJ5ZTXdjCXrQdszzuRp",
  "contractName": "Your contract name",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "imageHash":
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null,
  "proofs": [

↪"4vqLnpJRFpcDgM5vgi78DpZnVfqztsARHNb7Hbmq3mQBjS3SRnzFAiYjRvPazEVMhBM9cE4Rcp6H5K29kk75Uxyh
↪"
  ]
}
```

Version 4

Подписание:

```
{
  "type": 103,
  "version": 4,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "password": "signing-key-password",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "contractName": "Your contract name",
  "imageHash":
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null,
  "validationPolicy": {
    "type": "majority"
  },
  "apiVersion": "1.0"
}
```

Публикация:

```
{
  "id": "4WVhw3QdiinpE5QXDG7QfqLiLanM7ewBw4ChX4qyGjs2",
  "type": 103,
  "version": 4,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "senderPublicKey": "YNpp7chAaudMqEtSZZPyN4GYLJ5ZTXdjCXrQdszzuRp",
  "contractName": "Your contract name",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "imageHash":
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null,
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  "proofs": [

↪"4vqLnpJRFpcDgM5vgi78DpZnVfqztsARHNb7Hbmq3mQBjS3SRnzFAiYjRvPazEVMhBM9cE4Rcp6H5K29kk75Uxyh
↪"
  ]
}
```

Version 5

Подписание:

```
{
  "type": 103,
  "version": 5,
  "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
  "password": "signing-key-password",
  "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
  "contractName": "Your contract name",
  "imageHash":
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
  "params": [
    {
      "type": "string",
      "key": "test_key",
      "value": "test_value"
    }
  ],
  "fee": 100000000,
  "timestamp": 1651487626477,
  "feeAssetId": null,
  "atomicBadge": null,
  "validationPolicy": {
    "type": "majority"
  },
  "apiVersion": "1.0"
}
```

Публикация:

```
{
   "id": "4WVhw3QdiinpE5QXDG7QfqLiLanM7ewBw4ChX4qyGjs2",
   "type": 103,
   "version": 5,
   "sender": "3NpN3HyHzGj7Ny1k5F9zMMQ2n54TZg86G9D",
   "senderPublicKey": "YNpp7chAaudMqEtSZZPyN4GYLJ5ZTXdjCXrQdszzuRp",
   "contractName": "SOME_CONTRACT_NAME",
   "image": "registry.yourdomain.com/test-docker-repo/contract:v1.0.0",
   "imageHash":
↪"573387bbf50cfdeda462054b8d85d6c24007f91044501250877392e43ff5ed50",
   "params": [
     {
```

(continues on next page)

(продолжение с предыдущей страницы)


```
      "key": "int",
      "type": "integer",
      "value": 24
    },
    {
      "key": "bool",
      "type": "boolean",
      "value": true
    },
    {
      "key": "blob",
      "type": "binary",
      "value": "base64:YWxpY2U="
    }
  ],
  "fee": 0,
  "timestamp": 1665267880,
  "feeAssetId": null,
  "atomicBadge": {
    "trustedSender": "SOME_SENDER_ACCOUNT_ADDRESS"
  },
  "proofs": [
↪"32mNYSefBTrkVngG5REkmmGAVv69ZvNhpbegmnqDReMTmXNyYqbECPgHgXrX2UwyKGLFS45j7xDFyPXjF8jcfw94
↪"
  ],
  "validationPolicy": {
    "type": "SOME_VALIDATION_POLICY_NAME"
  },
  "apiVersion": "SOME_CONTRACT_VERSION",
  "payments": [
    {
      "amount": 100
    },
    {
      "assetId": "SOME_ASSET_ID",
      "amount": 100
    }
  ]
 }
```


В версии **4** данной транзакции настраивается валидация результатов исполнения обновляемого смарт-контракта при помощи поля `validationPolicy.type` (см. раздел *Валидация смарт-контрактов*). Варианты политик валидации:

- `any` – сохраняется действующая в сети общая политика валидации: для майнинга загружаемого смарт-контракта, майнер подписывает соответствующую транзакцию *105*. Также этот параметр устанавливается, если в вашей сети нет ни одного зарегистрированного валидатора.
- `majority` – транзакция считается валидной, если она подтверждена большинством валидаторов: **2/3** от общего числа зарегистрированных адресов с ролью **contract_validator**.
- `majorityWithOneOf(List[Address])` – транзакция считается валидной, если собрано большинство валидаторов, среди которых присутствует хотя бы один из адресов, включенных в список параметра.

Адреса, включаемые в список, должны иметь действующую роль **contract_validator**.

Предупреждение: При выборе политики валидации `majorityWithOneOf(List[Address])`, заполните список адресов, передача пустого списка запрещена.

В версии **5** данной транзакции пользователь может перевести свои активы на баланс контракта. Для этого в поле `payments` указывается массив ассетов и их количество. Можно передать как системный токен WEST, так и любые другие ассеты, созданные в сети. Использование версии 5 данной транзакции возможно начиная с релиза 1.12 после *активации фичи* 1120.

При работе в частной сети транзакция 103 предусматривает загрузку Docker-образа контракта не только из репозиториев, 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:


```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "image": "customregistry.com:5000/stateful-increment-contract:latest",
  "fee" : 100000000,
  "imageHash" : "ad6d0f8a61222794da15571749bc9db08e76b6a120fc1db90e393fc0ee9540d8",
  "type" : 103,
  "params" : [ {
    "type" : "string",
    "value" : "Value_here",
    "key" : "data"
  }, {
    "type" : "integer",
    "value" : 500,
    "key" : "length"
  } ],
  "version" : 4,
  "atomicBadge" : null,
  "apiVersion" : "1.0",
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [
↪"L521YncSMJDPqwBjQyS7m7Q6tseAw51nYE8iiPChEALx7S2WvpSosCVtWkXxh2ZqJ6LHkCvjVjRVuVs793kzjw8
↪" ],
  "contractName" : "grpc_validatable_statefull here_often",
  "id" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
  "validationPolicy" : {
    "type" : "any"
  },
  "timestamp" : 1625732696641,
  "height" : 1028130
}
```

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.

Важно: Начиная с релиза 1.12 (после *активации фичи* 1120) вызов *REST контрактов* невозможен. Рекомендуется использовать gRPC контракты вместо REST контрактов.

В поле `contractVersion` транзакции указывается версия контракта:

- 1 - для нового контракта;
- 2 - для обновленного контракта.

Данное поле доступно только для транзакций второй версии и старше: если в поле `version` транзакции создания смарт-контракта указано значение >= 2. Контракт обновляется при помощи транзакции *107*.

Если контракт не выполнился или выполнился с ошибкой, то транзакции 103 и 104 удаляются и не попадают в блок.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
contract	ByteStr	ID смарт-контракта
fee	Long	*Комиссия за транзакцию в WE Mainnet*
sender	ByteStr	Адрес отправителя транзакции
passworc	String	Пароль от ключевой пары в keystore ноды - *опциональное поле*
type	Byte	Номер транзакции (**104**)
params	List[DataEr	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type` `value` и `key` через запятую - *опциональное поле*
params.k	Byte	Ключ параметра
params.t	Byte	Тип данных параметра. Возможные значения: `binary` `bool` `integer` `string`
params.v	Byte	Значение параметра
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**104**)
id	Byte	ID транзакции вызова контракта
sender	ByteSt	Адрес отправителя транзакции
senderP	PublicK	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timesta	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
proofs	List(By	Массив подтверждений транзакции (**в формате base58**)
version	Byte	Версия транзакции
contrac	ByteSt	ID смарт-контракта
params	List[Da	Входные и выходные данные смарт-контракта. Вносятся при помощи полей `type` `value` и `key` через запятую - *опциональное поле*
params	Byte	Ключ параметра
params	Byte	Тип данных параметра. Возможные значения: `binary` `bool` `integer` `string`
params	Byte	Значение параметра
paymer		Целое число. которое определяет количество передаваемых контракту ассетов; в поле `amount` младшие разряды соответствуют дробным частям количества передаваемого ассета. если его `decimals` не нулевой. Поле является опциональным.
paymer		Идентификатор передаваемого контракту ассета; для передачи системного токена WEST поле assetId должно быть пустым. Поле является опциональным.

JSON-представление:

Version 2

Подписание:

```
{
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "fee": 10,
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "password": "",
    "type": 104,
    "params":
    [
        {
           "type": "integer",
           "key": "a",
           "value": 1
        },
        {
           "type": "integer",
           "key": "b",
           "value": 100

        }
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    ],
    "version": 2,
    "contractVersion": 1
}
```

Публикация:

```
{
    "type": 104,
    "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
    "fee": 10,
    "timestamp": 1549365736923,
    "proofs": [
↪"2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v
↪" ],
    "version": 2,
    "contractVersion": 1,
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "params":
    [
        {
        "key": "a",
        "type": "integer",
        "value": 1
        },
        {
        "key": "b",
        "type": "integer",
        "value": 100
        }
    ]
}
```

Version 3

Подписание:

```
{
    "contractId": "Dgk1hR7xRnDT1KJreaXCVtZLrnd5LJ8uUYtoZyQrV1LJ",
    "fee": 10000000,
    "sender": "3NpkC1FSW9xNfmAMuhRSRArLgnfyGyEry7w",
    "password": "",
    "type": 104,
    "params":
    [ {
      "type" : "string",
      "value" : "value",
      "key" : "data"
    }, {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      "type" : "integer",
      "value" : 500,
      "key" : "length"
    } ],
    "version": 3,
    "contractVersion": 1,
}
```

Публикация:

```
{
  "senderPublicKey" : "9Kgnqqxr5MU3PNrLgf1dkZL2HH6LBktB5Pv9L1cVELi1",
  "fee" : 10000000,
  "type" : 104,
  "params" : [ {
    "type" : "string",
    "value" : "data_response",
    "key" : "action"
  }, {
    "type" : "string",
    "value" : "000008_regular_data_request_
↪2m3SgcnQz9LXVi9ETy3CFHVGM1EyiqJi3vvRRQUM3oPp",
    "key" : "request_id"
  }, {
    "type" : "string",
    "value" : "76.33",
    "key" : "value"
  }, {
    "type" : "string",
    "value" : "1627678789267",
    "key" : "timestamp"
  } ],
  "version" : 3,
  "contractVersion" : 1,
  "sender" : "3NpkC1FSW9xNfmAMuhRSRArLgnfyGyEry7w",
  "feeAssetId" : null,
  "proofs" : [
↪"4aanqYjaTVNot8Fbz5ixjwKSdqS5x3DdvzxQ4WsTaPcftYdoFx99xwLC3UPN91VAtez4RTMzaYb1TECaVxHHT9AH
↪" ],
  "contractId" : "Dgk1hR7xRnDT1KJreaXCVtZLrnd5LJ8uUYtoZyQrV1LJ",
  "id" : "55imLuEXyVpBXb1S64R5PRx9acQQHaEATPwYwUVpqjAT",
  "timestamp" : 1627678789267,
  "height" : 1076064
}
```

Version 4

Подписание:

```
{
    "contractId": "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
    "fee": 10000000,
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "password": "",
    "type": 104,
    "params":
    [ {
      "type" : "string",
      "value" : "value",
      "key" : "data"
    }, {
      "type" : "integer",
      "value" : 500,
      "key" : "length"
    } ],
    "version": 4,
    "contractVersion": 3,
    "atomicBadge" : null
}
```

Публикация:

```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "fee" : 10000000,
  "type" : 104,
  "params" : [ {
    "type" : "string",
    "value" : "value",
    "key" : "data"
  }, {
    "type" : "integer",
    "value" : 500,
    "key" : "length"
  } ],
  "version" : 4,
  "contractVersion" : 3,
  "atomicBadge" : null,
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [
↪"2bpALen4diR7DTFhNQCrZKPueCPds2gFFPxe1KVzQwfRuGaK6QfvtpN8oqaZMsStoEHAa5DrTkKM8AuzHPYyMPVP
↪" ],
  "contractId" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
  "id" : "GBfibn8VjGmDS9ex4Nd4JNRLvDyvJjj8jLUUcbYwFTCf",
  "timestamp" : 1625732766458,
  "height" : 1028132
}
```

Version 5

Подписание:

```
{
  "contractId": "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
  "fee": 10000000,
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "password": "",
  "type": 104,
  "params": [
    {
      "type" : "string",
      "value" : "value",
      "key" : "data"
    },
    {
      "type" : "integer",
      "value" : 500,
      "key" : "length"
    }
  ],
  "version": 4,
  "contractVersion": 3,
  "atomicBadge" : null
}
```

Публикация:

```
{
  "senderPublicKey": "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
    "fee": 0,
    "type": 104,
    "params": [
      {
        "key": "int",
        "type": "integer",
        "value": 24
      },
      {
        "key": "bool",
        "type": "boolean",
        "value": true
      },
      {
        "key": "blob",
        "type": "binary",
        "value": "base64:YWxpY2U="
      }
    ],
    "version": 5,
    "contractVersion": "3",
    "atomicBadge": {
```

(continues on next page)

(продолжение с предыдущей страницы)


```
      "trustedSender": "SOME_SENDER_ACCOUNT_ADDRESS"
    },
    "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "feeAssetId": null,
    "proofs": [

↪"32mNYSefBTrkVngG5REkmmGAVv69ZvNhpbegmnqDReMTmXNyYqbECPgHgXrX2UwyKGLFS45j7xDFyPXjF8jcfw94
↪"
    ],
    "contractId": "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
    "id": "GBfibn8VjGmDS9ex4Nd4JNRLvDyvJjj8jLUUcbYwFTCf",
    "timestamp": 1665267880,
    "payments": [
      {
        "amount": 100
      },
      {
        "assetId": "SOME_ASSET_ID",
        "amount": 100
      }
    ]
  }
```


В версии **5** данной транзакции пользователь может перевести свои активы на баланс контракта. Для этого в поле `payments` указывается массив ассетов и их количество. Можно передать как системный токен WEST, так и любые другие ассеты, созданные в сети. Использование версии 5 данной транзакции возможно начиная с релиза 1.12 после *активации фичи* 1120.

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, 104 или 107 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 на публикацию транзакции

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**105**)
id	Byte	ID транзакции исполнения контракта
sender	Byte	Адрес отправителя транзакции
senderP	Publi	Открытый ключ отправителя транзакции
passwo	Strin	Пароль от ключевой пары в keystore ноды - *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timesta	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
proofs	List(I	Массив подтверждений транзакции (**в формате base58**)
version	Byte	Версия транзакции
tx	Array	Тело транзакции 103 или 104 исполняемого смарт-контракта
results	List[I	Список возможных результатов исполнения смарт-контракта
height	Byte	Высота выполнения транзакции - *опциональное поле*
assetOp		Упорядоченный список действий смарт-контракта с доступными ему ассетами, в том числе выпуск нового ассета, перевыпуск ассета, сжигание ассета или перевод доступного контракту ассета другому пользователю
assetOp		Служебное поле, представляющее тип операции. Поле может принмать следующие значения: `issue`, `reissue`, `burn`, `transfer`
assetOp		Служебное поле, представляющее версию объекта
assetOp		При выпуске ассетов значение поля вычисляется посредством gRPC-метода CalculateAssetId сервиса ContractService. При перевыпуске или сжигании ассета идентификатор определяет, перевыпуск или сжигание какого токена осуществляется. При переводе ассета определяет, передача какого ассета осуществляется. В случае отправки системного токена WEST поле assetId должно быть опущено или равно null.
assetOp		Имя ассета
assetOp		Описание ассета
assetOp		При выпуске ассетов в поле задаётся суммарное количество выпускаемого ассета. При перевыпуске ассета – количество довыпускаемого ассета
assetOp		При выпуске ассетов в поле задаётся количество десятичных разрядов выпускаемого ассета
assetOp		Флаг, указывающий на возможность перевыпускать ассет
assetOp		При выпуске ассетов значение поля используется для расчёта assetId. Не может быть равным 0. Диапазон допустимых значений: от -128 до 127. Также не может быть выпущено в рамках одного вызова контракта несколько ассетов с одинаковым nonce
assetOp		При сжигании ассетов в поле задаётся количество сжигаемого ассета. При переводе ассета определяет количество передаваемого ассета.
assetOp		При переводе ассетов в поле задаётся адрес пользователя, которому контракт осуществляет передачу активов.

JSON-представление:

Version 2

Публикация:


```
{
    "type": 105,
    "id": "38GmSVC5s8Sjeybzfe9RQ6p1Mb6ajb8LYJDcep8G8Umj",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "password": "",
    "fee": 500000,
    "timestamp": 1550591780234,
    "proofs": [
↪"5whBipAWQgFvm3myNZe6GDd9Ky8199C9qNxLBHqDNmVAUJW9gLf7t9LBQDi68CKT57dzmnPJpJkrwKh2HBSwUer6
↪" ],
    "version": 2,
    "tx":
         {
            "type": 103,
            "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
            "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
            "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
            "fee": 500000,
            "timestamp": 1550591678479,
            "proofs": [
↪"yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv
↪" ],
            "version": 2,
            "image": "stateful-increment-contract:latest",
            "imageHash":
↪"7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
            "contractName": "stateful-increment-contract",
            "params": [],
            "height": 1619
          },
    "results": [],
    "height": 1619,
    "atomicBadge" : null
}
```


Version 3

Публикация:


```
{
    "type": 105,
    "id": "SOME_TX_ID",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsjMVT2M",
    "fee": 0,
    "timestamp": 1665267880,
    "proofs": [
```


(continues on next page)

(продолжение с предыдущей страницы)

```
↪"32mNYSefBTrkVngG5REkmmGAVv69ZvNhpbegmnqDReMTmXNyYqbECPgHgXrX2UwyKGLFS45j7xDFyPXjF8jcfw94
↪"
    ],
    "version": 3,
    "tx": { // inner (executed) tx json-object
      "id": "SOME_INNER_TX_ID",
      // ...
          },
    "results": [
        {
          "key": "int",
          "type": "integer",
          "value": 24
        },
        {
          "key": "bool",
          "type": "boolean",
          "value": true
        },
        {
          "key": "blob",
          "type": "binary",
          "value": "base64:YWxpY2U="
        }
      ],
    "assetOperations": [
        {
          "operationType": "issue",
          "version": 1,
          "assetId": "SOME_ASSET_ID",
          "name": "Gigacoin",
          "description": "Gigacoin",
          "quantity": 10000000000,
          "decimals": 8,
          "isReissuable": true,
          "nonce": 1 // SOME_NONCE
        },

          "operationType": "burn",
          "version": 1,
          "assetId": "SOME_ASSET_ID",
          "amount": 1000
        },
        {
          "operationType": "reissue",
          "version": 1,
          "assetId": "SOME_ASSET_ID",
          "quantity": 10000000000,
          "isReissuable": true
        },
        {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
          "operationType": "transfer",
          "version": 1,
          "recipient": "SOME_RECIPIENT_ACCOUNT_ADDRESS",
          "assetId": "SOME_ASSET_ID",
          "amount": 1000
        }
      ]
    "resultsHash": "SOME_RESULTS_HASH",
    "validationProofs": [],
}
```

В версии **3** данной транзакции в поле `assetOperations` можно передать последовательность операций над ассетами, например перевод токенов с баланса пользователя на баланс контракта. Можно передать как системный токен WEST, так и любые другие ассеты, созданные в сети. Использование версии 3 данной транзакции возможно начиная с релиза 1.12 после *активации фичи* 1120. После активации фичи 1120 в сети используется только версия 3 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**.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - *опциональное поле*
contractId	ByteStr	ID смарт-контракта
fee	Long	*Комиссия за транзакцию в WE Mainnet*
type	Byte	Номер транзакции (**106**)
version	Byte	Версия транзакции

Публикация:

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	*Комиссия за транзакцию в WE Mainnet*
proofs	List(ByteStr)	Массив подтверждений транзакции (**в формате base58**)
version	Byte	Версия транзакции
contractId	ByteStr	ID смарт-контракта
height	Byte	Высота выполнения транзакции

JSON-представление:

Version 2

Подписание:

```
{
    "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "password": "",
    "contractId": "HKftkVDTcQp6kxdqVYNdzB9d4rhND4YRKxwJV1thMXcr",
    "fee": 1000000,
    "type": 106,
    "version": 2,
}
```

Публикация:

```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : "7QpXWLGuaspzrMsESRaHTgksndq5mcvfbVrqBTuLbxuy",
  "proofs" : [
↪"3FKPGT8YbLVun5cffZi1sHkgr9JZVxkeN7z2kUqDVLfhB5CwMtCAfyStRz1tpZuriKsR3MaBqNfReGx5sM2qey8i
↪" ],
  "fee" : 1000000,
  "contractId" : "HKftkVDTcQp6kxdqVYNdzB9d4rhND4YRKxwJV1thMXcr",
  "id" : "5hXuHs5HVhZSfek153t76HfW6egmCLdZmi5AeFzYBFN",
  "type" : 106,
  "version" : 2,
  "timestamp" : 1625648619321,
  "height" : 1025992
}
```

Version 3

Подписание:

```
{
    "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "password": "",
    "contractId": "75PumcfCVxzV3v7RAPYQUwCtSpU21hxfaWFhureCRTLM",
    "fee": 1000000,
    "type": 106,
    "version": 3,
  "atomicBadge" : {
    "trustedSender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  }
}
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "atomicBadge" : {
    "trustedSender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
  },
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [
↪"22tK24qHhgbTDjtRmR86z3WeLLqLnqPvhUhQrz8ohfbCwQ9nrwmHESuT9aFuwABeBRJ7MfVob1FiJnqg3y2PHLSj
↪" ],
  "fee" : 1000000,
  "contractId" : "75PumcfCVxzV3v7RAPYQUwCtSpU21hxfaWFhureCRTLM",
  "id" : "7opPrLd6x1hATRr9R5oXnEbYjYQzo5cn4Qpkiz12Mw9b",
  "type" : 106,
  "version" : 3,
  "timestamp" : 1619186857911,
  "height" : 861644
}
```


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**.

Важно: Начиная с релиза 1.12 (после *активации фичи* 1120) создание и вызов *REST контрактов* невозможны. Также нельзя обновить REST контракт на gRPC контракт. Рекомендуется использовать gRPC контракты вместо REST контрактов.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
image	Array[Byt	Имя Docker-образа смарт-контракта
sender	ByteStr	Адрес отправителя транзакции
password	String	Пароль от ключевой пары в keystore ноды - *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
contractId	ByteStr	ID смарт-контракта
imageHash	Array[Byt	Хэш Docker-образа смарт-контракта
type	Byte	Номер транзакции (**107**)
version	Byte	Версия транзакции
apiVersion	Byte	Версия API для gRPC-методов смарт-контракта (см. *Сервисы gRPC используемые смарт-контрактом*).
validationPolic	String	Тип политики валидации смарт-контрактов.

Публикация:

Поле	Тип данных	Описание
senderPublicKey	PublicKeyAccount	Открытый ключ отправителя транзакции
tx	Array	Тело транзакции 105 обновляемого контракта

JSON-представление:

Version 2

Подписание:

```
{
 "image" : "we-sc/grpc-contract-example:2.2-test-update",
 "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
 "password": "",
 "fee" : 100000000,
 "contractId" : "BWzX4mRBEnHKgn3HB78My5DZzDAqnCLWCCNpCuRkZrJA",
 "imageHash" :
↪"075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
 "type" : 107,
 "version" : 2
}
```

Публикация:

```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "image" : "we-sc/grpc-contract-example:2.2-test-update",
  "fee" : 100000000,
  "imageHash" :
↪"075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
  "type" : 107,
  "version" : 2,
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [
↪"RetQwzuWZWxpSNMqwB7k7o6hSm6nhFCc49zKUpwZEedzBYcohj9NVEPwAbKLW9RzRKX168xApV7Nu2qV2jaHAMg
↪" ],
  "contractId" : "BWzX4mRBEnHKgn3HB78My5DZzDAqnCLWCCNpCuRkZrJA",
  "id" : "6oopqcEf4AF943SCAqkBPrghyeQhmwn64TrhtCZbAn3v",
  "timestamp" : 1625649822957,
  "height" : 1026022
}
```

Version 3

Подписание:

```
 {
 "image" : "registry.wavesenterpriseservices.com/we-sc/grpc-contract-example:2.
↪2-test-update",
 "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
 "password": "",
 "fee" : 100000000,
 "contractId" : "HTqdjXUPTHZqGen2KKUkEenTELAqQ8irN58LA8EcP17q",
 "imageHash" :
↪"075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
 "type" : 107,
 "version" : 3,
 "atomicBadge" : null
}
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "image" : "registry.wavesenterpriseservices.com/we-sc/grpc-contract-
↪example:2.2-test-update",
  "fee" : 100000000,
  "imageHash" :
↪"075ad1607f193cc6fdb5e85c201f9ca3907c622718d75706bbc2a94a330de5b5",
  "type" : 107,
  "version" : 3,
  "atomicBadge" : null,
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [
↪"3ncWfFPqBAdgh65YceCCvF2RhUWWokQc9MsnHk27YLrYmPj9gWgrbRcousymJVA7ARFSz5UJcdW4Sa62FFhR5en3
↪" ],
  "contractId" : "HTqdjXUPTHZqGen2KKUkEenTELAqQ8irN58LA8EcP17q",
  "id" : "B7qjgCa9N6M6FwV63PbLwvtVpFo4bzB5gRZzGjwJpKJV",
  "timestamp" : 1619187337697,
  "height" : 861650
}
```

Version 4

Подписание:

```
 {
 "image" : "we-sc/grpc_validatable_stateless:0.1",
 "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
 "password": "",
 "fee" : 100000000,
 "contractId" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
 "imageHash" :
```

(continues on next page)

(продолжение с предыдущей страницы)


```
↪"bd98a7d3e55506ff936d8ea15e170a24d27662edd1b47e4fd20801d10655af8d",
 "type" : 107,
 "version" : 4,
 "atomicBadge" : null
}
```


Публикация:


```
{
  "senderPublicKey" : "CgqRPcPnexY533gCh2SSvBXh5bca1qMs7KFGntawHGww",
  "image" : "we-sc/grpc_validatable_stateless:0.1",
  "fee" : 100000000,
  "imageHash" :
↪"bd98a7d3e55506ff936d8ea15e170a24d27662edd1b47e4fd20801d10655af8d",
  "type" : 107,
  "version" : 4,
  "atomicBadge" : null,
  "apiVersion" : "1.0",
  "sender" : "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "feeAssetId" : null,
  "proofs" : [
↪"fZr9LpqSWbPcUzArSZxFDEuygN62hR63j2Cz1GyTFxPNRrNvVwkDhTDcC8zwRp235gA1gSM8fvPps9mvPTWDQ4p
↪" ],
  "contractId" : "HSLdKYqLq4LcZpq9LPki8Yv4ZRkFapVyHEYw1vZW2MoG",
  "id" : "HWZy7219Nx5oxj2QnK3ReEuZiqsjoULbmfdQz8YysFSz",
  "validationPolicy" : {
    "type" : "any"
  },
  "timestamp" : 1625732772746,
  "height" : 1028132
}
```


В версии **4** данной транзакции настраивается валидация результатов исполнения обновляемого смарт-контракта при помощи поля `validationPolicy.type` (см. раздел *Валидация смарт-контрактов*).

Варианты политик валидации:

- `any` – сохраняется действующая в сети общая политика валидации: для майнинга обновляемого смарт-контракта майнер подписывает соответствующую транзакцию *105*. Также этот параметр устанавливается, если в вашей сети нет ни одного зарегистрированного валидатора.
- `majority` – транзакция считается валидной, если она подтверждена большинством валидаторов: **2/3** от общего числа зарегистрированных адресов с ролью **contract_validator**.
- `majorityWithOneOf(List[Address])` – транзакция считается валидной, если собрано большинство валидаторов, среди которых присутствует хотя бы один из адресов, включенных в список параметра. Адреса, включаемые в список, должны иметь действующую роль **contract_validator**.

Предупреждение: При выборе политики валидации `majorityWithOneOf(List[Address])`, заполните список адресов, передача пустого списка запрещена.

110. GenesisRegisterNode Transaction

Регистрация ноды в генезис-блоке при старте блокчейна.

Данная транзакция не требует подписания.

Структура данных на публикацию транзакции

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**110**)
id	Byte	ID транзакции регистрации ноды в генезис-блоке
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
signature	ByteStr	Подпись транзакции (**в формате base58**)
version	Byte	Версия транзакции
targetPubK	Byte	Публичный ключ регистрируемой ноды
height	Byte	Высота выполнения транзакции

111. RegisterNode Transaction

Регистрация новой ноды в блокчейне или ее удаление.

Структуры данных транзакции

Подписание:

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 - *опциональное поле*
targetPubKey	Byte	Публичный ключ регистрируемой или удаляемой ноды
NodeName	Byte	Имя ноды
fee	Long	*Комиссия за транзакцию в WE Mainnet*

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**111**)
id	Byte	ID транзакции регистрации ноды
sender	ByteStr	Адрес отправителя транзакции
senderPublicK	PublicKeyAccc	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
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 - *опциональное поле*

JSON-представление:

Version 1

Подписание:

```
{
  "type": 111,
  "opType": "add",
  "sender":"3NgSJRdMYu4ZbNpSbyRNZLJDX926W7e1EKQ",
  "password": "",
  "targetPubKey": "6caEKC1UBgRvgAe9A7L5PWcrawrnEZGxtsXynGESwSj7",
  "nodeName": "GATEs node",
  "fee": 1100000,
}
```

Публикация:

```
{
    "senderPublicKey" : "FWz5gZ2w2ZxXbKEiwhgEcZKT4we1Wneh9XqmCeGPsA4r",
    "nodeName" : "GATEs node",
    "fee" : 1100000,
    "opType" : "add",
    "type" : 111,
    "version" : 1,
    "target" : "3NtieMGjVAH1nDsvnSEJ37BSW3hpJV2CneY",
    "sender" : "3NgSJRdMYu4ZbNpSbyRNZLJDX926W7e1EKQ",
    "proofs" : [
↪"FHEexr8MqMCkdqaVRrfxv7dnQFwo1VQxQFb4rW2VKh1NkuAhjhtzftKybBQCVbpKcCD1ZTRhwATpwERF9re2Viz
↪" ],
    "id" : "6WnDGkBDeSjg5y6QqVdy3BFHUy5nnr4QsxZCeNXZtZoq",
    "targetPubKey" : "6caEKC1UBgRvgAe9A7L5PWcrawrnEZGxtsXynGESwSj7",
    "timestamp" : 1619078302988,
    "height" : 858895
}
```

112. CreatePolicy Transaction

Создание группы доступа к конфиденциальным данным из указанных адресов.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
sender	ByteStr	Адрес отправителя транзакции
policyNa	String	Имя создаваемой группы доступа
passworc	String	Пароль от ключевой пары в keystore ноды - *опциональное поле*
recipient	Array[By	Массив адресов участников группы доступа к конфиденциальным данным через запятую
fee	Long	*Комиссия за транзакцию в WE Mainnet*
descripti	Array[byt	Произвольное описание транзакции (**в формате base58**)
owners	Array[By	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
type	Byte	Номер транзакции (**112**)
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**112**)
id	Byte	ID транзакции создания группы доступа
sender	ByteStr	Адрес отправителя транзакции
senderPubl	PublicKeyA	Открытый ключ отправителя транзакции
policyNam	String	Имя создаваемой группы доступа
recipients	Array[Bytes	Массив адресов участников группы доступа к конфиденциальным данным через запятую
owners	Array[Bytes	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) - *опциональное поле*
proofs	List(ByteSt	Массив подтверждений транзакции (**в формате base58**)
height	Byte	Высота выполнения транзакции
description	Array[byte]	Произвольное описание транзакции (**в формате base58**)
version	Byte	Версия транзакции

JSON-представление:

Version 2

Подписание:

