



Техническое описание платформы  
Waves Enterprise  
*Выпуск master*

<https://wavesenterprise.com>

мар. 18, 2021

---

## Блокчейн-платформа Waves Enterprise

---

---

## Обзор возможностей

---

Блокчейнплатформа Waves Enterprise — это универсальное решение для масштабируемой цифровой инфраструктуры, сочетающее в себе свойства публичных и частных блокчейнов для корпоративного и государственного использования. Корпоративная блокчейнплатформа решает проблему доверия между участниками отношений на уровне протокола работы платформы. Поддерживаемые типы консенсусов *PoS*, *PoA* и *CFT* гарантируют корректность добавляемых в блокчейн данных, а децентрализация обеспечивает независимый от контрагентов доступ к данным.

### 1.1 Блокчейн Waves Enterprise

- Построен на языке программирования Scala.
- Реализует лучшие технологии и практики использования, унаследованные от публичной блокчейнплатформы Waves.
- Адаптирован для использования в корпоративном и государственном секторах.
- Поддерживает три алгоритма консенсуса *PoS*, *PoA* и *CFT*: при развёртывании сети можно выбрать наиболее подходящий под вашу задачу.
- Обладает высокой пропускной способностью.
- Поддерживает Тьюрингполные *смартконтракты* на Docker.
- Поставляется в виде набора микросервисов.
- Использует алгоритмы сертифицированной государственной криптографии.
- Позволяет отправлять конфиденциальные данные через блокчейн адресно при помощи групп доступа, без публикации данных в открытом доступе.
- Реализует на уровне консенсуса систему управления полномочиями.
- Вебклиент Waves Enterprise реализует следующие функции: просмотр *транзакций*, кошелек, отправка всех типов транзакций, разработка смартконтрактов, мониторинг состояния блокчейна, управление полномочиями участников сети.

### 1.1.1 Варианты развертывания сети Waves Enterprise

Платформа Waves Enterprise предлагает три варианта развертывания блокчейна:

1. Работа в основной публичной сети.
2. Работа в частной сети с анкорингом в основную сеть.
3. Работа в независимой частной сети.

## 1.2 Основная сеть

Основная сеть поддерживается консорциумом крупных компаний из различных областей: банковская деятельность, промышленность, девелопмент, логистика и т.д. Компании, поддерживающие основную сеть, реализуют свои проекты в публичном блокчейне, а также обеспечивают сопутствующие процессы. К примеру, банки предоставляют фиатные *шлюзы*, регистраторы — доступ к облачной ГОСТкриптографии и т.д.

## 1.3 Независимая частная сеть

Коробочное решение Waves Enterprise позволяет развернуть собственную частную сеть для тех компаний, которые не готовы публично вести свои корпоративные процессы. Сеть конфигурируется в соответствии с потребностями каждой отдельной компании.

Основные конфигурируемые свойства решения:

- Тип консенсуса.
- Криптография.
- Число нод.
- Параметры работы блокчейна.

## 1.4 Частная сеть с публикацией хешей в основную сеть

Данное решение объединяет преимущества двух предыдущих вариантов и может быть актуально для небольших компаний и/или партнёров компаний, поддерживающих основную сеть. Использование частной сети позволяет избежать публичной демонстрации транзакций. При этом регулярная публикация хешей приватных блоков в основной сети позволяет повысить надежность информации внутри частной сети за счёт эффекта масштаба основной сети.

---

### Официальные ресурсы

---

- Официальный сайт блокчейнплатформы [Waves Enterprise](#)
- Страница проекта в [Github](#)
- Официальный сайт блокчейнплатформы [Waves](#)

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

- **мастерблокчейна** (Waves Enterprise Mainnet), который обеспечивает работу сети в целом, выступая в качестве глобального арбитра и опорной цепи, и ряда пользовательских;
- отдельных **сайдчейнов**, легко настраиваемых в соответствии с конкретной бизнесзадачей.

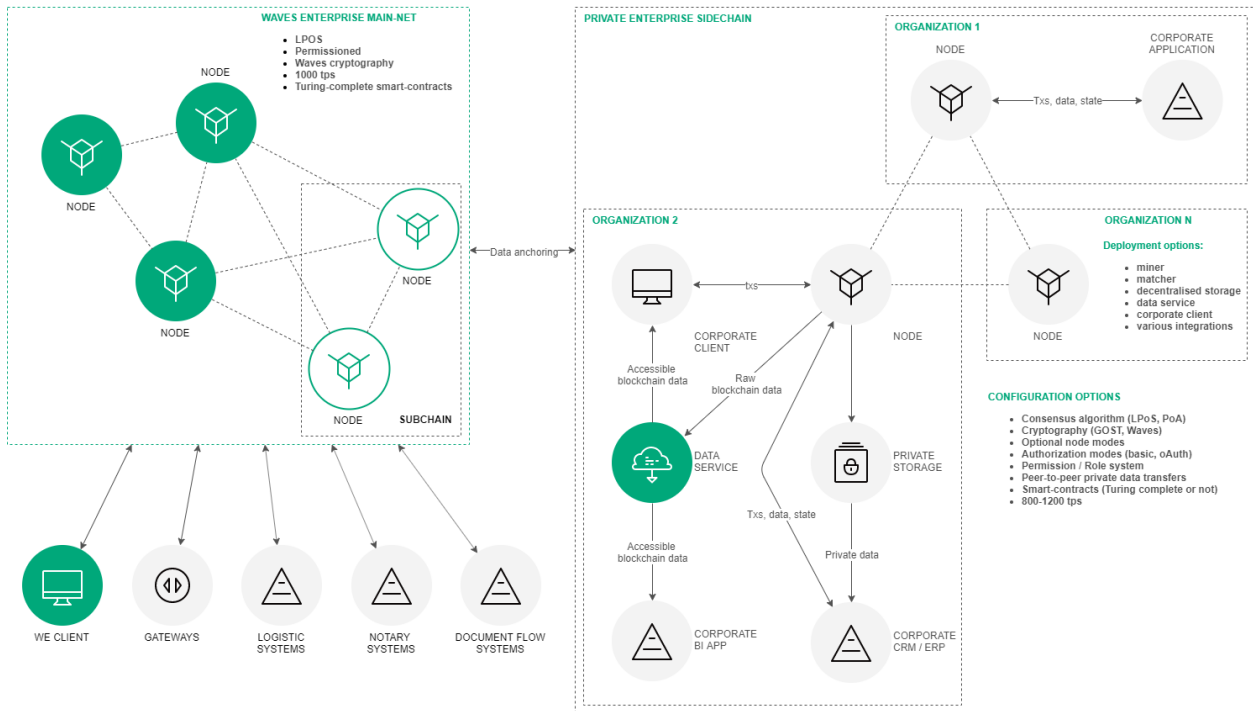
Применение такого принципа построения позволяет добиться оптимизации для более высоких скоростей или больших объемов вычислений, согласованности и доступности данных, а также устойчивости к злонамеренному изменению информации.

Также в Waves Enterprise реализован *механизм анкоринга сетей*, позволяющий создать конфигурацию сети, использующую сильные стороны обоих алгоритмов консенсуса. Например, основной блокчейн Waves Enterprise базируется на алгоритме консенсуса ProofofStake, 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 ProofofAuthority или Crash Fault Tolerance. Сайдчейны встраиваются в основной блокчейн с помощью механизма анкоринга (помещая криптографические доказательства транзакций в основную блокчейнсеть).

### 3.1 Архитектура ноды и дополнительных сервисов

ПО ноды может быть установлено без дополнительных сервисов, поскольку оно обеспечивает функционирование и взаимодействие внутри блокчейнсети. Дополнительные сервисы, в свою очередь, облегчают и значительно упрощают взаимодействие пользователя с блокчейнплатформой. Платформа Waves Enterprise состоит из пяти основных модулей и нескольких дополнительных микросервисов. В состав основных модулей входят:

- **Нода** основное приложение, устанавливаемое на компьютер и настраиваемое для работы в блокчейне по любому сценарию.



- **Корпоративный клиент вебприложение**, предоставляющее современный и многофункциональный интерфейс для взаимодействия с блокчейном.
- **Среда разработки смартконтрактов** среда для развёртывания и выполнения Тьюрингполных Docker смартконтрактов. Разворачивание Dockerконтейнеров со смартконтрактами происходит на удалённой виртуальной машине для обеспечения дополнительной безопасности.
- **Датасервис** сервис агрегирует данные из блокчейна в хранилище RDBMS (PostgreSQL) и обеспечивает полнотекстовый поиск по любой информации, содержащейся в блокчейне, через вебсервис RESTfull.
- **Приватное хранилище** компонент обеспечивают обработку и хранение приватных данных, а также коммуникации через зашифрованное соединение peertopeer. Приватное хранилище реализуется на базе БД PostgreSQL или S3 на Minio.

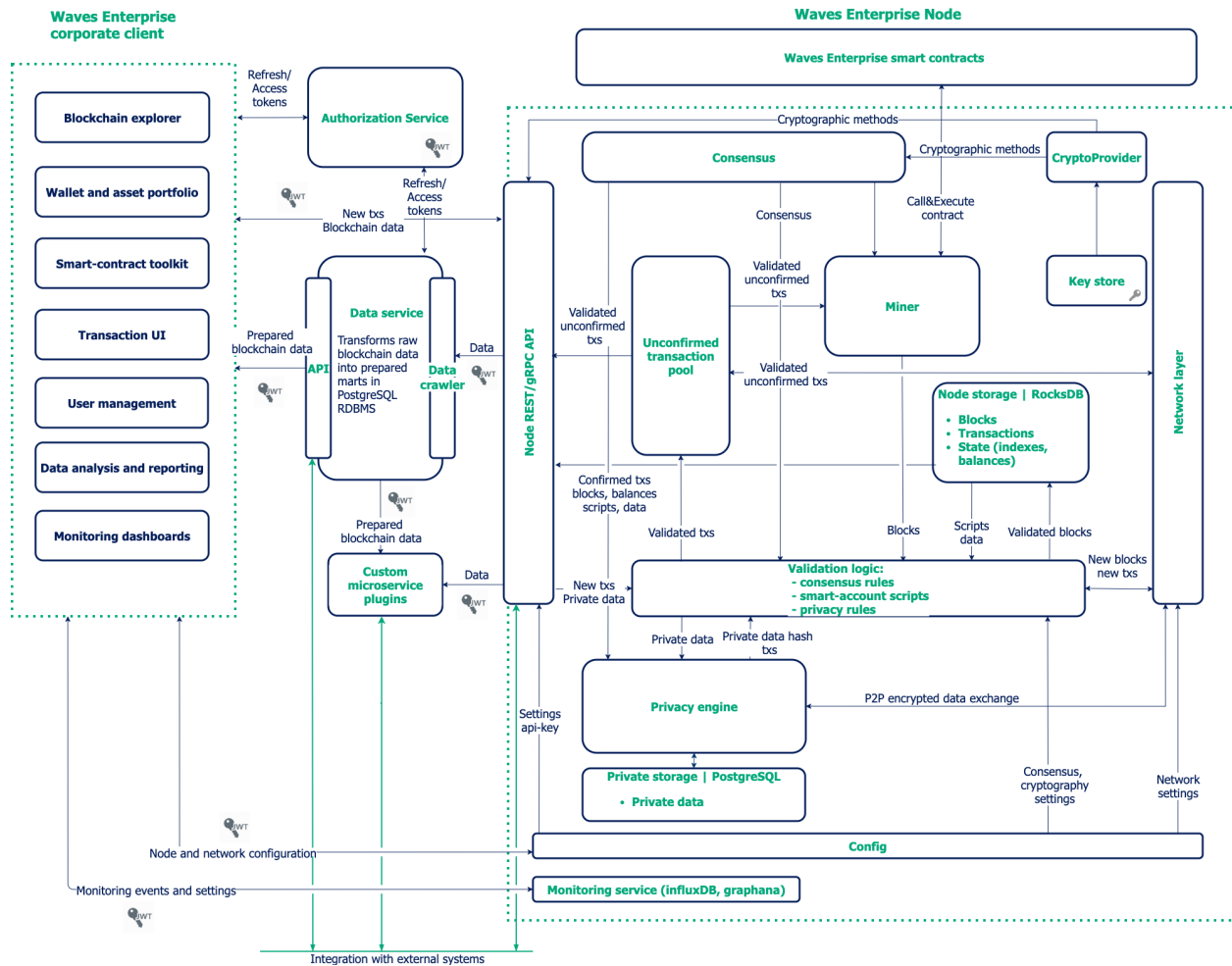
Дополнительные микросервисы включают в себя:

- **Сервис авторизации** сервис обеспечения авторизации для всех компонентов.
- **Датакраулер** сервис извлечения данных с ноды и загрузки извлечённых данных в датасервис.
- **Генератор** сервис генерации ключевых пар для новых аккаунтов и создания `apikeyhash`.
- **Плагины кастомизации данных** набор плагинов для обработки и кастомизации данных, передаваемых и принимаемых от внешних систем.
- **Сервис мониторинга** внешний сервис мониторинга, использующий базу данных (InfluxDB) для хранения временных рядов с данными и метриками приложения. БД InfluxDB является ПО с открытым исходным кодом и устанавливается клиентом отдельно.

## Компоненты ноды

Нода имеет следующие внутренние компоненты:

- **Node API** – интерфейсы gRPC и REST API ноды, позволяющие получать данные из блокчейна, подписывать и отправлять транзакции, отправлять конфиденциальные данные, создавать и выполнять





смартконтракты и др.

- **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.
- **Unconfirmed transaction pool (UTX pool)** – компонент, обеспечивающий хранение неподтвержденных транзакций до момента их проверки и отправки в блокчейн.
- **Consensus and cryptolibraries** – компоненты, отвечающие за механизм достижения согласия между узлами, а также за криптографические алгоритмы.
- **Key store** хранилище ключевых пар ноды и пользователей, все ключи защищены паролем.
- **Miner** – компонент, отвечающий за формирование блоков транзакций для записи в блокчейн, а также за взаимодействие с Docker смартконтрактами.
- **Network layer** – слой логики, обеспечивающий взаимодействие нод на прикладном уровне по сетевому протоколу поверх TCP.
- **Validation logic** – слой логики, содержащий такие правила проверки транзакций, как базовая проверка подписи и расширенная проверка по сценарию.
- **Config** – конфигурационные параметры ноды, задаваемые в файле `nodename.conf`.
- **Monitoring Service** – внешний сервис мониторинга, использующий базу данных (InfluxDB) для хранения временных рядов с данными и метриками приложения. БД InfluxDB является ПО с открытым исходным кодом и устанавливается клиентом отдельно.

Описание протокола работы блокчейна Waves Enterprise, обеспечивающего преимущество производительности относительно других блокчейнов.

### 4.1 Термины

- **Блок** — зафиксированный в блокчейне набор транзакций, подписанный майнером и содержащий ссылку на подпись предыдущего блока. Ограничен 1 Мб или 6000 транзакций.
- **Раунд** — период времени между выпуском ключевых блоков. Плавающее значение, регулируется алгоритмом консенсуса в зависимости от нагрузки на сеть, в среднем 40 секунд.
- **Доказательство доли владения** — получение права майнинга в консенсусе PoS.
- **Нода** — узел сети с запущенным приложением блокчейна Waves Enterprise.
- **Майнер** — нода, адрес которой обладает достаточным для майнинга балансом и ролью *miner*.
- **Ключевой блок** — блок, содержащий только служебную информацию:
  - публичный ключ майнера — для проверки подписи микроблоков;
  - сумму комиссии майнера за предыдущий блок;
  - подпись майнера;
  - ссылку на предыдущий ключевой блок.
- **Liquid Block** — служебный термин, описывающий состояние блока до выпуска следующего ключевого блока, т.е. завершения его майнинга.
- **Микроблоки** — наборы транзакций, применяемых к состоянию блокчейна раз в 5 секунд. Ограничен 500 транзакциями. Каждый микроблок подписан приватным ключом майнера.

## 4.2 Описание протокола

**WavesNG** — протокол, разработанный Waves Platform на основе **BitcoinNG** для повышения пропускной способности блокчейна Waves, на архитектуре которого реализован Waves Enterprise. Основная концепция протокола — непрерывное создание микроблоков вместо одного большого блока в каждом раунде майнинга, поскольку микроблоки гораздо быстрее пересылаются и проверяются.

Раунды майнинга начинаются с выпуска ключевого блока. Момент появления каждого ключевого блока и адрес указанного в нём майнера определяются *консенсусом*. Ключевой блок не содержит транзакций и быстро формируется. Далее, до появления следующего блока, раз в 5 секунд формируются микроблоки с транзакциями без доказательства доли, что также повышает скорость обработки. Каждый микроблок ссылается на предыдущий. Ключевой блок добавляется в блокчейн, как только следующий майнер выпустит свой ключевой блок со ссылкой на него.

Такой подход снижает время подтверждения транзакции по сравнению с другими блокчейнами.

### 4.2.1 1. Процесс создания Liquid Block

1. Консенсусом определяется майнящий адрес.
2. Майнер создает и рассылает по сети ключевой блок.
3. Каждые 5 секунд майнер создает и рассылает по сети микроблок, который содержит транзакции. Он должен ссылаться на предыдущий микроблок или ключевой блок.
4. Процесс продолжается до тех пор, пока в сети не появится новый валидный ключевой блок.

### 4.2.2 2. Механизм вознаграждения майнеров в WavesNG

В протоколе предусмотрена финансовая мотивация соблюдения правил для участников. Комиссия от транзакций в блоке распределяется в следующей пропорции: 40 % комиссии получает майнеру, создавший блок, 60 % майнер следующего блока. Транзакция по начислению комиссии происходит каждые 100 блоков для обеспечения доверительного интервала проверок.

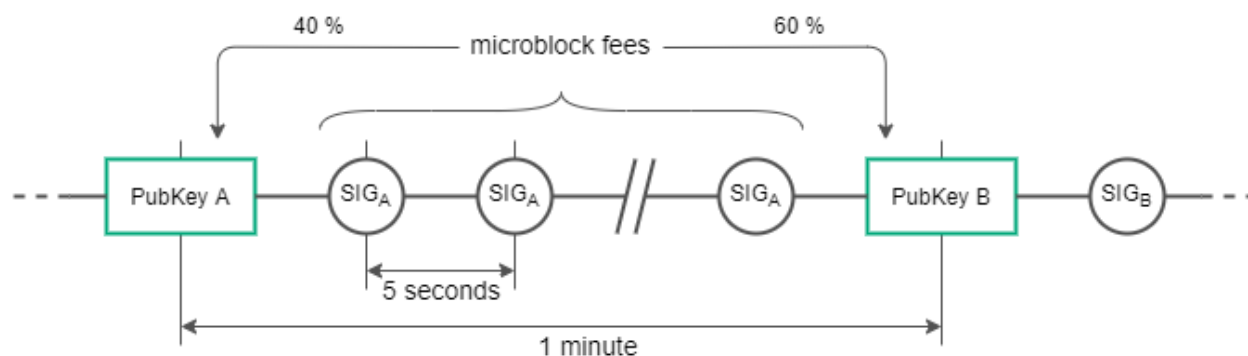


Рис. 1: Схема разделения комиссий

### 4.2.3 3. Разрешение конфликтов

Если майнер продолжает цепочку, создавая два микроблока с одним и тем же родительским блоком, он наказывается отменой дохода от комиссий; тот, кто обнаруживает мошенничество, получает награду майнера за блок. Блокчейн — распределенная система, и каждая нода хранит копию состояния всей сети. При появлении очередного микроблока, нода применяет полученные изменения к своей копии состояния сети и сверяет с остальными узлами сети. В этот момент происходит обнаружение несогласованности транзакций.

---

## Алгоритмы консенсуса

---

Блокчейн — это децентрализованная система, в которой нет центрального органа власти. Это делает систему некоррупционной, но создает сложности с итоговым принятием решений и организацией работы. Эти задачи решает механизм консенсуса, который является способом достижения согласия в группе участников. Голосование происходит в пользу большинства, не учитывая интересы меньшинства, но с другой стороны, это гарантирует достижение соглашения, которое несет пользу всей сети.

В Waves Enterprise вы можете выбрать механизм консенсуса при первичной конфигурации сети. Описание доступных механизмов, а также их плюсы и минусы, разобраны в соответствующих разделах.

### 5.1 Алгоритм консенсуса LPoS

Доказательство доли владения с правом аренды. В PoS (Proof of Stake) системах создание блока не требует энергозатратных вычислений, задача майнера — создание цифровой подписи блока.

#### 5.1.1 Proof of Stake

Механизм распределения прав создания блоков основан на количестве токенов на счету пользователя. Чем больше у пользователя токенов, тем выше вероятность, что он сможет создать блок.

В консенсусе Proof of Stake (доказательство доли) право выпуска блока определяется псевдослучайным образом, поскольку зная предыдущего майнера и балансы всех пользователей в системе можно вычислить следующего майнера. Это возможно, благодаря детерминированному вычислению генерирующей подписи блока, которая получается путем SHA256 хеширования генерирующей подписи текущего блока и публичного ключа аккаунта. Первые 8 байт полученного хеша преобразуются в число, называемое  $hit$  аккаунта  $X_n$ , и являются указателем на следующего майнера. Время генерации блока для аккаунта  $i$ , рассчитывается как:

$$T_i = T_{min} + C_1 \log\left(1 - C_2 \frac{\log \frac{X_n}{X_{max}}}{b_i A_n}\right)$$

где:

- $b_i$  — это стейк (доля баланса участника от общего баланса системы);
- $A_n$  — baseTarget, адаптивный коэффициент, регулирующий среднее время выпуска блока;
- $X_n$  — hit аккаунта;
- $T_{min}$  — 5 секунд, константа, определяющая минимальный временной интервал между блоками;
- $C_1$  — константа, равная 70 и корректирующая форму распределения интервала между блоками;
- $C_2$  — константа, равная 5E17 и предназначенная для регулировки значения baseTarget (сложности).

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 — сложность вычислений, параметр, удерживающий время генерации блоков в заданном диапазоне. BaseTarget в свою очередь вычисляется как:

$$(S > R_{max} \rightarrow T_b = T_p + \max(1, \frac{T_p}{100})) \wedge (S < R_{min} \wedge \wedge T_b > 1 \rightarrow T_b = T_p - \max(1, \frac{T_p}{100}))$$

где

- $R_{max} = 90$  — максимальное уменьшение сложности, когда время генерации блока в сети превышает 40 секунд;
- $R_{min} = 30$  — минимальное увеличение сложности, когда время генерации блока в сети меньше 40 секунд;
- $S$  — среднее время генерации, как минимум для трех последних блоков;
- $T_p$  — предыдущее значение baseTarget;
- $T_b$  — вычисленное значение baseTarget.

Глубокое описание технических особенностей и доработок классического PoS алгоритма вы можете найти в этой [статье](#).

## Преимущества перед PoW

Отсутствие сложных вычислений позволяет PoS сетям снизить требования к аппаратному обеспечению участников системы, что снижает стоимость разворачивания частных сетей. Также не нужна дополнительная эмиссия, которая в PoW (Proof of Work) системах используется для вознаграждения майнеров за нахождение нового блока. В PoS системах майнер получает вознаграждение в виде комиссий за транзакции, которые попали в его блок.

### 5.1.2 Leased Proof of Stake

Для пользователя, который обладает стейком, 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, за что вы можете получать часть от комиссий за транзакций, которые этот майнер поместил в свои блоки. Лизинг является полностью безопасной операцией. Токены не покидают ваш кошелек, вы передаете право учитывать свой баланс при розыгрыше права майнинга другому участнику сети.

## 5.2 Алгоритм консенсуса PoA

В приватном блокчейне не всегда нужны токены — например, блокчейн может быть использован для хранения хешей документов, которыми обмениваются организации. В таком случае, при отсутствии токенов и комиссий с транзакций, решение на базе алгоритма консенсуса PoS является избыточным. В Waves Enterprise можно выбрать 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 системой. Блоки и транзакции проверяются заранее утвержденными участниками, которые выступают в качестве модераторов системы.

### 5.2.1 Описание алгоритма

На базе приведенных ниже параметров формируется алгоритм определения майнера текущего блока. Параметры консенсуса указываются в блоке `consensus` конфигурационного файла ноды.

- $t$  — длительность раунда в секундах (параметр конфигурационного файла ноды: `roundduration`).
- $t_s$  — длительность периода синхронизации, вычисляется как  $t \cdot 0,1$ , но не более 30 секунд (параметр конфигурационного файла ноды: `syncduration`).
- $N_{\text{ban}}$  — количество пропущенных подряд раундов для выдачи бана майнеру (параметр конфигурационного файла ноды: `warningsforban`);
- $P_{\text{ban}}$  — доля максимального количества забаненных майнеров, в процентах от 0 до 100 (параметр конфигурационного файла ноды: `maxbanspercentage`);
- $t_{\text{ban}}$  — продолжительность бана майнера в блоках (параметр конфигурационного файла ноды: `bandurationblocks`).
- $T_0$  — unix time создания genesis блока.
- $T_H$  — unix time создания блока  $H$  — ключевой блок для NG.
- $r$  — номер раунда, вычисляется как  $(T_{\text{Current}} - T_0) \div (t + t_s)$ .
- $A_r$  — лидер раунда  $r$ , имеющий право на создание ключевых блоков и микроблоков для NG в раунде  $r$ .
- $H$  — высота цепочки, на которой создается ключевой блок и микроблоки для NG. Право на выпуск блока на высоте  $H$  имеет лидер раунда  $A_r$ .
- $M_H$  — майнер, выпустивший блок на высоте  $H$ .
- $Q_H$  — очередь активных на высоте  $H$  майнеров.

Очередь  $Q_H$  формируется из адресов, которым `permission` транзакцией выдано разрешение на майнинг, у которых оно не было отозвано до высоты  $H$ , и не истекло до момента времени  $T_H$ .

Очередь сортируется по временной метке транзакции предоставления прав на майнинг — узел, которому права были предоставлены раньше, будет выше в очереди. Для согласованной сети эта очередь будет одинакова на каждой ноды.

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  $M_{H1}$ , который создал предыдущий ключевой блок на высоте  $H1$ .
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, в текущий раунд не производится блоков, раунд «пропускается». 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  $(T_0 + (r1) * (t + t_s); T_0 + (r1) * (t + t_s) + t]$ . Блок, созданный майнером не в свою очередь или не вовремя, не считается валидным. После раунда длительностью  $t$  сеть синхронизирует данные в течение  $t_s$ . У лидера  $A_r$  есть 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: тип (PoS или PoA),  $t$ ,  $t_s$  задаются в конфигурационном файле узла сети. Параметр  $t$  при этом должен совпадать у всех участников сети, иначе произойдет форк сети.

### 5.2.2 Синхронизация времени между узлами сети

Каждый узел сети должен синхронизировать время приложения с доверенным NTP-сервером в начале каждого раунда. Адрес и порт сервера указывается в конфигурационном файле ноды. Сервер должен быть доступен каждой ноды сети.

### 5.2.3 Исключение неактивных майнеров

Если каким-либо майнером  $N_{ban}$  раз подряд было пропущено создание блока, этот майнер исключается из очереди на  $t_{ban}$  последующих блоков (параметр `bandurationblocks` в конфигурационном файле). 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.

### 5.2.4 Мониторинг

Мониторинг консенсуса PoA помогает выявлять факты создания и распространения невалидных блоков, а также пропуски очереди майнерами. Дальнейшие действия по выявлению и устранению неисправностей, а также блокировке вредоносных узлов выполняются администраторами сети.

В целях мониторинга процесса формирования блоков для алгоритма PoA в InfluxDB размещаются данные:

- Активный список майнеров, отсортированный в порядке предоставления прав на майнинг.
- Плановая временная метка раунда.



- Фактическая временная метка раунда.
- Текущий майнер.

### 5.2.5 Изменение параметров консенсуса

Изменение параметров консенсуса (время раунда и периода синхронизации) выполняется на основании данных конфигурационного файла ноды (см. врезку) на высоте «fromheight». Если одна из нод не укажет новые параметры, то произойдет форк.

Пример конфигурации:

```
// specifying inside of the blockchain parameter
consensus {
  type = poa
  sync-duration = 10s
  round-duration = 60s
  ban-duration-blocks = 100
  changes = [
    {
      from-height = 18345
      sync-duration = 5s
      round-duration = 60s
    },
    {
      from-height = 25000
      sync-duration = 10s
      round-duration = 30s
    }
  ]
}
```

## 5.3 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 какойлибо 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 откаты блоки начинают майниться заново и на некоторое время становятся недоступны в блокчейне а это, в свою очередь, может повлиять на использующие блокчейн бизнеспроцессы. Алгоритм консенсуса CFT (Crash Fault Tolerance) исключает возникновение таких ситуаций.

### 5.3.1 Описание алгоритма

В основе реализации CFT лежит алгоритм консенсуса *PoA* с добавленной фазой голосования **валидаторов раунда майнинга** участников сети, автоматически назначаемых алгоритмом консенсуса. Такой подход гарантирует следующее:

- блок известен более чем половине участников сети и завалидирован ими;
- блок не будет откаты и попадет в цепочку;
- в блокчейне не произойдет образования параллельной цепочки.

Все это достигается посредством финализации выпущенного блока. Сама финализация блока опирается на консенсус большинства валидаторов раунда ( $50\% + 1$ ), в соответствии с которым и принимается решение о добавлении блока в сеть. В случае отсутствия такого большинства майнинг останавливается до восстановления связности сети.

Консенсус CFT, также как и PoA, зависит от текущего времени, а время начала и окончания каждого раунда рассчитывается на основе временной метки *генезис блока*. Основные параметры, на основе которых формируется алгоритм для определения майнера текущего блока, также идентичны параметрам алгоритма PoA (см. раздел Описание алгоритма). Для валидации блоков в блок consensus конфигурационного файла ноды были добавлены два новых параметра:

- **maxvalidators** – лимит валидаторов, участвующих в голосовании в конкретном раунде;
- **finalizationtimeout** – время, в течение которого майнер ждет финализации последнего блока в цепочке. По прошествии этого времени майнер вернет транзакции обратно в UTXпул и начнет майнить раунд заново.

Для приведенного ниже описания функциональности CFT используются следующие обозначения:

- $t$  длительность раунда в секундах (параметр конфигурационного файла ноды: roundduration).
- $t_{start}$  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: finalizationtimeout).
- $V_{max}$  лимит валидаторов, участвующих в голосовании (параметр конфигурационного файла ноды: maxvalidators).

### 5.3.2 Голосование

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 maxvalidators ( $V_{max}$ ). 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.

### 5.3.3 Особенности майнинга

Основные правила майнинга в рамках консенсуса CFT идентичны правилам консенсуса PoA. При этом был введен дополнительный механизм, обеспечивающий отказоустойчивость консенсуса.