```
 {
    "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "policyName": "Policy# 7777",
    "password":"sfgKYBFCF@#$fsdf()%",
    "recipients": [
      "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
      "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
      "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
          "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
          "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
    ],
    "fee": 15000000,
    "description": "Buy bitcoin by 1c",
    "owners": [
      "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
      "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
      "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
    ],
    "type": 112,
    "version": 2,
}
```

Публикация:

```
 {
    "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "policyName": "Policy# 7777",
    "password":"sfgKYBFCF@#$fsdf()%",
    "recipients": [
      "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
      "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
      "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
          "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
          "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx"
    ],
    "fee": 15000000,
    "description": "Buy bitcoin by 1c",
    "owners": [
      "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
      "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
      "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
    ],
    "type": 112,
    "version": 2,
}
```

Version 3

Подписание:

```
 {
    "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "policyName": "Policy_v3_for_demo_txs",
    "password":"sfgKYBFCF@#$fsdf()%",
    "recipients" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
↪"3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
↪ "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn
↪" ],
    "fee": 100000000,
    "description": "",
    "owners" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
↪"3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
↪ "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn
↪" ],
    "type": 112,
    "version": 3
}
```

Публикация:

```
{
    "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
    "policyName" : "Policy_v3_for_demo_txs",
    "fee" : 100000000,
    "description" : "",
    "owners" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
↪"3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
↪ "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn
↪" ],
    "type" : 112,
    "version" : 3,
    "atomicBadge" : null,
    "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "feeAssetId" : null,
    "proofs" : [
↪"4NccZyPCgchDjeMdMmFKu7kxyU8AFF4e9cWaPFTQVQyYU1ZCCu3QmtmkfJkrDpDwGs4eJhYUVh5TnwYvjZYKPhLp
↪" ],
    "recipients" : [ "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
↪"3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF", "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
↪ "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d", "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn
↪" ],
    "id" : "5aYtmTr1AYYG8BrYvTTSqKzfJZxfgorx1BLGVwSAhwrz",
    "timestamp" : 1619186864092,
    "height" : 861637
}
```

113. UpdatePolicy Transaction

Изменение группы доступа к конфиденциальным данным.

Структуры данных транзакции

Подписание:

Поле	Тип данных	Описание
policyIc	String	Идентификатор создаваемой группы доступа
passwo	String	Пароль от ключевой пары в keystore ноды - *опциональное поле*
sender	ByteStr	Адрес отправителя транзакции
recipier	Array[Byt	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[Byt	Массив адресов-администраторов группы доступа через запятую: администраторы имеют право изменять группу доступа
type	Byte	Номер транзакции (**113**)
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**113**)
id	Byte	ID транзакции изменения группы доступа
sender	ByteStr	Адрес отправителя транзакции
senderPubl	PublicKeyA	Открытый ключ отправителя транзакции
policyId	String	Идентификатор создаваемой группы доступа
recipients	Array[Bytes	Массив адресов для добавления или удаления участников группы доступа к конфиденциальным данным через запятую
owners	Array[Bytes	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) - *опциональное поле*
proofs	List(ByteSt	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

JSON-представление:

Version 2

Подписание:

```
{
  "policyId": "UkvoboGXiwWpASr1GLG9M1MUbhrEMo4NBS7kquxVMw5",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
   "fee": 50000000,
   "opType": "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type": 113,
  "version": 2
 }
```

Публикация:

```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "fee" : 50000000,
  "opType" : "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type" : 113,
  "version" : 2,
  "policyId" : "UkvoboGXiwWpASr1GLG9M1MUbhrEMo4NBS7kquxVMw5",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [
↪"2CKd57kU3wbxdrHxMPNbrWHptnf5ZcydYjqxMPk46miMcUUAxgFGXcy621cjYFXC3vjpKNNrB2QcgtKe1Yx9TcLY
↪" ],
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "id" : "6o4azRwzmMg9SqWUq6rv6GAe5gzTYJvE5ek1v9VM3Mb",
  "timestamp" : 1619004195630,
  "height" : 856970
}
```

Version 3

Подписание:

```
{
  "policyId": "5aYtmTr1AYYG8BrYvTTSqKzfJZxfgorx1BLGVwSAhwrz",
  "password":"sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
   "fee": 50000000,
   "opType": "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type": 113,
  "version": 3
 }
```

Публикация:


```
{
  "senderPublicKey" : "7GiFGcGaEN87ycK8v71Un6b7RUoeKBU4UvUHPYbeHaki",
  "fee" : 50000000,
  "opType" : "remove",
  "owners" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "type" : 113,
  "version" : 3,
  "atomicBadge" : null,
  "policyId" : "5aYtmTr1AYYG8BrYvTTSqKzfJZxfgorx1BLGVwSAhwrz",
  "sender" : "3NxAooHUoLsAQvxBSqjE91WK3LwWGjiiCxx",
  "feeAssetId" : null,
  "proofs" : [
↪"2QMGoz6rycNsDLhN3mDce2mqGRQQ8r26vDDw551pnYcAecpFBDA8j38FVqDjLTGuFHs6ScX32fsGcaemmptpCFHk
↪" ],
  "recipients" : [ "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF" ],
  "id" : "Hwqf8LgpQfEcUYX9nMNG8uU2Cw1xSuGFqYxmuACpvU1L",
  "timestamp" : 1619187450552,
  "height" : 861653
}
```


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 на публикацию транзакции

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**114**)
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicK	PublicKeyAcco	Открытый ключ отправителя транзакции
policyId	String	Имя создаваемой группы доступа
dataHash	String	Хэш конфиденциальных данных для отправки
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
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 исполняется полностью (ни одна из включенных транзакций не отклоняется) или не исполняется 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
- *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*

Структуры данных транзакции

Подписание:

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 - *опциональное поле*
fee	Long	*Комиссия за транзакцию в WE Mainnet*
version	Byte	Версия транзакции

Публикация:

Поле	Тип данных	Описание
type	Byte	Номер транзакции (**114**)
id	Byte	ID транзакции
sender	ByteStr	Адрес отправителя транзакции
senderPublicK	PublicKeyAccc	Открытый ключ отправителя транзакции
fee	Long	*Комиссия за транзакцию в WE Mainnet*
timestamp	Long	Временная метка в формате **Unix Timestamp** (в миллисекундах) - *опциональное поле*
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 - *опциональное поле*
version	Byte	Версия транзакции

JSON-представление:

Version 1

Подписание:

```
{
    "sender": sender_0,
    "transactions": [
        signed_transfer_tx,
        signed_transfer_tx2
    ],
    "type": 120,
    "version": 1,
    "password":"lskjbJJk$%^#298",
    "fee": 0,
}
```

Публикация:

```
{
  "sender": "3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP",
  "transactions": [
        signed_transfer_tx,
        signed_transfer_tx2
  ],
  "type": 120,
  "version": 1,
}
```

Смотрите также

Описание транзакций

Комиссии в сети Mainnet

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	*Genesis Transaction*	Без версии
3	*Issue Transaction*	2
4	*Transfer Transaction*	3
5	*Reissue Transaction*	2
6	*Burn Transaction*	2
8	*Lease Transaction*	2
9	*Lease Cancel Transaction*	2
10	*Create Alias Transaction*	3
11	*Mass Transfer Transaction*	2
12	*Data Transaction*	2
13	*Set Script Transaction*	1
14	*Sponsorship Transaction*	1
15	*Set Asset Script Transaction*	1
101	*Genesis Permission Transaction*	Без версии
102	*Permission Transaction*	2
103	*Create Contract Transaction*	5
104	*Call Contract Transaction*	5
105	*Executed Contract Transaction*	3
106	*Disable Contract Transaction*	3
107	*Update Contract Transaction*	4
110	*Genesis Resgister Node Transaction*	1
111	*Register Node Transaction*	1
112	*Create Policy Transaction*	3
113	*Update Policy Transaction*	3
114	*Policy Data Hash Transaction*	3
120	*Atomic Transaction*	1

Смотрите также

Транзакции блокчейн-платформы

Описание транзакций

Комиссии в сети Mainnet

Смотрите также

Описание транзакций

Актуальные версии транзакций

Комиссии в сети Mainnet

1.25 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 AtomicTransaction*, 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*.

1.25.1 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 ее «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.

1.25.2 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 Mainnet

1.26 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:

1.26.1 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.

Proof of Stake

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 хэша преобразуются в число X_n , которое указывает на следующего майнера. Время генерации блока для аккаунта i, рассчитывается следующим образом:

$$T_i = T_{min} + C_1 \log(1 - C_2 \frac{\log \frac{X_n}{X_{max}}}{b_i A_n})$$

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 приватных сетей. Также в таких сетях не требуется 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.

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

1.26.2 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.

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*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 ($T_{Current}$ -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.

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.

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.

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.

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
    },
    {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      from-height = 25000
      sync-duration = 10s
      round-duration = 30s
    }]
}
```

Смотрите также

Общая настройка платформы: настройка консенсуса

Алгоритмы консенсуса

Алгоритм консенсуса LPoS

Алгоритм консенсуса CFT

1.26.3 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.

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, так же 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 *Алгоритм консенсуса PoA*). 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.

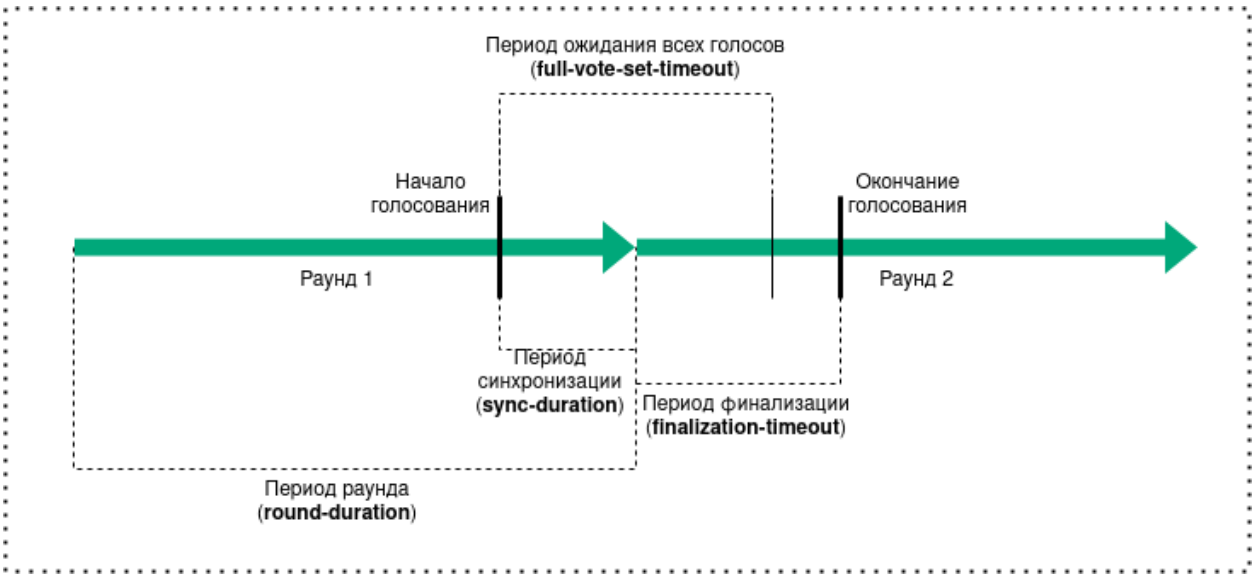
- **full-vote-set-timeout** – опциональный параметр, определяющий, сколько времени после окончания раунда (параметр конфигурационного файла ноды: `round-duration`) майнер ожидает полный набор голосов от всех валидаторов.

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-validators`).

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.

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 стейту не применен микроблок с набранными валидными голосами. При этом, если попытки майнинга выходят за временные рамки $t_{start} + t_{fin}$, 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.

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.

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

1.27 Криптография

Платформа Waves Enterprise предоставляет возможность выбора используемого криптографического алгоритма в зависимости от особенностей проекта. Доступны два типа криптографии: Waves и ГОСТ.

Примечание: ГОСТ криптография доступна только в *корпоративной* версии платформы, и не может быть использована в opensource версии платформы.

В таблице ниже представлены криптографические функции, используемые при выборе того или иного типа криптографии.

Таблица 3: **Используемые криптографические функции и алгоритмы**

Тип криптографии	Waves	ГОСТ
Функциональность		
Хэширование	Функциями **Blake2b256** и **Keccak256** последовательно	Функцией **Стрибог** в соответствии со стандартом **ГОСТ Р 34.11-2012** «Информационная технология. Криптографическая защита информации. Функция хэширования»
Электронная подпись	На базе эллиптической кривой **Curve25519** (ED25519 с ключами X25519)	В соответствии со стандартом **ГОСТ Р 34.10-2012** «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи»
Шифрование данных	Симметричное шифрование данных по стандарту **AES**	В соответствии со стандартом **ГОСТ Р 34.12-2015** «Информационная технология. Криптографическая защита информации. Блочные шифры» – симметричный алгоритм блочного шифрования **Kuznyechik**
Защита конфиденциальных данных	TLS v1.2 с криптонабором TLS_RSA_WITH_AES_256_CB(	TLS v1.2 для ГОСТ криптографии с криптонаборами: • TLS_CIPHER_2012; • TLS_CIPHER_2001; • TLS_GOSTR341112_256_WITH_KUZNYEC • TLS_GOSTR341112_256_WITH_MAGMA_ • TLS_CIPHER_2012_IANA.

1.27.1 Поддержка PKI

На платформе Waves Enterprise реализована инфраструктура открытых ключей (Public Key Infrastructure, PKI). Инфраструктура PKI используется только с ГОСТ криптографией.

Примечание: Инфраструктура PKI поддерживается только в *корпоративной* версии платформы, и не может быть использована в opensource версии платформы.

PKI имеет три режима функционирования:

- `отключен` – инфраструктура PKI отключена,
- `включен` – инфраструктура PKI включена. В этом случае
  - проверяется, что TLS включён на сетевом уровне, то есть параметр `node.network.tls` в файле **node.conf** имеет значение `true`;
  - ряд API методов, которые подразумевают работу с закрытым ключом на ноде, недоступны:
    * методы подписания транзакций через API ноды,
    * методы шифрования,
    * методы отправки конфиденциальных данных.
- `тестовый режим` – инфраструктура PKI функционирует в тестовом режиме. Доступны следующие API методы, которые подразумевают работу с закрытым ключом на ноде:
  - REST API методы:
    * методы подписания транзакций: *transactions/sign и transactions/signAndBroadcast*;
    * методы шифрования: *crypto/encryptCommon, crypto/encryptSeparate, crypto/decrypt*;
    * методы обмена конфиденциальными данными: */privacy/sendData, /privacy/sendDataV2 и /privacy/sendLargeData*;
    * методы подписания сообщений в блокчейне: *addresses/sign и addresses/signText*;
    * метод формирования электронной подписи данных */pki/sign*;
  - gRPC API методы:
    * методы обмена конфиденциальными данными: *PrivacyPublicService.SendData и PrivacyPublicService.SendLargeData*.

Режим PKI настраивается в разделе *crypto.pki.mode* конфигурационного файла ноды.

1.27.2 Хэширование

Как указано в таблице выше, операции хэширования выполняются функциями **Blake2b256** и **Keccak256** последовательно (для Waves криптографии), либо функцией **«Стрибог»** в соответствии с **ГОСТ Р 34.11-2012** «Информационная технология. Криптографическая защита информации. Функция хэширования» (для ГОСТ криптографии).

Размер блока выходных данных: **256 бит**.

1.27.3 Электронная подпись

Как указано в таблице выше, алгоритмы генерации ключей, формирования и проверки электронной подписи реализованы на базе эллиптической кривой **Curve25519** (ED25519 с ключами X25519) для Waves криптографии, либо в соответствии с **ГОСТ Р 34.10-2012** «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи» для ГОСТ криптографии.