При использовании консенсуса CFT очередная попытка майнинга считается неудачной, если последний полученный блок не был финализирован иными словами, к стейту не применен микроблок с набранными валидными голосами. При этом, если попытки майнинга выходят за временные рамки  $t_{\text{start}} + t_{\text{fin}}$ , нода принимает решение вернуть все транзакции из последнего блока обратно в УТХпул, после чего раунд начинается майниться заново.

Чтобы избежать возможного возврата транзакций в УТХпул, рекомендуется работать не с последним (жидким) блоком блокчейна, а с финализированным подтвержденным валидаторами сети.

### 5.3.4 Выбор канала для синхронизации

Для алгоритмов консенсуса PoS и PoA используется модуль, выбирающий для синхронизации наиболее сильную цепочку на основе сравнения данных задействованных нод. В CFT применяется иной механизм выбора, также увеличивающий отказоустойчивость системы: выбирается случайный канал из активных на момент синхронизации. Перечень активных каналов постоянно обновляется в ходе работы системы, а для равномерного распределения нагрузки на сеть время синхронизации с конкретным каналом ограничено.

### 5.3.5 Изменение параметров консенсуса

Как и в случае с алгоритмами консенсуса PoS и PoA, параметры консенсуса настраиваются на основе конфигурационного файла ноды. Ниже приведен пример конфигурации:

```
// specifying inside of the blockchain parameter
consensus {
  type = cft
  sync-duration = 10s
  round-duration = 60s
  ban-duration-blocks = 100
  max-validators = 16
  finalization-timeout = 5s
}
```

Платформа Waves Enterprise предоставляет возможность выбора используемой криптографии в зависимости от особенностей реализуемого проекта и юрисдикции заказчика.

#### 6.1 Хеширование

Операции хеширования в платформе выполняются функциями Blake2b256 и Кескак256 последовательно, либо функцией «Стрибог» в соответствии с ГОСТ Р 34.112012 «Информационная технология. Криптографическая защита информации. Функция хеширования». Размер блока выходных данных 256 бит.

#### 6.2 Электронная подпись

Алгоритмы генерации ключей, формирования и проверки электронной подписи реализованы на базе эллиптической кривой Curve25519 (ED25519 с ключами X25519), либо в соответствии с ГОСТ Р 34.102012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи».

#### 6.3 Шифрование данных

В платформе реализована возможность шифрования данных при помощи сессионных ключей на базе протокола ДиффиХелмана. Операция применяется для шифрования любого вида текстовой информации, например, данных смартконтрактов, которые не должны быть доступны для других участников блокчейна. Шифрование может выполняться как индивидуально для каждого получателя, с формированием уникального экземпляра шифротекста, так и с формированием единого шифротекста для группы получателей.

Используемые алгоритмы для симметричного шифрования соответствуют стандарту AES, либо ГОСТ Р 34.122015 «Информационная технология. Криптографическая защита информации. Блочные шифры».

Для операций шифрования/расшифрования данных применяются симметричные ключи СЕК и КЕК. СЕК (Content Encryption Key) используется для шифрования текстовых данных, КЕК (Key Encryption Key) используется для шифрования СЕК. Ключ СЕК формируется блокчейн-узлом случайным образом с применением соответствующих алгоритмов хеширования. Ключ КЕК формируется нодой на базе алгоритма Диффи-Хелмана, используя публичные и приватные ключи отправителя и получателей, и применяется для шифрования ключа СЕК.

Описание методов шифрования и их использования приведено в разделе *Операции шифрования данных*.

---

**Важно:** Используемый алгоритм шифрования оффчейн-протокола передачи приватных данных зависит от используемой версии ноды. К примеру, в актуальной версии 1.5 при выборе ГОСТ-шифрования протокол устанавливает зашифрованное TLSlike соединение с применением алгоритма шифрования «Кузнечик».

---

---

## Управление полномочиями

---

Блокчейнплатформа Waves Enterprise реализует закрытую (permissioned) модель блокчейна, доступ к которому возможен только для *авторизованных* администратором участников.

В платформе также присутствует ролевая модель, где каждая роль наделяет участника той или иной дополнительной возможностью. Подробнее о ролях в блокчейнсети Waves Enterprise читайте в следующем подразделе.

### 7.1 Описание ролей

#### **permissioner**

Участник с ролью `permissioner` является администратором сети и имеет право назначать или удалять любые роли участников сети. Как правило, роль `permissioner` присваивается участникам при запуске блокчейнсети.

#### **sender**

Участник с ролью `sender` имеет право отправлять транзакции в сеть.

**Внимание:** Роль `sender` определяется в генезисблоке, поэтому недоступна в сетях, созданных на версии платформы младше 1.5.0. Пример настройки роли `sender` для генезис блока в файле конфигурации ноды см. в разделе *Конфигурационный файл ноды*.

#### **blacklister**

Участник с ролью `blacklister` имеет право отправлять назначать или удалять роль `banned` другим участникам.

#### **miner**

Участник с ролью `miner` 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 (развёртывание) Docker смартконтрактов в блокчейне. Подробнее о смартконтрактах можно почитать в подразделе *Смартконтракты Docker*.

#### **connectionmanager**

Участник с ролью `connectionmanager` имеет право на подключение или отключение блокчейн-узлов от сети. Подробнее о подключении новых нод в сеть рассказано в подразделе *Управление доступом*.

#### **banned**

Роль `banned` получают ноды, которые временно или постоянно ограничены в действиях в блокчейн-сети.

## 7.2 Обновление списка полномочий

Изменить список полномочий может только нода с ролью `permissioner`. Для добавления или удаления ролей используется транзакция *102 Permission Transaction*. При изменении списка разрешений нода выполняет следующие проверки:

1. Отправитель транзакции *102* не находится в списке **blacklist**.
2. У отправителя есть роль `permissioner`.
3. Роль `permissioner` активна в настоящий момент у отправителя транзакции.
4. Указанная в транзакции *102* роль неактивна в случае её добавления адресу, и активна в случае её удаления у адреса.

Чтобы поместить выбранную ноду в список **blacklist**, нода с ролью `permissioner` назначает роль `banned` выбранному адресу, отправляя транзакцию *102* в блокчейн с соответствующими параметрами.

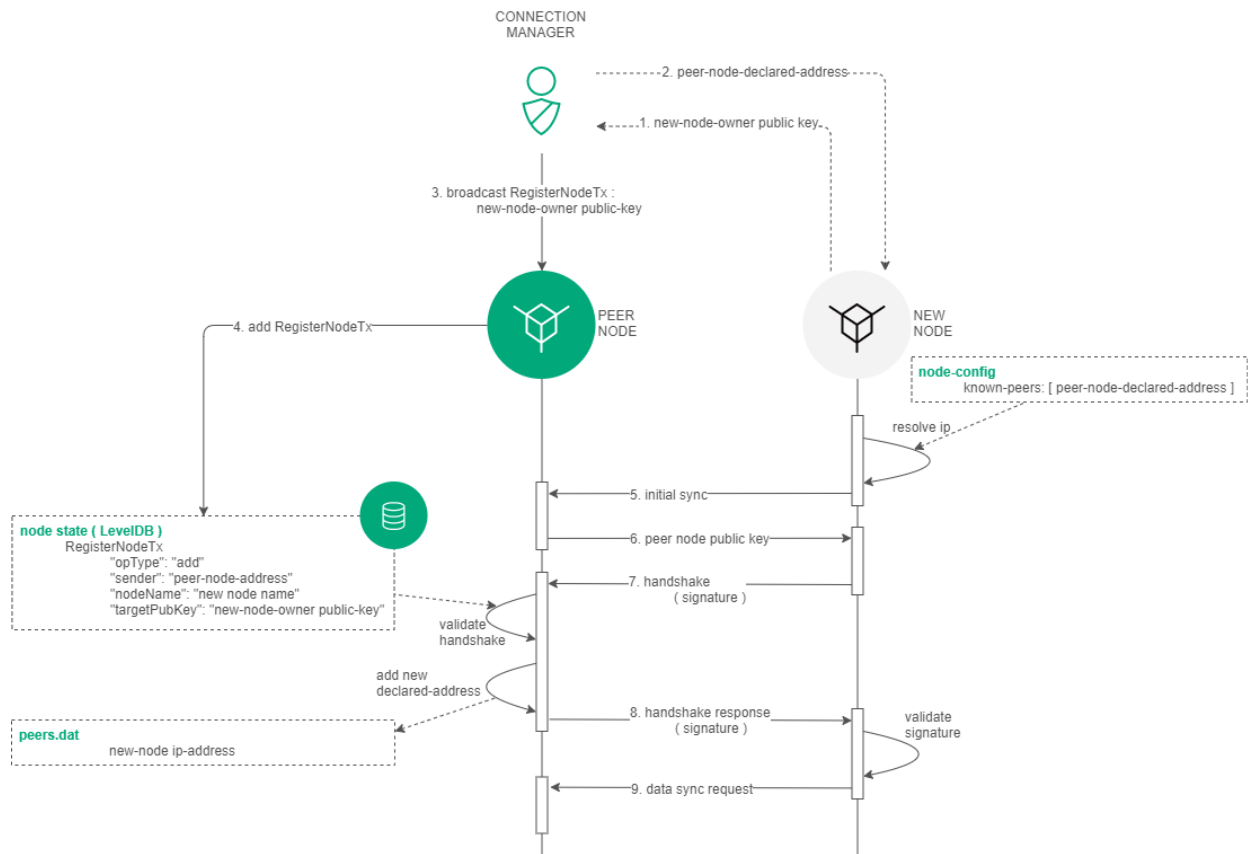
Для назначения любых других ролей (майнер, разработчик контрактов, управляющая токенами нода) нода с ролью `permissioner` выпускает транзакцию *102* с соответствующими параметрами. После попадания транзакции в блокчейн изменения полномочий у выбранных нод вступят в силу.

## 7.3 Управление доступом в блокчейн

В блокчейне Waves Enterprise правом на подключение участников к сети обладает пользователь с ролью «Connection Manager». Доказательством возможности подключения к блокчейн-сети является отдельная транзакция *111 RegisterNode*. В данной транзакции указываются учетные данные подключаемого узла. По результатам добавления подобных транзакций у каждого узла формируется таблица разрешенных участников.

Каждая попытка подключения участника сопровождается *handshake*-общением, в котором помимо служебной информации указывается область данных с доказательством принадлежности к подключаемой сети — в упрощенном виде это совокупность публичного ключа с электронной подписью участника. Поскольку публичный ключ подключаемого участника уже сохранен в хранилище остальных пиров, то участник получивший *handshake*-запрос сверяет подпись и предоставленный ранее в блокчейне публичный ключ. Если проверка завершилась успехом, то участник формирует ответный *handshake*-запрос, при успехе которого устанавливается соединение между сторонами. После успешного подключения участники выполняют синхронизацию с сетью, а так же синхронизацию таблицы соответствия блокчейн и сетевых адресов узлов, что необходимо в дальнейшем в процессе пересылки *конфиденциальных данных*.

Процесс отключения какого-либо участника от сети аналогичен ранее описанному процессу за тем исключением, того что пользователь с ролью «Connection Manager» выпускает транзакцию *111 RegisterNode* с





параметром "opType": "remove". Поскольку, handshakeзапрос выполняется с частотой 1 раз в 30 секунд, то следующий, после удаления участника из сети запрос, будет запрещен, ввиду отсутствия учетных данных подключаемого участника в таблице блокчейнузла.

---

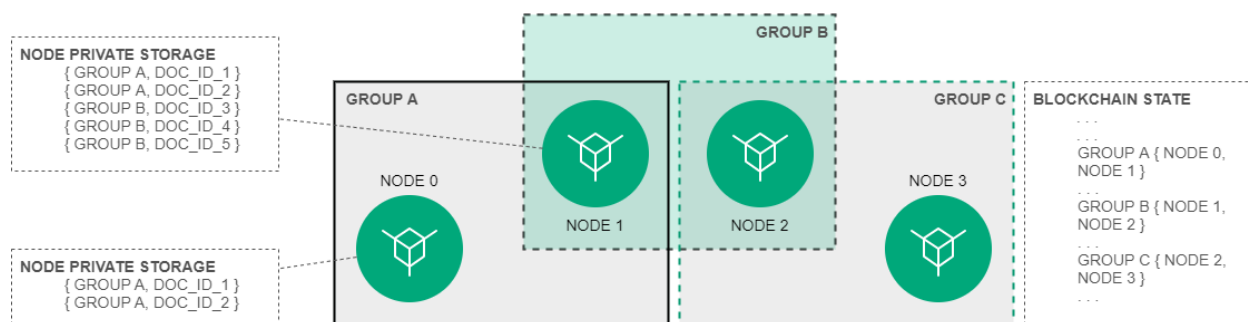
Конфиденциальность данных

---

Блокчейнплатформа Waves Enterprise позволяет организовать передачу и хранение конфиденциальных данных между участниками сетевого взаимодействия. Защита конфиденциальных данных при их передаче и хранении обеспечивается набором групп, участники которых могут обмениваться приватными данными между собой.

Платформа Waves Enterprise поддерживает два варианта хранения конфиденциальных данных:

- PostgreSQL
- S3 на базе серверов Minio



## 8.1 Группы доступа

Группа доступа создаётся участниками, которым необходимо обмениваться приватными данными. Группу доступа может создать любой участник сети и включить в неё любой состав других нод сети. Обмениваться информацией внутри группы могут только ноды.

Группа доступа имеет следующие параметры:

- имя (policyName);
- описание (Description);

- список получателей конфиденциальных данных (Recipients);
- список участников с правами на редактирование состава участников группы (Owners).

Создание группы доступа происходит при помощи отправки в блокчейн транзакции *CreatePolicy* (type = 112, создание группы).

Владельцы группы имеют право изменять состав участников группы доступа. Для изменения состава участников группы необходимо отправить в блокчейн транзакцию *UpdatePolicy* (type = 113, редактирование группы доступа).

Для внешних приложений в *API ноды* реализованы запросы, которые возвращают сведения по группе и данным, передаваемым внутри группы: GET /privacy/{policy}/recipients, GET /privacy/{policy}/getHashes, GET /privacy/getInfo/{hash}.

## 8.2 Отправка и получение данных

Отправляемые данные пересылаются посредством POST /privacy/sendData запросом через собственную ноду организации, в которой проверяется принадлежность отправителя к указанной им группе. Если проверка выполнена успешно, то данные записываются в хранилище ноды, и инициируется транзакция *PolicyDataHash* (type = 114, отправка хешсуммы данных в сеть) с посчитанной хешсуммой от передаваемых данных. Передать в сеть данные можно размером не более 20 МБ.

При получении транзакции с хешсуммой от передаваемых данных принимающая сторона проверяет принадлежность блокчейн-узла организации к указанной в транзакции группе. Если участник состоит в группе, то выполняется запрос *getPrivateData* на получение конфиденциальных данных. Запрос выполняется к сетевому адресу участника группы по установленному P2P соединению. Для обеспечения безопасности при передачи данных по незащищенному каналу связи используется криптографический протокол Диффи Хеллмана.

---

## Активация функциональных возможностей

---

Блокчейнплатформа Waves Enterprise поддерживает возможность активации функциональных возможностей блокчейна путем голосования нод. Иными словами, **механизм софтфорка блокчейна**. Активация новых функциональных возможностей — необратимое действие, поскольку блокчейн не поддерживает отката софтфорка.

В голосовании могут участвовать только ноды с ролью `miner`, поскольку голос ноды сохраняется в созданный ей блок.

### 9.1 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:

- `featurecheckblocksperiod` — период проведения голосования (в блоках);
- `blocksforfeatureactivation` — 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.

## 9.2 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 `featurecheckblocksperiod` происходит голосование всех нод, имеющих роль `miner`.
2. По прошествии интервала `featurecheckblocksperiod` производится подсчет голосов идентификаторов каждой функциональной возможности в созданных блоках.
3. Если возможность, вынесенная на голосование, набирает количество голосов, большее или равное параметру `blocksforfeatureactivation`, то она приобретает статус **APPROVED** (утверждена).
4. Утвержденная функциональная возможность активируется по прошествии интервала `featurecheckblocksperiod` от текущей высоты блокчейна.

## 9.3 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.

## 9.4 Предварительная активация функциональных возможностей

Все функциональные возможности, за которые предусмотрена возможность голосования, могут быть активированы принудительно при старте нового блокчейна. Для этого предусмотрен блок `preactivatedfeatures` 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.

## 9.5 Список идентификаторов функциональных возможностей

Идентификатор	Название
100	Алгоритм консенсуса LPoS
101	Поддержка gRPC смартконтрактами 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 liquidblock и микроблока

---

### Атомарные транзакции

---

Платформа Waves Enterprise поддерживает выполнение атомарных операций. Такие операции состоят из нескольких действий, и либо выполняются полностью, либо не выполняются вообще. Для этого в системе существует *120* транзакция, представляющая собой контейнер, в который помещаются две и более подписанные транзакции.

*120* транзакция поддерживает следующие типы транзакций:

- *4* *Трансфер ассета*, версия 3
- *102* *Добавление / удаление прав*, версия 2
- *103* *Создание контракта*, версия 3
- *104* *Вызов контракта*, версия 4
- *105* *Исполнение контракта*, версии 1 и 2
- *106* *Деактивация контракта*, версия 3
- *107* *Обновление контракта*, версия 3
- *112* *Создание группы приватности*, версия 3
- *113* *Обновление группы приватности*, версия 3
- *114* *Добавление приватных данных*, версия 3

Ключевым отличием новых версий транзакций, которые поддерживаются атомарной транзакцией *120*, является наличие полей *atomicBadge*. Это поле содержит доверенный адрес отправителя транзакции *trustedSender* для добавления в контейнер транзакции *120*. Если адрес отправителя не указывается, тогда отправителем становится адрес, с которого в блокчейн отправляется *120* транзакция.

## 10.1 Обработка атомарной транзакции

120 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 120 транзакции в пул неподтверждённых транзакций 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. Внутри атомарной транзакции, попавшей в блок, не должно быть исполняемых (executable) транзакций, и поле `miner` не должно быть пустым.

После исполнения атомарной транзакции в блок попадает её «копия», сформированная по следующим правилам:

- Поле `miner` не участвует в формировании подписи транзакции и заполняется публичным ключом майнера блока.
- Майнером блока формируется массив `proofs`, источником которого служат идентификаторы транзакций, входящих в атомарную транзакцию. При включении в блок атомарная транзакция имеет 2 подписи – подпись исходной транзакции и подпись майнера.
- Если в списке присутствуют executable транзакции, то они заменяются на executed транзакции. При валидации атомарной транзакции в составе блока проверяются обе подписи.

## 10.2 Создание атомарной транзакции

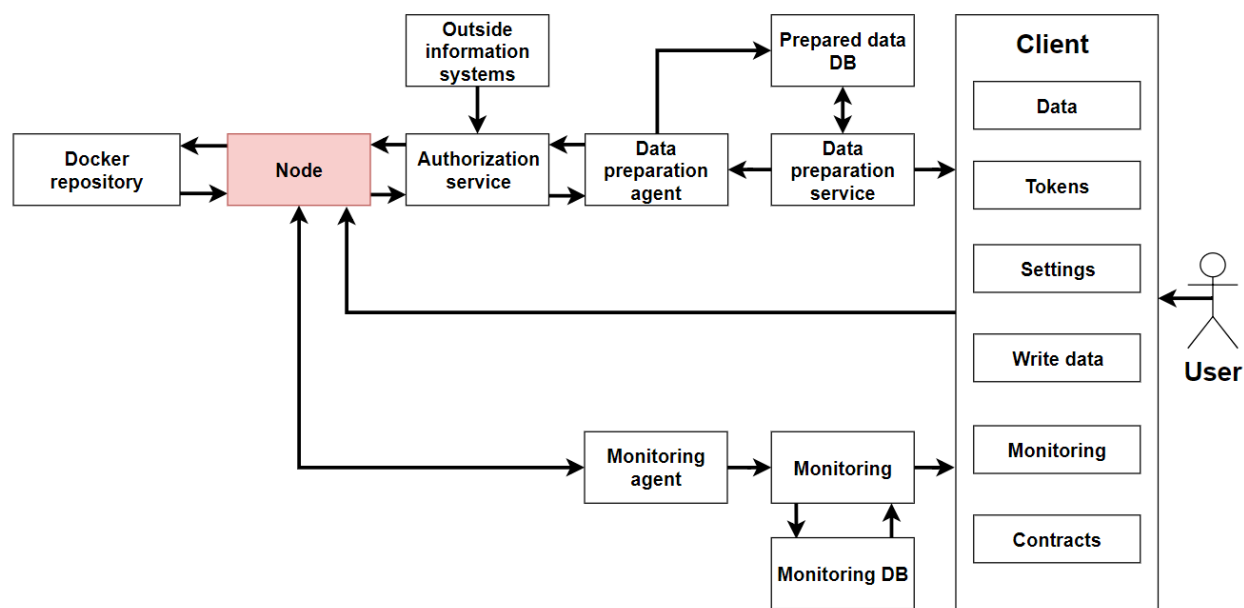
Для создания атомарной транзакции необходим доступ к *REST API* ноды.

1. Пользователь подбирает из списка поддерживаемых транзакций те транзакции, которые должны выполняться как атомарная операция.
2. Корректно заполняет поля всех транзакций и подписывает их, используя метод *sign*.
3. Далее пользователь заполняет поле `transactions` 120 транзакции данными подписанных, но не отправленных в блокчейн транзакций.
4. После внесения всех данных о транзакциях пользователь подписывает и отправляет в блокчейн 120 транзакцию.

**Внимание:** Если вы создаёте атомарную транзакцию с включением 114 транзакции, то при её подписании установите значение параметра `broadcast = false`.



Клиент *Waves Enterprise* — это удобный способ управления блокчейном *Waves Enterprise*. Клиент предназначен для работы в *публичной сети* *Waves Enterprise*.



Клиент включает в себя разделы для использования всех возможностей блокчейна:

- **Статистика сети** — страничка содержит общую информацию о сети и различные статистические данные.
- **Эксплорер** — страничка позволяет найти информацию о транзакциях или пользователях, благодаря гибкому поиску и развитой системе фильтров.
- **Токены** — страничка позволяет переводить, выпускать, передавать в аренду токены.

- **Контракты** страничка предоставляет инструменты для публикации и вызова Docker-контрактов. Для публикации доступны контракты из репозитория, адрес которого был указан при сборке клиента.
- **Передача данных** страничка позволяет отправлять из интерфейса транзакции с данными и файлы, а также работать с группами доступа для передачи конфиденциальных данных.
- **Настройки сети** страничка позволяет посмотреть информацию о нодах в сети и рассчитать сумму лизинга.
- **Написать нам** форма связи с технической поддержкой Waves Enterprise. Вы можете оставить комментарий в свободной форме, и он обязательно будет рассмотрен технической поддержкой.

Настройки профиля вы можете найти в верхнем правом углу интерфейса, нажав на иконку с Email. Форма выбора адреса ноды или создания нового блокчейнадреса для привязки аккаунта клиента к нему открывается при нажатии на кнопку Адрес.

Клиент поддерживает все современные типы браузеров. Если вебинтерфейс клиента некорректно отображается, или возникают какие-либо ошибки в процессе загрузки страниц, обновите ваш браузер до последней версии.

### Статистика сети

На вкладке «Общая информация» вы можете посмотреть следующие данные:

- Загрузку сети в процентах.
- Средний размер блока в байтах.
- Общее количество блоков.
- Количество отправителей транзакций.
- Количество нод в сети.
- Информацию о последних запущенных смартконтрактах с временем их исполнения.

### Эксплорер

Раздел содержит информацию о транзакциях в блокчейне. Для получения информации используйте фильтры и поиск с указанием полей транзакций для поиска.

Доступны фильтры транзакций:

- Все транзакции отображение всех транзакций.
- Транзакции с данными работа с транзакциями с данными (*Data Transaction*).
- Токены выборка транзакций с токенами. При выборе доступны контекстные фильтры по типам операций с токенами (например, перевод, аренда или эмиссия токенов).
- Разрешения выборка транзакций по пользовательским разрешениям. При выборе доступны контекстные фильтры по типам разрешений (например, майнинг, публикация контрактов или управление доступом).
- Группы выборка транзакций по операциям с группами доступа к конфиденциальным данным. При выборе доступны контекстные фильтры по типам операций (например, создание или редактирование группы доступа).
- Контракты выборка транзакций по операциям с контрактами. При выборе доступны контекстные фильтры по Docker-контрактам.
- неподтвержденные транзакции.
- Пользователи информация о пользователях. При выборе доступны контекстные фильтры по типам разрешений (например, майнинг, публикация контрактов или управление доступом).

## Токены

Раздел отображает баланс авторизованной учетной записи, а также позволяет переводить токены другим участникам сети, передавать токены в аренду и выпускать токены. Выпуск токенов требует разрешения «Управление токенами».

## Контракты

Раздел отображает информацию по существующим контрактам в сети, позволяет публиковать и запускать выбранные контракты. Доступна фильтрация в поисковой строке по параметрам транзакций. Для публикации контрактов нужно разрешение «Публикация контрактов».

## Передача данных

Раздел позволяет создавать транзакции с данными и просматривать информацию о таких транзакциях, а также создавать группы доступа к конфиденциальным данным и обмениваться ими внутри таких групп.

## Настройки сети

В разделе можно посмотреть информацию о нодах в сети, а также рассчитать сумму лизинга. Для этого вам нужно указать следующие данные:

- Адрес лизингового пула.
- Начало и конец расчётного периода лизинга.
- Процент выплат.

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.

---

**Примечание:** Средства должны оставаться в лизинге не менее 1000 блоков без движения, прежде чем начнут приносить прибыль.

---

## Напишите нам

В свободной форме вы можете отправить комментарии и отзывы в нашу техническую поддержку. Все обращения будут обязательно рассмотрены.

## Адрес

Переход на страничку осуществляется при нажатии кнопки Адрес в правом верхнем углу интерфейса. Раздел отображает информацию об аккаунте пользователя (публичный и приватный ключи, секретная фраза). Также в настройках выдаются разрешения другим пользователям. Для выдачи необходимо разрешение «Управление разрешениями». Если блокчейнадрес не привязан к аккаунту клиента, то на этой страничке его можно создать или указать адрес ноды из ключевого хранилища.

## Настройки аккаунта

Переход на страничку осуществляется при нажатии иконки с Email в правом верхнем углу интерфейса. Раздел отображает информацию о текущей версии клиента и позволяет сменить язык интерфейса.

## Блоки, транзакции, сообщения

## 12.1 Блоки

В этом разделе приведена структура хранения блоков в блокчейнплатформе Waves Enterprise.

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Version (0x02 for Genesis block, 0x03 for common block)	Byte	1
2	Timestamp	Long	8
3	Parent block signature	Bytes	64
4	Consensus block length (always 40 bytes)	Int	4
5	Base target	Long	8
6	Generation signature*	Bytes	32
7	Transactions block length (N)	Int	4
8	Transaction #1 bytes	Bytes	M1
...	...	...	...
8 + (K - 1)	Transaction #K bytes	Bytes	MK
9 + (K - 1)	Generator's public key	Bytes	32
10 + (K - 1)	Block's signature	Bytes	64

Генерирующая подпись (Generation signature) вычисляется на основе хеша (Blake2b256) от следующих полей:

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Previous block's generation signature	Bytes	32
2	Generator's public key	Bytes	32

Подпись блока вычисляется на основе следующих данных:

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Version (0x02 for Genesis block, 0x03 for common block)	Byte	1
2	Timestamp	Long	8
3	Parent block signature	Bytes	64
4	Consensus block length (always 40 bytes)	Int	4
5	Base target	Long	8
6	Generation signature*	Bytes	32
7	Transactions block length (N)	Int	4
8	Transaction #1 bytes	Bytes	M1
...	...	...	...
8 + (K - 1)	Transaction #K bytes	Bytes	MK
9 + (K - 1)	Generator's public key	Bytes	32

## 12.2 Транзакции

В этом разделе приведено описание формата данных в транзакциях, а также структура хранения транзакций в блокчейнплатформе Waves Enterprise. Для некоторых типов транзакций введено версионирование.

### 12.2.1 Формат данных в транзакциях

Все транзакции используют поле `timestamp`, содержащее временную метку в формате **Unix Timestamp** в миллисекундах.

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**.

Поля «ключзначение» секции `data` транзакции 12 и секции `params` транзакции 104 поддерживают 4 типа данных: *string*, *integer*, *boolean*, *binary*.

На странице проекта в [GitHub](#) доступны протофайлы, определяющие формат ответа, возвращаемого нодой.

### 12.2.2 Структура хранения транзакций

Значения в jsonзапросах для подписания и отправки транзакций в блокчейн являются примерными. Перед отправкой запроса на подписание транзакции проверьте соответствие параметров запроса актуальным данным. Например, если вы отправляете транзакцию в Mainnet, необходимо убедиться, что вы указали правильный размер комиссии за транзакцию. В противном случае транзакция не пройдет валидацию, и нода вернет ошибку 105 InvalidFee.

Дополнительная информация о комиссиях за транзакции приведена в разделе *Комиссии в сети «Waves Enterprise Mainnet»*

Таблица 1: Типы транзакций

№	Тип транзакции	Описание
1	<i>Genesis transaction</i>	Первоначальная привязка баланса к адресам, создаваемым при старте блокчейна нод
3	<i>Issue Transaction</i>	Выпуск токенов
4	<i>Transfer Transaction</i>	Перевод токенов
5	<i>Reissue Transaction</i>	Перевыпуск токенов
6	<i>Burn Transaction</i>	Сжигание токенов
8	<i>Lease Transaction</i>	Передача токенов в аренду
9	<i>Lease Cancel Transaction</i>	Отмена аренды токенов
10	<i>Create Alias Transaction</i>	Создание псевдонима
11	<i>MassTransfer Transaction</i>	Массовый перевод токенов. Указана минимальная комиссия
12	<i>Data Transaction</i>	Транзакция с данными в виде полей с парой ключ-значение. Указана минимальная комиссия
13	<i>SetScript Transaction</i>	Транзакция, привязывающая скрипт с RIDEконтрактом к аккаунту
14	<i>Sponsorship Transaction</i>	Транзакция, подписывающая спонсорский ассет
15	<i>SetAssetScript</i>	Транзакция, привязывающая скрипт с RIDEконтрактом к ассету
101	<i>Genesis Permission Transaction</i>	Назначение первого администратора сети для дальнейшей раздачи прав
102	<i>Permission Transaction</i>	Выдача/отзыв прав у аккаунта
103	<i>CreateContract Transaction</i>	Создание Dockerконтракта
104	<i>CallContract Transaction</i>	Вызов Dockerконтракта
105	<i>ExecutedContract Transaction</i>	Выполнение Dockerконтракта
106	<i>DisableContract Transaction</i>	Отключение Dockerконтракта
107	<i>UpdateContract Transaction</i>	Обновление Dockerконтракта
110	<i>GenesisRegisterNode Transaction</i>	Регистрация ноды в генезисблоке при старте блокчейна
111	<i>RegisterNode Transaction</i>	Регистрация новой ноды в сети
112	<i>CreatePolicy Transaction</i>	Создание группы доступа к конфиденциальным данным
113	<i>UpdatePolicy Transaction</i>	Изменение группы доступа
114	<i>PolicyDataHash Transaction</i>	Отправка в сеть хеша данных
120	<i>AtomicTransaction Transaction</i>	Упаковывание нескольких транзакций в одну для атомарного выполнения

**Важно:** Транзакция 101 также предназначена для включения роли *sender*. Эта роль может быть выдана участникам как при помощи 101, так и 102 транзакции. Однако если хотя бы один участник не был назначен *sender* при помощи 101 транзакции, выдать эту роль при помощи 102 транзакции будет невозможно.