Подробнее генерация и проверка электронной подписи с использованием API методов описаны в разделах *gRPC: проверка электронной подписи данных (PKI)* и *REST API: формирование и проверка электронной подписи данных (PKI)*.

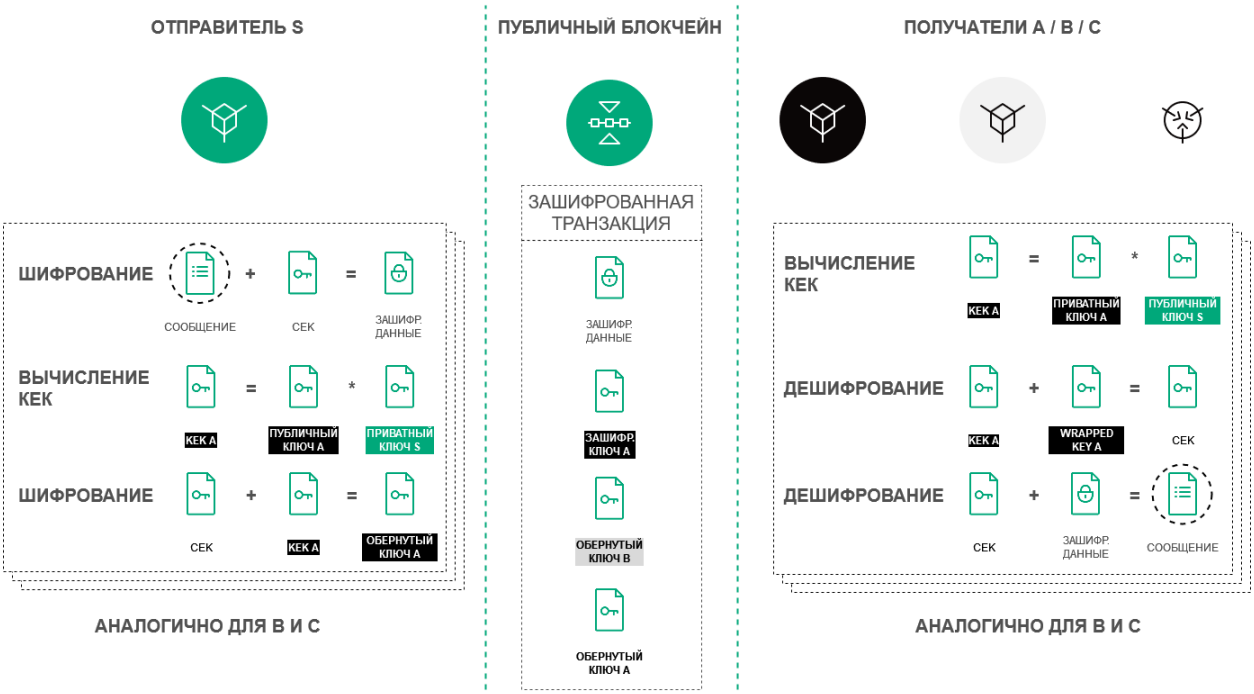
1.27.4 Защита конфиденциальных данных

Платформа Waves Enterprise предоставляет возможность использовать протокол TLS для защиты передаваемых между нодами данных. Поддерживаемые протоколы при использовании Waves и ГОСТ криптографии указаны в таблице выше.

Чтобы активировать TLS, необходимо в конфигурационном файле ноды **node.conf** задать параметру `node.network.tls` значение `true`.

Если протокол TLS не используется для создания соединений между нодами (параметру `node.network.tls` присвоено значение `false`), то для защиты передаваемых конфиденциальных данных (privacy) используется TLS-подобная схема сквозного шифрования (end-to-end encryption) при помощи сессионных ключей на базе **протокола Диффи-Хеллмана**. Такая защита будет применена только к конфиденциальным данным при их передаче между нодами peer-to-peer, то есть между двумя участниками сети.

Ниже приведено схематичное описание процедуры шифрования текстовых данных на базе протокола Диффи-Хеллмана:



Примечание: Платформа также использует протокол TLS при работе со смарт-контрактами для следующих соединений:

- соединение с Docker-хостом (Docker-TLS);
- соединение от смарт-контракта к ноде по gRPC и REST API.

Настройка и использование TLS в этих случаях описаны в разделе *Общая настройка платформы: настройка исполнения смарт-контрактов*.

Смотрите также

Общая настройка платформы: настройка криптографии

Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды

Тонкая настройка платформы: настройка TLS

REST API: реализация методов шифрования

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

contract_pki_service.proto

1.28 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*.

1.28.1 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*.

banned

Роль `banned` временно или постоянно ограничивает отправку транзакций от этого участника. Адрес с ролью `banned` попадает в черный список нод (**blacklist**) – список адресов, от которых не принимаются транзакции.

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*.

contract_validator

Участник с ролью `contract_validator` имеет право на валидацию обновляемых и загружаемых смарт-контрактов.

Подробнее о применении этой роли: *Валидация смарт-контрактов*.

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*.

1.28.2 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

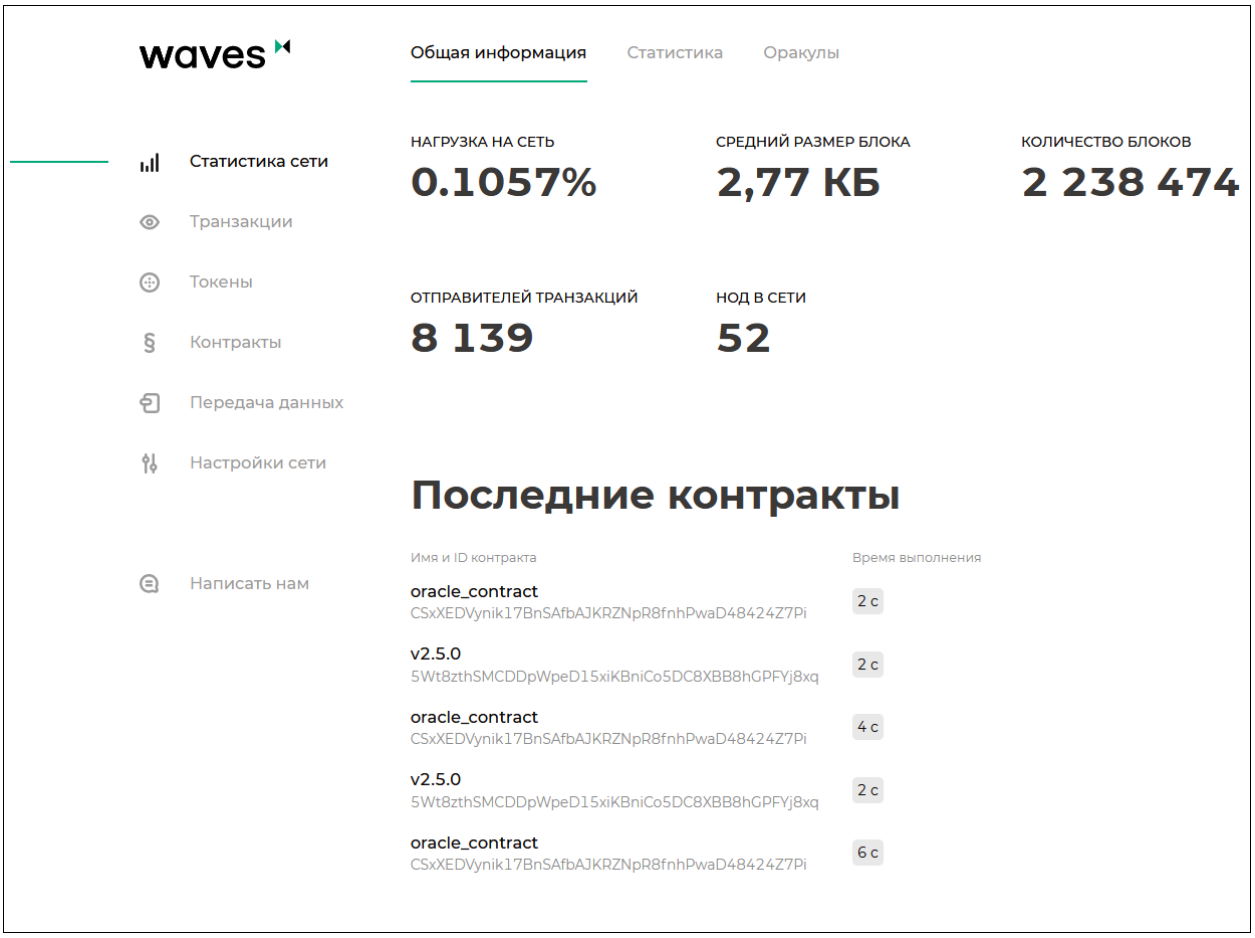
Описание транзакций

1.29 Клиент

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;


waves
Общая информация
Статистика
Оракулы
Статистика сети
Транзакции
Токены
Контракты
Передача данных
Настройки сети
Написать нам
НАГРУЗКА НА СЕТЬ
0.1057%
СРЕДНИЙ РАЗМЕР БЛОКА
2,77 КБ
КОЛИЧЕСТВО БЛОКОВ
2 238 474
ОТПРАВИТЕЛЕЙ ТРАНЗАКЦИЙ
8 139
НОД В СЕТИ
52
Последние контракты
Имя и ID контракта
Время выполнения
oracle_contract
CSxXEDVynik17BnSAfbAJKRZNpR8fnhPwaD48424Z7Pi
2 с
v2.5.0
5Wt8zthSMCDDpWpeD15xiKBniCo5DC8XBB8hGPFYj8xq
2 с
oracle_contract
CSxXEDVynik17BnSAfbAJKRZNpR8fnhPwaD48424Z7Pi
4 с
v2.5.0
5Wt8zthSMCDDpWpeD15xiKBniCo5DC8XBB8hGPFYj8xq
2 с
oracle_contract
CSxXEDVynik17BnSAfbAJKRZNpR8fnhPwaD48424Z7Pi
6 с

- *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*.

Работа с **Ledger Nano** описана в разделе

1.29.1 Использование Ledger Nano с клиентом блокчейн-платформы Waves Enterprise

Введение

Ledger Nano – это аппаратный кошелек для хранения цифровых активов. Ledger Nano использует автономный метод генерации приватных ключей (холодное хранение), поэтому он считается одним из самых надежных способов хранения цифровых активов и многие пользователи криптовалют выбирают именно его. Ниже описаны настройки, необходимые для использования Ledger Nano с клиентом Waves Enterprise. Клиент Waves Enterprise позволяет переводить токены с помощью устройства Ledger Nano.

Предварительные условия использования Ledger Nano

1. Вы инициализировали ваше устройство **Ledger Nano**.

   Примечание: Waves Enterprise поддерживает работу с моделями Ledger Nano S, Ledger Nano S+ и Ledger Nano X.

   ---

2. Установлена последняя версия прошивки.
3. Ledger Live готов к использованию.
4. Установлены браузеры Google Chrome или Firefox.

Установка приложения Waves Enterprise на вашем устройстве Ledger Nano

1. Скачайте и запустите Ledger Live, откройте Manager.
2. Подключите и разблокируйте ваше устройство Ledger Nano.
3. При необходимости разрешите синхронизацию с Ledger Live, нажав правую кнопку на устройстве.
4. В каталоге Ledger Live, найдите приложение Waves Enterprise и нажмите кнопку `Install`.

Примечание: Для установки приложения Waves Enterprise требуется около 40 кБ. Точный размер приложения указан в каталоге Ledger Live.

Появится окно установки, и на вашем устройстве Ledger Nano отобразится сообщение `Processing...`, после чего установка приложения будет завершена.

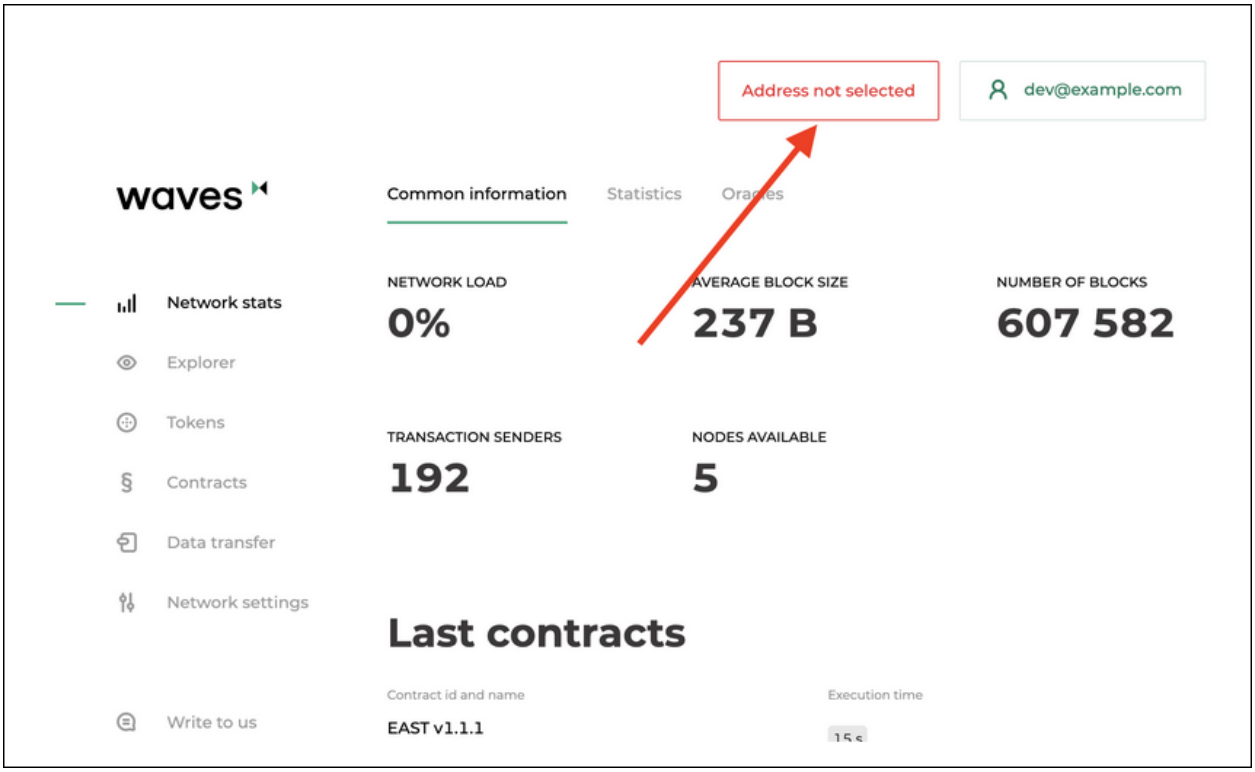
Запуск приложения Waves Enterprise на вашем устройстве Ledger Nano

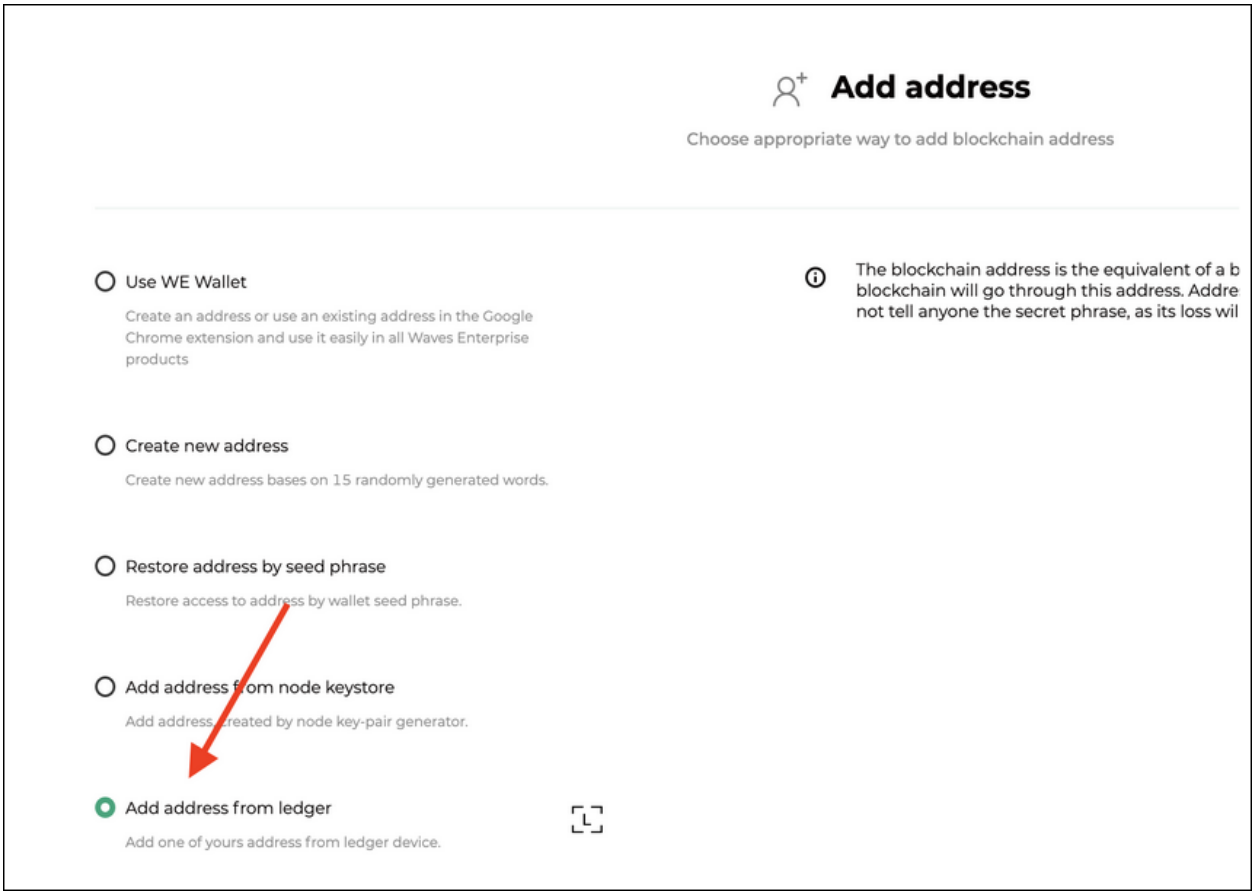
1. После установки приложения Waves Enterprise, используйте левую или правую кнопку, чтобы найти его в меню.
2. Для запуска приложения нажмите обе кнопки одновременно.

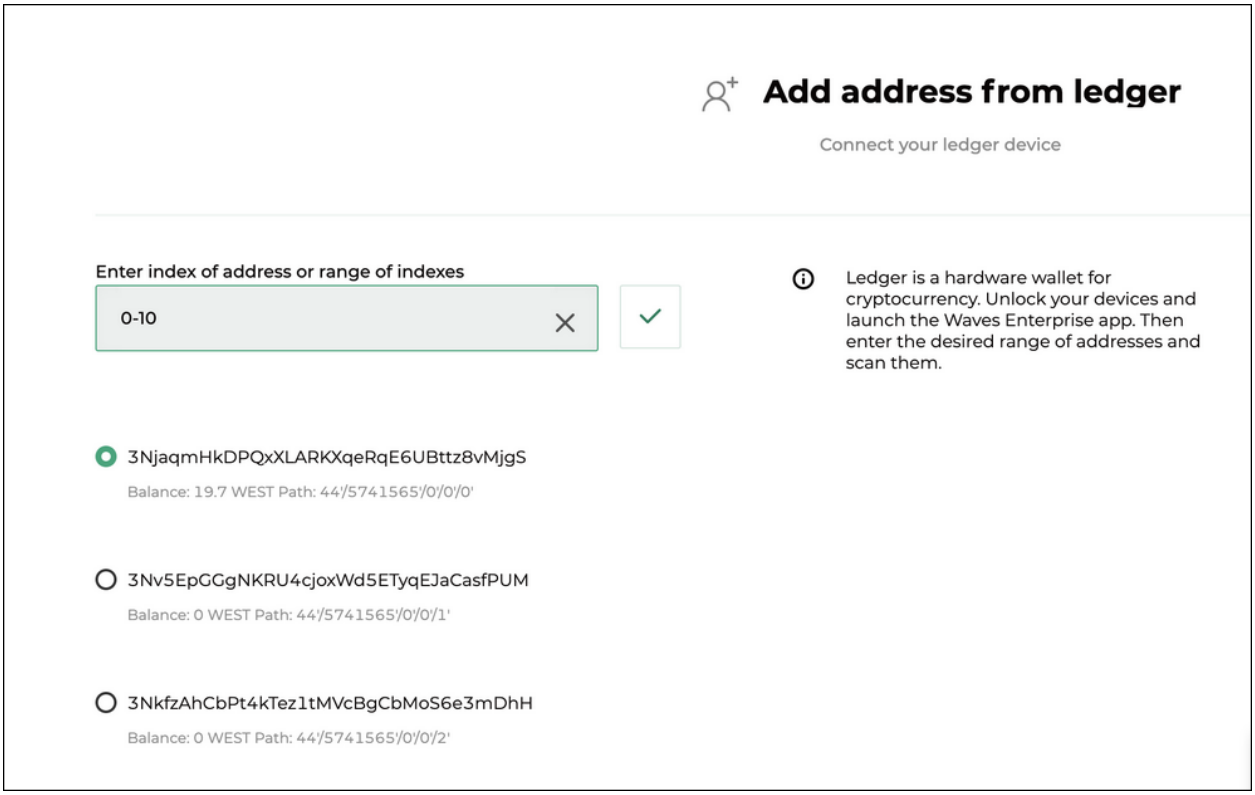
Использование устройства Ledger Nano с приложением Waves Enterprise

1. Убедитесь, что ваше устройство Ledger Nano подключено и разблокировано. Другие приложения для криптовалют не должны быть запущены и не должны перехватывать соединение между Ledger Nano и приложением Waves Enterprise.
2. Откройте Клиент Waves Enterprise в браузере Google Chrome или Firefox.
3. Войдите в свой аккаунт и нажмите левую кнопку в верхнем меню, чтобы выбрать или добавить адрес.

1. Нажмите кнопку `Add address`.
2. Затем выберите `Add address from ledger`.
3. На следующей странице введите идентификатор адреса или диапазон идентификаторов, затем нажмите кнопку `Submit`.
4. Выберите нужный адрес, задайте ему имя и используйте как текущий адрес.


Address not selected
dev@example.com
waves
Common information
Statistics
Network stats
Explorer
Tokens
Contracts
Data transfer
Network settings
Write to us
NETWORK LOAD
0%
AVERAGE BLOCK SIZE
237 B
NUMBER OF BLOCKS
607 582
TRANSACTION SENDERS
192
NODES AVAILABLE
5
Last contracts
Contract id and name
Execution time
EAST v1.1.1


Add address
Choose appropriate way to add blockchain address
Use WE Wallet
Create an address or use an existing address in the Google Chrome extension and use it easily in all Waves Enterprise products
Create new address
Create new address bases on 15 randomly generated words.
Restore address by seed phrase
Restore access to address by wallet seed phrase.
Add address from node keystore
Add address created by node key-pair generator.
Add address from ledger
Add one of yours address from ledger device.
The blockchain address is the equivalent of a b
blockchain will go through this address. Addre
not tell anyone the secret phrase, as its loss wil



Перевод токенов

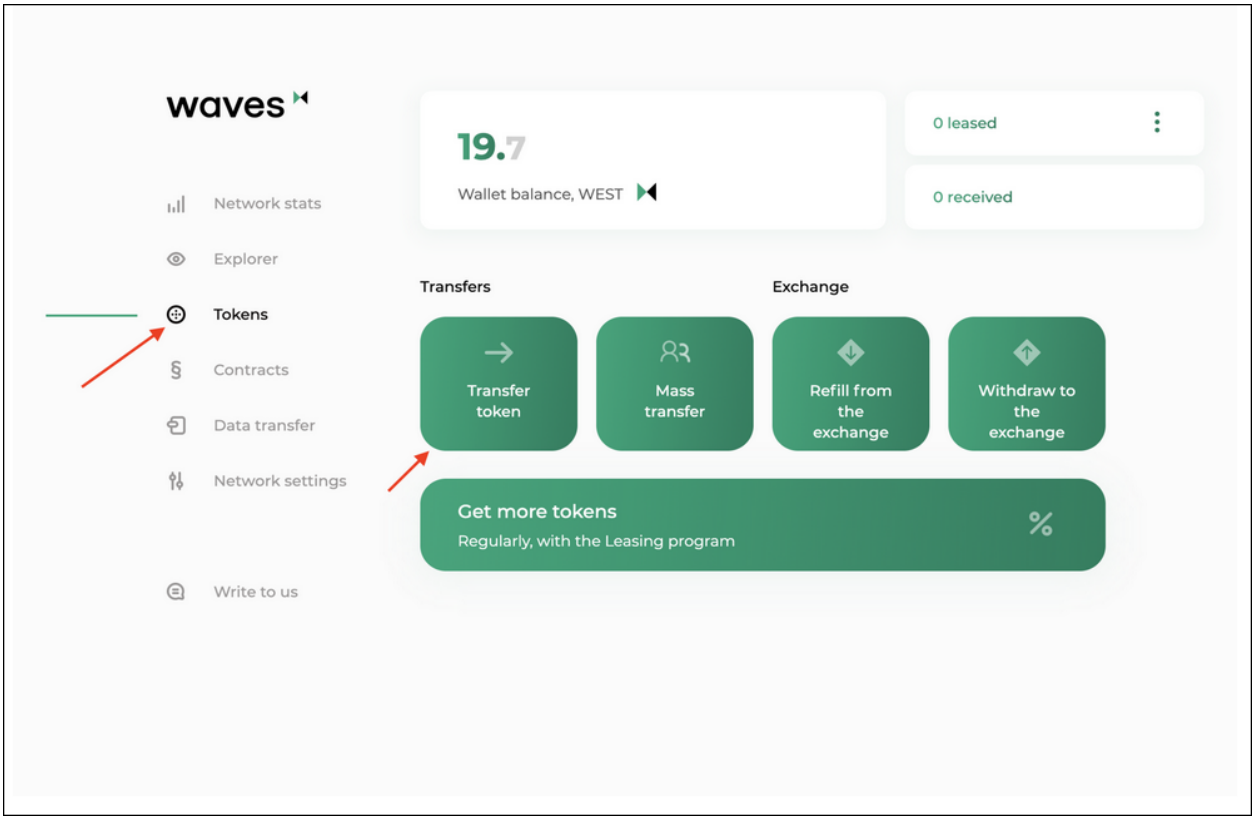
На настоящий момент поддерживаются только транзакции перевода токенов.

Для перевода токенов выполните следующие шаги:

1. В клиенте Waves Enterprise перейдите на вкладку `Токены` (`Tokens`) и нажмите кнопку `Перевести токен` (`Transfer token`).
2. На следующей странице введите адрес получателя, количество токенов и описание перевода.
3. Проверьте данные в вашем Ledger Nano и подпишите транзакцию.

Примечание: Если вы открываете клиент Waves Enterprise на новой машине или в новом браузере, то необходимо валидировать его на вашем устройстве Ledger Nano.

Если вам необходима помощь в настройке работы с Ledger Nano, свяжитесь с нами.