## 1. Genesis transaction

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+		Byte
fee	+		Long
timestamp	+	+	Long
signature	+		ByteStr
recipient	+	+	ByteStr
amount	+	+	Long
height	+		

## 3. issueTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
assetId		+		ByteStr
name	+	+	+	Array[Byte]
quantity	+	+	+	Long
reissuable	+	+	+	Boolean
decimals	+	+	+	Byte
description	+	+	+	Array[Byte] ( <i>base58</i> )
chainId		+	+	Byte
script	+ (opt)	+	+	Bytes
password	+ (opt)			String
height		+		

### JSON для вызова метода sign

```
{
  "type": 3,
  "version": 2,
  "name": "Test Asset 1",
  "quantity": 10000000000,
  "description": "Some description",
  "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
  "password": "",
  "decimals": 8,
  "reissuable": true,
  "fee": 100000000
}
```

### Broadcasted JSON



```
{
  "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. TransferTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
recipient	+	+	+	ByteStr
assetId	+ (opt)	+	+	ByteStr
feeAssetId	+ (opt)	+	+	Bytes
amount	+	+	+	Long
attachment	+ (opt)	+	+	Bytes ( <i>base58</i> )
password	+ (opt)			String
height		+		
atomicBadge	+	+	+	

#### JSON для вызова метода sign

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

**Broadcasted JSON**

```
{
  "senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "amount": 200000000,
  "fee": 100000,
  "type": 4,
  "version": 2,
  "attachment": "3uaRTtZ3taQtRSmquqeC1DniK3Dv",
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
  "feeAssetId": null,
  "proofs": [
    "2hRxJ2876CdJ498UCpErNfDSYdt2mTK4XUnmZNqZiq63RupJs5WTrAqR46c4rLQdq4toBZk2tSYCeAQWEQyi72U6"
  ],
  "assetId": null,
  "recipient": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
  "id": "757aQzJiQZRfVRuJNnP3L1d369H2oTjUEazwtYxGngCd",
  "timestamp": 1558952680800
}
```

**5. ReissueTransaction**

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
assetId	+	+	+	ByteStr
quantity	+	+	+	Long
reissuable	+	+	+	Boolean
password	+ (opt)			String
height				

**JSON для вызова метода sign**

```
{
  "type": 5,
  "version": 2,
  "quantity": 10000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
  "reissuable": true,
  "fee": 100000001
}
```

**Broadcasted JSON**

```
{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "quantity": 10000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "chainId": 84,
  "proofs": [
    "3gmgGM6rYpxuuR5QvJkugPsERG7yWYF7JN6QzpUGJwT8Lw6SUHkzzk8R22A7cGQz7TQQ5NifKxvAQzwPyDQbwmBg" ],
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
  "fee": 100000001,
  "id": "GsNvk15Vu4kqtRmMSpYW21WzgJpZrLBwjCREHWuwnvh5",
  "type": 5,
  "version": 2,
  "reissuable": true,
  "timestamp": 1551447859299,
  "height": 1190
}
```

## 6. BurnTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
assetId	+	+	+	ByteStr
quantity	+		+	Long
amount		+		Long
password	+ (opt)			String
height				

### JSON для вызова метода sign

```
{
  "type": 6,
  "version": 2,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "password": "",
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
  "quantity": 1000,
  "fee": 100000,
  "attachment": "string"
}
```

### Broadcasted JSON

```
{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "amount": 1000,

```

(continues on next page)

(продолжение с предыдущей страницы)

```

"sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"chainId": 84,
"proofs": [
↪ "kzTwsNXjJkzk6dpFFZZXyeimYo6iLTVbCnCXBD4xBtyrNjysPqZfGKk9NdJUTP3xeAPhtEgU9hsdwzRV01hKMgS" ],
"assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg",
"fee": 100000,
"id": "3yd2HZq7sgun7GakisLH88UeKcpYMUEL4sy57aprAN5E",
"type": 6,
"version": 2,
"timestamp": 1551448489758,
"height": 1190
}

```

## 8. LeaseTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
amount	+	+	+	Long
recipient	+	+	+	ByteStr
status		+		
password	+ (opt)			String
height		+		

### JSON для вызова метода sign

```

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

```

### Broadcasted JSON

```

{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "amount": 1000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "proofs": [
↪ "5jvmWKmU89HnxXFXNAd9X41zmiB5fSGoXMirsaJ9tNeyiCajmjm7MR48g789VucckQw2UExaVXfhdsEBuUrchvrq" ],
  "fee": 100000,
  "recipient": "3N1ksBqc6uSksdiYjCzMtvEpiHhS1JjkbPh",
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "id": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
    "type": 8,
    "version": 2,
    "timestamp": 1551449299545,
    "height": 1190
  }

```

## 9. LeaseCancelTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
leaseId	+ (txId)	+	+	Byte
lease		+		
password	+ (opt)			String
height		+		

### JSON для вызова метода sign

```

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

```

### Broadcasted JSON

```

{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "leaseId": "6Tn7ir9MycHW6Gq2F2dGok2stokSwXJadPh4hW8eZ8Sp",
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "chainId": 84,
  "proofs": [
    ↪ "2Gns72hraH5yay3eiWeyHQEA1wTqiiAztalJHinEYX91FEv62HFW38Hq89GnsEJFHUvo9KHYtBBrb8hgTA9wN7DM" ],
  "fee": 100000,
  "id": "9vhxB2ZDQcqiumhQbCPnAoPBLuir727qgJhFeBNmPwmu",
  "type": 9,
  "version": 2,
  "timestamp": 1551449835205,
  "height": 1190
}

```

## 10. CreateAliasTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
alias	+	+	+	Bytes
password	+ (opt)			String
height		+		

### JSON для вызова метода sign

```
{
  "type": 10,
  "version": 2,
  "fee": 100000,
  "sender": "3N9vL3apA4j2L5PoJHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "alias": "hodler"
}
```

### Broadcasted JSON

```
{
  "type": 10,
  "id": "DJTtaiMpb7eLuPW5GcE4ndeE8jWswPjx8gPYmbZPJjpag",
  "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "fee": 0,
  "timestamp": 1549290335781,
  "signature":
  ↪ "2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbBDzRZYEY",
  "proofs": [
  ↪ "2qYepod9DhpxVad1yQDbv1QzU4KLKcbjjdtGY7De2272K76nbQfaXsRnyd31hUE8bhvLjjpHRdtoLVzbBDzRZYEY" ],
  "version": 1,
  "alias": "testperson4",
  "height": 59245
}
```

## 11. MassTransferTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
assetId	+ (opt)	+	+	ByteStr
attachment	+ (opt)	+	+	(base58)
transfers	+	+	+	List[Transfer]
transferCount		+	+	
totalAmount		+		
password	+ (opt)			String
height		+		

### JSON для вызова метода sign

```
{
  "type": 11,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "fee": 2000000,
  "version": 1,
  "transfers":
  [
    { "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB", "amount": 100000 },
    { "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUetrRfFHM", "amount": 100000 }
  ]
}
```

### Broadcasted JSON

```
{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "fee": 2000000,
  "type": 11,
  "transferCount": 2,
  "version": 1,
  "totalAmount": 200000,
  "attachment": "",
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "proofs": [
    ↪ "2gWpMWdgZCjbygCX5US3aAfftKtGPRSK3aWGJ6RDnWJf9hend5sBFAgY6u3Mp4jN8cqwaJ5o8qrKNedGN5CPN1GZ" ],
  "assetId": null,
  "transfers":
  [
    {
      "recipient": "3MtHszoTn399NfsH3v5foeEXRRrchEVtTRB",
      "amount": 100000
    },
    {

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "recipient": "3N7BA6J9VUBfBRutuMyjF4yKTUEtrRFfHMc",
    "amount": 100000
  },
  ],
  "id": "D9jUSHHcJqVAvkFMiRfDBhQbUzoSfQqd9cjaunMmtjdu",
  "timestamp": 1551450279637,
  "height": 1190
}

```

## 12. DataTransaction

**Предупреждение:** Транзакция имеет ограничения:

1. Количество данных в секции «data» передаваемого JSON должно быть не более 100 пар "key": "value",

```

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

```

2. Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

**Подсказка:** Параметр senderPublicKey не требуется указывать, если подписывается транзакция, в которой автор и отправитель совпадают.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size (Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
senderPublicKey	+ (opt)	+	+	PublicKeyAccount	3264
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+		Byte	1
authorPublicKey		+	+	PublicKeyAccount	3264
author	+	+			3264
data	+	+	+		3264
password	+ (opt)			String	32767
height		+			8

### JSON для вызова метода sign

```

{
  "type": 12,

```

(continues on next page)



(продолжение с предыдущей страницы)

```

"version": 1,
"sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"password": "",
"senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
"author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
"data": [
  {
    "key": "objectId",
    "type": "string",
    "value": "obj:123:1234"
  }
],
"fee": 100000
}

```

**Broadcasted JSON**

```

{
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "authorPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "data":
  [
    {
      "type": "string",
      "value": "obj:123:1234",
      "key": "objectId"
    }
  ],
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "proofs": [
    ↪ "2T7WQm5XW8cFHfiFkdDEic9oNiT7aFiH3TyKkARERopr1VJvzRKqHAVnQ3eiYZ3uYN8uQnPopQEH4XV8z5SgSwsf" ],
  "author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "fee": 100000,
  "id": "7dMMCQNTusahZ7DWtNGjCwAhRYpjaH1hsepRMbpn2BkD",
  "type": 12,
  "version": 1,
  "timestamp": 1551680510183
}

```

### 13. SetScriptTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
chainId		+	+	Byte
version	+	+	+	Byte
script	+ (opt)	+	+	Bytes
name	+	+	+	Array[Byte]
description	+ (opt)	+	+	Array[Byte] ( <i>base58</i> )
password	+ (opt)			String
height		+		

#### JSON для вызова метода sign

```
{
  "type": 13,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "fee": 1000000,
  "name": "faucet",
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAAACdHgG+RXSszQ=="
}
```

#### Broadcasted JSON

```
{
  "type": 13,
  "id": "HPDypnQJHJskN8kwszF8rck3E5tQiuM1fEN42w6PLmt",
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "fee": 1000000,
  "timestamp": 1545986757233,
  "proofs": [
    ↪ "2QiGYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTzobjmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFgG" ],
  "chainId": 84,
  "version": 1,
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAAACdHgG+RXSszQ==",
  "name": "faucet",
  "description": "",
  "height": 3805
}
```

## 14. SponsorshipTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
assetId	+ (opt)	+	+	ByteStr
fee	+	+	+	Long
isEnabled	+	+	+	Boolean
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
chainId		+	+	Byte
version	+	+	+	Byte
script	+ (opt)	+	+	Bytes
name	+	+	+	Array[Byte]
description	+ (opt)	+	+	Array[Byte] ( <i>base58</i> )
password	+ (opt)			String
height		+		

### JSON для вызова метода sign

```
{
  "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
  "assetId": "G16FvJk9vabwxjQswH9CQAhbZzn3QrwqWjwnZB3qNVox",
  "fee": 100000000,
  "isEnabled": false,
  "type": 14,
  "password": "1234",
  "version": 1
}
```

### Broadcasted JSON

```
{
  "type": 14,
  "id": "Ht6kpnQJHJskN8kwszF8rck3E5tQiuM1fEN42wGfdk7",
  "sender": "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t",
  "senderPublicKey": "Gt55fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUophy89",
  "fee": 100000000,
  "assetId": "G16FvJk9vabwxjQswH9CQAhbZzn3QrwqWjwnZB3qNVox",
  "timestamp": 1545986757233,
  "proofs": [
    ↪ "5TfgYS2dqh8QyN7Vu2tAYaioX5WM6rTSDPGbt4zrWS7QKTz0jmR2kjppvGNj4tDPsYPbcDunqBaqhaudLyMeGFh7" ],
  "chainId": 84,
  "version": 1,
  "isEnabled": false,
  "height": 3865
}
```

## 15. SetAssetScriptTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
chainId		+	+	Byte
assetId	+	+	+	ByteStr
script	+ (opt)	+	+	Bytes
password	+ (opt)			String
height		+		

## JSON для вызова метода sign

```
{
  "type": 15,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "password": "",
  "fee": 100000000,
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAAACdHgG+RXSzQ==",
  "assetId": "7bE3JPwZC3QcN9edctFrLAKYysjfMEk1SDjZx5gitSGg"
}
```

## Broadcasted JSON

```
{
  "type": 15,
  "id": "CQpEM9AEDvgxKfgWLH2HxE82iAzpXrtqsDDcgZGPAF9J",
  "sender": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "senderPublicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "fee": 100000000,
  "timestamp": 1549448710502,
  "proofs": [
    ↪ "64eodpuXQjaKQQ4GJBaBrqiBtmkjSxseKC97gn6EwB5kZtMr18mAUHPRkZaHJeJxaDyLzGEZKqhYoUknWfNhXnkf" ],
  "version": 1,
  "chainId": 84,
  "assetId": "DnK5Xfi2wXUJx9BjK9X6ZpFdTLdq2GtWH9pWrcxcmrhB",
  "script": "base64:AQQAAAAHJG1hdGNoMAUAAAAACdHgG+RXSzQ==",
  "height": 61895
}
```

**101. GenesisPermitTransaction**

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte	
id	+		Byte	
fee	+		Long	
timestamp	+	+	Long	
signature	+		ByteStr	
target	+	+	ByteStr	
role	+	+	String	
height				

**102. PermitTransaction**

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee		+		Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+		+	Byte
target	+	+	+	ByteStr
PermissionOp			+	PermissionOp
opType	+	+		String
role	+	+		String
dueTimestamp	+ (opt)	+		Option[Long]
password	+ (opt)			String
height		+		
atomicBadge	+	+	+	

**JSON для вызова метода sign**

```
{
  "type": 102,
  "sender": "3GLWx8yUFcNSL3DER8kZyE4TypAyNiEYsKG",
  "password": "",
  "senderPublicKey": "4WnvQPit2DiliYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
  "fee": 0,
  "target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL",
  "opType": "add",
  "role": "contract_developer",
  "dueTimestamp": null,
  "version": 1,
}
```

**Broadcasted JSON**

```
{
  "senderPublicKey": "4WnvQPit2DiliYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
```

(continues on next page)

(продолжение с предыдущей страницы)

```

"role": "contract_developer",
"sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
"proofs": [
  "5ABJCRTKGo6jmdZCRWcLQc257CCeczmcjmtfJmbBE7TP3KsVkwvisH9kEkfYPckVCzEMKZTCd3LKAPcN8o4Git3j"
],
"fee": 0,
"opType": "add",
"id": "8zVUH7nsDCcpwyfxiq8DCTgqL7Q23FW1KWepB9EZcFG6",
"type": 102,
"dueTimestamp": null,
"timestamp": 1559048837487,
"target": "3GPtj5osoYqHpyfmsFv7BMiyKsVzbG1ykfL"
}

```

### 103. CreateContractTransaction

**Предупреждение:** Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

Поле feeAssetId опционально и используется только для *gRPC* контрактов (значение поля version = 2).

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
senderPublicKey		+	+	PublicKeyAccount	3264
password	+ (opt)			String	32767
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+	+	Byte	1
feeAssetId	+ (opt)			Byte	1
image	+	+	+	Array[Byte]	32767
imageHash	+	+	+	Array[Byte]	32767
contractName	+	+	+	Array[Byte]	32767
params	+	+	+	List[DataEntry[_]]	32767
height		+			8
atomicBadge	+	+	+		32767

### JSON для вызова метода sign

```

{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2",
  "password": "",
  "params": [],
  "type": 103,

```

(continues on next page)

(продолжение с предыдущей страницы)

```
"version": 1,
}
```

### Broadcasted JSON

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhAv38Dws5skqDsMVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "params": [],
  "height": 1619
}
```

## 104. CallContractTransaction

**Предупреждение:** Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

В поле `contractVersion` указывается версия контракта, значение 1 для нового контракта, значение 2 для обновленного контракта. Данное поле доступно только для второй версии транзакции "version": 2,. Контракт обновляется при помощи 107 транзакции. При создании контракта автоматически создается транзакция 104, вызывающая контракт для его проверки. Если контракт не выполнен или выполнен с ошибкой, то транзакции 103 и 104 отбрасываются и не попадают в блок.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
senderPublicKey		+	+	PublicKeyAccount	3264
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+	+	Byte	1
contractVersion	+	+	+	Byte	1
contractId	+	+	+	ByteStr	32767
params	+	+	+	List[DataEntry[_]]	32767
height		+			8
password	+ (opt)			String	32767
atomicBadge	+	+	+		32767

### JSON для вызова метода sign

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

### Broadcasted JSON

```
{
  "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
    }
  ]
}
```



## 105. ExecutedContractTransaction

**Предупреждение:** Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+		Byte
sender	+		PublicKeyAccount
senderPublicKey	+	+	PublicKeyAccount
fee	+		Long
timestamp	+	+	Long
proofs	+	+	List[ByteStr]
version	+	+	Byte
tx	+	+	ExecutableTransaction
results	+	+	List[DataEntry[_]]
height	+		
password	+ (opt)		String
atomicBadge	+	+	

## Broadcasted JSON

```
{
  "type": 105,
  "id": "38GmSVC5s8Sjeybzfe9RQ6p1Mb6ajb8LYJDcep8G8Umj",
  "sender": "3N3YTjtNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "password": "",
  "fee": 500000,
  "timestamp": 1550591780234,
  "proofs": [
    ↪ "5whBipAWQgFvm3mNZe6GDd9Ky8199C9qNxLBHqDNmVAUJW9gLf7t9LBQDi68CKT57dzmnP JpJkrwKh2HBSwUer6" ],
  "version": 1,
  "tx": {
    "type": 103,
    "id": "ULc9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTjtNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
    "fee": 500000,
    "timestamp": 1550591678479,
    "proofs": [
      ↪ "yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
    "version": 1,
    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
  },
  "results": [],
  "height": 1619
}
```

**106. DisableContractTransaction**

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version	+	+	+	Byte
contractId	+	+	+	ByteStr
height		+		
password	+ (opt)			String
atomicBadge	+	+	+	

**JSON для вызова метода sign**

```
{
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "password": "",
  "contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
  "fee": 500000,
  "type": 106,
  "version": 1,
}
```

**Broadcasted JSON**

```
{
  "type": 106,
  "id": "8Nw34YbosEVhCx18pd81HqYac4C2pGjyLKck8NhSoGYH",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 500000,
  "proofs": [
    "5GqPQkuRvG6LPXgPoCr9FogAdmhAaMbyFb5UfjQPUKdSc6BLuQsz75LAWix1ok2Z6PC5ezPpjzqnr15i3RQmaEc" ],
  "version": 1,
  "contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
  "height": 1632
}
```

**107. UpdateContractTransaction**

**Предупреждение:** Байтовое представление транзакции после подписания не должно превышать размер в 150 КБ.

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type	Size(Bytes)
type	+	+	+	Byte	1
id		+		Byte	1
sender	+	+		PublicKeyAccount	3264
senderPublicKey		+	+	PublicKeyAccount	3264
image	+	+	+	Array[Bytes]	32767
imageHash	+	+	+	Array[Bytes]	32767
fee	+	+	+	Long	8
timestamp	+ (opt)	+	+	Long	8
proofs		+	+	List[ByteStr]	32767
version	+	+	+	Byte	1
contractId	+	+	+	ByteStr	32767
height		+			8
password	+ (opt)			String	32767
atomicBadge	+	+	+		32767

### JSON для вызова метода sign

```
{
  "image": "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender": "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "password": "",
  "fee": 100000000,
  "contractId": "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRcWA",
  "imageHash": "0e5d280b9acf6efd8000184ad008757bb967b5266e9ebf476031fad1488c86a3",
  "type": 107,
  "version": 1
}
```

### Broadcasted JSON

```
{
  "senderPublicKey":
  ↪ "5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "tx": {
    "senderPublicKey":
    ↪ "5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
    "image": "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
    "sender": "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
    "proofs": [
    ↪ "3tNsTyteeZrxEbVSv5zPT6dr247nXsVWR5v7Khx8spypgZQUdorCQZV2guTomutUTcyxhJUjNkQW4VmSgbCtgm1Z"],
    "fee": 0,
    "contractId": "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRcWA",
    "id": "HdZdhXVveMT1vYzGTviCoGQU3aH6ZS3YtFpYujWeGCH6",
    "imageHash": "17d72ca20bf9393eb4f4496fa2b8aa002e851908b77af1d5db6abc9b8eae0217",
    "type": 107, "version": 1, "timestamp": 1572355661572},
    "sender": "3HfRBedCpWi3vEzFSKEZDFXkyNWbWLWQmmG",
    "proofs": [
    ↪ "28ADV8miUVN5EFjhqefJ6MADsXYjbxA3TsxSwFVs18jXAsHVaBczvnyoUSaYJsJRnmaWgXbpbduccRxpKGTs6tro"],
    "fee": 0, "id": "7niVY8mjzeKqLBePvhTxFRfLu7BmcVvfqaqtbWAN8AA2",
    "type": 105,
    "version": 1,
    "results": [],
    "timestamp": 1572355666866
  }
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

}
}

```

## 110. GenesisRegisterNodeTransaction

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+		Byte
fee	+		Long
timestamp	+	+	Long
signature	+		Bytes
version		+	Byte
targetPubKey	+	+	
height	+		

## 111. RegisterNodeTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+		Byte
sender	+	+		PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+		Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
version			+	Byte
targetPubKey	+	+	+	PublicKeyAccount
nodeName	+	+	+	String
opType	+	+	+	
height		+		
password	+ (opt)			String

### JSON для вызова метода sign

```

{
  "type": 111,
  "opType": "add",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "",
  "targetPubKey": "apgJP9atQccdBPAGJPwH3NBVqYXrapgJP9atQccdBPAGJPwHapgJP9atQccdBPAGJPwHDKkh6A8",
  "nodeName": "Node #1",
  "fee": 500000,
}

```

## 112. CreatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
policyName	+	+	+	String
recipients	+	+	+	Array[Byte]
owners	+	+	+	Array[Byte]
fee	+	+	+	Long
timestamp	+ (opt)	+	+	Long
proofs		+	+	List[ByteStr]
height			+	Long
description	+	+	+	String ( <i>base58</i> )
password	+ (opt)			String
version	+	+	+	Byte
atomicBadge	+	+	+	

## JSON для вызова метода sign

```
{
  "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": 1,
}
```

## 113. UpdatePolicyTransaction

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
policyId	+	+	+	String
recipients	+	+	+	Array[Byte]
owners	+	+	+	Array[Byte]
fee	+	+	+	Long
timestamp		+	+	Long
proofs		+	+	List[ByteStr]
height			+	Long
opType	+	+	+	
description	+	+	+	String ( <i>base58</i> )
password	+			String
version	+	+	+	Byte
atomicBadge	+	+	+	

## JSON для вызова метода sign

```
{
  "policyId": "7wphGbhqbmUgzun5wzgcwqtViTiMdFezSa11fxRV58Lm",
  "password": "sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAoohUoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "3NwJfjG5RpaDfxEhkwXgwD7oX21NMFCxJHL"
  ],
  "fee": 15000000,
  "opType": "add",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 113,
  "version": 1,
}
```

## 114. PolicyDataHashTransaction

Когда пользователь отправляет конфиденциальные данные в сеть при помощи *POST /privacy/sendData*, нода автоматически формирует транзакцию 114.

Field	Broadcasted JSON	Blockchain state	Type
type	+	+	Byte
id	+	+	Byte
sender	+	+	PublicKeyAccount
senderPublicKey	+	+	PublicKeyAccount
policyId	+	+	String
dataHash	+	+	String
fee	+	+	Long
timestamp	+	+	Long
proofs	+	+	List[ByteStr]
height		+	Long
version	+	+	Byte
atomicBadge	+	+	

## 120. AtomicTransaction

Транзакция помещает в контейнер другие транзакции для их атомарного выполнения. Транзакция поддерживает следующие типы транзакций:

- 4 *Трансфер ассета*, версия 3
- 102 *Добавление / удаление прав*, версия 2
- 103 *Создание контракта*, версия 3
- 104 *Вызов контракта*, версия 4
- 105 *Исполнение контракта*, версии 1 и 2
- 106 *Деактивация контракта*, версия 3
- 107 *Обновление контракта*, версия 3
- 112 *Создание политики приватности*, версия 3
- 113 *Обновление политики приватности*, версия 3
- 114 *Добавление приватных данных*, версия 3

Field	JSON to sign	Broadcasted JSON	Blockchain state	Type
type	+	+	+	Byte
id		+	+	Byte
sender	+	+	+	PublicKeyAccount
senderPublicKey		+	+	PublicKeyAccount
fee	+	+	+	Long
timestamp		+	+	Long
proofs		+	+	List[ByteStr]
height			+	Long
transactions	+	+	+	
miner		+	+	String
password	+			String
version	+		+	Byte

## JSON для вызова метода sign

```
{
  "sender": sender_0,
  "transactions": [
    signed_transfer_tx,
    signed_transfer_tx2
  ],
  "type": 120,
  "version": 1,
  "password": "lskjbJJk$%^#298",
  "fee": 0,
}
```

## Пример запроса

```
{'sender': '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP',
'transactions': [
  {'senderPublicKey':
    ↪ '5nGi8XoiGjjybPmjLNy1k2bus4yXLaeuA3Hb7BikwD9tboFWFXJYUmt05Jo0x76c3pp2Mr1LjgodUJuxryCJofQ',
    ↪ 'amount': 10, 'fee': 10000000, 'type': 4, 'version': 3, 'atomicBadge': {'trustedSender':
    ↪ '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP'}, 'attachment': '', 'sender':
    ↪ '3Mv79dyPX2cvLtRXn1MDDWiCZMBrkW9d97c', 'feeAssetId': None, 'proofs': [
    ↪ 'XQ7iAqkarmm14AATc2Y9cR3Z9WnirSH4kH6RUL4QdT82rEwsmWBbBfWrADLE9o4cp2VR39W6b3vdrwFgg1dX7m3'],
    ↪ 'assetId': None, 'recipient': '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'id':
    ↪ 'FZ59wAZnkFUqXjn61vvyj59fRa3cuS6nzuW3vqoRMsM5', 'timestamp': 1602857131666}, {'senderPublicKey
    ↪ ': '56rV5kcR9SBsxQ9LtNrmp6V72S4BDkZUJaA6ujZswDneDmCTmeSG6UE2FQP1rPXdfpWQNunRw4aijGXxoK3o4puj',
    ↪ 'amount': 20, 'fee': 10000000, 'type': 4, 'version': 3, 'atomicBadge': {'trustedSender':
    ↪ '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP'}, 'attachment': '', 'sender':
    ↪ '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'feeAssetId': None, 'proofs': [
    ↪ '5KaXUFan2JD6VsJeGNyBCxEwqCjUF1nASAzxnPZzBydXA5RJyXQGaL6N9MQ8GDNori1nXw5FsDLBqc3CPM3ezsk'],
    ↪ 'assetId': None, 'recipient': '3Mv79dyPX2cvLtRXn1MDDWiCZMBrkW9d97c', 'id':
    ↪ '8GTqE1cc6zTVxYgQxgHJWJitVsDFRc6GmU5FJcnp5gu2', 'timestamp': 1602857132314}
  ],
'type': 120,
'version': 1}
```

## 12.3 Сетевые сообщения

В этом разделе приведена структура сетевых сообщений в блокчейнплатформе Waves Enterprise.

### 12.3.1 Network message

Все сетевые сообщения, за исключением Handshake, базируются на следующей структуре:

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Payload	Bytes	N



Magic Bytes следующие: 0x12, 0x34, 0x56, 0x78. Контрольная сумма полезной нагрузки это первые 4 байта от `_FastHash_` от байтов `_Payload_`. FastHash это хешфункция Blake2b256(data).

### 12.3.2 Handshake message

Handshake сообщение предназначена для первичного обмена данными между двумя нодами. Авторизованный Handshake содержит блокчейнадрес владельца ноды и подпись. Неподписанные Handshake сообщения не принимаются.

#### Авторизованный Handshake

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	HandshakeType	byte	1
2	Application name length (N)	Byte	1
3	Application name (UTF8 encoded bytes)	Bytes	N
4	Application version major	Int	4
5	Application version minor	Int	4
6	Application version patch	Int	4
7	Consensus name length (P)	Byte	1
8	Consensus name length (UTF8 encoded bytes)	Bytes	P
9	Node name length (M)	Byte	1
10	Node name (UTF8 encoded bytes)	Bytes	M
12	Node nonce	Long	8
13	Declared address length (K) or 0 if no declared address was set	Int	4
14	Declared address bytes (if length is not 0)	Bytes	K
15	Peer port	Int	4
16	Node owner address	Bytes	26
17	Signature	Bytes	64

### 12.3.3 GetPeers message

GetPeers сообщение отправляется для запроса сетевых адресов участников сети.

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x01)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4

### 12.3.4 Peers message

Peers сообщение является ответом на запрос GetPeers.

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x02)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Peers count (N)	Int	4
7	Peer #1 IP address	Bytes	4
8	Peer #1 port	Int	4
...	...	...	...
$6 + 2 * N - 1$	Peer #N IP address	Bytes	4
$6 + 2 * N$	Peer #N port	Int	4

### 12.3.5 GetSignatures message

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x14)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block IDs count (N)	Int	4
7	Block #1 ID	Bytes	64
...	...	...	...
$6 + N$	Block #N ID	Bytes	64

### 12.3.6 Signatures message

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x15)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block signatures count (N)	Int	4
7	Block #1 signature	Bytes	64
...	...	...	...
$6 + N$	Block #N signature	Bytes	64

### 12.3.7 GetBlock message

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x16)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block ID	Bytes	64

### 12.3.8 Block message

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x17)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Block bytes (N)	Bytes	N

### 12.3.9 Score message

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x18)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Score (N bytes)	BigInt	N

### 12.3.10 Transaction message

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x19)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Transaction (N bytes)	Bytes	N

### 12.3.11 Checkpoint message

Порядковый номер поля	Поле	Тип	Размер поля в байтах
1	Packet length (BigEndian)	Int	4
2	Magic Bytes	Bytes	4
3	Content ID (0x64)	Byte	1
4	Payload length	Int	4
5	Payload checksum	Bytes	4
6	Checkpoint items count (N)	Int	4
7	Checkpoint #1 height	Long	8
8	Checkpoint #1 signature	Bytes	64
...	...	...	...
$6 + 2 * N - 1$	Checkpoint #N height	Long	8
$6 + 2 * N$	Checkpoint #N signature	Bytes	64

---

## Смартконтракты Docker

---

Платформа Waves Enterprise предоставляет возможность разработки и использования Тьюрингполных смартконтрактов.

### 13.1 Смартконтракты на платформе Waves Enterprise

Тьюрингполные смартконтракты позволяют реализовать любую логику, заложенную в программный код. Для отделения запуска и работы самих смартконтрактов от платформы Waves Enterprise используется контейнеризация на базе [Docker](#). При этом для написания смартконтракта может использоваться любой язык программирования. Каждый смартконтракт запускается в Dockerконтейнере для изоляции его работы и управления ресурсами запущенного смартконтракта.

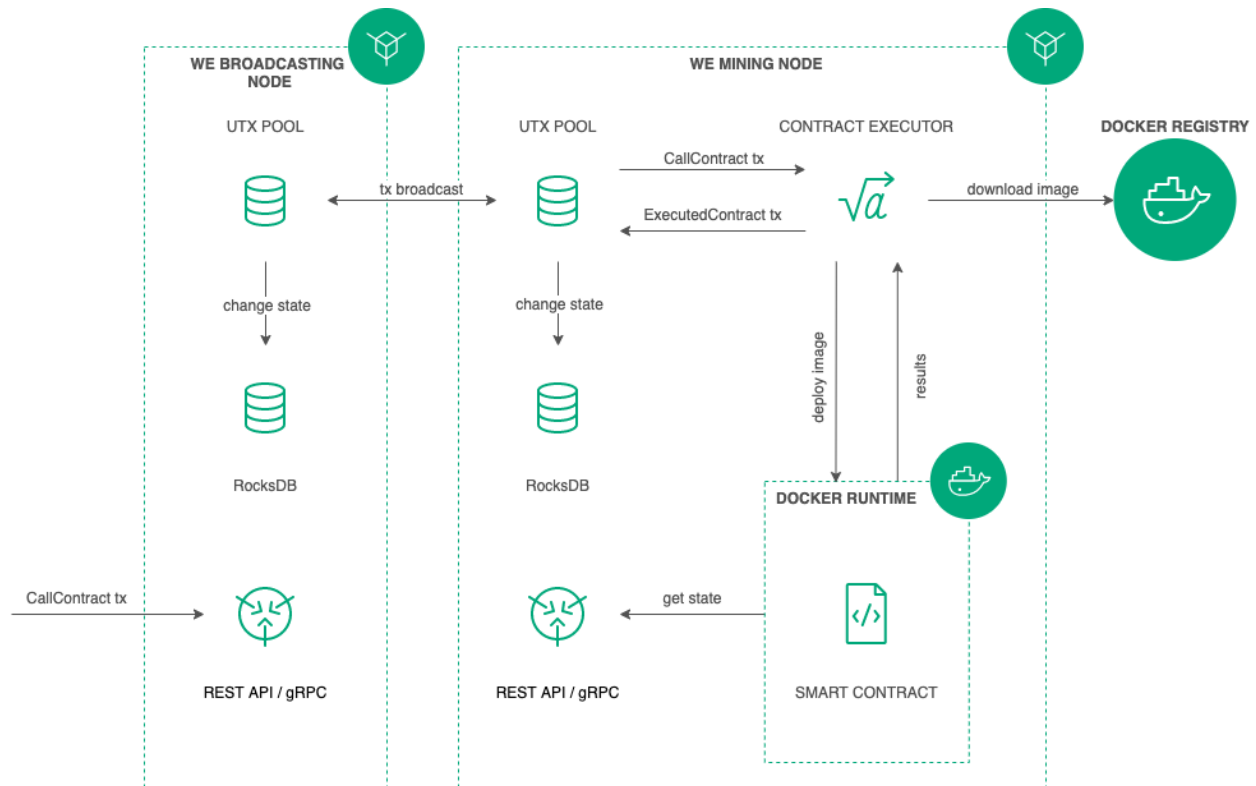
Когда смартконтракт запускается в блокчейн сети, его код нельзя произвольно изменить, заменить или запретить его выполнение без вмешательства в работу всей сети. Такое свойство делает смартконтракты практически незаменимым инструментом в блокчейн сети.

Для хранения смартконтрактов используется [Docker Registry](#) с доступом на чтение образов (Docker images) контрактов для машин с нодами. Компания Waves Enterprise предоставляет открытый репозиторий для Docker смартконтрактов, куда любой разработчик может добавить свой смартконтракт. Открытый репозиторий находится по адресу [registry.wavesenterprise.com/wavesenterprisepublic](https://registry.wavesenterprise.com/wavesenterprisepublic). Для добавления своего смартконтракта в открытый репозиторий необходимо написать заявку в нашу [техническую поддержку](#). После одобрения заявки смартконтракт будет добавлен в открытый репозиторий, и вы сможете вызывать его из клиента или REST API ноды.

Если вы используете приватную блокчейн сеть, то вам необходимо иметь свой собственный Dockerрепозиторий для публикации и вызова смартконтрактов.

Доступ к состоянию ноды может выполняться через [REST API ноды](#) или через [gRPC](#).

Смартконтракт может создавать и вызывать любой участник сети независимо от того, есть у него нодамайнер или нет. Достаточно [зарегистрироваться](#) в сети Mainnet через *клиентский интерфейс*.



## 13.2 Создание контракта

Создание смартконтракта начинается с подготовки Dockerобраза, который состоит из программного кода контракта, необходимого окружения и из специального сценарного файла Dockerfile. Подготовленный Dockerобраз собирается (build) и отправляется в Docker Registry. Для отправки нового смартконтракта создайте заявку на портале [технической поддержки](#). После проверки смартконтракта сотрудники технической поддержки размещают его в открытом Docker репозитории. В *конфигурационном файле ноды* по умолчанию уже присутствуют настройки секции `dockerengine` для работы с открытым Docker репозиторием, а также по умолчанию установлены рекомендованные значения параметров для оптимальной работы смартконтрактов в блокчейн сети Mainnet.

Пример Dockerfile при использовании REST API:

```
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"
```

Пример Dockerfile при использовании gRPC:

```
FROM python:3.9-rc-buster
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 /
```

(continues on next page)

(продолжение с предыдущей страницы)

```
RUN chmod +x run.sh
ENTRYPOINT ["/run.sh"]
```

Установка контракта реализуется через публикацию специальной (CreateContractTransaction) транзакции, содержащей ссылку на образ в Docker Registry. Для использования REST API или gRPC необходимо указать версию транзакции 103. После получения транзакции нода скачивает образ по указанной в поле «image» ссылке, образ проверяется и запускается в виде Dockerконтейнера.

## 13.3 Исполнение контракта

Исполнение смартконтрактов инициируется специальной (CallContractTransaction) транзакцией, в которой содержится идентификатор контракта и параметры для его вызова. По идентификатору транзакции определяется Dockerконтейнер. Контейнер запускается, если не был запущен ранее. В контейнер передаются параметры запуска контракта. Смартконтракты изменяют своё состояние через обновление пар ключ значение.

## 13.4 Параллельное исполнение контрактов

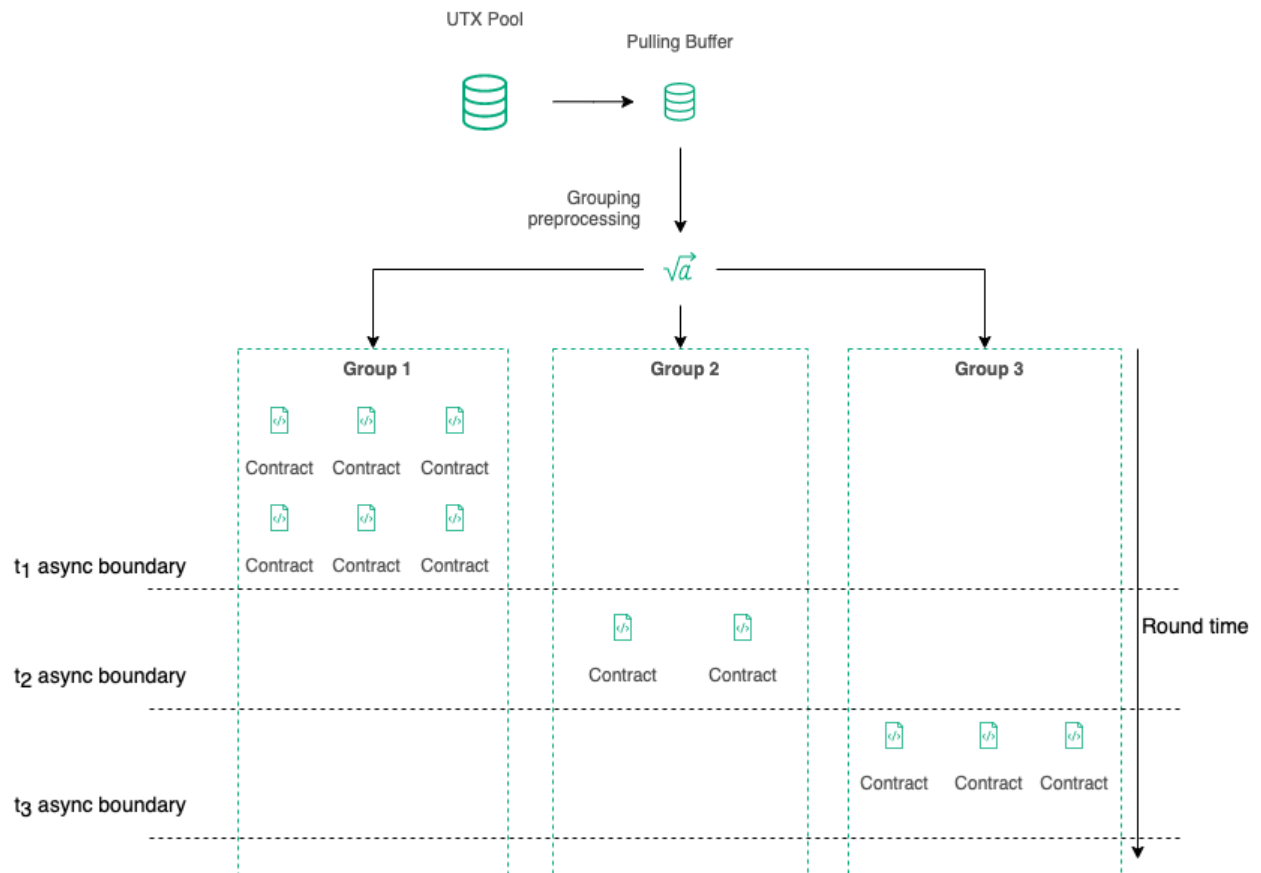
Платформа Waves Enterprise позволяет запускать несколько Docker контрактов одновременно. Такая опция поддерживается только *gRPCконтрактами*. Как это работает:

1. Разработчик смартконтракта указывает параметр `asynsfactor` в коде контракта (подробнее см. *Создание смартконтракта*). Данный параметр определяет допустимое количество одновременно выполняемых транзакций по смартконтракту.
2. При старте контракт передает в ноду значение параметра `asynsfactor`.
3. Когда запускается исполнение контрактов, буфер для контрактов начинает наполняться. Из UTX пула берутся необработанные транзакции с контрактами, пока буфер не заполнится.
4. Далее набранные транзакции разбиваются на группы по идентификаторам контрактов. В один момент времени допускается выполнение только одной группы, при этом внутри группы параллельная обработка контрактов определяется параметром `asynsfactor`.
5. При переходе очередного контракта к выполнению освобождается одна ячейка в буфере, при этом при поступлении транзакции из UTX пула ячейка блокируется. Таким образом, операции заполнения буфера и обработки вызовов контракта происходят параллельно, что позволяет избежать пауз на подтягивание транзакций, когда все текущие транзакции уже были обработаны.
6. На параллельное исполнение контрактов влияет значение параметра `pullingbuffersize`. Данный параметр настраивается в секции `dockerengine` конфигурационного файла ноды и указывает на величину буфера для обработки транзакций с контрактами.

На схеме ниже показан примерный принцип параллельной обработки смартконтрактов.

Буфер позволяет всегда иметь запас транзакций с контрактами, готовых для обработки. Помимо этого размер буфера влияет на максимальный промежуток времени, который может быть использован для обработки контрактов из одной группы. Чем больше размер буфера, тем больше это время. Таким образом размер буфера также является ограничением для количества транзакций, ожидающих обработки. При превышении этого предела обработка транзакций переносится на следующую группу.

Логика кода смартконтракта, как и выбранные средства разработки (язык программирования, на котором написан контракт), должны учитывать специфику параллельной обработки контракта. Например, если





смартконтракт с функцией инкремента будет исполняться параллельно, то результат получится некорректным, поскольку используется общий ключ во время каждого вызова контракта. Контракт, реализующий процедуру голосования на платформе Waves Enterprise, не использует общие ключи и поддерживает параллельное исполнение, что увеличивает эффективность его обработки и гарантирует более быстрое получение результатов.

Подробнее о создании смартконтракта читайте на страничке *Создание смарт контракта*.

## 13.5 Изменение контракта

Изменять Docker смартконтракт может только его разработчик, который создал транзакцию 103 и сохранил свою роль `contract_developer` в момент изменения смартконтракта. Изменение смартконтракта выполняется при помощи транзакции 107. Необходимо, чтобы смартконтракт был активным.

После включения 107 транзакции в блок ноды майнеры скачивают образ контракта и запускают его для проверки корректности исполнения. Далее выпускается 105 транзакция с включением в неё 107 транзакции.

## 13.6 Запрет вызова контракта

При необходимости разработчик контракта может запретить его вызов. Для этого публикуется специальная (`DisableContractTransaction`) транзакция с указанием идентификатора контракта. Контракт становится недоступным после его отключения, но по нему можно получить информацию из блокчейна впоследствии.

## 13.7 Описание транзакций

Для реализации взаимодействия между блокчейном и Docker контрактом реализованы следующие транзакции:

Код	Тип транзакции	Назначение
103	<code>CreateContractTransaction</code>	Инициализация контракта. Подписание транзакции производится пользователем с ролью « <code>contract_developer</code> »
104	<code>CallContractTransaction</code>	Вызов контракта. Подписание транзакции производится инициатором исполнения контракта
105	<code>ExecutedContractTransaction</code>	Заключение результата исполнения контракта на стейт контракта. <b> br </b> Подписание транзакции производится нодой, формирующей блок
106	<code>DisableContractTransaction</code>	Запрет вызова контракта. <b> br </b> Подписание транзакции производится пользователем с ролью « <code>contract_developer</code> »
107	<code>UpdateContractTransaction</code>	Обновление кода контракта. <b> br </b> Подписание транзакции производится пользователем с ролью « <code>contract_developer</code> » <b> br </b> Изменять контракт может только его разработчик и инициатор 103 транзакции

## 13.8 Конфигурация ноды

Скачивание и исполнение Dockerконтрактов, инициированных транзакциями с кодами 103–107 выполняется на нодах с включенной опцией `dockerengine.enable = yes` (подробнее в разделе «Установка и настройка» > «Запуск Dockerконтрактов»).

## 13.9 REST API

Описание методов REST API, которые может использовать Dockerконтракт, приведено в разделе *Методы API, доступные смартконтракту*.

## 13.10 gRPC

Описание методов gRPC, которые может использовать Dockerконтракт, приведено в разделе *Сервисы gRPC, используемые смартконтрактом*.

## 13.11 Примеры реализации

- Создание простого контракта

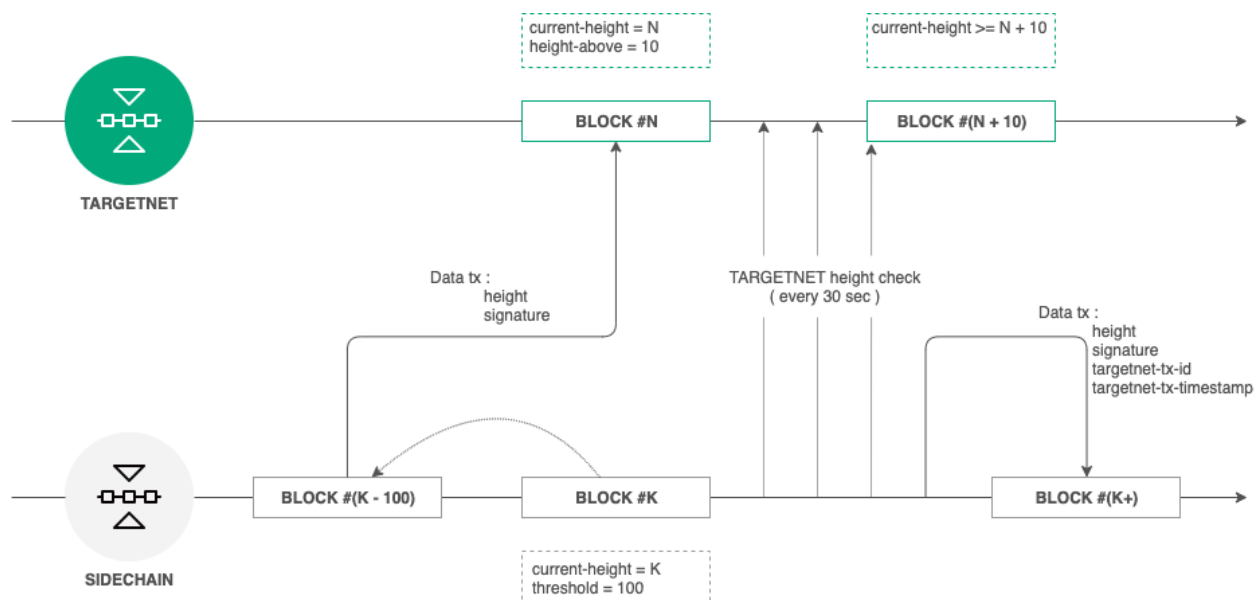
В приватном блокчейне транзакции обрабатываются определенным списком участников, каждый из которых заранее известен. Малое, по сравнению с публичной сетью, количество участников блоков и транзакций в приватном блокчейне несёт угрозу подмены информации. Перезапись цепочки блоков и транзакций, особенно в случае использования PoS консенсуса, становится реальной.

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 маловероятна из-за большого количества участников и блоков. Публикуемые данные — подпись и высота блоков приватной сети. Взаимная связность двух и более сетей повышает их стойкость, т.к. для подлога или изменения данных в результате *longrange атаки* необходимо атаковать все связанные сети.

## 14.1 Как работает анкоринг в блокчейне Waves Enterprise

Анкоринг работает следующим образом:

1. Выполняются *настройки анкоринга* в конфигурационном файле ноды приватного блокчейна. При указании параметров, ответственных за работу анкоринга, устанавливайте рекомендованные значения, чтобы избежать сложностей в работе приватного блокчейна.
2. Через каждый заданный диапазон блоков `heightrange` нода фиксирует информацию о блоке на высоте `currentheight threshold` в виде транзакции в Targetnet. В качестве такой транзакции используется *Data Transaction* со списком пар полей `keyvalue`, описание которых приведено *ниже*. После отправки транзакции нода получает её высоту в Targetnet.
3. Нода выполняет проверку высоты блокчейна в Targetnet каждые 30 секунд, пока высота не достигнет значения **высота созданной транзакции + heightabove**.
4. При достижении высоты блокчейна Targetnet, определённой в пункте 3, и положительной проверке наличия первой транзакции в блокчейне Targetnet нода создаёт вторую транзакцию с данными для анкоринга уже в приватном блокчейне.



## 14.2 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.
- `targetnettxid` идентификатор транзакции для анкоринга в Targetnet.
- `targetnettxtimestamp` дата и время создания транзакции для анкоринга в Targetnet.

## 14.3 Ошибки, возникающие в процессе анкоринга

Ошибки в анкоринге могут возникать на любом этапе. В случае возникновения ошибок в приватном блокчейне всегда публикуется *Data Transaction* 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.
- `errorcode` код ошибки.
- `errormessage` описание ошибки.

Таблица 1: Типы ошибок при анкоринге

Код	Сообщение об ошибке	Возможная причина
0	Unknown error	При отправке транзакции в Targetnet произошла неизвестная ошибка
1	Fail to create data transaction for Targetnet	Создание транзакции для отправки в Targetnet завершилась ошибкой
2	Fail send 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	В результате публикации транзакции в Targetnet вернулся отличный от 200 код
4	Fail to parse http body of response from Targetnet transaction broadcast	В результате отправки транзакции в Targetnet вернулся нераспознаваемый JSONзапрос
5	Targetnet return transaction with id='\$TargetnetTxId' but it differ from transaction that we sent id='\$sentTxId'	В результате отправки транзакции в Targetnet вернулся отличный от первой транзакции идентификатор
6	Targetnet didn't respond on transaction info request	Targetnet не ответил на запрос об информации о транзакции
7	Fail to get current height in Targetnet	Не удалось получить текущую высоту в Targetnet
8	Anchoring transaction in Targetnet disappeared after height rise enough	Анкоринг транзакция пропала из Targetnet после увеличения высоты на значение heightabove
9	Fail to create sidechain anchoring transaction	Не удалось опубликовать анкоринг транзакцию в приватном блокчейне
10	Anchored transaction in sidechain was changed during Targetnet height arise await, looks like a rollback has happened	Ожидалось подтверждение транзакции в Targetnet произошел откат приватного блокчейна, идентификатор анкоринг транзакции был изменен

## 15.1 Сервис авторизации

Сервис авторизации является внешним по отношению к ноде и обеспечивает авторизацию всех компонентов блокчейнсети. Сервис авторизации построен на базе протокола [OAuth 2.0](#). OAuth 2.0 является открытым фреймворком для реализации механизма авторизации, позволяющим предоставлять третьей стороне ограниченный доступ к защищенным ресурсам пользователя без передачи третьей стороне логина и пароля. В общем виде поток данных между участниками информационного взаимодействия на базе протокола OAuth 2.0 приведен ниже.

Средством авторизации является [JSON Web Token](#). Токены используются для авторизации каждого запроса от клиента к серверу и имеют ограниченное время жизни. Клиент получает два вида 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.

В общем виде схема авторизацию включает в себя следующие операции:

1. Клиент (компонент блокчейнсети, такой как корпоративный клиент, датасервис 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, такие как идентификатор и роль. Аутентификационные данные клиентов хранятся в конфигурационном файле сервиса авторизации. Каждый раз перед отправкой запроса стороннему сервису клиент проверяет время жизни access токена и, в случае истечения срока жизни токена, обращается к сервису авторизации для получения нового access токена. Для запросов к сервису авторизации используется refresh токен.
3. Используя актуальный access токен, клиент отправляет запрос на получение данных стороннего сервиса.

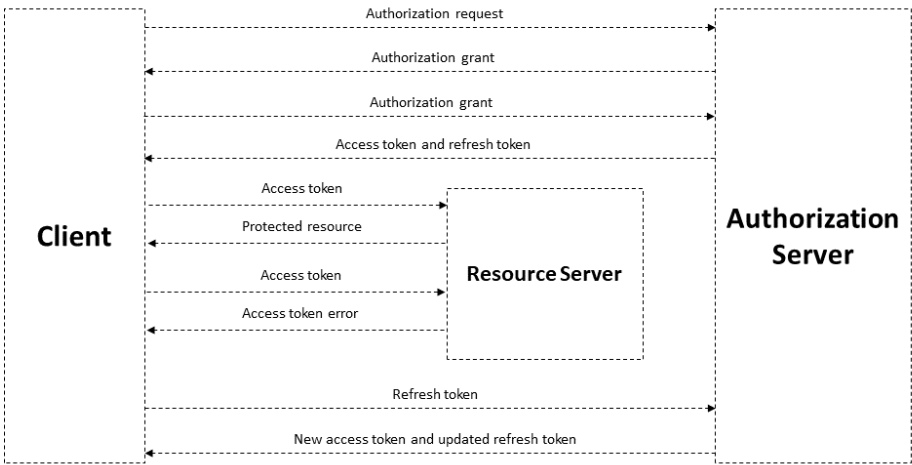


Рис. 1: Общая схема авторизации на базе протокола OAuth 2.0

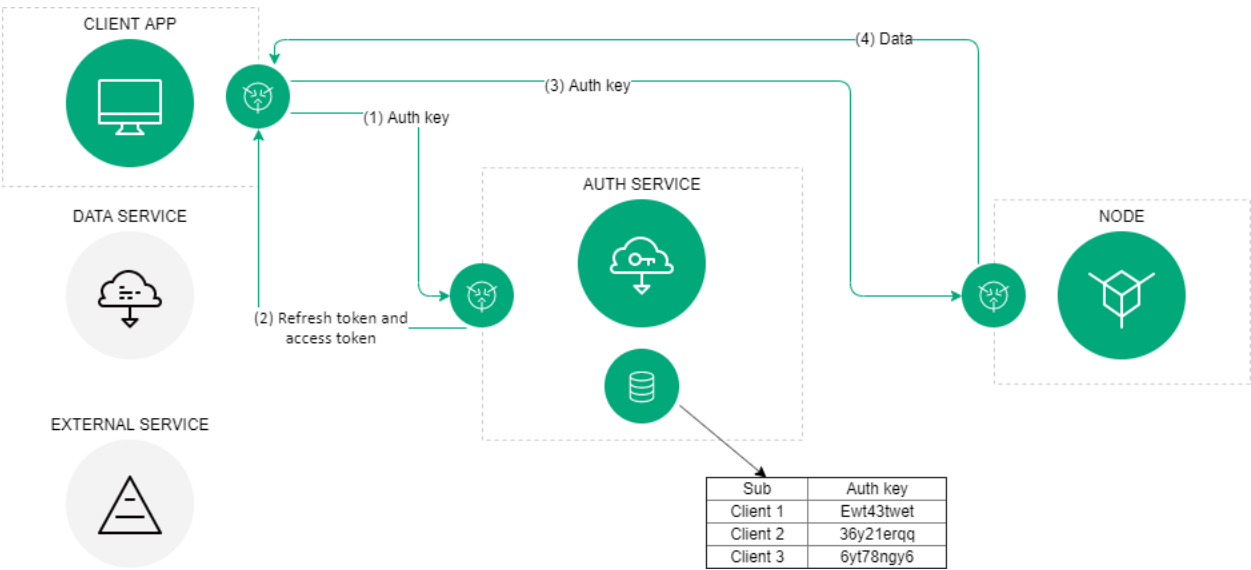


Рис. 2: Схема авторизации на блокчейнплатформе Waves Enterprise

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.

## 15.2 Сервис подготовки данных

Сервис агрегирует данные из блокчейна в реляционную БД и предоставляет API для доступа к этим данным. Функциональные возможности сервиса спроектированы под потребности клиента Waves Enterprise. Для запросов доступны уточняющие параметры.

Для использования сервиса необходимо развернуть свой экземпляр клиента и ноды из комплекта поставки. На данный момент доступ к API сервиса подготовки данных в публичной сети ограничен. Описание REST API сервиса подготовки данных приведено в разделе *REST API сервиса подготовки данных*.



---

Системные требования

---

Ниже приведены аппаратные и системные требования.

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

---

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

---

**Требования к окружению для платформы Waves Enterprise**

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

---

## Установка и запуск платформы

---

На данный момент мы поддерживаем операционные системы на базе Unix (например, популярные дистрибутивы Linux или MacOS). Однако платформа Waves Enterprise может быть запущена и под Windows в экспериментальном режиме. Вы также можете использовать такие решения, как виртуальные машины с Unix подобной системой и среду Docker для установки и запуска платформы Waves Enterprise на операционной системе Windows.

Для установки платформы вам потребуется наличие установленных [Docker Engine](#) и [Docker Compose](#) в среде развертки.

---

**Важно:** Убедитесь, что у вас установлена версия Docker Engine, поддерживающая версию формата файлов dockercompose 3.0 и выше. Подробности вы можете узнать на официальной странице [Docker](#).

---

### 17.1 Варианты установки платформы

В зависимости от сценария использования платформы Waves Enterprise предлагаются следующие варианты установки:

#### 17.1.1 Развёртывание платформы в режиме проверки возможностей (Sandbox)

В ознакомительном режиме вы можете взаимодействовать с блокчейном через клиентское приложение, либо REST/gRPC интерфейсы ноды: отправлять транзакции, получать данные из блокчейна, устанавливать и вызывать смартконтракты, а также передавать конфиденциальные данные между нодами.

1. Создайте рабочую директорию и поместите туда файл `dockercompose.yml`, который вы можете скачать с официальной страницы Waves Enterprise на [GitHub](#), выбрав самый свежий релиз. Также скачать этот файл вы можете при помощи `wget` в терминале:

```
wget https://github.com/waves-enterprise/WE-releases/releases/download/v1.5.0/docker-compose.yml
```

2. Для установки платформы в режиме Sandbox откройте терминал, перейдите в директорию, содержащую файл `dockercompose.yml`, и выполните следующую команду:

```
docker run --rm -ti -v $(pwd):/config-manager/output wavesenterprise/config-manager:v1.5.0
```

После развёртывания платформы все созданные пароли и адреса хранятся в файле `credentials.txt`, который находится в рабочей директории.

3. Дождитесь результатов выполнения предыдущей команды и выполните следующую команду:

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

**Внимание:** На ОС Linux для выполнения команд могут понадобиться права администратора (права root).

После запуска контейнеров клиентское приложение будет доступно по адресу `http://localhost`, REST API ноды `http://localhost/node0`.

Для остановки запущенных нод и сервисов выполните следующую команду:

```
docker-compose down
```

Без лицензии сеть будет работать в деморежиме до высоты в 30 000 блоков. Для деморежима получать лицензию необязательно.

Команда Waves Enterprise предлагает полностью автоматический режим развёртывания для целей ознакомления с возможностями платформы. В этом режиме будет установлена блокчейнсеть из трёх нод, а также дополнительные компоненты *сервис авторизации*, *сервис подготовки данных* и *корпоративный клиент*. Все ключевые пары, используемые для подписания транзакций и блоков будут сгенерированы случайным образом. Деморежим позволяет протестировать все доступные опции и возможности системы. Сеть работает до высоты 30 000 блоков.

### 17.1.2 Подключение одной ноды к сети Mainnet

Полная инструкция по подключению ноды к Mainnet представлена на странице *Подключение ноды в сеть «Waves Enterprise Mainnet»*.

1. Создайте рабочую директорию и поместите туда файл `dockercompose.yml`, который вы можете скачать с официальной страницы Waves Enterprise на [GitHub](#), выбрав самый свежий релиз.
2. Поместите в рабочую директорию конфигурационный файл ноды с именем `private_network.conf`. Для подключения к Mainnet используется конфигурационный файл ноды `mainnet.conf`, взятый с официальной страницы Waves Enterprise на [GitHub](#). Скачайте его и переименуйте в `private_network.conf`.
3. Запустите следующую команду и дождитесь результатов выполнения команды:

```
docker run --rm -ti -v $(pwd):/config-manager/output/ wavesenterprise/config-manager:v1.4.0
```

После развёртывания ноды все созданные пароли и адреса хранятся в файле `credentials.txt`, который находится в рабочей директории.