Смотрите также

Клиент

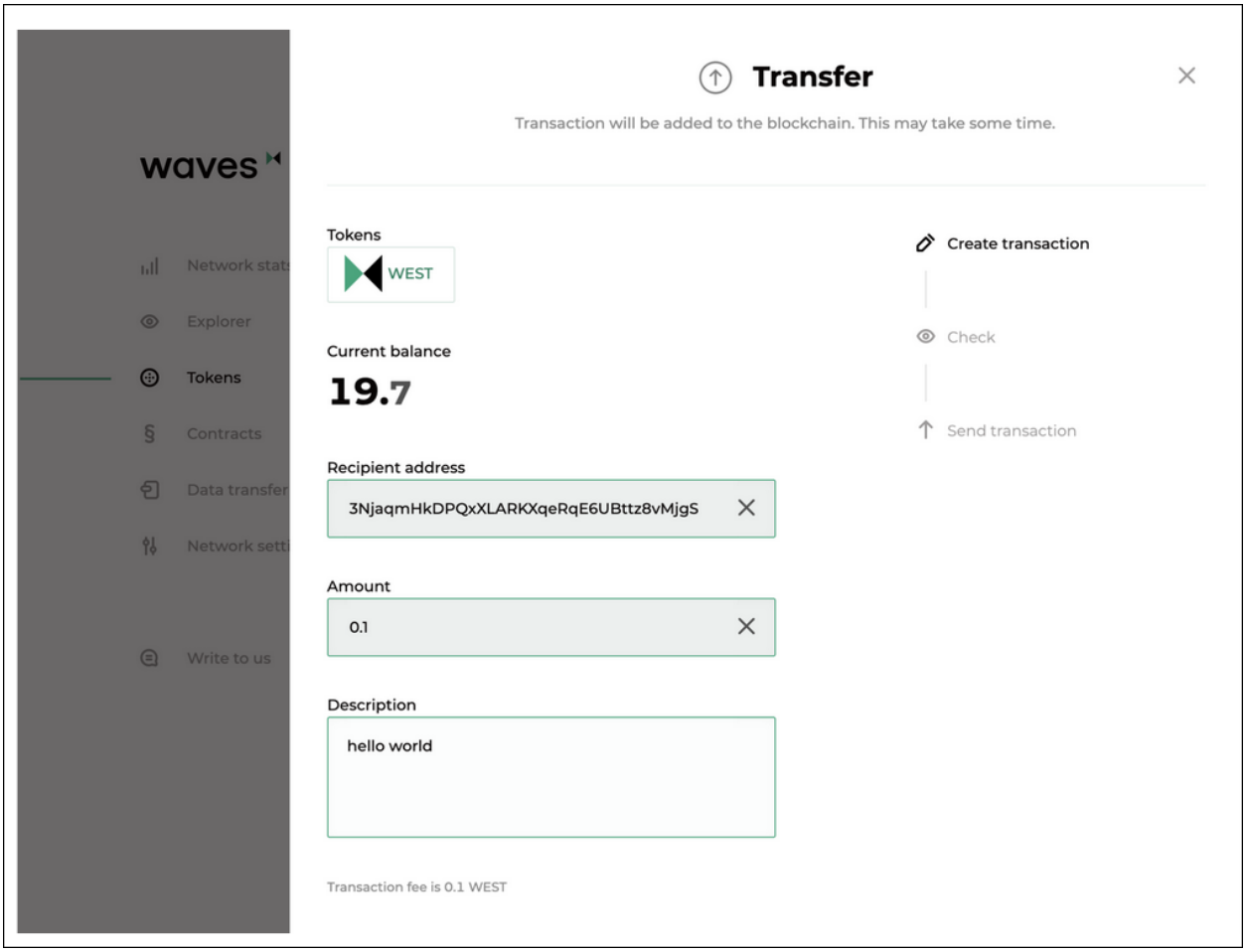
1.29.2 Статистика сети

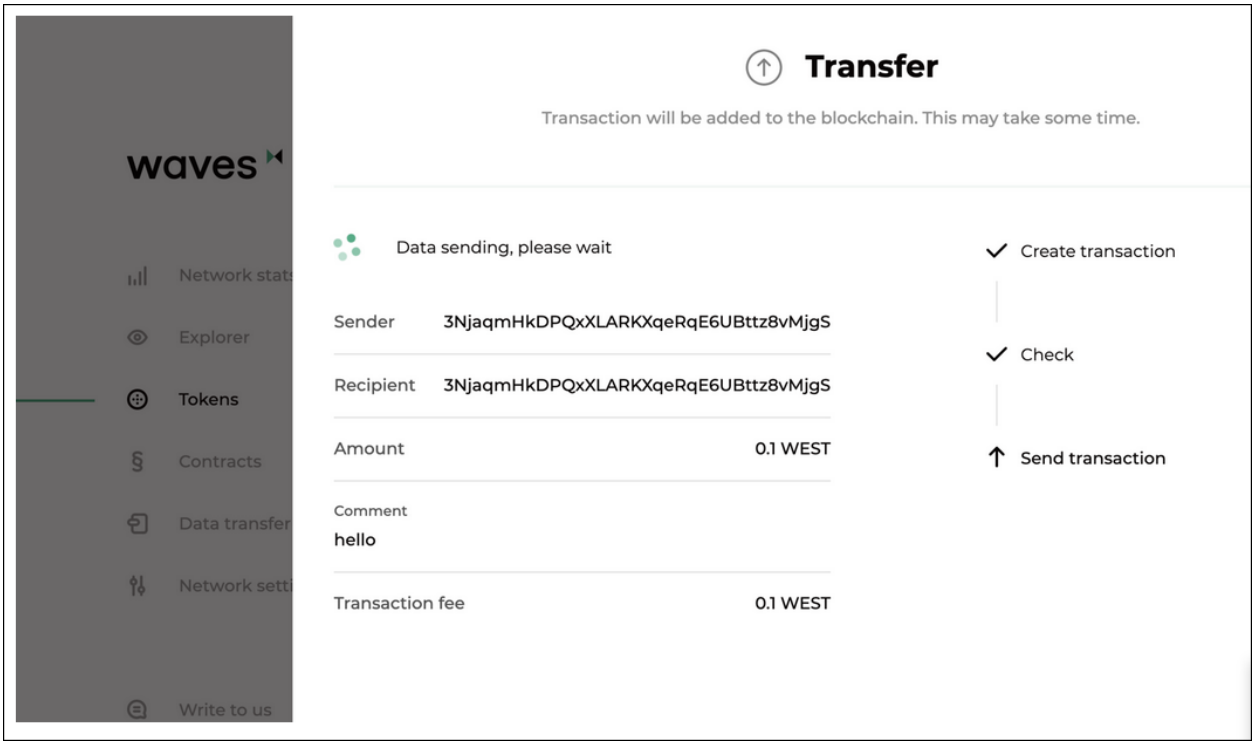
На вкладке **Общая информация** раздела «Статистика сети» представлено текущее состояние Waves Enterprise Mainnet:

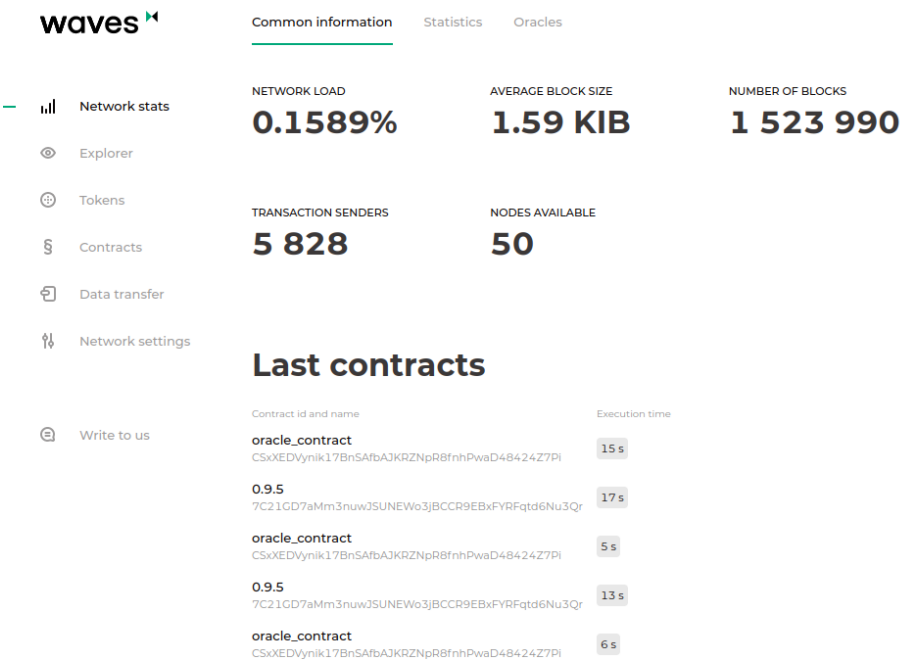
- нагрузка на сеть;
- средний размер одного блока;
- общее количество блоков в сети;
- количество нод и отправителей транзакций;
- последние вызванные смарт-контракты.

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;


waves
Network stats
Explorer
Tokens
Contracts
Data transfer
Network setti
Write to us
Transfer
Transaction will be added to the blockchain. This may take some time.
Tokens
WEST
Current balance
19.7
Recipient address
3NjaqmHkDPQxXLARKXqeRqE6UBttz8vMjgS
Amount
0.1
Description
hello world
Transaction fee is 0.1 WEST
Create transaction
Check
Send transaction


waves
Network stats
Explorer
Tokens
Contracts
Data transfer
Network setti
Write to us
Transfer
Transaction will be added to the blockchain. This may take some time.
Data sending, please wait
Sender 3NjaqmHkDPQxXLARKXqeRqE6UBttz8vMjgS
Recipient 3NjaqmHkDPQxXLARKXqeRqE6UBttz8vMjgS
Amount 0.1 WEST
Comment
hello
Transaction fee 0.1 WEST
Create transaction
Check
Send transaction



- 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.

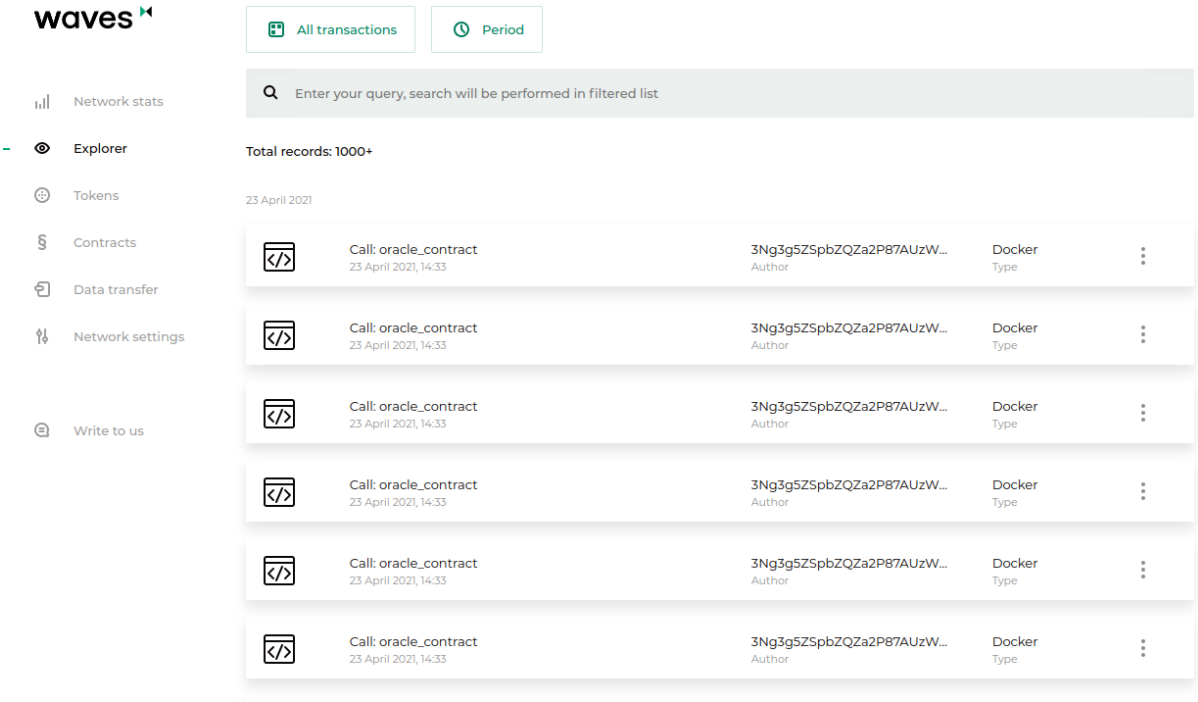
Относительный график отображает зависимость колебаний стоимости WEST и традиционных активов по следующим парам:

- WEST - USDN;
- BTC - USD;
- BRENT - USD;
- Золото - USD;

График стоимости WEST отображает стоимость WEST в других криптовалютах:

- WEST - USDN;
- WEST - WAVES;
- WEST - BTC.

1.29.3 Транзакции



Раздел «Транзакции» содержит информацию о транзакциях в блокчейне. Для поиска доступна фильтрация по периоду публикации, а также по следующим категориям:

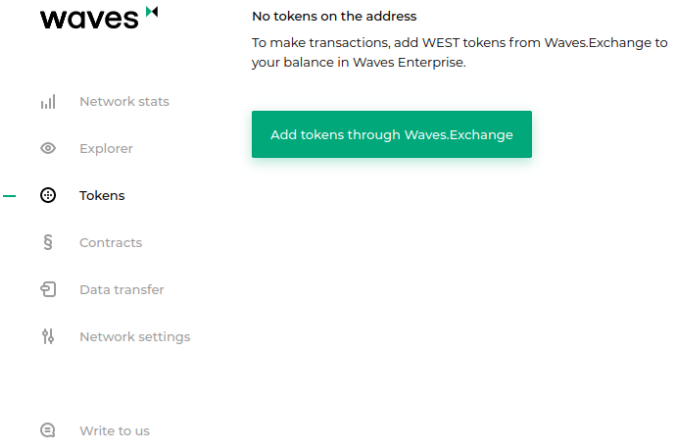
- по участникам;
- по транзакциям с данными;
- по идентификаторам транзакций;
- по именам смарт-контрактов;
- по подписям транзакций;
- по номеру блока, содержащего транзакции.

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.

1.29.4 Токены



При отсутствии токенов на вашем адресе, в разделе «Токены» 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*.

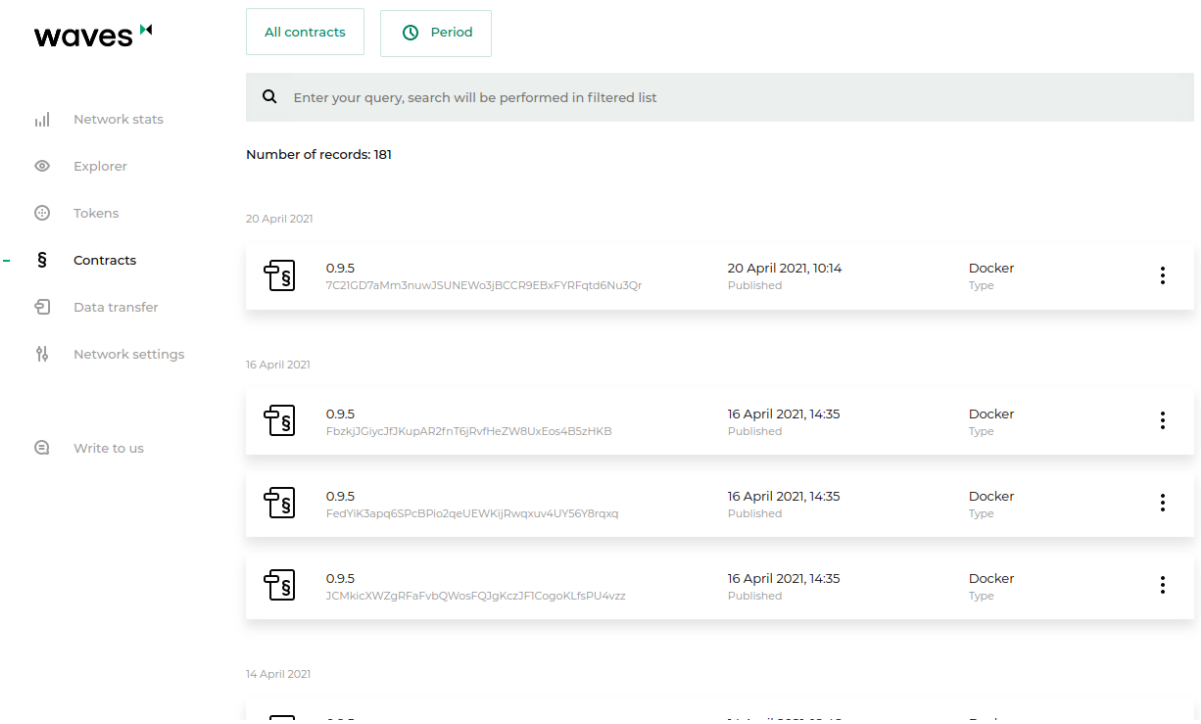
1.29.5 Контракты

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*.



При выборе контракта открывается его карточка.

В карточке каждого смарт-контракта вы увидите следующие 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;
- **Обновление** – информация о последнем обновлении контракта;
- **История версий** – таблица с именами образов, датами создания и контрольными суммами для каждой версии смарт-контракта.

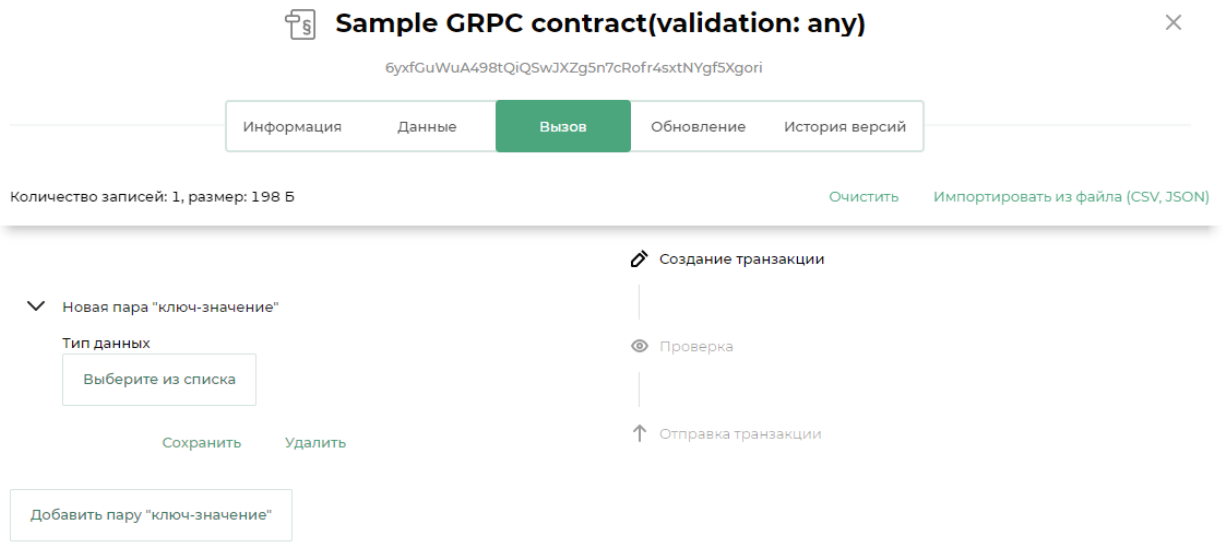
Вызов контракта

Клиент позволяет загрузить параметры для следующих транзакций с помощью csv или json:

- *CallContract Transaction*;
- *Data Transaction*,

Для загрузки параметров на вкладке **Вызов** нажмите ссылку **Импортировать из файла (CSV, JSON)**, затем загрузите файл. Файл json должен представлять собой массив объектов, каждый из которых имеет следующие ключи:

- `value` – значение;
- `key` – строка, название ключа;
- `type` – тип; может принимать одно из следующих значений:



- `integer`;
- `string`;
- `boolean`;
- `binary (base64)`.

Подробнее о смарт-контрактах блокчейн-платформы Waves Enterprise см. статью *Смарт-контракты*.