4. При наличии файла *лицензии* положите файл лицензии в директорию `working_directory/configs/nodes/node0/license`, которая создаётся при развёртывании окружения в рабочей директории.
5. Выполните команду для запуска ноды:

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

После запуска контейнера REST API ноды будет доступен по адресу <http://localhost:6862>.

**Внимание:** При наличии ошибок убедитесь, что не запущены другие конкурирующие контейнеры или программы. Для вывода списка запущенных контейнеров и их состояния введите командой `docker ps a`. Для остановки выбранного контейнера командой `docker stop [myContainer]`. Для остановки всех контейнеров вы можете ввести `docker stop $(docker ps a q)`. Команда `docker rm [myContainer]` удалит выбранный, `docker rmi $(docker images a q)` удалит все контейнеры.

Для остановки запущенной ноды выполните следующую команду:

```
docker-compose down
```

Для подключения к основной сети Waves Enterprise Mainnet достаточно установить одну ноду. Полная процедура по подключению к Mainnet описана на странице [Подключение ноды в сеть «Waves Enterprise Mainnet»](#).

Для развёртывания полной блокчейн сети из N нод обратитесь в нашу [техническую поддержку](#) для консультации.

## 17.2 Полезная информация для установки и использования платформы

Также в данном разделе представлены такие полезные темы, как:

### 17.2.1 Первые шаги после установки платформы Waves Enterprise

В числе первых действий после развёртывания платформы Waves Enterprise вы можете совершить следующие операции:

- *Привязка адреса ноды к клиенту*
- *Отправка транзакций*
- *Активация опций платформы*

#### Привязка адреса ноды к вебклиенту

После того, как блокчейнплатформа стартовала, выполните следующие действия:

1. Откройте браузер и введите в адресную строку значение `http://localhost`.
2. Зарегистрируйтесь в вебклиенте, используя любой действительный электронный адрес, и зайдите в вебклиент.
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 «Продолжить».
5. Заполните поля, указанные ниже. Значения вы можете взять из файла `credentials.txt` для первой ноды в рабочей директории.

- *Имя адреса* укажите наименование ноды.
- *URL ноды* укажите значение `http://localhost/nodeAddress`.
- *Тип авторизации на ноде* выберите тип авторизации на ноде (токен или `apikey`).
- *Блокчейнадрес* укажите адрес ноды.
- *Пароль от ключевой пары* укажите пароль от ключевой пары ноды.

Теперь можно отправлять транзакции из вебклиента от адреса ноды, на котором есть токены.

## Отправка транзакций

Транзакции можно отправлять из вебклиента или при помощи REST API ноды. В вебклиенте вы можете выполнять следующие действия:

- Операции с токенами. Вам необходимо *привязать адрес ноды* к клиенту для совершения операций с токенами.
- Работа с приватными группами для обмена конфиденциальными данными.
- Операции с Dockerконтрактами.
- Использование опции анкоринга.
- Отправка транзакций с данными.

Все действия выполняются в интуитивно понятном и дружелюбном вебинтерфейсе. Каждое действие сопровождается отправкой соответствующей транзакции в блокчейн.

При помощи REST API ноды можно отправить любую транзакцию в блокчейн. Для отправки транзакции через REST API ноды выполните следующие действия:

1. Откройте REST API ноды, перейдя в браузере по адресу `http://localhost/node-0`.
2. Введите `apikey`, значение которого можно взять из файла `credentials.txt` и поля «API key», в форму *авторизации по API key* и нажмите `Authorize`.
3. Выберите методы *Transactions*, далее метод `POST /transactions/signAndBroadcast` и нажмите «Try it out».
4. При помощи *таблицы транзакций* выберите транзакцию, которую хотите отправить в блокчейн.
5. Сформируйте jsonзапрос со своими параметрами, используя примеры запросов из раздела *Транзакции* для каждого вида транзакции. В основном, это такие параметры, как:
  - `sender` адрес нодыотправителя транзакции;
  - `password` пароль от файла хранилища ключей `keystore.dat`;
  - `recipient` адрес нодыполучателя;
  - различного вида идентификаторы.
6. Вставьте запрос в соответствующую форму `body` REST API интерфейса, в котором вы также можете найти примеры запросов для отправки транзакций в блокчейн.
7. Нажмите «Execute» и посмотрите результат отправки транзакции в поле `Response body`. Код успешного ответа 200.

## Включение дополнительных опций

По умолчанию в деморежиме включены две опции — работа с Dockerконтрактами и майнинг. Авторизация установлена по *apikeyhash*. Для использования Dockerконтрактов в конфигурационном файле ноды уже присутствуют настройки по умолчанию для локального Dockerхоста. Настройки майнинга так же установлены по умолчанию в соответствии с рекомендованными значениями.

Дополнительные опции платформы Waves Enterprise включаются и настраиваются при помощи соответствующих секций конфигурационного файла ноды. Зайдите в конфигурационный файл ноды, на которой вы хотите включить дополнительные опции или настроить используемые, и отредактируйте секции выбранных опций.

- *Настройка Dockerконтрактов*
- *Настройка анкоринга*
- *Настройка майнинга*
- *Настройка типа авторизации*
- *Настройка групп доступа*

Конфигурационные файлы нод хранятся в индивидуальных директориях каждой ноды, например, `../working directory/configs/nodes/node0/node.conf`. В зависимости от секции конфигурационного файла ноды рекомендованные значения либо уже установлены в примерах файлов, либо их можно найти на страничке описания секции. Секцию необходимо раскомментировать или скопировать из документации с соответствующей странички с описанием.

По всем возникающим вопросам о настройке секций конфигурационного файла ноды обращайтесь в [техническую поддержку](#).

### 17.2.2 Обновление ноды, подключённой к Mainnet

При работе с блокчейн сетью Waves Enterprise Mainnet мы рекомендуем своевременно обновлять подключённые ноды. После выхода нового релиза всем клиентам приходит письмо с уведомлением об обновлении версии ноды. Если вам не пришло такое письмо, напишите, пожалуйста, запрос в нашу [техническую поддержку](#).

Представленная ниже инструкция предназначена для нод, развёрнутых и запущенных при помощи файла `dockercompose.yml`. Для обновления других версий нод обратитесь, пожалуйста, в нашу [техническую поддержку](#).

Для обновления ноды выполните следующие действия:

1. Скачайте последнюю версию файла `dockercompose.yml` с официальной страницы Waves Enterprise на [GitHub](#), выбрав последний релиз.
2. Поместите файл `dockercompose.yml` в рабочую директорию ноды.
3. Если нода работает, то остановите ноду, выполнив команду:

```
docker-compose down
```

4. После остановки ноды выполните команду:

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

**Внимание:** При первом запуске ноды релиза 1.4.0 будет запущен мигратор данных. Миграция выполняется в автоматическом режиме и занимает несколько минут. Если миграция завершилась успешно вы увидите сообщение «Migration finished sucessfully» и запуск ноды будет продолжен.

### 17.2.3 Работа в вебклиенте

При помощи *вебинтерфейса* пользователь выполняет основные операции с блокчейном. В этом разделе мы рассмотрим наиболее популярные операции через него.

- *Расчёт выплат лизинга*
- *Публикация и вызов контракта*
- *Отправка транзакции с данными*
- *Работа с группами доступа*

#### Расчёт выплат лизинга

Об алгоритме расчёта выплат лизинга можно почитать на страничке описания *клиента*. Доход от лизинга рассчитывается на страничке **Настройки сети > Расчёт выплат лизинга**. Заполните следующие поля:

- Адрес лизингового пула.
- Начало расчётного периода (высота блокчейна), но не глубже 200 000 блоков от текущей высоты.
- Конец расчётного периода (высота блокчейна). По умолчанию берётся текущая высота блокчейна.
- Процент выплат.

Нажмите **Рассчитать выплаты**. Ниже на страничке появится результат расчёта выплат.

Ноды	Расчёт выплат лизинга
<b>Как отработал пул</b>	
Период 1000000-1071605 блоков	
Обработал: 6322 блоков	
Заработал: 2 014,44 WEST	
<b>Список выплат</b>	
Ha 3NmhwpV1K7GTBNDEQwd4rEYPMwmG2XA4HJT: 0,259 446 97 WEST	
Ha 3Nn2cFkxfqbxG4Dh9xpuq7R4VniSyaFbQVV: 0,023 060 72 WEST	
Ha 3NubonpWEAS6MudNVbqRsVpnngoq7NSUdDT: 2,468 356 87 WEST	
Ha 3NgFKN1zBQfDYK6Knq3PccKcBDxZmZgbibF: 0,348 390 73 WEST	
Ha 3NpyKBukUJf7jHKLajq2EprkzFbKQTpfBk: 0,243 837 81 WEST	
Ha 3NsiS9tp39FqG32B77FRECqaiD5XZviJg5: 0,000 069 68 WEST	
Ha 3NeypmSd2FhhuG69NRYCbHK11F3z5NijkYi: 0,000 252 54 WEST	
Ha 3NdngWgvoRgUkSSmTnCj3VTnLeXMHMGoadD: 0,007 181 66 WEST	
Ha 3NgsSubNBHPZLp36wZydzueqRXGZgWp39gX: 0,000 123 01 WEST	
Ha 3Nu67AQMo1knNGVmo3DBrMSmkqg3SPcb21P: 1,309 166 48 WEST	
Ha 3NvbaHefhoBYifBuilEqX5A8MLCZ4YSzhzE: 0,523 389 99 WEST	
Ha 3Nhqmtg5QJS77SAZWnt8KM7tNQwz472NAB: 0,453 006 87 WEST	

Вы также можете выгрузить информацию о расчёте выплат лизинга в формате *json* для рассылки заинтересованным адресам.

## Публикация и вызов контракта

Для работы с контрактами необходима роль `contract_developer`. Образы контрактов, собранные на одноимённой вкладке, не публиковались в блокчейне, эти данные берутся из Dockerрепозитория напрямую. Вкладка **Опубликованные контракты** содержит список всех контрактов, созданных и опубликованных в блокчейне.

Через клиентский интерфейс вы можете публиковать и вызывать *контракты* только второй версии, работающие через gRPC. Однако страничка **Контракты** показывает контракты всех версий. Версия контракта указана в его карточке, которая вызывается при клике на запись в списке опубликованных контрактов.

Выполните следующие действия для публикации контракта в блокчейне:

1. На вкладке **Образы контрактов** выберите контракт, который хотите опубликовать в блокчейне, и откройте его карточку.
2. В карточке контракта перейдите на вкладку **Публикация** и заполните поле *Имя контракта*.
3. Укажите в поле *Новая пара ключзначение* столько пар значений, сколько хотите, чтобы обработал контракт. Выберите тип данных для каждой пары (строка, целое число, булево, бинарные данные base64). Эти данные необходимо указать в соответствии с логикой кода контракта. После ввода всех пар значений нажмите кнопку *Далее*.
4. Проверьте корректность всех введённых данных.
5. Для публикации контракта нажмите кнопку *Далее*. Контракт опубликуется.
6. Далее можно ещё раз опубликовать другой образ контракта с новыми значениями или вернуться к списку контрактов.

Сразу после опубликования в блокчейне контейнеру с контрактом присваивается идентификатор. Такой контейнер можно в дальнейшем вызывать и обновлять, используя этот идентификатор. Из одного образа контракта в репозитории блокчейна можно создавать любое количество контейнеров с контрактами.

Выполните следующие действия для вызова уже опубликованного контракта:

1. На вкладке **Опубликованные контракты** выберите контракт, который хотите вызвать, и откройте его карточку.
2. Перейдите на вкладку **Вызов** и укажите в поле *Новая пара ключзначение* столько пар значений, сколько хотите, чтобы отработал контракт. Выберите тип данных для каждой пары (строка, целое число, булево, бинарные данные base64). Эти данные необходимо указать в соответствии с логикой кода контракта. После ввода всех пар значений нажмите кнопку *Далее*. Произойдёт вызов контракта.
3. Далее можно ещё раз вызвать этот контракт с другими значениями или вернуться к списку контрактов.

Публикация контракта зависит от его размера, если код контракта достаточно объёмный, то транзакция с контрактом попадёт в блокчейн в течение примерно 510 минут.



## Отправка транзакции с данными

На страничке **Передача данных** можно отправлять транзакции с данными в формате «ключзначение». Выполните следующие действия для создания транзакции с данными на вкладке **Запись**:

1. Нажмите кнопку *Создать транзакцию с данными*.
2. Укажите в поле *Новая пара ключзначение* столько пар значений, сколько хотите, чтобы поместилось в транзакции. Всего можно добавить до 100 пар «ключзначение». Выберите тип данных для каждой пары (строка, целое число, булево, бинарные данные base64). После ввода всех пар значений нажмите кнопку *Далее*.
3. Проверьте корректность всех введённых данных и нажмите кнопку *Далее*. Транзакция с данными опубликуется.

## Работа с группами доступа

На вкладке **Группы** создаются группы обмена приватными данными. Подробнее о группах доступа читайте в разделе *Конфиденциальность данных*. Для работы с данными в приватных группах необходимо добавить адрес действующей ноды блокчейнсети. Также у вашего электронного адреса в клиенте должна быть роль `privacy`. Обратитесь к администратору сервиса авторизации для получения такой роли.

Выполните следующие действия для привязки адреса ноды к аккаунту клиента:

1. Откройте форму управления адресами аккаунта, нажав кнопку **Адрес не выбран** или на название уже привязанного адреса в верхнем правом углу интерфейса.
2. Нажмите кнопку **Добавить адрес** и выберите вариант *Добавить адрес из ключевого хранилища ноды*.
3. Заполните следующие поля:
  - Имя адреса.
  - URL ноды.
  - Тип авторизации на ноде. Тип авторизации должен совпадать с установленным типом на ноде.
  - Блокчейнадрес.
  - Пароль от ключевой пары.
4. Нажмите кнопку **Продолжить** для привязки адреса ноды к аккаунту.

Группы создаются на вкладке «Передача данных Группы». Выполните следующие действия для создания новой приватной группы:

1. Нажмите кнопку *Новая группа*.
2. Укажите название группы и добавьте адреса участников в приватную группу. При желании можно добавить описание группы. Для каждого участника вы можете выбрать одну из двух ролей: доступ к данным или управление участниками. Для обмена сообщениями внутри группы каждый адрес должен принадлежать ноде блокчейнсети и иметь роль *Управление данными* в данной группе. Пользователи с клиентским адресом в блокчейнсети могут быть участниками групп, но не имеют возможности обмениваться сообщениями внутри самой группы.
3. При необходимости добавьте модератора в группу. Модератор имеет возможность редактировать состав участников группы и не имеет доступа к самим данным в отличие от пользователя с ролью управления участниками.

---

**Примечание:** Удаление группы из блокчейна невозможно, вы можете только исключить всех участников из группы.

---

4. Нажмите кнопку *Далее* и проверьте правильность введённых данных. При необходимости отредактируйте данные группы.
5. Нажмите кнопку *Далее* для создания группы доступа к приватным данным.

Сообщения создаются и хранятся на вкладке «Сообщения» в карточке группы приватных данных. Карточка открывается при нажатии на запись о группе приватных данных.

---

## Ручная конфигурация ноды

---

Конфигурация ноды включает в себя следующие шаги:

### 18.1 Подготовка конфигурационных файлов

В конфигурации ноды используются следующие файлы:

- `accounts.conf` конфигурационный файл для генерации аккаунтов.
- `apikeyhash.conf` конфигурационный файл для генерации значений полей `apikeyhash` и `privacyapikeyhash` при выборе авторизации по хешу ключевой строки `apikey`.
- `node.conf` основной конфигурационный файл ноды, определяющий ее принципы работы и набор опций.

#### 18.1.1 Конфигурационный файл для создания аккаунтов `accounts.conf`

При указании пути в параметрах файла `accounts.conf` необходимо использовать символ «прямого слэша» / как разделитель уровней иерархии директорий. При работе в ОС Linux значение `wallet` должно соответствовать структуре каталогов операционной системы, например, `/home/contract/we/keystore.dat`. При настройке ноды не допускается использование кириллических символов при указании путей до рабочей директории, хранилища ключей и т.д.

```
// accounts.conf listing

accounts-generator {
    waves-crypto = yes
    chain-id = V
    amount = 1
    wallet = ${user.home}/node/keystore.dat
    wallet-password = "some string as password"
    reload-node-wallet {
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    enabled = false
    url = "http://localhost:6862/utils/reload-wallet"
  }
}

```

Описание параметров конфигурационного файла представлено ниже.

- `wavescrypto` – выбор криптографического алгоритма («yes» использовать *криптографию Waves*, «no» использовать *ГОСТкриптографию*);
- `chainid` – идентифицирующий байт сети, значение потребуется дальше для внесения в параметр `addressschemecharacter` в конфигурационный файл ноды;
- `amount` – количество генерируемых ключевых пар;
- `wallet` – путь до каталога хранения ключей на ноды, значение потребуется дальше для внесения в параметр `wallet > file` в конфигурационный файл ноды. Для криптографии Waves указывается путь до файла `keystore.dat` (пример, `${user.home}/we/keystore.dat`), для ГОСТкриптографии путь до директории (`${user.home}/we/keystore/`);
- `walletpassword` – пароль для доступа к закрытым ключам ноды, значение потребуется дальше для внесения в параметр `wallet > password` в конфигурационный файл ноды;
- `reloadnodewallet` опция для обновления `keyStore` ноды без перезапуска приложения, по умолчанию установлено в значение «Выключено» (`false`). В параметре `url` указывается путь до метода `/utils/reloadwallet` REST API ноды.

### 18.1.2 Конфигурационный файл `apikeyhash.conf`

Конфигурационный файл `apikeyhash.conf` нужен только для генерации значений полей `apikeyhash` и `privacyapikeyhash` при выборе авторизации по хешу ключевой строки `apikey`.

```

// api-key-hash.conf listing

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

```

Описание параметров:

- `wavescrypto` – выбор криптографического алгоритма («yes» использовать *криптографию Waves*, «no» использовать *ГОСТкриптографию*);
- `apikey` – ключ, который необходимо придумать. Значение данного ключа потребуется указать в запросах к REST API ноды (подробнее на странице *REST API ноды*).

### 18.1.3 Конфигурационный файл ноды node.conf

Если планируется подключение к существующей сети, то для упрощения подключения запросите готовый конфигурационный файл ноды у одного из участников сетевого взаимодействия или у администратора вашей сети. При создании сети с нуля или подключении к сети «Waves Enterprise Mainnet» пример конфигурационного файла ноды можно взять на странице проекта на [GitHub](#).

Файл node.conf выполнен в формате HOCON.

Об изменениях в конфигурационном файле ноды можно почитать в разделе *Изменения в конфигурационном файле ноды*.

**Предупреждение:** Для нод версии 1.0 и выше в конфигурационном файле ноды в корневой секции node необходимо наличие следующего параметра:

```
"features": {
  "supported": [100]
}
```

Данная опция становится активной после достижения суммарного количества блоков из параметров featurecheckblocksperiod = 15000 и blocksforfeatureactivation = 10000 (25000 блоков), которые находятся в секции blockchain. При подключении к Mainnet или Partnernet данные параметры не могут быть изменены. Ноды без активации данной опции не смогут подключиться к сети.

Пример конфигурационного файла ноды представлен ниже. В данном примере отключены опции *анкоринга*, *Docker* смартконтрактов и *групп* доступа к приватным данным. Продемонстрировано включение роли sender в блоке genesis.

**Внимание:** Генезис это первый блок сети, от которого формируется блокчейн. Для включения роли sender необходимо явно указать версию генезиса (2), а также добавить сам параметр включения роли. Описание роли sender см. в разделе *Управление полномочиями*.

Также установлена *авторизация* по хешу ключевой строки apikey и криптография Waves. Описание параметров конфигурационного файла ноды вы можете найти [тут](#).

**Примечание:** Если вы планируете использовать дополнительные опции, установите поле enable выбранной опции в значение yes или true и настройте секцию опции в соответствии с описанием её настройки.

**Предупреждение:** Заполните **ТОЛЬКО** те поля, где в качестве значений указано слово */FILL/*.

```
node {
  # Type of cryptography
  waves-crypto = yes

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

  # NTP settings
  ntp.fatal-timeout = 5 minutes
```

(continues on next page)

(продолжение с предыдущей страницы)

```

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

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

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

    # Access password
    password = " /FILL/ "
}

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

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

```

(continues on next page)

(продолжение с предыдущей страницы)

```

# Initial token distribution:
# - recipient: target's blockchain address (base58 string)
# - amount: amount of tokens, multiplied by 10e8 (integer)
#
# Example: { recipient: "3HQSr3VFCiE6JcWuV1yX8xttYbAGKTLV3Gz", amount: 30000000 WEST }
}

#
# Note:
# Sum of amounts must be equal to initial-balance above.
#
{ recipient: " /FILL/ ", amount: 1000000 WEST },
{ recipient: " /FILL/ ", amount: 1500000 WEST },
{ recipient: " /FILL/ ", amount: 500000 WEST },
]
network-participants = [
# Initial participants and role distribution
# - public-key: participant's base58 encoded public key;
# - roles: list of roles to be granted;
#
# Example: {public-key: "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjgkgE7ZW1Tt", roles:
[permissioner, miner, connection_manager, contract_developer, issuer]}
#
# Note:
# There has to be at least one miner, one permissioner and one connection_manager for
the network to start correctly.
# Participants are granted access to the network via GenesisRegisterNodeTransaction.
# Role list could be empty, then given public-key will only be granted access to the
network.
#
{ public-key: " /FILL/ ", roles: [permissioner, sender, miner, connection_manager,
contract_developer, issuer]},
{ public-key: " /FILL/ ", roles: [miner, sender]},
{ public-key: " /FILL/ ", roles: []},
]
}
}

# 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

```

(continues on next page)

(продолжение с предыдущей страницы)

```

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

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

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

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

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

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

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

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

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

    # Port to listen to REST API requests
    port = 6862

    # Enable/disable TLS for REST
    tls = no
  }
}

```

(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

  akka-http-settings {
    akka {
      http.server.idle-timeout = infinite
    }
  }
}

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 {
  # Max parallel data crawling tasks
  crawling-parallelism = 100

  storage {
    vendor = none

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

```

(continues on next page)

(продолжение с предыдущей страницы)

```

# }

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

cleaner {
  enabled: no

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

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

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

# 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
  }
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```

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

```

## 18.2 Изменения в конфигурационном файле ноды

В этом разделе приведены изменения в конфигурационном файле в зависимости от версии ноды.

**Предупреждение:** Если вы обновляете версию ноды, необходимо также обновить конфигурационный файл ноды. Без обновления конфигурационного файла нода работать не будет!

### 18.2.1 Изменения в конфигурационном файле ноды версии 1.5.0

В связи с добавлением алгоритма консенсуса *CFT* и роли *sender* внесены изменения в следующие секции конфигурационного файла ноды:

#### Секция **Blockchain**

Для блока consensus добавлен тип консенсуса *cft*, а также два параметра, необходимых для настройки валидации блоков:

- `maxvalidators` максимальное количество валидаторов, определяемых для конкретного раунда голосования.
- `finalizationtimeout` время ожидания майнером финализации последнего блока (в секундах).

### Секция genesis

- Добавлен параметр `version` для явного указания используемой версии генезиса. Версия по умолчанию 1, версия 2 применяется для включения роли `sender`.
- Добавлен параметр `senderroleenabled` для включения роли `sender`. Для включения роли установите значение параметра `true`, для отключения `false`.

**Внимание:** Роль `sender` определяется в генезис блоке. Соответственно, она будет работать только для новых сетей, сформированных на платформе версии 1.5.0 и выше.

## 18.2.2 Изменения в конфигурационном файле ноды версии 1.4.0

Изменилась база данных для хранения стейта внутри ноды, теперь вместо LevelDB используется [RockDB](#). Если вы переходите на релиз 1.4.0, то откат обратно невозможен. Миграция БД происходит автоматически при переходе на релиз 1.4.0 и может занять продолжительное время. Если у вас в сети несколько нод, то ноды необходимо обновлять строго последовательно! Рекомендуется сделать резервную копию старой БД.

### Секция api

Секция `restapi` была переименована в секцию `api`. Теперь секция включает в себя *настройки* REST API и gRPC интерфейса.

### Секция privacy

В секции произошли следующие изменения:

- Добавился блок `cleaner`
- Добавился вариант хранения данных [S3 Minio](#)

## 18.2.3 Изменения в конфигурационном файле ноды версии 1.2.2

### Секция blockchain

Для сети **Mainnet** секцию `blockchain` необходимо поменять с полного варианта на следующий:

```
blockchain.type = MAINNET
```

**Предупреждение:** Если ноды, подключённые к Mainnet, будут иметь старые настройки секции `blockchain`, то может произойти рассинхронизация сети (форк от сети Mainnet)!

Во всех остальных случаях секция `blockchain` соответствует установленным настройкам.

## 18.2.4 Изменения в конфигурационном файле ноды версии 1.2.0

### Секция `dockerengine`

В секцию `dockerengine` добавлен параметр `grpcserver`, отвечающий за настройку gRPC сервера для работы docker контрактов с gRPC API:

```
grpc-server {
  # gRPC server port
  port = 6865
  # Optional node host
  # host = "192.168.65.2"
}
```

## 18.2.5 Изменения в конфигурационном файле ноды более ранних версий

Версия ноды 1.1.2

Версия ноды 1.1.0

## 18.3 Описание основных параметров и секций конфигурационного файла ноды

Для параметров в конфигурационном файле применяется несколько типов значений:

- Числовое значение, используемое для указания точного количества элементов. Это может быть количество транзакций, блоков, соединений.
- Числовое значение с указанием единиц измерения, используемое для определения временного периода или объёма памяти. В таком виде, как правило, указываются временные периоды в днях, часах или секундах, или объём кэшпамяти, например, `memory = 256M` или `connectiontimeout = 30s`.
- Строковое значение, используемое для указания адресов, путей к директориям, паролям и т.д. Путь к директории указывается в формате, приемлемом для текущей ОС. Значение указывается в кавычках.
- Массив, используемый для указания списков значений, например, адресов или публичных ключей. Значение указывается в квадратных скобках через запятую.
- Логический тип `no` или `yes`, используемый для активации различных опций.

Пример конфигурационного файла приведен на странице [подготовки конфигурационных файлов](#). В состав файла входят следующие секции:

- `node` общая секция, куда входят все секции и дополнительные параметры для настройки ноды.
- `synchronization.transactionbroadcaster` настройка параметров синхронизации для отправки неподтверждённых транзакций в блокчейн.
- `additionalcache` настройка параметров дополнительной кэшпамяти для временного хранения входящих блоков.
- `loggers` детализированная настройка логов.
- `ntp` настройка параметров NTPсервера.
- `blockchain` настройка основных параметров блокчейна.
- `features` настройка дополнительных параметров ноды.

- *tls* включение и настройка TLS ноды.
- *network* сетевые настройки.
- *wallet* настройка доступа к закрытым ключам ноды.
- *miner* настройка майнинга.
- *api* настройка REST API/gRPC и типа авторизации.
- *privacy* настройка групп доступа к конфиденциальной информации.
- *dockerengine* настройка Docker смартконтрактов.

### 18.3.1 Секция `node`

Дополнительные параметры секции:

- *wavescrypto* тип *шифрования* в блокчейне. Возможные значения: *yes* выбор криптографии Waves, *no* выбор ГОСТкриптографии.
- *directory* основная директория для хранения ПО ноды.
- *datadirectory* директория для хранения данных блокчейна в RocksDB: блоки, транзакции, стейт ноды.
- *logginglevel* уровень логирования работы ноды. Возможные значения: *DEBUG*, *INFO*, *WARN*, *ERROR*, по умолчанию установлено значение *INFO*.
- *owneraddress* адрес ноды, которая будет владельцем конфигурационного файла. Если владелец ноды будет начальным участником сети, то его адрес и публичный ключ должны быть в *genesis* блоке.

### 18.3.2 Секция `synchronization.transactionbroadcaster`

- *maxbatchsize* и *maxbatchtime* – технические параметры, позволяющие регулировать скорость уменьшения очереди транзакций.
- *minbroadcastcount* – минимальное число соединений, которые можно использовать для отправки каждой транзакции в блокчейн. Значение не должно превышать число нод в сети минус один (нодаотправитель в расчёт не принимается).
- *retrydelay* – интервал повторной отправки транзакции, если количества текущих соединений не хватило, или произошли ошибки во время отправки.
- *extensionbatchsize* количество блоков в серии, используемое для запроса расширения от пиров.
- *knowntxcachesize* максимальное количество неподтверждённых транзакций в кешпамяти.
- *knowntxcachetime* максимальное время жизни неподтверждённых транзакций в кешпамяти.

### 18.3.3 Секция additionalcache

- rocksdb параметры БД RocksDB:
  - maxcachesize максимальный размер кешпамяти.
  - maxrollbackdepth количество блоков, на которые ноду можно откатить в ручном режиме.
  - rememberblocksintervalincache период хранения блоков к кешпамяти.
- blockids параметры кеша для входящих блоков.
  - maxsize максимальный размер кешпамяти.
  - expireafter период, через который хранимые блоки удаляются из кеша.

### 18.3.4 Секция loggers

Секция предназначена для перечисления логгеров с заданным индивидуально уровнем логгирования. Список логгеров можно узнать при использовании метода *GET /node/logging*. Логгеры указываются в следующем виде:

```
"com.wavesplatform.mining.MinerImplPoa": TRACE
"com.wavesplatform.utx.UtxPoolImpl": DEBUG
```

### 18.3.5 Секция ntp

- servers список адресов NTPсерверов. Рекомендуемое значение [ "0.pool.ntp.org", "1.pool.ntp.org", ... "10.pool.ntp.org" ].
- requesttimeout таймаут для одного запроса на NTPсервер. Рекомендуемое значение 10 секунд.
- expirationtimeout таймаут синхронизации запросов к NTPсерверу. Рекомендуемое значение 1 минута.
- fataltimeout таймаут подключения к NTPсерверу. Рекомендуемое значение 1 минута.

### 18.3.6 Секция blockchain

- type тип блокчейна. Возможные значения MAINNET или CUSTOM. Значение MAINNET позволяет использовать genesisблок, консенсус и настройки сети Mainnet. При выборе значения MAINNET в конфигурационном файле ноды, которая подключается к сети Mainnet, не нужно указывать параметры блоков custom, genesis и consensus.
- consensus.type тип консенсуса в блокчейне. Возможные значения: pos, poa или cft. Вы можете подробно почитать о настройке консенсуса [здесь](#).

#### Блок fees

- enabled опция использования комиссий за выпуск *транзакций*. Возможные значения false или true.

#### Блок custom

- addressschemecharacter байт сети, для «Waves Enterprise Mainnet» V, для «Waves Enterprise Partnetnet» P. Данный параметр используется для предотвращения конфликта адресов из разных сетей. Для сайдчейна или для тестовых версий блокчейнплатформы Waves Enterprise можно использовать любые буквы. Ноды в одной блокчейнсети должны иметь одинаковый байт сети.

- `functionality` блок настройки основных параметров блокчейна.
- `genesis` блок настройки параметров генезисблока.

#### Блок `functionality`

- `featurecheckblocksperiod` количество блоков, через которые выполняется проверка и активация опций блокчейна.
- `blocksforfeatureactivation` количество блоков, через которые применяется активированная опция.
- `preactivatedfeatures` набор опций блокчейна.

#### Блок `genesis`

- `averageblockdelay` средняя задержка создания блоков. Данный параметр используется для консенсуса *PoS*.
- `initialbasetarget` начальное базовое число для регулирования процесса майнинга. Данный параметр используется для консенсуса *PoS*. От значения параметра зависит частота формирования блоков чем больше значение, тем чаще создаются блоки. Также величина баланса майнера влияет на использование данного параметра в майнинге чем больше баланс майнера, тем меньше становится значение `initialbasetarget` для выбора нодымайнера. При выставлении значения данному параметру рекомендуется учитывать комбинацию балансов майнеров и ожидаемый интервал между блоками.
- `blocktimestamp` числовой код даты и времени. Время указывается в миллисекундах, значение должно состоять из 13 цифр. Если вы берёте стандартное значение `timestamp`, состоящее из 10 цифр, то в конце необходимо добавить три любые цифры.
- `initialbalance` начальный баланс сети. Значение этого параметра влияет на процесс майнинга при *PoS* консенсусе. Чем больше баланс майнера, тем меньше становится значение `initialbasetarget` для определения нодымайнера текущего раунда.
- `genesispublickeybase58` хеш публичного ключа генезисблока, зашифрованный в Base58.
- `signature` подпись генезисблока, зашифрованная в Base58.
- `transactions` список участников сети с первоначальным балансом, создание которых войдёт в генезисблок в виде генезистранзакций.
- `networkparticipants` список сетевых участников с ролями, создание которых войдёт в генезисблок в виде генезистранзакций.

### 18.3.7 Секция `tls`

Секция предназначена для работы с TLS в ноде.

- `type` состояние режима TLS. Возможные опции: `DISABLED` (отключен, в этом случае остальные опции не указываются или комментируются) и `EMBEDDED` (включен, сертификат подписывается провайдером ноды и упаковывается в JKS-файл (`keystore`) при этом директория, в которой располагается сертификат, и параметры доступа к сертификату и `keystore` указывается пользователем вручную в последующих полях).
- `keystorepath` относительный путь к `keystore`, размещаемому в директории ноды: `${node.directory}/we_tls.jks`.
- `keystorepassword` пароль для `keystore`.
- `privatekeypassword` пароль для приватного ключа.



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

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

- **keystore** имя файла keystore.
- **storepass** пароль от keystore, указывается в конфиге в секции **keystorepassword**.
- **keypass** пароль от приватного ключа, указывается в конфиге в секции **privatekeypassword**.
- **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 IPадреса для работы сертификата в различных сетях.

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

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

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

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

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

Затем также укажите относительный путь к **we.jks** в файле конфигурации ноды.

### 18.3.8 Секция **network**

- **bindaddress** сетевой адрес ноды.
- **port** номер порта.
- **knownpeers** список известных сетевых адресов нод. Этот параметр должен быть обязательно заполнен. Список адресов передаётся пользователю администратором сети до подключения новой ноды.
- **declaredaddress** сетевой адрес ноды вместе с номером порта для процедуры handshake.
- **maxsimultaneousconnections** максимальное количество одновременно поддерживаемых соединений. Параметр ограничен количеством нод в блокчейне, т.е. максимальное количество одновременных соединений будет не больше числа нод в сети.
- **peersrequestinterval** интервал запроса списка пиров. Значение указывается в секундах или минутах. Рекомендуемое значение 12 минуты.
- **attemptconnectiondelay** интервал запроса на подключение к любому из известных ноды пиров.

### 18.3.9 Секция wallet

- `file` директория для хранения приватных ключей.
- `password` пароль для доступа к файлу с приватными ключами.

### 18.3.10 Секция miner

- `enable` активация опции майнинга.
- `quorum` необходимое количество нодмайнеров для создания блока. Значение 0 позволит генерировать блоки оффлайн. При указании значения необходимо учитывать, что собственная нодамайнер не суммируется со значением этого параметра, т.е., если вы указываете `quorum = 2`, то для майнинга нужно минимум 3 нодымайнера.
- `intervalafterlastblockthengenerationisallowed` создание блока только в том случае, если последний блок не старше указанного периода времени.
- `microblockinterval` интервал между микроблоками.
- `minmicroblockage` минимальный возраст микроблока.
- `maxtransactionsinmicroblock` максимальное количество транзакций в микроблоке.
- `minimalblockgenerationoffset` минимальный временной интервал между блоками.

### 18.3.11 Секция features

- `supported` список поддерживаемых опций.

## 18.4 Создание аккаунтов

Аккаунт ноды включает в себя адрес и ключевую пару публичный и приватный ключи. Адрес и публичный ключ демонстрируются пользователю во время создания аккаунта в командной строке. Приватный ключ ноды записывается в хранилище ключей `keystore.dat`.

### 18.4.1 Генерирование ключевых пар

Адрес ноды и ключевая пара будущих участников сети создаются при помощи утилиты генератора. Получить последнюю версию генератора можно на странице проекта на [GitHub](#). Для запуска утилиты необходимо в качестве одного из параметров указать файл `accounts.conf`, в котором определяются параметры генерации ключей. При создании ключевой пары придумайте и введите пароль, сохраните его для последующей конфигурации. Данный пароль будет использоваться при создании глобальной переменной `WE_NODE_OWNER_PASSWORD` далее. Если не хотите устанавливать пароль от ключевой пары ноды, нажмите `enter`. Команда для запуска генератора:

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

## 18.4.2 Глобальные переменные

Для обеспечения дополнительной безопасности рекомендуется использовать пароль для ключевой пары ноды. Платформа Waves Enterprise поддерживает два способа использования пароля:

1. Ввод пароля в ручном режиме при каждом старте ноды.
2. Создание глобальных переменных в операционной системе.

При использовании ручного ввода пароля создавать глобальные переменные необязательно. Если вы планируете разворачивать ноду в контейнерах или подобных сервисах, то удобнее будет задать в ОС следующие глобальные переменные:

1. `WE_NODE_OWNER_PASSWORD` пароль от ключевой пары ноды, который вводится на этапе создания этой ключевой пары.
2. `WE_NODE_OWNER_PASSWORD_EMPTY` `true` или `false`, установите значение `true`, если не хотите устанавливать пароль на ключевую пару ноды, в таком случае создавать переменную `WE_NODE_OWNER_PASSWORD` необходимости нет. Если вы используете пароль, установите значение `false` и пропишите в переменную `WE_NODE_OWNER_PASSWORD` пароль от ключевой пары ноды.

## 18.5 Подпись genesis блока

Подпишите genesis-блок утилитой `generatorsx.x.jar`. Команда для подписания: `java jar generatorsx.x.jar GenesisBlockGenerator node.conf`, где `node.conf` это отредактированный в этом [пункте](#) конфигурационный файл ноды. После подписания поля `genesispublickeybase58` и `signature` конфигурационного файла будут заполнены значениями открытого ключа и подписи genesis-блока.

Пример:

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

## 18.6 Настройка консенсуса

Блокчейн-платформа Waves Enterprise поддерживает три типа консенсуса *PoS*, *PoA* и *CFT*. Настройки консенсуса располагаются в секции *blockchain*.

### 18.6.1 Настройка PoS

Если вы не указали тип консенсуса в поле `consensus.type` секции *blockchain*, то консенсус PoS будет использоваться по умолчанию. За работу майнинга с консенсусом PoS отвечают следующие параметры, расположенные в блоке `genesis` секции *blockchain*:

- `averageblockdelay` средняя задержка создания блоков. Значение по умолчанию 60 секунд. Значение этого параметра игнорируется, если выбран консенсус PoA.
- `initialbasetarget` 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 `initialbasetarget` при расчёте очереди нод майнера в текущем раунде.

- `initialbalance` начальный баланс сети. Чем больше доля баланса майнера от изначального баланса сети, тем меньше становится значение `initialbasetarget` для определения нодымайнера текущего раунда.

Для демонстрационной работы с нодой мы рекомендуем использовать значения, которые установлены по умолчанию в примерах конфигурационных файлов, приведенных на странице проекта на [GitHub](#).

### 18.6.2 Настройка PoA

Для использования консенсуса *PoA* раскомментируйте или добавьте блок `consensus` в секции *blockchain*:

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

Приведенные в вышеуказанном блоке `consensus` параметры используются только для консенсуса *PoA*.

- `type` тип консенсуса. Возможные значения `pos`, `poa` или `cft`. При указании значения `pos` другие параметры блока учитываться не будут. Тип консенсуса CFT описан *ниже*.
- `roundduration` длина раунда майнинга блока в секундах.
- `syncduration` период синхронизации майнинга блока в секундах. Полное время раунда складывается из суммы `roundduration` и `syncduration`.
- `bandurationblocks` количество блоков, на которые нодамайнер попадает в бан.
- `warningsforban` количество раундов, которое нодамайнер предупреждается о попадании в бан.
- `maxbanspercentage` процент нодмайнеров от общего числа нод в сети, который может быть помещён в бан.

Использование консенсуса *PoA* позволяет регулировать очередность создания блоков путем ограничения функции майнинга для определенных нод. Это делается для того, чтобы распределить равномерно нагрузку в сети, если какие-либо нодымайнеры вышли из сети или стали неактивными. Нодамайнер может попасть в бан по следующим причинам:

- если нода пропустит свою очередь майнинга;
- если нода предоставит невалидный блок;
- если нода вышла из сети.

Перед попаданием в `blacklist` нодамайнер получает предупреждения о попадании в бан на такое количество раундов, какое указано в параметре `warningsforban`. После окончания количества блоков, определенного в параметре `bandurationblocks`, нодамайнер получает возможность участвовать в создании блоков.

### 18.6.3 Настройка CFT

Для использования консенсуса *CFT* также необходимо раскомментировать или добавить блок `consensus` в секции *blockchain*: Основные параметры настройки CFT идентичны параметрам консенсуса *PoA*:

```
consensus {
  type: "cft"
  round-duration: 7s
  sync-duration: 3s
  ban-duration-blocks: 14
  warnings-for-ban: 2
  max-bans-percentage: 33
  max-validators: 7
  finalization-timeout: 2s
}
```

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

- `maxvalidators` – лимит валидаторов, участвующих в голосовании в конкретном раунде.
- `finalizationtimeout` – время, в течение которого майнер ждет финализации последнего блока в цепочке. По прошествии этого времени майнер вернет транзакции обратно в UTXпул и начнет майнить раунд заново.

### 18.6.4 Настройка консенсуса в секции `miner`

При настройке параметров консенсуса необходимо учитывать следующие параметры секции *miner*:

- `microblockinterval` интервал между микроблоками. Значение указывается в секундах.
- `minmicroblockage` минимальный возраст микроблока. Значение указывается в секундах и не должно превышать значения параметра `microblockinterval`.
- `minimalblockgenerationoffset` минимальный временной интервал между блоками. Значение указывается в миллисекундах.

Значения параметров создания микроблоков не должны конфликтовать со значениями параметров `averageblockdelay` для PoS и `roundduration` для PoA и CFT. Количество микроблоков в блоке не ограничено, но зависит от размера транзакций, попавших в микроблок.

## 18.7 Настройка Docker

Настройка исполнения Dockerконтрактов для ноды задается секцией `dockerengine` в *конфигурационном файле*.

Подробнее о настройке смартконтрактов Docker см. в разделе *Подготовка к работе* главы *Смартконтракты Docker*.

## 18.8 Настройка авторизации и REST API и gRPC интерфейсов ноды

Блокчейнплатформа Waves Enterprise поддерживает следующие два типа авторизации для доступа к REST API/gRPC интерфейсам ноды:

- авторизация по хешу ключевой строки `apikey`;
- авторизация с помощью `JWT` токена.

Авторизация по ключевой строке `apikey` является простым средством управления доступом к ноде с низким уровнем безопасности. В случае попадания ключевой строки `apikey` к злоумышленнику, тот получает полный доступ к ноде. Применение авторизации с использованием отдельного сервиса авторизации, где доступ к ноде предоставляется по специальному токenu, повышает безопасность блокчейн сети до высокого уровня. Подробнее о получении JWT токена можно почитать в разделе *Сервис авторизации*.

### 18.8.1 Секция `api` конфигурационного файла ноды

Секция `api` содержит настройки авторизации и REST API/gRPC интерфейсов.

```
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
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

# Akka HTTP settings for gRPC server
akka-http-settings {
  akka {
    http.server.idle-timeout = infinite

    # Uncomment these settings if you want detailed logging for gRPC calls
    # loggers = ["akka.event.slf4j.Slf4jLogger"]
    # loglevel = "DEBUG"
    # logging-filter = "akka.event.slf4j.Slf4jLoggingFilter"
    # stdout-loglevel = "DEBUG"
    # log-dead-letters = 10
    # log-dead-letters-during-shutdown = on
    #
    # actor {
    #   debug {
    #     # enable function of LoggingReceive, which is to log any received message at
    #     # DEBUG level
    #     receive = on
    #     # enable DEBUG logging of all AutoReceiveMessages (Kill, PoisonPill etc.)
    #     autoreceive = on
    #     # enable DEBUG logging of actor lifecycle changes
    #     lifecycle = on
    #     # enable DEBUG logging of unhandled messages
    #     unhandled = on
    #     # enable DEBUG logging of subscription changes on the eventStream
    #     event-stream = on
    #     # enable DEBUG logging of all LoggingFSMs for events, transitions and timers
    #     fsm = on
    #   }
    # }
    #
    # io.tcp.trace-logging = on
    # http.server.http2.log-frames = yes
  }
}

# Authorization strategy should be either 'oauth2' or 'api-key', default is 'api-key'
auth {
  type = "api-key"

  # Hash of API key string
  api-key-hash = "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"

  # Hash of API key string for PrivacyApi routes
  privacy-api-key-hash = "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"
}

# For OAuth2:
# auth {
#   type: "oauth2"

#   # OAuth2 service public key to verify auth tokens
#   public-key: "AuthorizationServicePublicKeyInBase64"

# }
}

```

**Описание параметров блока `api.rest`**

- `enable` активация опции REST API на ноду.
- `bindaddress` сетевой адрес ноды, на котором будет доступен REST API интерфейс.
- `port` порт прослушивания REST API запросов.
- `tls` включение/отключение TLS для REST API запросов.
- `cors` поддержка кроссдоменных запросов к REST API.
- `transactionsbyaddresslimit` максимальное количество транзакций, возвращаемых методом `/transactions/address/{address}/limit/{limit}`.
- `distributionaddresslimit` максимальное количество адресов, указываемых в поле `limit` и возвращаемых методом `GET /assets/{assetId}/distribution/{height}/limit/{limit}`.

**Описание параметров блока `api.grpc`**

- `enable` активация gRPC интерфейса на ноду.
- `bindaddress` сетевой адрес ноды, на котором будет доступен gRPC интерфейс.
- `port` порт прослушивания gRPC запросов.
- `tls` включение/отключение TLS для gRPC запросов.

**Секция `auth` для типа `apikey`**

- `type` тип авторизации, установите значение `apikey` авторизация по хешу ключевой строки.
- `apikeyhash` хеш от ключевой строки доступа к REST API.
- `privacyapikeyhash` хеш от ключевой строки доступа к методам `privacy`.

**Секция `auth` для типа `oauth2`**

- `type` тип авторизации, установите значение `oauth2` авторизация по токenu.
- `publickey` публичный ключ сервиса авторизации.

REST API и gRPC интерфейсы используют одинаковые значения `apikey` и JWT токена.

**18.8.2 Использование авторизации по ключевой строке**

В параметре `authtype` установите значение `apikey`. Используя утилиту `generatorsx.x.x.jar`, создайте `apikeyhash` для доступа к REST API ноды. Для запуска утилиты требуется в качестве одного из параметров указать *файл `apikeyhash.conf`*, в котором определяются параметры создания `apikeyhash`. Команда для запуска утилиты:

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

Полученное в результате исполнения утилиты значение укажите в параметре `apikeyhash` конфигурационного файла ноды.

Для доступа к методам `privacy` создайте `privacyapikeyhash` аналогичным методом, как и `apikeyhash`, описанным выше. Полученное значение укажите в параметре `privacyapikeyhash` конфигурационного файла ноды.



### 18.8.3 Использование авторизации по токenu

В параметре `authtype` установите значение `oauth2`, в параметре `publickey` укажите публичный ключ сервиса авторизации.

## 18.9 Настройка анкоринга

Если используете опцию *анкоринга*, необходимо настроить блок `anchoring.targetnet` в данном случае блокчейнсет, в которую нода из сайдчейна будет выполнять анкорингтранзакции.

```
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

- `heightrange` число блоков, через которое нода приватного блокчейна отправляет в Targetnet транзакции для анкоринга.
- `heightabove` число блоков в Targetnet, через которое нода приватного блокчейна создаёт подтверждающую анкоринг транзакцию с данными первой транзакции. Рекомендуется устанавливать значение, не превышающее максимальную величину отката в Targetnet `maxrollback`.
- `threshold` число блоков, которое отнимается от текущей высоты приватного блокчейна. В транзакцию для анкоринга, отправляемую в Targetnet, попадёт информация из блока на высоте `currentheight - threshold`. Если устанавливается значение 0, то берётся информация из текущего блока. Рекомендуется устанавливать значение, близкое к максимальной величине отката в приватном блокчейне `maxrollback`.

- `txminingcheckdelay` время ожидания между проверками доступности транзакции для анкоринга в Targetnet.
- `txminingcheckcount` 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 `heightrange` задаёт приблизительный интервал между транзакциями анкоринга. Реальное время попадания транзакций анкоринга в смайненый блок сети Targetnet может превышать время, потраченное на майнинг количества блоков `heightrange` в сети Targetnet.

#### Параметры авторизации при использовании анкоринга

- `type` тип авторизации при использовании анкоринга. `apikey` авторизация по `apikeyhash`, `authservice` авторизация по специальному токenu.

В случае выбора авторизации по `apikeyhash` достаточно указать значение ключа в параметре `apikey` ниже. Если вы выбираете авторизацию по токenu, необходимо указать `type = "authservice"`, раскомментировать параметры ниже и установить для них значения:

- `authorizationtoken` постоянный авторизационный токен.
- `authorizationseviceurl` URLадрес сервиса авторизации.
- `tokenupdateinterval` 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.

- `targetnetschemebyte` байт сети Targetnet.
- `targetnetnodeaddress` 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` как в примере `http://node.weservices.com:6862/NodeAddress`.
- `targetnetnoderecipientaddress` адрес ноды в сети Targetnet, на который будут записываться транзакции для анкоринга, подписанные ключевой парой данного адреса.
- `targetnetprivatekeypassword` пароль от приватного ключа ноды для подписи анкорингтранзакций.

Сетевой адрес и порт для анкоринга в сеть Targetnet/Partnernet можно получить у сотрудников технической поддержки Waves Enterprise. Если используется несколько приватных блокчейнов с взаимным анкорингом, то необходимо использовать 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 нодe приватной сети.
- `password` пароль от файла с ключевой парой.

#### Параметры комиссий

- `targetnetfee` плата за выпуск транзакции для анкоринга в сети Targetnet.
- `sidechainfee` плата за выпуск транзакции в приватном блокчейне.

## 18.10 Настройка групп доступа к конфиденциальным данным

При использовании методов *privacy* активируйте функциональность и заполните блок `storage` параметрами настройки БД для хранения конфиденциальных данных:

```
privacy {
  # Max parallel data crawling tasks
  crawling-parallelism = 100

  storage {
    vendor = none

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

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

  cleaner {
    enabled: no

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

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

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

### Описание параметров

- `vendor` выбор варианта хранения данных: `s3` облачное или локальное хранение на базе Amazon Simple Storage Service (S3), `postgres` локальное хранение в БД PostgreSQL. Для хранения информации на базе S3 используется сервер [Minio](#).

Параметры для БД PostgreSQL:

- `url` адрес БД PostgreSQL;
- `driver` имя драйвера JDBC;
- `profile` имя профиля для доступа к JDBC;
- `user` имя пользователя для доступа к БД;
- `password` пароль для доступа к БД;
- `connectionPool` имя пула соединений, по умолчанию HikariCP.
- `connectionTimeout` таймаут для соединения;
- `connectionTestQuery` имя тестового запроса;
- `queueSize` размер очереди запросов;
- `numThreads` количество одновременных подключений;
- `schema` схема взаимодействия;
- `migrationdir` директория для миграции данных.

Параметры для S3:

- `url` адрес сервера S3 для хранения данных, поддерживаются сервера [Minio](#);
- `bucket` имя таблицы БД S3 для хранения данных;
- `region` название региона S3, значение параметра `awsglobal`;
- `accesskeyid` идентификатор ключа доступа к данным;
- `secretaccesskey` ключ доступа к данным в хранилище S3;
- `pathstyleaccessenabled = true` неизменяемый параметр указания пути к таблице S3;
- `connectiontimeout` таймаут для соединения;
- `connectionacquisitiontimeout` таймаут для получения соединения;
- `maxconcurrency` число параллельных обращений к хранилищу;
- `readtimeout` таймаут на чтение данных.

Секция `cleaner`

- `enabled` включение/отключение периодического удаления файлов, которые не попали в блокчейн;
- `interval` интервал очистки файлов;
- `confirmationblocks` период времени в блоках, за который существует хэштранзакция данных в блокчейне, после чего она будет удалена;
- `pendingtime` максимальный период времени, за который сохраняется файл с данными, не попадая в блокчейн.

Остальные параметры:

- `requesttimeout` таймаут ожидания всех ответов от пиров на запрос данных.
- `initretrydelay` задержка от момента получения датахеша до старта его поиска по пирам.

- `crawlingparallelism` ограничение максимального количества одновременных процессов в синхронизаторе.
- `maxattemptcount` максимальное количество раундов по запросу данных от пиров, спустя которое данные считаются «утерянными».
- `lostdataprocessingdelay` интервал раундов запросов «утерянных» данных.
- `cache` настройки кешпамяти запросов.

Для хранения конфиденциальных данных используется БД PostgreSQL. База данных устанавливается на машину с нодой, также создается аккаунт доступа к БД. Вы можете воспользоваться справочной документацией на PostgreSQL для скачивания и установки версии, которая соответствует вашему типу операционной системы.

Во время установки БД система предложит создать аккаунт для доступа к БД. Эти логин и пароль для доступа необходимо внести в соответствующие параметры `user/password`.

Внесите URL подключения к PostgreSQL в параметр `url`. В URL входят следующие параметры:

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

Можно указывать URL до БД PostgreSQL в одной строке с данными аккаунта. Пример приведен ниже, где `user=user_privacy_node_0@wdev` это логин пользователя, `password=7nZL7Jr41q0WUHz5qKdypA&sslmode=require` пароль с опцией требования его ввода при авторизации.

#### Пример

```
privacy.storage.url = "jdbc:postgresql://vostk-dev.postgres.database.azure.com:5432/
↳privacy_node_0?user=user_privacy_node_0@we-dev&password=7nZL7Jr41q0WUHz5qKdypA&
↳sslmode=require"
```

На странице [GitHub Waves Enterprise](#) вы можете получить примеры конфигурационных файлов и дистрибутивы последнего релиза платформы.

---

Использование лицензии

---

Блокчейнплатформа Waves Enterprise коммерческая и рассчитана в первую очередь на применение в крупных компаниях и государственном секторе. Для использования технологии необходимо получить лицензию на платформу. Быстрый и удобный доступ к списку лицензий обеспечивается [сервисом управления лицензиями](#).

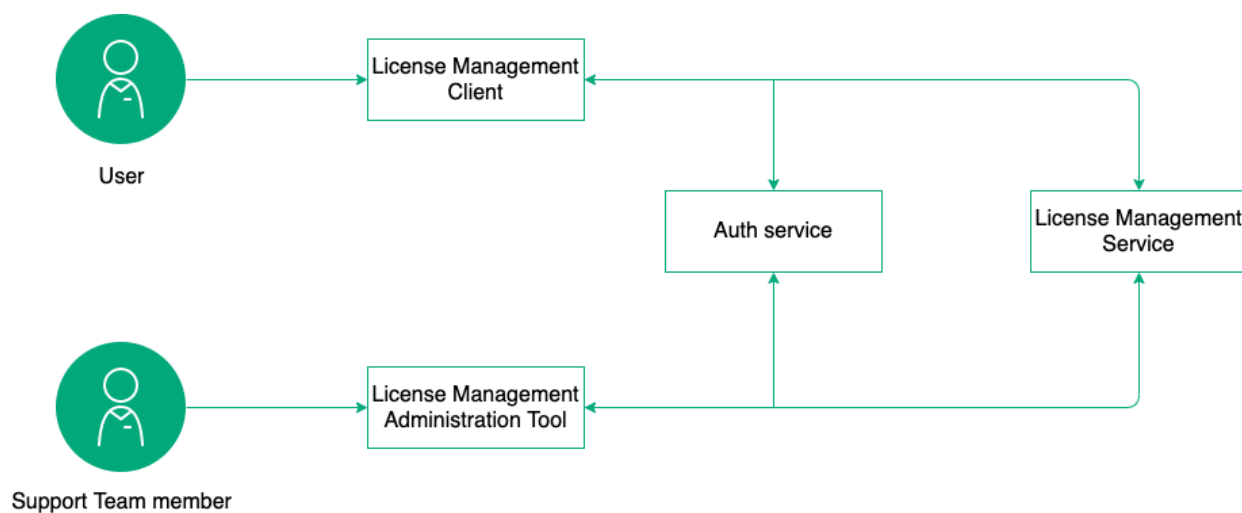


Рис. 1: Схема получения лицензии блокчейнплатформы Waves Enterprise

Для ознакомления с возможностями платформы лицензия на продукт не требуется. Платформа сохраняет полную функциональность до достижения высоты блокчейна равной **30 000** блоков, что при времени раунда блока, равного 30 секундам, составляет 10 дней работы без ограничений.

Пользователям блокчейнплатформы Waves Enterprise предлагаются к использованию следующие типы лицензии:

- **Коммерческая лицензия** позволяет использовать платформу для реализации коммерческих проектов. Выдается на срок, определяемый договорными отношениями с партнёром.

- **Некоммерческая лицензия** позволяет использовать платформу для реализации некоммерческих проектов. Выдается на срок, определяемый договорными отношениями с партнёром.
- **Пробная лицензия** позволяет ознакомиться с платформой и технологией. Выдается на срок пилотного проекта по договору, либо на время разработки и отладки продукта.
- **Лицензия для работы в сети Mainnet** специальная лицензия, позволяющая запустить ноду в сети *Mainnet*. Для работы в сети необходимо иметь на балансе или в лизинге не менее **50 000 WEST**. При снижении указанного остатка вводятся ограничения на формирование блоков и доступ к API ноды. Отправка заявки на регистрацию новых участников осуществляется в системе [Service Desk](#).

**Внимание:** Одна лицензия распространяется на одну ноду!

Лицензии различаются по сроку действия:

- Бессрочные.
- 2 года.
- 1 год.
- 3 месяца (пробная).
- Аренда на время использования платформы.

После истечения срока действия лицензии узел, на который данная лицензия установлена, теряет возможность формировать новые блоки и записывать новые транзакции в сеть.

## 19.1 Получение лицензии

Чтобы оформить запрос на лицензию, выполните следующие действия:

1. Перейдите в [сервис управления лицензиями](#) и создайте новую учётную запись, если она не была создана ранее.
2. Отправьте заявку на получение лицензии в [службу поддержки Waves Enterprise](#). Представитель службы поддержки свяжется с вами, согласует детали, создаст профиль компании и привяжет к нему созданную учётную запись.
3. После активируйте лицензию указав адрес вашей ноды (*node\_owner\_address*).
4. Указанный файл лицензии в виде JSON отправьте в запросе *POST /licenses/upload* к ноде.
5. Для просмотра состояния лицензии воспользуйтесь запросом *GET /licenses/status*.

---

Подключение к Mainnet и Partnetnet

---

## 20.1 Работа в сети «Waves Enterprise Mainnet»

### 20.1.1 Подключение ноды в сеть «Waves Enterprise Mainnet»

**Предупреждение:** При подключении своей ноды к сети «Waves Enterprise Mainnet» и майнинге на балансе аккаунта должно быть не менее **50 000 WEST**! Информация о генерирующем балансе в сети Mainnet обновляется один раз в 1000 блоков, майнинг будет доступен только после обновления генерирующего баланса.

Для подключения своей ноды к сети «Waves Enterprise Mainnet» выполните следующие действия:

1. Зайдите на [сайт Waves Enterprise](#) и создайте учётную запись, следуя подсказкам вебинтерфейса.
2. Переведите токены в сеть «Waves Enterprise Mainnet».
3. Передайте токены в аренду в любом количестве на адрес `3NrKDuHjUG7vSCiMMD259msBKcPRm4MvaJu` и сохраните идентификатор этой транзакции. В дальнейшем вы можете отозвать токены из аренды, поскольку эта операция нужна для верификации владения вами данным адресом и балансом.
4. *Разверните одну ноду.*
5. Зайдите на [сайт службы поддержки Waves Enterprise](#) и зарегистрируйтесь.
6. Выберите тип заявки «Подключение участника» для юридического или физического лица.
7. Заполните все необходимые поля формы. Если вы хотите майнить, отметьте поле **Прошу предоставить права майнинга**.
8. В поле **Подтверждение владения токенами WEST** введите идентификатор транзакции передачи токенов в аренду.
9. Дождитесь рассмотрения заявки на подключение. После успешной регистрации можете начинать работу в сети «Waves Enterprise Mainnet».



10. После получения разрешения на подключение к сети «Waves Enterprise Mainnet» и получения лицензии *запустите* ноду, публичной ключ которой вы указали в заявке.
11. Для выполнения майнинга и работы в сети переведите или передайте в аренду токены на адрес подключённой ноды.



## 20.1.2 Комиссии в сети «Waves Enterprise Mainnet»

№	Тип транзакции	Комиссия	Описание
1	<i>Genesis transaction</i>	отсутствует	Первоначальная привязка баланса к адресам создаваемых при старте блокчейна нод
2	Payment Transaction (не используется)		
3	<i>Issue Transaction</i>	1WES	Выпуск токенов. Комиссия взимается только в WEST
4	<i>Transfer Transaction</i>	0.01WES	Перевод токенов
5	<i>Reissue Transaction</i>	1WES	Перевыпуск токенов
6	<i>Burn Transaction</i>	0.05WES	Сжигание токенов
8	<i>Lease Transaction</i>	0.01WES	Передача токенов в аренду
9	<i>Lease Cancel Transaction</i>	0.01WES	Отмена аренды токенов
10	<i>Create Alias Transaction</i>	1WES	Создание псевдонима
11	<i>MassTransfer Transaction</i>	0.05WES	Массовый перевод токенов. Указана минимальная комиссия, размер комиссии зависит от количества адресов в транзакции
12	<i>Data Transaction</i>	0.05WES	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
13	<i>SetScript Transaction</i>	0.5WES	Транзакция, привязывающая скрипт с RIDEконтрактом к аккаунту
14	SponsorFee Transaction (не используется)		
15	<i>SetAssetScript</i>	1WES	Транзакция, привязывающая скрипт с RIDEконтрактом к ассету
101	<i>Genesis Permission Transaction</i>	отсутствует	Назначение первого администратора сети для дальнейшей раздачи прав
102	<i>Permission Transaction</i>	0.01WES	Выдача/отзыв прав у аккаунта
103	<i>CreateContract Transaction</i>	1WES	Создание Dockerконтракта
104	<i>CallContract Transaction</i>	0.1WES	Вызов Dockerконтракта
105	<i>ExecutedContract Transaction</i>	отсутствует	Выполнение Dockerконтракта
106	<i>DisableContract Transaction</i>	0.01WES	Отключение Dockerконтракта
107	<i>UpdateContract Transaction</i>	1WES	Обновление Dockerконтракта
110	<i>GenesisRegisterNode Transaction</i>	отсутствует	Регистрация ноды в генезисблоке при старте блокчейна

## 20.2 Работа в сети «Waves Enterprise Partnernet»

### 20.2.1 Подключение ноды в сеть «Waves Enterprise Partnernet»

Выполните следующие действия для подключения ноды в сеть «Waves Enterprise Partnernet»:

1. *Разверните* одну ноду по аналогии с подключением к сети «Waves Enterprise Mainnet».
2. Зайдите на [сайт службы поддержки Waves Enterprise](#) и зарегистрируйтесь.
3. Выберите тип заявки «Подключение участника» для юридического или физического лица.
4. Заполните все необходимые поля формы. Если вы хотите майнить, отметьте поле **Прошу предоставить права майнинга**.
5. Дождитесь рассмотрения заявки на подключение. После успешной регистрации и получения лицензии можете начинать работу в сети «Waves Enterprise Partnernet».
6. После получения разрешения на подключение к сети «Waves Enterprise Partnernet» *запустите* ноду.

Блокчейнплатформа Waves Enterprise предоставляет возможность взаимодействия с блокчейном как в части получения данных (транзакции, блоки, балансы и др.), так и в части записи информации в блокчейн (подписание и отправка транзакций) при помощи gRPCИнтерфейса и RESTful API ноды.

## 21.1 gRPC

gRPC — это высокопроизводительный фреймворк для вызов удаленных процедур (RPC), который работает поверх HTTP/2. В качестве инструмента описания типов данных и сериализации используется протокол [Protobuf](#). Официально фреймворк [gRPC](#) поддерживает 10 языков программирования. Список используемых языков доступен в [официальной документации gRPC](#).

### 21.1.1 Как использовать фреймворк gRPC

Перед использованием gRPCИнтерфейса необходимо выполнить следующие подготовительные действия:

1. Определиться с языком программирования, на котором будет осуществляться взаимодействие с нодой.
2. Установить фреймворк gRPC.
3. Скачать protobuf файлы, содержащие структуру данных для разработки обращений к ноде или смарт-контрактов, со страницы проекта в [GitHub](#).

На этой странице доступен архив `waveventsproto.zip`, в котором необходимо выбрать используемую версию ноды и соответствующие файлы.

Для генерации кода на базе структуры данных из protobuf файлов используется плагин `protoc` фреймворка gRPC.

Включение gRPCИнтерфейса и его настройка выполняются через *конфигурационный файл ноды*. Для взаимодействия с нодой предусмотрен порт **6865**.

gRPCИнтерфейс платформы предназначен для выполнения следующих задач:

- Отслеживание определенных групп событий, происходящих в блокчейне.
- Реализация методов подписи с сертификатом (PKI).
- Реализация методов шифрования.
- Получение информации о транзакции по ее ID.
- Получение параметров конфигурации ноды.
- Получение информации о состоянии смартконтракта

Для каждой из этих задач предусмотрены отдельные наборы методов:

- gRPC методы ноды
- сервисы *gRPC*, используемые смартконтрактом.

## 21.2 REST API

REST API позволяет пользователям удалённо взаимодействовать с нодой через запросы и ответы в формате JSON. Работа с API происходит по протоколу https. Удобным интерфейсом к API служит известный фреймворк Swagger.

### 21.2.1 Методы REST API ноды

Полное описание REST API вы можете найти на странице [Документация API](#). Почти все методы REST API закрыты *авторизацией*. Если метод открыт, то вы увидите значок .

#### Activation

---

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

---

#### GET /activation/status

Возвращает статус активации нового функционала в ноде/нодах.

**Ответ метода:**

```
{
  "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",

```

(continues on next page)

(продолжение с предыдущей страницы)

```

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

```

## Addresses

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

**GET /addresses/info/{address}**

Получение публичного ключа по адресу. Метод возвращает только те публичные ключи, которые хранятся в файле ноды `keystore.dat`.

**Ответ метода:**

```
{
  "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
  "publicKey": "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjkgE7ZW1Tt"
}
```

**GET /addresses**

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

**Ответ метода:**

```
[
  "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```

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

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

**Ответ метода:**

```
[
  "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```

**GET /addresses/balance/{address}**

Получение баланса для адреса `{address}`.

**Ответ метода:**

```
{
  "address": "3N3keodUiS8WEw9W4BKDNxgNdUpwSnpb3K",
  "confirmations": 0,
  "balance": 100945889661986
}
```



**POST /addresses/balance/details**

Получение балансов для списка адресов.

**Запрос метода:**

```
{
  "addresses": [
    "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ", "3N11u447zghwj9MemYkrkt9v9xDaMwTY9nG"
  ]
}
```

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

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

**Ответ метода:**

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

**GET /addresses/effectiveBalance/{address}**

Возвращает общий баланс аккаунта.

**Ответ метода**

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

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

Возвращает генерирующий баланс адреса на указанной высоте.

**Примечание:** Метод показывает генерирующий баланс, определённый на высоте не ниже, чем 2000 блоков назад.

**Запрос метода:**

```
{
  "address": "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ",
  "height": 1000
}
```

**Ответ метода:**

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

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

Возвращает подробные сведения о балансе адресата {address}.

**Запрос метода:**

```
{
  "addresses": [
    "3N65yEf31ojBZUvpu4LCo7n8D73juFtheUJ"
  ]
}
```

**Ответ метода:**

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

### Параметры ответа

- Regular — общий баланс участника, включая средства переданные в лизинг
- Available — общий баланс участника, за исключением средств переданных в лизинг
- Effective — общий баланс участника, включая средства переданные участнику за лизинг (Available + средства переданные вам в лизинг)
- Generating — минимальный баланс участника, включая средства переданные участнику за лизинг, за последние 1000 блоков (используется для майнинга)

### GET /addresses/scriptInfo/{address}

Получение данных об установленном скрипте на адресе {address}.

**Ответ метода:**

```
{
  "address": "3N3keodUiS8WLEw9W4BKDNxgNdUpwSnpb3K",
  "script":
    ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SoCMRkRKfGzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
    ↪",
  "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,
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

"extraFee": 10001
}

```

**Параметры ответа**

- «address» адрес в формате Base58
- «script» Base64 представление скрипта
- «scriptText» исходный код скрипта
- «complexity» сложность скрипта
- «extraFee» комиссия за исходящие транзакции, установленные скриптом

**POST /addresses/sign/{address}**

Возвращает закодированное в формат Base58 сообщение, подписанное приватным ключом адресата {address}, сохраненным в keystore ноды. Сообщение сначала подписывается, после этого выполняется преобразование.

**Запрос метода:**

```

{
  "message": "mytext"
}

```

**Ответ метода:**

```

{
  "message": "wWshKhJj",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
    ↪ "62PFG855ThsEHUZ4N8VE8kMyHCK9GWnvtTZ3hq6JHYv12BhP1eRjegA6nSa3DAoTTMammhamadvizDUYZAZtKY9S"
}

```

**POST /addresses/verify/{address}**

Проверяет подпись сообщения, выполненную адресатом {address}, в т.ч. созданную через метод POST /addresses/sign/{address}.

**Запрос метода:**

```

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

```

**Ответ метода:**

```

{
  "valid": true
}

```

**POST /addresses/signText/{address}**

Возвращает сообщение, подписанное приватным ключом адресата {address}, сохраненным в keystore ноды.

**Запрос метода:**

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

**Ответ метода:**

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
    ↪ "5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjqwHK7Zjnrzs9QYRwo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u"
}
```

**POST /addresses/verifyText/{address}**

Проверяет подпись сообщения, выполненную адресатом {address}, в т.ч. созданную через метод POST /addresses/signText/{address}.

**Запрос метода:**

```
{
  "message": "message",
  "publicKey": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
    ↪ "5kVZfWfFmoYn38cJfNhkdct5WCyksMgQ7kjqwHK7Zjnrzs9QYRwo6HuJoGc8WRMozdYcAVJvojJnPpArqPvu2uc3u"
}
```

**Ответ метода:**

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

**GET /addresses/validate/{addressOrAlias}**

Проверяет корректность заданного адресата или его псевдонима {addressOrAlias} в блокчейнсети работающей ноды.

**Ответ метода:**

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

**POST /addresses/validateMany**

Проверяет валидность адресов или алиасов.

**Запрос метода:**

```
{
  addressesOrAliases: [
    "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
    "alias:T:asdfghjk",
    "alias:T:1nvAliDA11ass99911%~&$$$ "
  ]
}
```

**Ответ метода:**

```
{
  validations: [
    {
      addressOrAlias: "3HSVTtjim3FmV21HWQ1LurMhFzjut7Aa1Ac",
      valid: true
    },
    {
      addressOrAlias: "alias:T:asdfghjk",
      valid: true
    },
    {
      addressOrAlias: "alias:T:1nvAliDA11ass99911%~&$$$ ",
      valid: false,
      reason: "GenericError(Alias should contain only following characters: -.0123456789@_
↪abcdefghijklmnopqrstuvwxyz)"
    }
  ]
}
```

**GET /addresses/publicKey/{publicKey}**

Возвращает адрес участника на основании его публичного ключа.

**Ответ метода:**

```
{
  "address": "3N4WaaaNAVLMQgVKTRSePgWBuAKvZTjAQbq"
}
```

**GET /addresses/data/{address}**

Возвращает все данные, записанные на аккаунт адресата {address}.

**Ответ метода:**

```
[
  {
    "key": "4yR7b6Gv2rzLrhYBHpVCmLH42raPGTF4Ggi1N36aWnY",
    "type": "integer",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
[
  {
    "value": 1500000
  }
]
```

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

Возвращает данные, записанные на аккаунт адресата {address} по ключу {key}.

#### Ответ метода:

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

### Alias

---

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

---

### GET /alias/byalias/{alias}

Получает адрес участника по его псевдониму {alias}.

#### Ответ метода

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

### GET /alias/byaddress/{address}

Получает {alias} псевдоним участника по его адресу {address}.

#### Ответ метода

```
[
  "alias:HUMANREADABLE1",
  "alias:HUMANREADABLE2",
  "alias:HUMANREADABLE3",
]
```

## Anchoring

### GET /anchoring/config

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

Получение секции *анкоринга* конфигурационного файла ноды.

#### Ответ метода

```

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

```

## Assets

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

### GET /assets/balance/{address}

Возвращает баланс всех ассетов адресата {address}.

#### Ответ метода:

```

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

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    }
  },
  {
    "assetId": "49KfHPJcKvSAvNKwM7CTofjKHZL87SaSx8eyADBjv5Wi",
    "balance": 10,
    "quantity": 10000000000,
    "reissuable": false,
    "issueTransaction" : {
      "type" : 3,
      ...
    }
  }
]
}

```

**Параметры метода:**

- «address» адрес участника
- «balances» объект с балансами участника
- «assetId» ассет ID
- «balance» баланс ассета
- «quantity» колво выпущенных ассетов
- «reissuable» признак может быть ассет перевыпущен или нет
- «issueTransaction» транзакция создания ассета
- «minSponsoredAssetFee» минимальное значение комиссии для спонсорских транзакций
- «sponsorBalance» средства, выделенные для оплаты транзакций по спонсируемым ассетам

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

Возвращает баланс адресата {address} по ассету {assetId}.

**Ответ метода:**

```

{
  "address": "3Mv61qe6egMSjRDZiiuvJDnf3Q1qW9tTZDB",
  "assetId": "Ax9T4grFxx5m3KPUEKjMdnQkCKtBktf694wU2wJYvQUUD",
  "balance": 4879179221
}

```

**GET /assets/details/{assetId}**

Возвращает описание ассета {assetId}.

**Ответ метода:**

```

{
  "assetId" : "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ",
  "issueHeight" : 140194,
  "issueTimestamp" : 1504015013373,

```

(continues on next page)



(продолжение с предыдущей страницы)

```

"issuer" : "3NCBMxgdghg4tUhEEffSXy11L6hUi6fcBpd",
"name" : "name",
"description" : "Sponsored asset",
"decimals" : 1,
"reissuable" : true,
"quantity" : 1221905614,
"script" : null,
"scriptText" : null,
"complexity" : 0,
"extraFee": 0,
"minSponsoredAssetFee" : 100000 // null assume no sponsorship, number - amount of assets for
↪ minimal fee
}

```

**GET /assets/{assetId}/distribution**

Возвращает распределение ассета {assetId}.

**Ответ метода:**

```

{
  "3P8GxcTEyZtG6LEfnn9knp9wu8uLKrAFHCb": 1,
  "3P2voHxcJg79csj4YspNqlakepX8TSmGhTE": 1200
}

```

**POST /assets/balance**

Возвращает баланс активов для одного или нескольких адресов.

**Ответ метода**

```

{
  "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG": [],
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZyNh84pn": []
}

```

**Blocks**

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

Последний блок в течение периода его создания может содержать разное количество транзакций. Это связано с тем, что пока блок не принят нодами майнерами, количество транзакций в нем может постоянно меняться. Поэтому при использовании методов, предоставляющих информацию о последнем блоке, следует иметь в виду, что количество транзакций в последнем блоке может измениться.

**GET /blocks/height**

Возвращает номер блока текущего состояния блокчейна.

**Ответ метода:**

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

**GET /blocks/height/{signature}**

Возвращает высоту (номер) блока по его подписи.

**GET /blocks/first**

Возвращает содержимое первого блока (genesis block).

**GET /blocks/last**

Возвращает содержимое последнего блока.

**Ответ метода:**

```
{
  "version": 2,
  "timestamp": 1479313809528,
  "reference":
  ↪ "4MLXQDbARiJDEAoy5vZ8QYh1yNnDhdGhGWkDKna8J6QXb7agVpFEi16hHBGUxxnq8x4myG4w66DR4Ze8FM5dh8Gi",
  "nxtconsensus": {
    "basetarget": 464,
    "generationsignature": "7WUV2TufaRAyjiCPFDnAWbn2Q7Jk7nBmWbnnDXKDEeJv"
  },
  "transactions": [
    {
      "type": 2,
      "id":
      ↪ "64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
      "fee": 100000,
      "timestamp": 1479313757194,
      "signature":
      ↪ "64hxaxZvB9iD1cfRf1j8KPTXs4qE7SHaDWTZKoUvgfVZotaJUtSGa5Bxi86ufAfp5ifoNAGknBqS9CpxBKG9RNVR",
      "sender": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8",
      "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHmM3Uki7pLw",
      "recipient": "3N8UPtqiy322NVr1fLP7SaK1AaCU7oPaVuy",
      "amount": 1000000000
    }
  ],
  "generator": "3N5GRqzDBhjVXnCN44baHcz2GoZy5qLxtTh",
  "signature":
  ↪ "4ZhZdLAvaGneLU4K4b2eTgRQvbBjEZrtwo1qAhM9ar3A3weGEutbfNKM4WJ9JZnV8BXenx8JRGVNwpfx3prGaxd",
  "fee": 100000,
  "blocksize": 369
}
```

**GET /blocks/at/{height}**

Возвращает содержимое блока на высоте {height}.

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

Возвращает содержимое блоков в диапазоне от {from} до {to}.

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

Возвращает содержимое блоков с расширенной информацией о транзакциях в диапазоне от {from} до {to}.

**GET /blocks/signature/{signature}**

Возвращает содержимое блока по его подписи {signature}.

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

Возвращает все блоки сформированные (смайненные) адресатом {address}.

**GET /blocks/child/{signature}**

Возвращает унаследованный блок от блока с подписью {signature}.

**GET /blocks/headers/at/{height}**

Возвращает заголовок блока на высоте {height}.

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

Возвращает заголовки блоков диапазоне от {from} до {to}.

**GET /blocks/headers/last**

Возвращает заголовок последнего блока в блокчейне.

## Consensus

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

### GET /consensus/algo

Возвращает тип алгоритма консенсуса, используемый в сети.

**Ответ метода:**

```
{
  "consensusAlgo": "Fair Proof-of-Stake (FairPoS)"
}
```

### GET /consensus/settings

Возвращает параметры консенсуса, заданные в конфигурационном файле ноды.

**Ответ метода:**

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

### GET /consensus/minersAtHeight/{height}

Возвращает очередь майнеров на высоте {height}.

**Ответ метода:**

```
{
  "miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxFxLanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
  "height": 1
}
```

**GET /consensus/miners/{timestamp}**

Возвращает очередь майнеров на время {timestamp}.

**Ответ метода:**

```
{
  "miners": [
    "3Mx5sDq4NXef1BRzJRAofa3orYFxLanxmd7",
    "3N2EsS6hJPYgRn7WFJHLJNnrsm92sUKcXkd",
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
    "3N6pfQJyqjLCmMbU7G5sNABLmSF5aFT4KTF",
    "3NBbipRYQmZFudFCoVJXg9JMkkyZ4DEdZNS"
  ],
  "timestamp": 1547804621000
}
```

**GET /consensus/bannedMiners/{height}**

Возвращает список заблокированных майнеров на высоте {height}.

**Ответ метода:**

```
{
  "bannedMiners": [],
  "height": 1000
}
```

**GET /consensus/basetarget/{blockId}**

Возвращает значение базовой сложности (basetarget) создания блока {blockId}.

**GET /consensus/basetarget**

Возвращает значение базовой сложности (basetarget) создания последнего блока.

**GET /consensus/generatingbalance/{address}**

Возвращает генерирующий баланс доступный для майниновой ноды {address} минимальный баланс участника, включая средства переданные участнику за лизинг, за последние 1000 блоков.

**GET /consensus/generationsignature/{blockId}**

Возвращает значение **генерирующей подписи** (generation signature) создания блока {blockId}.

**GET /consensus/generationsignature**

Возвращает значение **генерирующей подписи** (generation signature) последнего блока.

**Contracts**


---

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

---

**GET /contracts**

Возвращает информацию по контрактам.

**Ответ метода**

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

**POST /contracts**

Возвращает некоторые параметры по одному или нескольким заданным в запросе идентификаторам контрактов.

**Параметры запроса:**

```
{
  "contracts": [
    "string"
  ]
}
```

**Ответ метода**

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

```

(continues on next page)

(продолжение с предыдущей страницы)

```

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

```

**GET /contracts/info/{contractId}**

Возвращает актуальную информацию по версии указанного контракта, его локации и хешсуммы образа.

**Параметры запроса:**

```
"Contract id"
```

**Ответ метода**

```

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

```

**GET /contracts/status/{id}**

Возвращает статус исполняемой транзакции по контракту.

**Параметры запроса:**

```
"id" - Transaction ID
```

**Ответ метода**

```

[
  {
    "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)

```

"sender": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
"senderPublicKey": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
"txId": "4q5Q8vLeGBpcdQofZikyrrjHUS4pB1AB4qNE2yHRKWU",
"status": "Success",
"code": null,
"message": "Smart contract transaction successfully mined",
"timestamp": 1558961376012,
"signature":
↪ "3Vhqc9DvNhMvFFtWnBuV4XwQ62ZcTAvLNZYmeGc7mGzMcnGZ3RlshDs393fnQu1WTh8CmL58YnvnjyULEEi5yorV"
}
]

```

### GET /contracts/{contractId}

Возвращает результат исполнения смартконтракта по его идентификатору (id транзакции создания контракта).

#### Параметры запроса

```

"contractId" - Contract ID
"offset" - Offset number
"matches" - String for matches search
"limit" - Limit number

```

#### Ответ метода:

```

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

```

### POST /contracts/{contractId}

Возвращает ключи смартконтрактов по их идентификатору (id транзакции создания контракта).

#### Параметры запроса

```

"Contract Id"
{
  "keys": [
    "string"
  ]
}

```

#### Ответ метода:



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

### GET /contracts/executedtxfor/{id}

Возвращает результат исполнения смартконтракта по идентификатору транзакции исполнения контракта.

#### Параметры запроса:

"id" - Transaction ID

#### Ответ метода:

```
{
  "type": 105,
  "id": "2UAHvs4KsfBbRVPm2dCigWtqUHuaNQou83CXy6DGDiRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqLCqouVrFZynjfoTEuPNV9GrdauNpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
    "4BoG6wQnYyZWYUKzAwh5n1184tsEWUqUTWmXMExvvcU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
  ],
  "version": 1,
  "tx": {
    "type": 103,
    "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
    "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
    "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhAv38Dws5skqDsjMVT2M",
    "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 исполнения смартконтракта по его идентификатору (id транзакции создания контракта) и ключу {key}.

**Параметры запроса:**

```
"Contract id"
"key" - Key name
```

**Ответ метода:**

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

**Crypto**

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

**POST /crypto/encryptSeparate**

Шифрует текст отдельно для каждого получателя уникальным ключом.

**Запрос метода**

```
{
  "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
  "password": "some string as a password",
  "encryptionText": "some text to encrypt",
  "recipientsPublicKeys": [
    ↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc",
    "9LopMj2GqWxBYgnZ2gxaNwxXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S"]
}
```

**Пример ответа**

```
{
  "encryptedText": "IZ5Kk5YNspMWl/jmlTizVxD6Nik=",
  "publicKey":
    ↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc",
  "wrappedKey":
    ↪ "uWVoXJAzruwTDDsbphDS31TjSQX6CSWXivp3x34uE3XtnMqK9swoaZ3LyAgFDR7o6CfkgzFkWmTen4qAZewPfBbwR"
},
{
  "encryptedText": "F9u010RGvSEDe6dWm1pzJQ+3xqE=",
  "publicKey":
    ↪ "9LopMj2GqWxBYgnZ2gxaNwxXqxXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S",
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

"wrappedKey":
↪ "LdzdoKadUzBTmwcZGYgu1AM4YrbBLr9Uh1MvQ3MPcLZUhCD9herz4dv1m6ssaVHPiBNUGgqKnLZ6Si4Cc64UvhXBbG"
}

```

## POST /crypto/encryptCommon

Шифрует данные единым ключом СЕК для всех получателей, СЕК оборачивается уникальными КЕК для каждого получателя.

### Запрос метода

```

{
  "sender": "3MCUfX4P4U56hoQwSqXnLJenB6cDkxBjisL",
  "password": "some string as a password",
  "encryptionText": "some text to encrypt",
  "recipientsPublicKeys": [
↪ "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc",
    "9LopMj2GqWxBYgnZ2gxaNxxXqXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S"]
}

```

### Пример ответа

```

{
  "encryptedText": "NpCCig2i3jzo0xBnfqjfedbti8Y=",
  "recipientToWrappedStructure": {
    "5R65oLxp3iwPekwirA4VwwUXaySz6W6YKXBKBRl352pwwcpsFcjRHJ1VVHLp63LkrkxsNod64V1pffeizZ5i2qXc":
    "M8pAe8HnKiWLE1HsC1ML5t8b7giWxiHfvagh7Y3F7rZL8q1tqMCJMYJo4qz4b3xjcuuUiV57tY3k7oSig53Aw1Dkkw",
    "9LopMj2GqWxBYgnZ2gxaNxxXqXHuWd6ZAdVqkprR1fFMNvDUHYUCwFxsB79B9sefgxNdqwNtqzuDS8Zmn48w3S":
    "Doqn6gPvBBesSu2vdwgfYmBDHM4knEGMbqPn8Np76mNRRoZXLdioofyVbSSaTTER4cvXwzEwVMugiy2wuzFWk3zCiT3"
  }
}

```

## POST /crypto/decrypt

Расшифровывает данные. Расшифровка доступна в случае, если ключ получателя сообщения находится в keystore ноды.

### Запрос метода

```

{
  "recipient": "3M5F8B1qxSY1W6kA2ZnQiDB4JTGz9W1jvQy",
  "password": "some string as a password",
  "encryptedText": "oiKFJijfid8HkjsjdhKHHud987d",
  "wrappedKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy"
  "senderPublicKey": "M5F8B1qxSY1W6kA2ZnQiDB4JTGzA2ZnQiDB4JTGz9W1jvQy",
}

```

### Пример ответа

```

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

```

## Debug

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

### GET /debug/blocks/{howMany}

Отображает размер и полный хеш последних блоков. Количество блоков указывается при запросе.

#### Ответ метода

```
[
  {
    "226": "7CkZxrAjU8bnat8CjVAPagobNYazyv1HASubmp7YYqGe"
  },
  {
    "226": "GS3y9fUHAkCamq52TPsjizDVir8J7iGoe8P2XZLasxsC"
  },
  {
    "226": "B9LmhGGDdvfcUA9JEWvyVrT9sazZE6gibpAN13xUN7KV"
  },
  {
    "226": "Byb9MHtwYf3MFyi2tbhQ3GTdCct5phKq9REkbjQTzdne"
  },
  {
    "226": "HSxSHbiV4tZc8RaN6jxdhgkAhjxuLn76uHxerMRUefA"
  }
]
```

### GET /debug/info

Отображает необходимую информацию для отладки и тестирования.

#### Ответ метода

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

(continues on next page)

(продолжение с предыдущей страницы)

```
{
  "minerState": "mining microblocks"
}
```

## POST /debug/rollback

Убирает из блокчейна все блоки после указанной высоты.

### Запрос метода

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

### Ответ метода

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

## POST /debug/validate

Валидирует транзакции и измеряет затраченное время в миллисекундах.

### Параметры запроса

```
"id" - Transaction ID
```

### Ответ метода

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

## GET /debug/minerInfo

Отображает информацию о майнере, необходимую для отладки.

### Ответ метода

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

## GET /debug/historyInfo

Отображает историю последнего блока, необходимую для отладки.

### Ответ метода

```
{
  "lastBlockIds": [
    "37P4fvexYHPUzNPRRqYbRYxGz7x3r5jFznck7amaS6aWnHL5oQqrqCzsSh1HvYKnd2ZhU6n6sWYPb3hxsY8FBfmZ",
    "5RRu1qtesz4KvrVp4fzxQHebq2fRanNsg3HJKwD4uChqySm7vFHCdHKU6iZYXJDVmfSxiE9Maeb6sM2JireaWlBx",
    "3Lo27JfjekcZnJsYEe7st7evDZ6TgmCUBtiZrSxUCobKL48DZQ4dXMfp89WYjEykh15HEHSXzqMSTQigE8vEcN2r",
    "r4RuxEXAqgfDMKVXRWmZcGMaWKDsAvVxfXDtw8d6bamLR61J1gaoesargYSoZQqRbDrBcefLprk7D78fA728719",
    "3F4Up46crZbpKVWUeieL6GeSrVMYm7JJ7aX6aHD6B8wedFggSKv8d3H39Qy9MLEauFBu9m3qZV1U8emhmqwmLbg",
    "QSuBkEtVe9nik5T5S33ogeCbgKy7ihBkS2pwYayK23m4ANier83ThpajEzvpbyPy9pPWzc5St8mYUKxXDscKuRC",
    "4udpNnz3e1M1GbVZxtwfg8gpF6EbiKxRCRBwi6iRMylsvh5J2Ec9Wqyu2sq2KYL75o12yiP8TszworeUfuxNmJ5g",
    "5BZYZ4RZAJjM5KKCaHpyUsXnb4uunnM5kcfTojc5QzQo3vyP2w3YD4qrALizkkQQR4ziS77BoAGb56QCecUtHFFN",
    "5JwfLaF1oGxRXVCdDbFuKpxrvxgLCGU3kCFwxUhlL8G3xV211MrKBuAuQ4MaC5uN574uV9U8M6HfHTMERnfr5jGJ",
    "4bysMhz14E1rC7dLYScfVVqPmHqzi8jdchcnkruJmCNL86TwV2cbF7G9YVchvTrv9qbQZ7JQownV59gRRcD26zm16"
  ],
  "microBlockIds": []
}
```

## GET /debug/configInfo

Отображает конфигурационный файл ноды.

### Ответ метода

```
{
  "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/rollbackto/{signature}**

Откатывает блокчейн до блока с указанной подписью.

**Параметры запроса**

"signature" - Block signature

**Ответ метода**

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

**GET /debug/portfolios/{address}**

Отображает текущий портфель неучтённых транзакций в UTX пуле.

**Параметры запроса**

"address" - Node address

**Ответ метода**

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

**POST /debug/print**

Распечатывает строку при уровне логирования DEBUG в логфайл.

**Запрос метода**

```
{
  "message": "string"
}
```

## GET /debug/state

Отображает текущий стейт ноды.

### Ответ метода

```
{
  "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}

Отображает стейт ноды на указанной высоте.

### Параметры запроса

"height" - Block height

### Ответ метода

```
{
  "3JPWx147Xf3f9fE89YtfvRhtKWBHy9rWnMK": 17528100000,
  "3JU5tCoswHH7FKPBuowySWBnQwpbZiYyNhB": 300020907600000,
  "3JCJChsQ2CGyHc9Ymu8cnsES6YzjjJELu3a": 75000350600000,
  "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3": 6847000000,
  "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF": 1248960085800000,
  "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwD9t": 994280900000
}
```

## Leasing

**Подсказка:** Правила формирования запросов к ноды приведены в разделе *Как использовать REST API*.



**GET /leasing/active/{address}**

Возвращает список транзакций создания лизинга, в которых принимал участие {address}, как отправитель или получатель.

**Ответ метода:**

```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEWoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVvRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
    ↪ "25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPfyegP82sNSfM2bXNX3K7p6D3HD",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```

```
[
  {
    "type": 8,
    "id": "2jWhz6uGYsgvfoMzNR5EEGdi9eafyCA2zLFfkM4NP6T7",
    "sender": "3PP6vdkEWoif7AZDtSeSDtZcwiqSfhmwttE",
    "senderPublicKey": "DW9NKLeyoEWDqJKhWv87EdFfTqpFtJBWoCqfCVvRhsY",
    "fee": 100000,
    "timestamp": 1544390280347,
    "signature":
    ↪ "25kpwh7nYjRUtfbAbWYRyMDPCUCoyMoUuWTJ6vZQrXsZYXbdiWHa9iGscTTGnPfyegP82sNSfM2bXNX3K7p6D3HD",
    "version": 1,
    "amount": 31377465877,
    "recipient": "3P3RD3yJW2gQ9dSVwVVDVCQiFWqaLtZcyzH",
    "height": 1298747
  }
]
```

**Licenses**

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

## GET /licenses

Возвращает список всех загруженных лицензий.