1.29.6 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.

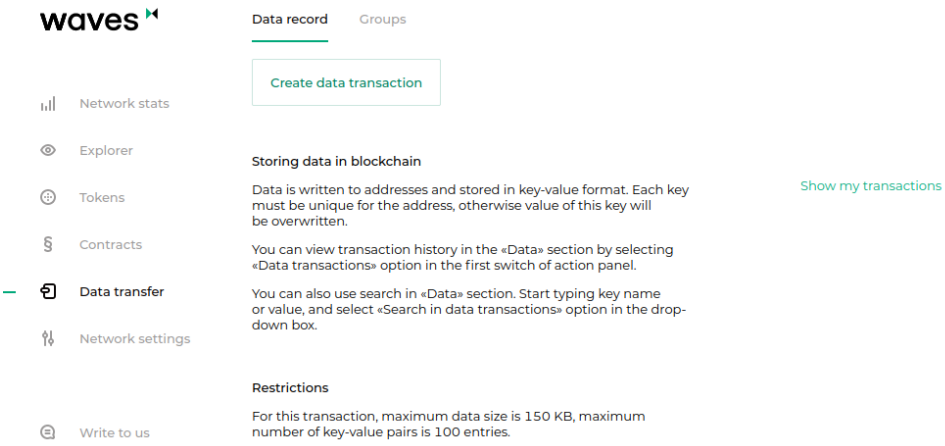
На вкладке **Группы** вы можете создавать и редактировать группы доступа к конфиденциальным данным и отправлять в них транзакции с данными. Также на вкладке отображается информация о группах доступа, в которых вы состоите.

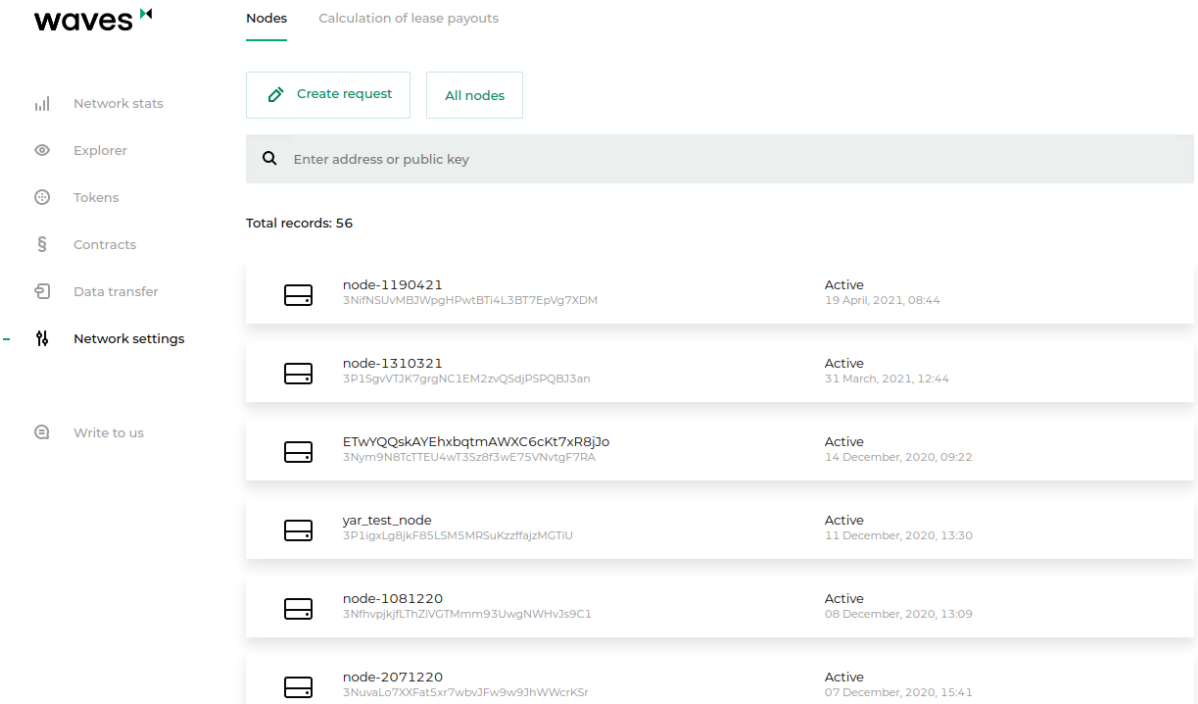
1.29.7 Настройки сети

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;


waves
Network stats
Explorer
Tokens
Contracts
Data transfer
Network settings
Write to us
Data record
Groups
Create data transaction
Storing data in blockchain
Data is written to addresses and stored in key-value format. Each key must be unique for the address, otherwise value of this key will be overwritten.
You can view transaction history in the «Data» section by selecting «Data transactions» option in the first switch of action panel.
You can also use search in «Data» section. Start typing key name or value, and select «Search in data transactions» option in the drop-down box.
Show my transactions
Restrictions
For this transaction, maximum data size is 150 KB, maximum number of key-value pairs is 100 entries.


waves
Network stats
Explorer
Tokens
Contracts
Data transfer
Network settings
Write to us
Nodes
Calculation of lease payouts
Create request
All nodes
Enter address or public key
Total records: 56
node-1190421
3NifNSUvMBJWpgHPwtBTi4L3BT7EpVg7XDM
Active
19 April, 2021, 08:44
node-1310321
3P1SgvVTJK7grgNC1EM2zvQSdjPSPQBJ3an
Active
31 March, 2021, 12:44
ETwYQQskAYEhxbqtmAWXC6cKt7xR8jJo
3Nym9N8TcTTEU4wT3Sz8f3wE75VNvtgF7RA
Active
14 December, 2020, 09:22
yar_test_node
3P1igxLg8jkF85LSM5MRSuKzzffajzMGTiU
Active
11 December, 2020, 13:30
node-1081220
3NfhvpJkjfLThZiVGTMmm93UwgNWHvJs9C1
Active
08 December, 2020, 13:09
node-2071220
3NuvaLo7XXFat5xr7wbvJFw9w9JhWWcrKSr
Active
07 December, 2020, 15:41

- 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.

1.29.8 Написать нам

В разделе «Написать нам» вы можете оставить любой комментарий или сообщение для службы технической поддержки Waves Enterprise.

Смотрите также

Привязка блокчейн-платформы к клиенту

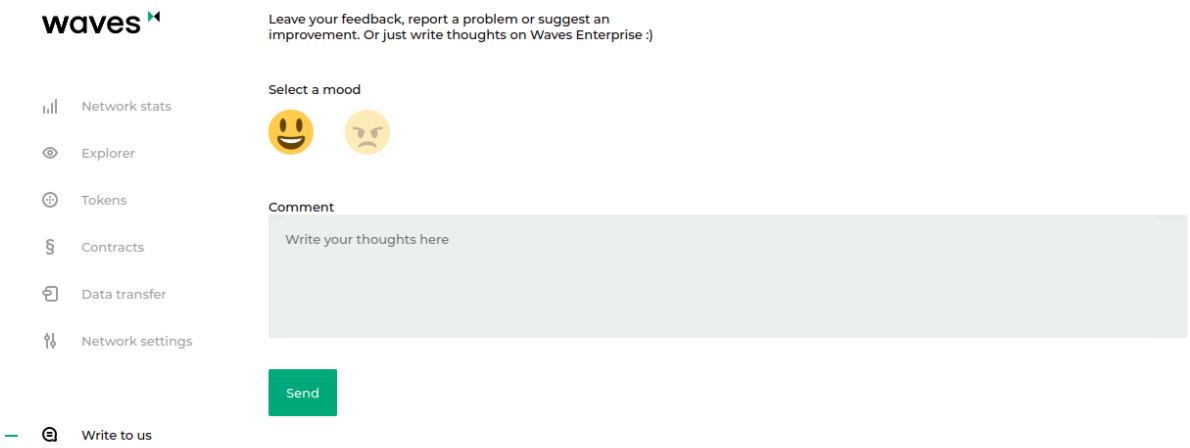
1.30 Генераторы

Генераторы – это набор утилит, входящий в комплект поставки блокчейн-платформы Waves Enterprise. Генераторы поставляются 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;

1.30.1 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.

Пример конфигурационного файла accounts.conf:

```
accounts-generator {
  crypto {
    type = WAVES
    pki {
      mode = OFF
      required-oids = []
    }
  }
  chain-id = T
  amount = 5
  wallet = ${user.home}"/node/wallet/wallet1.dat"
  wallet-password = "some string as password"
  reload-node-wallet {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    enabled = false
    url = "http://localhost:6869/utils/reload-wallet"
  }
}
```

Запуск **AccountsGeneratorApp**:

```
java -jar generator-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*.

1.30.2 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 generator-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*.

1.30.3 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 generator-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.

Пример файла api-key-hash.conf:

```
apikeyhash-generator {
  crypto {
    type = GOST
    pki {
        mode = ON
        required-oids = ["1.2.3.4.5.6.7.8.9.10.11"]
    }
  }
  api-key = "some string for api-key"
}
```

Подробнее об авторизации gRPC и REST API см. раздел *Тонкая настройка платформы: настройка авторизации для gRPC и REST API*.

Смотрите также

Архитектура

1.31 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.

1.31.1 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:

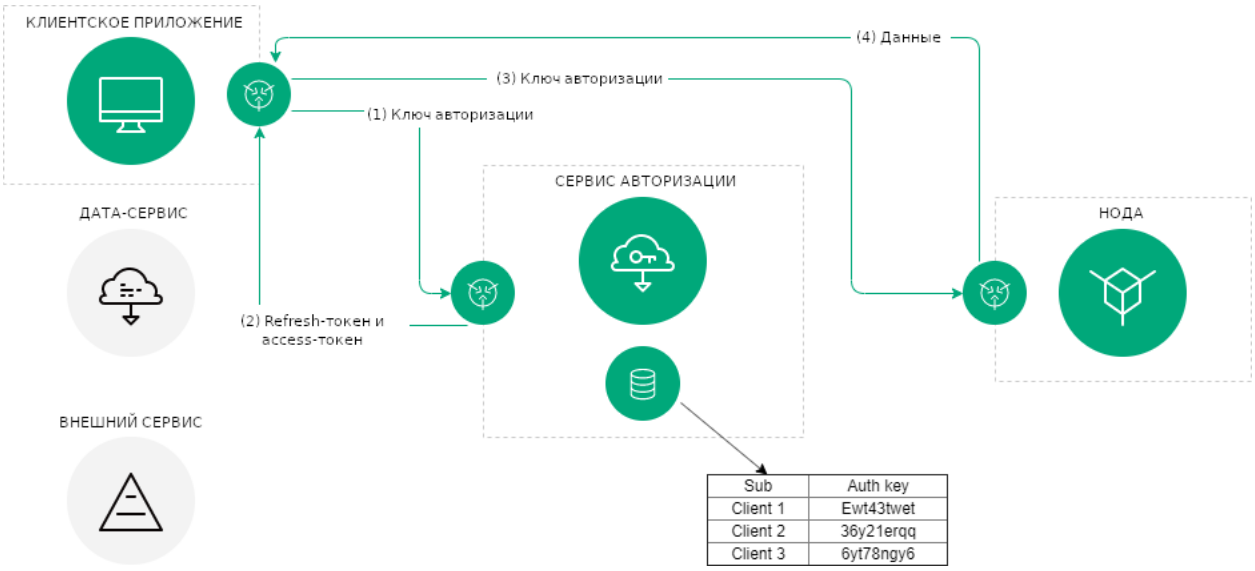
Средством авторизации является JSON Web Token (JWT). Токены используются для авторизации каждого запроса от клиента к серверу и имеют ограниченное время жизни. Клиент получает два токена – **access** и **refresh**. Access-токен используется для авторизации запросов на доступ к защищенным ресурсам и для 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

Клиент
Запрос авторизации
Предоставление авторизации
Предоставление авторизации
Access-токен и refresh-токен
Access-токен
Защищенный ресурс
Access-токен
Ошибка access-токена
Ресурсный сервер
Refresh-токен
Новый access-токен и обновленный refresh-токен
Сервер авторизации


КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ
(4) Данные
(3) Ключ авторизации
(1) Ключ авторизации
СЕРВИС АВТОРИЗАЦИИ
НОДА
ДАТА-СЕРВИС
(2) Refresh-токен и access-токен
ВНЕШНИЙ СЕРВИС
Sub
Auth key
Client 1
Ewt43twet
Client 2
36y21erqq
Client 3
6yt78ngy6

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.

1.31.2 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, которые описаны в разделе Сервис подготовки данных: ручная настройка.

1.31.3 API-методы интеграционных сервисов

Для обмена данными интеграционным сервисам доступны отдельные методы REST API:

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 отправки 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-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA7d9Oj/ZQTkkjf4UuMfUu
QIFDTYxYf6QBKMVJnq/wXyPYYkV8HVFYFizCaEciv3CXmBH77sXnuTlrEtvK7zHB
KvV870HmZuazjIgZVSkOnOY7F8UUVNXnlzVD1dPsOGJ6orM41DnC1W65mCrP3bjn
fV4RbmykN/lk7McA6EsMcLEGbKkFhmeq2Nk4hn2CQvoTkupJUnOCP1dhO4bq1lQ7
Ffj9K/FJq73wSXDoH+qqdRG9sfrtgrhtJHerruhv3456e0zyAcDO8+sJUQFKY80B
SZMEndVzFS2ub9Q8e7BfcNxTmQPM4PhHO5wuTqL32qt3uJBx2OI4lu3OND44ZrDJ
BbVog73oPjRYXj+kTbwUZI66SP4aLcQ8sypQyLwqKk5DtLRozSNOOIrupJJ/pwZs
9zPEggL91T0rirbEhGlf5U8/6XN8GVXX4iMk2fD8FHLFJuXCD7Oj4JC2iWfFDC6a
uUkwUfqfjJB8BzIHkncoqOZbpidEE2lTWl+svuEu/wyP5rNlyMiE/e/fZQqM2+o0
cH5Qow6HH35BrloCSZciutUcd1U7YPqESJ5tryy1xn9bsMb+On1ocZTtvec/ow4M
RmnJwmOj1nd+cc19OKLG5/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

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 (ассетах).

GET /assets

Метод предназначен для получения списка доступных в блокчейне ассетов. 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/info/{contractId}

Метод возвращает информацию о смарт-контракте по его идентификатору `{contractId}`.

Пример ответа:

GET /contracts/id/{id}:

```
{
  "creator": "9yxe6Kw9eiCD2mTNvKdrcQ1EoQqzMy7p52USZftBtQhp",
  "contractId": "7zcrHAFZmcZ3EGs7JWL5jCrbizCppu2rcpDDuChNTF6K",
  "image": "registry.wvservices.com/waves-enterprise-public/east-contract:v1.2",
  "imageHash": "baef03e82e4ecc723b85876111cbe25ed390ad7c62169e8a3ba142b6a2ad3000",
  "version": 5,
  "active": true,
  "validationPolicy": {
     "type": "majority_with_one_of",
     "addresses": [
       "3NyJPnLBdEQiPdHoHHgQAYX6UVj6GKMxgMx",
       "3NmHrYoC8S2SUosy6UJp47bBwq2Cr2X6Yq1",
       "3NrKDuHjUG7vSCiMMD259msBKcPRm4MvaJu"
     ]
  },
  "apiVersion": "1.0"
}
```

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"
      }
```


(continues on next page)

(продолжение с предыдущей страницы)

```
    ]
  }
]
```

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"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  ]
}
```

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,
    "key": "string",
    "value": "string",
    "position_in_tx": 0
  }
]
```


GET /api/v1/txData/{key}/{value}

Метод 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
    }
  ],
  "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,
          "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"
```


(continues on next page)

(продолжение с предыдущей страницы)

```
    }
  ]
}
```

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",
      "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

1.32 Различия opensource и коммерческой версий блокчейн-платформы Waves Enterprise

Блокчейн платформа Waves Enterprise существует в коммерческой версии и в версии с открытым исходным кодом (opensource).

Коммерческая версия блокчейн-платформы Waves Enterprise ориентирована на использование в корпоративном и государственном секторах и распространяется при помощи *пользовательских лицензий*. Для приобретения коммерческой версии платформы Waves Enterprise свяжитесь с отделом продаж Waves Enterprise по электронной почте: sales@wavesenterprise.com.

Релиз opensource версии Waves Enterprise, распространяемой по лицензии Apache 2.0, доступен в официальном репозитории Waves Enterprise в GitHub. На opensource версию не распространяется *ограничение на высоту блокчейна в 30000 блоков*.

Таблица 4: Различия opensource и коммерческой версий платформы Waves Enterprise

Функциональность	Opensource версия	Коммерческая версия
Контейнеризированные Смарт-Контракты	✓	✓
Обмен конфиденциальными данными	✓	✓
Алгоритмы консенсуса: LPoS, PoA, CFT	✓	✓
Анкоринг	✓	✓
Криптография:		
• Waves (Curve25519, Blake2b256 и Keccak256)	✓	✓
• ГОСТ	✗	✓
Поддержка TLS	✗	✓
Поддержка PKI	✗	✓

Соответственно, отличаются *конфигурационные файлы ноды* для opensource и коммерческой версий платформы Waves Enterprise. Следующие разделы конфигурационного файла ноды недоступны в opensource версии:

- *node.tls*
- node.network.tls
- *node.api.rest.tls*
- *node.api.grpc.tls*
- *node.docker-engine.docker-tls*
- node.license
- *настройки ГОСТ криптографии*

Смотрите также

Лицензии блокчейн-платформы Waves Enterprise

1.33 Внешние компоненты платформы

Таблица 5: Список проприетарных компонентов

Название	Версия	Лицензия	Тип лицензии	Ссылка на лицензию	Компонент архитектуры
CryptoPro CSP, включая CryptoPro JCSP	5.0 R2	Лицензия ООО «КРИПТО-ПРО»	Proprietar	https://www.cryptopro.ru/download?pid=1417	Нода

Таблица 6: Список open-source компонентов

Название	Версия	Лицензия	Тип лицензии	Ссылка на лицензию	Компонент архитектуры
postgres	13.x	PostgreSQL License	Freeware, opensource	https://github.com/postgres/postgres/blob/master/COPYRIGHT	Дата-краулер
nodejs	12.21	MIT License	Freeware, opensource	https://raw.githubusercontent.com/nodejs/node/master/LICENSE	Дата-краулер, дата-сервис, клиент
npm	6.14.›	The Artistic License 2.0	Freeware, opensource	https://github.com/npm/cli/blob/latest/LICENSE	Дата-краулер, дата-сервис, клиент
netty	4.1.x	Apache License 2.0	Freeware, opensource	https://github.com/netty/netty/blob/4.1/LICENSE.txt	Нода
rocksdb	6.13.›	Apache License 2.0	Freeware, opensource	https://github.com/facebook/rocksdb/blob/master/LICENSE.Apache	Нода
docker-java	3.2.x	Apache License 2.0	Freeware, opensource	https://github.com/docker-java/docker-java/blob/master/LICENSE	Нода
akka (http, grpc)	10.1.›	Apache License 2.0	Freeware, opensource	https://github.com/akka/akka/blob/master/LICENSE	Нода
swagger-ui	3.23.›	Apache License 2.0	Freeware, opensource	https://github.com/swagger-api/swagger-ui/blob/master/LICENSE	Нода
nginx	1.18.›	BSD License	Freeware, opensource	https://nginx.org/LICENSE	Клиент

1.34 Официальные ресурсы и контакты

1.34.1 Официальные ресурсы блокчейн-платформы

- Официальный сайт блокчейн-платформы Waves Enterprise
- Страница проекта в Github
- Официальный сайт блокчейн-платформы Waves

1.34.2 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

1.35 Словарь терминов

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

Ассет
: Цифровой актив в блокчейне. Представляет собой набор токенов

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 стейту блокчейна. Количество транзакций в микроблоке ограничено 500 единицами. Микроблоки формируют блок сети. Микроблоки возникают исключительно под нагрузкой: если нет транзакций, то выпускаются только блоки.

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 транзакциям 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 (снепшот)
Набор всех данных блокчейна по аккаунтам, смарт-контрактам, группам доступа к конфиденциальным данным, ролям и зарегистрированным нодам, актуальный на момент снятия этого набора. Снимок данных не содержит истории изменения значений, транзакций и блоков.

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, использующаяся для мотивации участников к майнингу в сети.

   При использовании платформы *с подключением к сети Mainnet* используется системный токен WEST. Помимо системного токена, вы можете создать и использовать *другие токены*.

   В отличии от блокчейн платформ, где необходимо публиковать смарт-контакт стандарта *ERC-20* для создания нового токена, сеть Waves Enterprise предоставляет нативную возможность выпуска токенов при помощи *транзакции выпуска токена*.

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

УКЭП
Усиленная квалифицированная электронная подпись, созданная на базе инфраструктуры открытых ключей (PKI). УКЭП выдает аккредитованный удостоверяющий центр (УЦ). Срок действия УКЭП как правило ограничен одним годом

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

Эндпоинт (эндпойнт, Endpoint) сервиса
: http или https адрес, по которому обращается HTTP метод. Эндпоинт выполняют конкретную задачу, принимает параметры и возвращает данные.

API-метод
: Отдельная процедура, вызываемая участником при помощи API-интерфейса блокчейн-платформы (gRPC или REST API) и предназначенная для выполнения определенной операции в блокчейне

CEK
: Content Encryption Key – ключ шифрования данных. Используется для шифрования текстовых данных

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

KEK
: Key Encryption Key – ключ шифрования ключа. Используется для шифрования ключа шифрования данных (CEK)

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

MVCC (Multiversion concurrency control)
: Механизм управления параллельным доступом к состоянию смарт-контрактов посредством многоверсионности. Благодаря этому механизму нода поддерживает возможность параллельно выполнять несколько транзакций любых смарт-контрактов, при этом гарантируется согласованность данных.

JWT-токен (JSON Web Token)
: Объект в формате JSON, применяющийся для авторизации участника блокчейна по протоколу oAuth

PKI
: Public Key Infrastructure – инфраструктура открытых ключей, в которой каждый ключ представлен двумя частями: публичной и приватной. Подробнее см. Инфраструктура открытых ключей

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

1.36 Что нового в блокчейн-платформе Waves Enterprise

1.36.1 1.12.1

Версия 1.12.1 является последней выпущенной версией и в этой справочной системе имеет тег **latest**.

Добавлены разделы:

- *Токены блокчейн-платформы Waves Enterprise*
- *Управление токенами из смарт-контракта*

Изменены следующие разделы:

- *Сервисы gRPC, используемые смарт-контрактом*
- *contract_ contract_ service.proto*
- *Роли участников*
- *Инструментарий gRPC*
- *Смарт-контракты*
- *Список идентификаторов функциональных возможностей*
- *Словарь терминов*
- *POST /utils/hash/secure*

Версия 1.12.1 содержит важные исправления; подробнее см. описание релиза.

1.36.2 1.12.0

Добавлены разделы:

- *Общая настройка платформы: настройка криптографии*
- *REST API: получение сертификатов*
- *gRPC: получение сертификатов*

Изменены следующие разделы:

- *Анкоринг*
- *node.conf*
- *Примеры конфигурационных файлов ноды*
- *Лицензии блокчейн-платформы Waves Enterprise*
- *Получение лицензии для работы в частной сети*
- *Развертывание платформы в частной сети*
- *Настройка платформы для работы в частной сети*
- *gRPC: получение параметров конфигурации ноды*
- *REST API: информация о конфигурации и состоянии ноды, остановка ноды*

- *REST API: информация о смарт-контрактах*
- *Генераторы*
- *POST /utils/hash/fast*
- *POST /privacy/sendData*
- *POST /privacy/sendDataV2*
- *POST /privacy/sendLargeData*
- *Отправка в блокчейн конфиденциальных данных*
- *Отправка в блокчейн потока конфиденциальных данных*
- *Активация функциональных возможностей*
- *gRPC: получение параметров конфигурации ноды*
- *Использование Ledger Nano с клиентом блокчейн-платформы Waves Enterprise*
- *REST API: подписание и валидация сообщений в блокчейне*
- *REST API: реализация методов шифрования*
- *REST API: формирование и проверка электронной подписи данных (PKI)*
- *Подписание и отправка транзакций*
- *Отправка транзакций в блокчейн*
- *REST API: обмен конфиденциальными данными и получение информации о группах доступа*
- *gRPC: работа с конфиденциальными данными*
- *gRPC: получение информации о результатах исполнения вызова смарт-контракта*
- *103. CreateContract Transaction*
- *Внешние компоненты платформы*
- *Криптография*
- *Общая настройка платформы: настройка консенсуса*
- *Общая настройка платформы: настройка майнинга*
- *Подписание genesis-блока и запуск сети*
- *REST API: информация об ассетах и балансах адресов*
- *Разработка и применение смарт-контрактов*
- *Пример смарт-контракта с использованием REST API*
- *103. CreateContract Transaction*
- *104. CallContract Transaction*
- *107. UpdateContract Transaction*
- *Активация функциональных возможностей*

Версия 1.12.0 содержит важные исправления, подробнее см. описание релиза.

1.36.3 1.11.0

Добавлены следующие разделы:

- *Различия opensource и коммерческой версий блокчейн-платформы Waves Enterprise*
- *Клиент для WE contract SDK (Java/Kotlin Contract SDK)*

Изменены следующие разделы:

- *Развертывание платформы в частной сети*
- *Развертывание платформы с подключением к Mainnet*
- *Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды*
- *Тонкая настройка платформы: настройка TLS*
- *Общая настройка платформы: настройка исполнения смарт-контрактов*
- *Лицензии блокчейн-платформы Waves Enterprise*

Версия 1.11.0 содержит важные исправления, подробнее см. описание релиза.

1.36.4 1.8.4

Добавлены следующие разделы:

- *Использование Ledger Nano с клиентом блокчейн-платформы Waves Enterprise*
- *Создание смарт-контрактов с помощью JS Contract SDK*
- *Создание смарт-контрактов с помощью Java/Kotlin Contract SDK*

Изменены следующие разделы:

- *103. CreateContract Transaction*
- *Группа методов Contracts*
- *Общая настройка платформы: настройка исполнения смарт-контрактов*
- *Использование REST API*
- *Комиссии в сети Mainnet*

Версия 1.8.4 содержит важные исправления, подробнее см. описание релиза.

1.36.5 1.8.2

Версия 1.8.2 содержит важные исправления, подробнее см. описание релиза.

1.36.6 1.8.0

Изменены следующие разделы:

- *Тонкая настройка платформы: настройка групп доступа к конфиденциальным данным*
- *REST API: реализация методов шифрования*
- *Словарь терминов*
- *Системные требования*
- *Тонкая настройка платформы: настройка TLS*
- *Пример подготовки артефактов для TLS*
- *Тонкая настройка платформы: настройка инструментов gRPC и REST API ноды*
- *Общая настройка платформы: настройка исполнения смарт-контрактов*
- *node.conf*
- *Общая настройка платформы: настройка майнинга*
- *Требования к окружению для платформы Waves Enterprise*
- *Инструментарий gRPC*
- *gRPC: отслеживание событий в блокчейне*
- *gRPC: получение информации о ноде*
- *contract_transaction_service.proto*
- *gRPC: получение информации о результатах исполнения вызова смарт-контракта*
- *gRPC: получение информации о размере UTX-пула*
- *contract_pki_service.proto*
- gRPC: реализация методов шифрования
- *gRPC: работа с транзакциями*
- *gRPC: работа с конфиденциальными данными*
- *REST API: обмен конфиденциальными данными и получение информации о группах доступа*
- *gRPC: получение вспомогательной информации*
- *gRPC: получение информации об адресах участников сети*
- *Обмен конфиденциальными данными*
- *REST API: информация о конфигурации и состоянии ноды, остановка ноды*
- *Смарт-контракты*
- *Активация функциональных возможностей*
- *Клиент*
- *Неизменяемость данных в блокчейне*

Версия 1.8.0 содержит важные исправления, подробнее см. описание релиза.

1.36.7 1.7.3

Версия 1.7.3 содержит важные исправления, подробнее см. описание релиза.

1.36.8 1.7.2

Изменены следующие разделы:

- generating_balance
- *Создание аккаунта ноды*
- *Подписание genesis-блока и запуск сети*
- *Механизм вознаграждения валидаторов смарт-контрактов*
- *Словарь терминов*

1.36.9 1.7.0

Добавлен следующий раздел:

Тонкая настройка платформы: настройка ноды в режиме наблюдения

1.36.10 1.6.2

Изменены следующие разделы:

- *Описание транзакций*
- *Сервисы gRPC, используемые смарт-контрактом*
- *Смарт-контракты*
- *Роли участников*
- *Механизм создания снимка данных*
- *Активация функциональных возможностей*
- *Системные требования*

1.36.11 1.6.0

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*

1.36.12 1.5.2

Внесены изменения в раздел *Алгоритм консенсуса CFT*.

Версия 1.5.2 содержит важные исправления, подробнее см. описание релиза.

1.36.13 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

1.36.14 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

1.36.15 1.3.1

Добавлены следующие разделы:

- Параллельное исполнение контрактов

Изменены следующие разделы:

- Создание смарт-контракта
- Настройка Docker

1.36.16 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

1.36.17 1.2.3

Изменены следующие разделы:

- Смарт-контракты Docker
- Описание основных параметров и секций конфигурационного файла ноды
- Настройка групп доступа к конфиденциальным данным

1.36.18 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

1.36.19 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
- Обновлен раздел *Криптография*. Часть информации была перенесена в Операции шифрования данных
- Изменения в конфигурационном файле ноды
- Транзакции

1.36.20 1.1.2

Изменены следующие разделы:

- Демо-версия
- Изменения в конфигурационном файле ноды
- Раздел Установка ноды преобразован в раздел «Установка и запуск платформы»
- Подключение участников к сети
- Настройка анкоринга
- Настройка типа авторизации для доступа к REST API ноды
- Подключение ноды в сеть «Partnernet»
- Подключение ноды в сеть «Mainnet»
- *Системные требования*

1.36.21 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

1.36.22 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, 368

C

CEK, 368
Crash Fault Tolerance (*CFT*), 368

Авторизация, 364
Адрес, 364
Аккаунт, 364
Алиас (*псевдоним*), 364
Анкоринг, 364
Ассет, 364
Атомарная транзакция, 364
Баланс, 364
Блок, 364
Блокчейн, 365
Валидация, 365
Генератор, 365
Генерирующий баланс, 365
Группа доступа, 365
Дата-краулер, 365
Исполнение смарт-контракта, 365
Ключевой блок, 365
Комиссия, 365
Консенсус, 365
Лизинг, 365
Лицензия, 365
Майнер, 365
Майнинг, 365
Миграция, 365
Микроблок, 365
Нода (*узел*), 366
Обновление ноды, 366
Образ, 366
Откат, 366
Пир, 366
Подписание транзакции, 366
Приватная (частная) сеть, сайдчейн, 366
Приватный ключ, 366
Публикация транзакции, 366
Публичная сеть, 366
Публичный ключ, 366
Пул неподтвержденных транзакций (*UTX-пул*), 366
Раунд, 366
Репозиторий, 366
Роль, 366
Сетевое сообщение, 366
Смарт-контракт, 366
Снимок данных (*снепшот*), 367
Создание смарт-контракта, 367
Софт-форк, 367
Стейт, 367
Стейт адреса, 367
Стейт смарт-контракта, 367
Токен, 367
Транзакция, 367
УКЭП, 367
Участник, 367
Форк, 367
Хранилище ключей (*keystore*), 367
Хэш, 367
Хэш ключевой строки, 367
Эндпоинт (эндпойнт, Endpoint) сервиса, 368