### Ответ метода

```
[
  {
    "license": {
      "version": 1,
      "id": "3GLWx8yUFcNSL3DER8kZyE4TpyAyNiEYsKG",
      "license_type": null,
      "issued_at": "2020-02-27T16:11:14.784Z",
      "node_owner_address": "4WnvQPit2Di1iYXDgDcXnJZ5yroKW54vauNoxdNeMi2g",
      "valid_from": "2020-02-20",
      "valid_to": "2020-02-27",
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key": "dmLTlippM7tmfSC8u9P4wU6sBgHXGYy6JYxCq1CCh8i",
    "signature":
    ↪ "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957mLTlippM7tmfSC8u",
    "signer_id": "ff9b8af966b4c84e66d3847a514e65f55b2c1f63afcd8b708b9948a814cb8957"
  },
  {
    "license": {
      "version": 1,
      "id": "49KfHPJcKvSAvNKwM7CTofjKHZL87SaSx8eyADBjv5Wi",
      "license_type": null,
      "issued_at": "2020-02-27T16:12:34.327Z",
      "node_owner_address": "3N4WaaaNAVLMQgVKTRSePgwBuAKvZTjAQbq",
      "valid_from": "2020-02-29",
      "valid_to": null,
      "features": [
        "all_inclusive"
      ]
    },
    "signer_public_key": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
    "signature":
    ↪ "5kwwE9sDZzssNaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts",
    "signer_id": "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxvVRTQRnMQ"
  }
]
```

## GET /licenses/status

Возвращает статус активации лицензии ноды.

### Ответ метода

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

## POST /licenses/upload

Добавляет новую лицензию в JSON формате в ноду.

### Запрос метода

```
{
  "license": {
    "version": 1,
    "id": "49KfHPJcKvSAvNKwM7CTofjKHZL87SaSx8eyADBjv5Wi",
    "license_type": null,
    "issued_at": "2020-02-27T16:12:34.327Z",
    "node_owner_address": "3N4WaaaNAVLMQgVKTRSePgWbUAKvZTjAQbq",
    "valid_from": "2020-02-29",
    "valid_to": null,
    "features": [
      "all_inclusive"
    ]
  },
  "signer_public_key": "C1ADP1tNGuSLTiQrfNRPhgXx59nCrwrZFRV4AHpfKBpZ",
  "signature":
  ↪ "5kwwE9sDZss0NaoBSJnb8RLqfYGt1NDGbTWWXUeX8b9amRRJN3hr5fhs9vHBq6VES5ng4hqbCUoDEsoQNauRRts",
  "signer_id": "8tdULCMr598Kn2dUaKwHkvsNyFbDB1Uj5NxxvVRTQRnMQ"
}
```

### Ответ метода

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

## Node

**Подсказка:** Правила формирования запросов к ноду приведены в разделе *Как использовать REST API*.

## GET /node/config

Возвращает основные конфигурационные параметры ноды.

### Ответ метода:

```
{
  {
    "version": "1.3.0-RC7",
    "gostCrypto": false,
    "chainId": "V",
    "consensus": "POA",
    "minimumFee": {
      "3": 0,
      "4": 0,
      "5": 0,
```

(continues on next page)

(продолжение с предыдущей страницы)

```

        "6": 0,
        "7": 0,
        "8": 0,
        "9": 0,
        "10": 0,
        "11": 0,
        "12": 0,
        "13": 0,
        "14": 0,
        "15": 0,
        "102": 0,
        "103": 0,
        "104": 0,
        "106": 0,
        "107": 0,
        "111": 0,
        "112": 0,
        "113": 0,
        "114": 0
    },
    "additionalFee": {
        "11": 0,
        "12": 0
    },
    "maxTransactionsInMicroBlock": 500,
    "minMicroBlockAge": 0,
    "microBlockInterval": 1000,
    "blockTiming": {
        "roundDuration": 7000,
        "syncDuration": 700
    }
}

```

## GET /node/logging

Отображает список логгеров и их уровень логирования для каждого отдельно.

### Ответ метода:

```

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

```

## POST /node/logging

Устанавливает определённый уровень логирования для указанных логов.

### Запрос метода

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

### Ответ метода:

## POST /node/stop

Запрос останавливает ноду.

## GET /node/status

Возвращает основные конфигурационные параметры ноды.

### Ответ метода:

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

---

**Примечание:** Если возникают какие-либо ошибки при использовании ГОСТ-криптографии на ноду, этот метод укажет на возможные ошибки с JCP:

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

---

## GET /node/version

Возвращает версию приложения.

### Ответ метода:

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

**GET /node/owner**

Возвращает адрес и публичный ключ владельца ноды.

**Ответ метода:**

```
{
  "address": "3JFR1pmL6biTzr9oa63gJcjZ8ih429KD3aF",
  "publicKey": "EPxkVA9iQejsjQikovyxkkY8iHnbXsR3wjkgE7ZW1Tt"
}
```

**Peers**


---

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

---

**POST /peers/connect**

Запрос на подключение нового узла к ноде.

**Запрос метода:**

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

**Ответ метода:**

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

**GET /peers/connected**

Возвращает список подключенных нод.

**Ответ метода:**

```
{
  "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",

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "peerName": "zx 217",
    "peerNonce": 1021800
  }
]
}

```

**GET /peers/all**

Возвращает список всех известных нод.

**Ответ метода:**

```

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

```

**GET /peers/suspended**

Возвращает список suspended нод.

**Ответ метода:**

```

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

```

**POST /peers/identity**

Получение публичного ключа пира, к которому подключается нода для передачи конфиденциальных данных.

**Запрос метода:**

```

{
  "address": "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "signature":
    ↪ "6RwMUQcwrxtKDgM4ANes9Amu5EJgyfF9Bo6nTpXyD89ZKMAcpCM97igbWf2MmLXLdqNxdsUc68fd5TyRBEB6nqf"
}

```

**Параметры:**

- address блокчейнадрес, который соответствует параметру «privacy.owneraddress» в конфигурационном файле ноды;
- signature электронная подпись от значения поля «address».

**Ответ метода:**

```
{
  "publicKey": "3NBVqYXrapgJP9atQccdBPAGJPwHDKkh6A8"
}
```

**Параметры:**

- `publicKey` публичный ключ пира, связанный с параметром «`privacy.owneraddress`» в его конфигурационном файле. Если выключен режим проверки `handshakes`, то параметр `publicKey` не отображается.

**GET /peers/hostname/{address}**

Получение `hostname` и IPадреса ноды по ее адресу в сети Waves Enterprise.

**Ответ метода:**

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

**GET /peers/allowedNodes**

Получение актуального списка разрешенных участников сети на момент запроса.

**Ответ метода:**

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



## Permissions

---

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

---

### GET /permissions/{address}

Возвращает роли (permissions), назначенные на указанный адрес {address}, действительные на текущий момент времени.

**Ответ метода:**

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

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

Возвращает роли (permissions), назначенные на указанный адрес {address}, действительные на момент времени {timestamp}.

**Ответ метода:**

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

**POST /permissions/addresses**

Возвращает роли (permissions), назначенные на список адресов, действительные на текущий момент времени.

**Запрос метода:**

```
{
  "addresses": [
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w", "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7"
  ],
  "timestamp": 1544703449430
}
```

**Ответ метода:**

```
{
  "addressToRoles": [
    {
      "address": "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w",
      "roles": [
        {
          "role": "miner"
        },
        {
          "role": "permissioner"
        }
      ]
    },
    {
      "address": "3Mx5sDq4NXef1BRzJRAofa3orYFxFanxmd7",
      "roles": [
        {
          "role": "miner"
        }
      ]
    }
  ],
  "timestamp": 1544703449430
}
```

**PKI**

**Предупреждение:** Методы PKI работают только с ГОСТ криптографией.

В PKI используются форматы ЭП, приведенные в таблице ниже. Номер формата ЭП из таблицы соответствует значению поля sigtype.

Таблица 1: Форматы ЭП

№	Формат ЭП
1	CAdESBES
2	CAdESX Long Type 1
3	CAdEST

## POST /pki/sign

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

Метод формирует отсоединённую ЭП для данных, передаваемых в запросе. В данном запросе `inputData` это данные для формирования ЭП в виде массива байт в кодировке **Base64**, `keystoreAlias` это наименование ключевого контейнера закрытого ключа ЭП. Также необходимо указать пароль от ключевого контейнера в параметре `password`.

### Пример запроса

```
{
  "inputData" : "SGVsbG8gd29ybGQh",
  "keystoreAlias" : "key1",
  "password" : "password",
  "sigType" : "CAdES_X_Long_Type_1",
}
```

### Пример ответа

```
{
  "signature" :
  ↪ "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQjsndjfnksdnjfn="
}
```

## GET /pki/keystoreAliases

Метод возвращает список всех keystoreалиасов на ГОСТ криптографии.

### Пример ответа

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

## POST /pki/verify

Метод выполняет проверку УКЭП для данных, переданных в запросе. Поле `extendedKeyUsageList` является опциональным и может содержать массив значений OID (Объектный идентификатор) для определения области действия сертификата. Проверку сертификата может осуществлять любая нода, имеющая параметры запроса.

### Пример запроса

```
{
  "inputData" : "SGVsbG8gd29ybGQh",
  "signature" : "c2RmZ3NkZmZoZ2ZkZ2hmZGpkZ2ZoamhnZmtqaGdmamtkZmdoZmdkc2doZmQ=",
  "sigType" : "CAAdES_X_Long_Type_1",
  "extendedKeyUsageList": [
    "1.2.643.7.1.1.1.1",
    "1.2.643.2.2.35.2"
  ]
}
```

### Пример ответа

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

## Работа с методом POST /pki/verify

Нода имеет возможность проверять УКЭП (Усиленная квалифицированная электронная подпись), используя метод API `Post /pki/verify`. Для корректности работы метода API `Post /pki/verify` необходимо установить корневой сертификат на ноду. Корневой сертификат УЦ однозначно идентифицирует центр сертификации и является основанием в цепочке доверия.

### Как установить корневой сертификат на ноду

Корневой сертификат устанавливается в следующую папку со средой Java:

```
-keystore /Library/Java/JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/jre/lib/
↳ security/cacerts
```

Пароль по умолчанию на хранилище сертификатов Java `cacerts` `changeit`. При желании вы можете изменить пароль. Установка сертификатов выполняется следующей командой:

```
sudo keytool -import -alias testAliasCA_cryptopro -keystore /Library/Java/
↳ JavaVirtualMachines/jdk1.8.0_191.jdk/Contents/Home/jre/lib/security/cacerts -file ~/
↳ Downloads/cert.cer
```

## Privacy

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

## POST /privacy/sendData

Запись конфиденциальных данных в хранилище ноды.

## Запрос метода:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "password": "apgJP9atQccdBPA",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "info": {
    "filename": "Service contract #100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "some comments"
  },
  "data":
  ↪ "TWFuIGlzlGRpc3Rpbmd1aXNoZWQsIG5vdCBvbmx5IGJ5IGhpcyByZWZzb24sIGJ1dCBieSB0aGlzIHNpbmd1bGFyIHBhc3Npb24gZnJvbSBvdGhl",
  ↪ ",
  "hash": "FRog42mmzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta"
}
```

## Параметры:

- sender блокчейнадрес, от которого должны рассылаться данные (соответствуют значению параметра «privacy.owneraddress» в конфигурационном файле ноды);
- password пароль для доступа к закрытому ключу keystore ноды;
- policyId идентификатор группы, в рамках которой пересылаются данные;
- type тип данных;
- info информация о данных;
- data строка, содержащая данные в формате **base64**;
- hash sha256хеш данных в формате **base58**.

## Ответ метода:

```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrngLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mmzTA292ukng6PHoEK9Mpx9GZNrEHecfvpwmta",
  "proofs": [
    "2jM4tw4uDmspuXUBt6492T7opuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc"
  ],
  "fee": 110000000,
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
{
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
}
```

**POST /privacy/sendDataV2**

Вторая версия метода *POST /privacy/sendData*, позволяющего выполнять поточную загрузку файлов с конфиденциальными данными в хранилище ноды.

**Запрос метода:**

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "type": "file",
  "hash": "e67ad392ab4d933f39d5234asdd96c18c491140e119d590103e7fd6de15623f9",
  "info": {
    "filename": "Договор об оказании услуг №100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "la la fam"
  },
  "fee": 15000000,
  "password": "12345qwerty",
  "timestamp": 0
}
```

Параметры отличаются от параметров метода *POST /privacy/sendData* только отсутствием поля Data. Вместо заполнения данного поля вам необходимо выбрать и приложить файл с данными в соответствующем окне Swagger.

**Ответ метода:**

```
{
  "senderPublicKey": "Gt3o1ghh2M2TS65UrHZCTJ82LLcMcBrxuaJyrsgLk5VY",
  "policyId": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aGljaC",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "dataHash": "FRog42mmzTA292ukng6PHoEK9Mpx9GZNRHEcfvpwmta",
  "proofs": [
    "2jm4tw4uDmspuXUBt6492T7oPuZskYhFGW9gkbq532BvLYRF6RJn3hVGNLuMLK8JSM61GkVgYvYJg9UscAayEYfc"
  ],
  "fee": 110000000,
  "id": "H3bdFTatppjnMmUe38YWh35Lmf4XDYrgsDK1P3KgQ5aa",
  "type": 114,
  "timestamp": 1571043910570
}
```

**GET /privacy/{policyid}/recipients**

Получение адресов всех участников, записанных в группу {policyid}.

Ответ метода:

```
[
  "3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "3Mx2afTZ2KbRrLNbytyzTtXukZvqEB8SkW7"
]
```

**GET /privacy/{policyid}/owners**

Получение адресов всех владельцев, записанных в группу {policyid}.

Ответ метода:

```
[
  "3GCFaCWtvLDnC9yX29YftMbn75gwfdwGsBn",
  "3GGxcmNyq8ZAHzK7or14Ma84khW8peBohJ",
  "3GRLFi4rz3SniCuC7rbd9UuD2KUZyNh84pn",
  "3GKpShRQRtddF1yYhQ58ZnKMTnp2xdEzKqW"
]
```

**GET /privacy/{policyid}/hashes**

Получение массива идентификационных хешей, которые записаны в привязке к {policyid}.

Ответ метода:

```
[
  "FdfdNBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8",
  "eedfdNBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8"
]
```

**GET /privacy/{policyid}/getData/{policyItemHash}**

Получение пакета конфиденциальных данных по идентификационному хешу.

Ответ метода:

```
c29tZV9iYXN1NjRfZW5jb2RlZF9zdHJpbmc=
```

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

Получение метаданных для пакета конфиденциальных данных по идентификационному хешу.

Ответ метода:

```
{
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUgEytUUz",
  "policy": "4gZnJvbSBvdGhlciBhbmltYWxzLCB3aG1jaC",
  "type": "file",
  "info": {
    "filename": "Contract №100/5.doc",
    "size": 2048,
    "timestamp": 1000000000,
    "author": "AIvanov@org.com",
    "comment": "Comment"
  },
  "hash": "e67ad392ab4d933f39d5723aeed96c18c491140e119d590103e7fd6de15623f1"
}
```

**POST /privacy/forceSync**

Запрос на принудительное получение пакета конфиденциальных данных.

Ответ метода:

```
{
  "result": "success" // or "error"
  "message": "Address '3NBVqYXrapgJP9atQccdBPAgJPwHDKkh6A8' not in policy 'policyName'"
}
```

**POST /privacy/getInfos**

Запрос на возвращение массива метаинформации о приватных данных по предоставленным идентификатору группы и хешу данных.

Пример запроса:

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

Ответ метода:



```

{
  "policiesDataInfo": [
    {
      "policyId": "somepolicyId_1",
      "datasInfo": [
        {
          "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"
          }
        }
      ]
    }
  ]
}

```

## Transactions

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

### GET /transactions/info/{id}

Запрос сведений по транзакции по ее ID.

#### Параметры запроса:

"id" - Transaction ID

#### Ответ метода:

```

{
  "type": 4,
  "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",

```

(continues on next page)

(продолжение с предыдущей страницы)

```

"sender": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
"senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMHm3Uki7pLw",
"recipient": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
"assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
"amount": 100000,
"fee": 100000,
"timestamp": 1549365736923,
"attachment": "string",
"signature":
↪ "GknccUA79dBcwWgKjQB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjtUm",
"height": 7782
}

```

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

Возвращает последние {limit} транзакций с адреса {address}.

**Ответ метода:**

```

[
  [
    {
      "type": 2,
      "id":
↪ "4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLnkFQjWp95CdddnBW",
      "fee": 100000,
      "timestamp": 1549365736923,
      "signature":
↪ "4XE4M9eSoVWVdHwDYXqZsXhEc4q8PH9mDMUBegCSBBVHJyP2Yb1ZoGi59c1Qzq2TowLmymLnkFQjWp95CdddnBW",
      "sender": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
      "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMHm3Uki7pLw",
      "recipient": "3N9iRMou3pgmyPbFZn5QZQvBTQBkL2fR6R1",
      "amount": 1000000000
    }
  ]
]

```

**GET /transactions/unconfirmed**

Возвращает все неподтвержденные транзакции из utxpool ноды.

**Ответ метода:**

```

[
  {
    "type": 4,
    "id": "52GG9U2e6foYRKp5vAzsTQ86aDAABfRJ7synz7ohBp19",
    "sender": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
    "senderPublicKey": "CRxqEuxhdZBEHX42MU4FfyJxuHmbDBTaHMHm3Uki7pLw",
    "recipient": "3NBVqYXrapgJP9atQccdBPagJPwHDKkh6A8",
    "assetId": "E9yZC4cVhCDfbjFJCc9CqkAtkoFy5KaCe64iaxHM2adG",
    "amount": 100000,
    "fee": 100000,

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "timestamp": 1549365736923,
    "attachment": "string",
    "signature":
    ↪ "GknccUA79dBcwWgKjqB7vYHcnsj7caYETfncJhRkkaetbQon7DxbpMmvK9LYqUkirJp17geBJCRTNkHEoAjsUm"
  }
]

```

**GET /transactions/unconfirmed/size**

Возвращает количество транзакций, находящихся в УТХпуле.

**GET /unconfirmed/info/{id}**

Запрос сведений по транзакции из УТХпула по ее ID.

**POST /transactions/calculateFee**

Расчитывает размер комиссии по переданной транзакции.

**Параметры запроса**

```

"type" - Transaction type
"senderPublicKey" - Public key of sender
"sender" is ignored
"fee" is ignored and all the other parameters appropriate for a transaction of the given type.

```

**Запрос метода**

```

{
  "type": 10,
  "timestamp": 1549365736923,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "alias": "ALIAS",
}

```

или

```

{
  "type": 4,
  "sender": "3MtrNP7AkTRuBhX4CBti6iT21pQpEnmHtyw",
  "recipient": "3P8JYPHrnXSfsWP1LVXySdzU1P83FE1ssDa",
  "amount": 1317209272,
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSzs3DfPuiXrURJ4AJS",
  "attachment": "string"
}

```

**Ответ метода**

```

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

```

или

```
{
  "feeAssetId": "8LQW8f7P5d5PZM7GtZEBgaqRPGSszS3DfPuiXrURJ4AJS",
  "feeAmount": 10000
}
```

## POST /transactions/sign

Подписывает транзакцию закрытым ключом отправителя, сохраненным в keystore ноды. После подписания ответ метода должен быть подан на вход метода *Broadcast*.

Для подписания запросов ключом из keystore ноды требуется обязательное указание пароля в поле password.

### Примеры запросов

ID	Тип транзакции
3	<i>Issue</i>
4	<i>Transfer</i>
5	<i>Reissue</i>
6	<i>Burn</i>
7	<i>Exchange</i>
8	<i>Lease</i>
9	<i>Lease Cancel</i>
10	<i>Alias</i>
11	<i>Mass Transfer</i>
12	<i>Data</i>
13	<i>Set Script</i>
14	<i>Sponsorship</i>
101	<i>Permission (for Genesis block)</i>
102	<i>PermissionTransaction</i>
103	<i>CreateContractTransaction</i>
104	<i>CallContractTransaction</i>
105	<i>ExecutedContractTransaction</i>
106	<i>DisableContractTransaction</i>
107	<i>UpdateContractTransaction</i>
110	<i>GenesisRegisterNode Transaction</i>
111	<i>RegisterNode Transaction</i>
112	<i>CreatePolicy Transaction</i>
113	<i>UpdatePolicy Transaction</i>
114	<i>PolicyDataHash Transaction</i>
120	<i>AtomicTransaction Transaction</i>

### 3. Issue

```
{
  "type": 3,
  "version": 2,
  "name": "Test Asset 1",
  "quantity": 100000000000,
  "description": "Some description",
  "sender": "3FSCKyfFo3566zwiJjSFLBwKvd826KXUaqR",
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "decimals": 8,
    "reissuable": true,
    "fee": 100000000
  }

```

#### 4. Transfer

```

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

```

#### 10. Alias

```

{
  "type": 10,
  "version": 2,
  "fee": 100000,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "alias": "hodler"
}

```

#### 12. Data

```

{
  "type": 12,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "senderPublicKey": "Fbt5fKHesnQG2CXmsKf4TC8v9oB7bsy2AY56CUopa6H3",
  "author": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "data":
  [
    {
      "key": "objectId",
      "type": "string",
      "value": "obj:123:1234"
    }
  ],
  "fee": 100000
}

```

#### 13. Set Script

```

{
  "type": 13,
  "version": 1,
  "sender": "3N9vL3apA4j2L5PojHW8TYmfHx9Lo2ZaKPB",
  "fee": 100000,
  "name": "faucet",
  "script": "base64:AQAAAAHJG1hdGNoMAUAAACdHgG+RXSzQ=="
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```
.. _tx-sponsorship:
```

## 14. Sponsorship

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

## 102. PermissionTransaction

### Пример запроса

```
{
  "type": 102,
  "sender": "3NA9hBG0VPfJVybremiFgWN8REi9oiDydEF",
  "password": "",
  "fee": 1000000,
  "target": "3N8YKU9W1491pbCnNbKBpTSNCJmBaH2nbiT",
  "opType": "add",
  "role": "permissioner"
}
```

## 103. CreateContractTransaction

### Пример запроса

```
{
  "fee": 100000000,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 1,
}
```

### Пример ответа

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLZZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yeCRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQq8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "image": "stateful-increment-contract:latest",
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
    "contractName": "stateful-increment-contract",
    "params": [],
    "height": 1619
  }

```

## 104. CallContractTransaction

### Пример запроса

```

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

```

### Пример ответа

```

{
  "type": 104,
  "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqlCqouVrFZynjfotEuPNV9GrdauNpgdWXLsq",
  "fee": 10,
  "timestamp": 1549365736923,
  "proofs": [
    "2q4cTBhdKEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
  ],
  "version": 2,
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "params": [
    {
      "key": "a",
      "type": "integer",
      "value": 1
    },
    {
      "key": "b",
      "type": "integer",

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    "value": 100
  }
]
}

```

## 105. ExecutedContractTransaction

### Пример ответа

```

{
  "type": 105,
  "id": "2UAHvs4KsfBbRVPM2dCigWtqUHuanQou83CXy6DGDiaRa",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrQLCqouVrFZynjfotEuPNV9GrdaunpgdWXLsq",
  "fee": 500000,
  "timestamp": 1549365523980,
  "proofs": [
    "4BoG6wQnYyZWYUKzAwh5n1184tsEWUqUTWmXMExvvCU95xgk4UFB8iCnHJ4GhvJm86REB69hKM7s2WLAwTSXquAs"
  ],
  "version": 1,
  "tx": {
    "type": 103,
    "id": "2sqPS2VAKmK77FoNakw1VtDTcBDSa7nqh5wTXvJeYGo2",
    "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
    "senderPublicKey": "2YvzcVLrQLCqouVrFZynjfotEuPNV9GrdaunpgdWXLsq",
    "fee": 500000,
    "timestamp": 1549365501462,
    "proofs": [
      "2ZK1Y1ecfQXeWsS5sfcTLM5W1KA3kwi9Up2H7z3Q6yVzMeGxT9xWJT6jREQsmuDBcvk3DCCiWBdFHaxazU8pbo41"
    ],
    "version": 1,
    "image": "localhost:5000/contract256",
    "imageHash": "930d18dacb4f49e07e2637a62115510f045da55ca16b9c7c503486828641d662",
    "params": []
  },
  "results": []
}

```

## 106. DisableContractTransaction

### Пример запроса

```

{
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "password": "",
  "contractId": "Fz3wqAWWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
  "fee": 500000,
  "type": 106
}

```

### Пример ответа

```

{
  "type": 106,
  "id": "8Nw34YbosEVhCx18pd81HqYac4C2pGjyLKck8NhSoGYH",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skQdsjMVT2M",

```

(continues on next page)



(продолжение с предыдущей страницы)

```

"fee": 500000,
"proofs": [
  ↪ "5GqPQkuRvG6LPXgPoCr9FogAdmhAaMbyFb5UfjQPUKdSc6BLuQSZ75LAWix1ok2Z6PC5ezPpjzqznr15i3RQmaEc" ],
"version": 1,
"contractId": "Fz3wq4WWcPMT4M1q6H7crLKtToFJvbeLSvqjaU4ZwMpg",
"height": 1632
}

```

## 107. UpdateContractTransaction

### Пример запроса

```

{
  "image" : "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
  "sender" : "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
  "password": "",
  "fee" : 100000000,
  "contractId" : "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRcWA",
  "imageHash" : "0e5d280b9acf6efd8000184ad008757bb967b5266e9ebf476031fad1488c86a3",
  "type" : 107,
  "version" : 1
}

```

### Пример ответа

```

{
  "senderPublicKey":
  ↪ "5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
  "tx":
  {
    "senderPublicKey":
    ↪ "5qBRDm74WKR5xK7LPs8vCy9QjzzqK4KCb8PL36fm55S3kEi2XZETHFgMgp3D13AwgE8bBkYrzvEvQZuabMfEyJwW",
    "image": "registry.wvservices.com/we-sc/tdm-increment3:1028.1",
    "sender": "3Mxxz9pBYS5fJMARJNQmzYUHxiWAtvMzSRT",
    "proofs": [
      ↪ "3tNsTyteeZrxEbVSv5zPT6dr247nXsVVR5v7Khx8spypgZQUdorCQZV2guTomutUTcyxhJUjNkQW4VmSgbCtgm1Z"],
    "fee": 0,
    "contractId": "EnsihTUHSNAB9RcWXJbiWT98X3hYtCw3SBzK8nHQRcWA",
    "id": "HdZdhXVveMT1vYzGTviCoGQU3aH6ZS3YtFpYujWeGCH6",
    "imageHash": "17d72ca20bf9393eb4f4496fa2b8aa002e851908b77af1d5db6abc9b8eae0217",
    "type": 107, "version": 1, "timestamp": 1572355661572},
    "sender": "3HfRBedCpWi3vEzFSKEZDFXkyNWbWLWQmmG",
    "proofs": [
      ↪ "28ADV8miUVN5EFjhqefj6MADsXYjbxA3TsxSsWfVs18jXAsHVaBczvnyoUSaYJsJRnmaWgXbpbduccRxpKGTs6tro"],
    "fee": 0, "id": "7niVY8mjzeKqLBePvhTxFRfLu7BmcwVfqaqtBWAN8AA2",
    "type": 105,
    "version": 1,
    "results": [],
    "timestamp": 1572355666866
  }
}

```

## 110. GenesisRegisterNode

### Пример запроса

```
{
  "type": 110,
  "id": "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkYmTmJMgYLpvwynbxHPTFPFEfFdyLpJ",
  "fee": 0,
  "timestamp": 1489352400000,
  "signature":
  ↪ "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkYmTmJMgYLpvwynbxHPTFPFEfFdyLpJ",
  "targetPublicKey": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMDrAEJpj",
  "target": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMDrAEJpj"
}
```

**Пример ответа**

```
{
  "signature":
  ↪ "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkYmTmJMgYLpvwynbxHPTFPFEfFdyLpJ",
  "fee": 0,
  "id": "2Xgbsqgfbp5fiq4nsaAoTkQsXc399tXdnKom8prEZqPW2Q7xZKNKCCqpkYmTmJMgYLpvwynbxHPTFPFEfFdyLpJ",
  "type": 110,
  "targetPublicKey": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMDrAEJpj",
  "timestamp": 1489352400000,
  "target": "3JNLQYuHYSHZiHr5KjJ89wwFJpDMDrAEJpj",
  "height": 1
}
```

**111. RegisterNode****Пример запроса**

```
{
  "type": 111,
  "opType": "add",
  "sender": "3HYW75PpAeVukmbYo9PQ3mzSHdKUGeytUUz",
  "password": "",
  "targetPubKey": "apgJP9atQccdBPAGJPwH3NBVqYXrapgJP9atQccdBPAGJPwHapgJP9atQccdBPAGJPwHDKkh6A8",
  "nodeName": "Node #1",
  "fee": 500000,
}
```

**112. CreatePolicy****Пример запроса**

```
{
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "policyName": "Policy# 7777",
  "password": "sfgKYBFcF0#$fsdf()*%",
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAoHuoLsAQvxBSqJE91WK3LwWGjiiCxx"
  ],
  "fee": 15000000,
  "description": "Buy bitcoin by 1c",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  ]
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

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

```

### 113. UpdatePolicy

#### Пример запроса

```

{
  "policyId": "7wphGbhqbmUgzun5wzggwqtViTiMdFezSa11fxRV58Lm",
  "password": "sfgKYBFCF@#$fsdf()*%",
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "proofs": [],
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAoHuoLsAQvxBSqjE91WK3LwWGjiiCxx",
    "3NwJfjG5RpaDfxEhkwXgWD7oX21NMFCxJHL"
  ],
  "fee": 15000000,
  "opType": "add",
  "owners": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
  ],
  "type": 113,
}

```

### 114. PolicyDataHash

Когда пользователь отправляет конфиденциальные данные в сеть при помощи *POST /privacy/sendData*, нода автоматически формирует транзакцию 114. **120. AtomicTransaction**

#### Пример запроса

```

{ 'sender': '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP',
  'transactions': [
    { 'senderPublicKey':
      ↪ '5nGi8XoiGjjyjbPmjLNy1k2bus4yXLaeuA3Hb7BikwD9tboFWFXJYUmt05Joxx76c3pp2Mr1LjgodUJuxryCJofQ',
      ↪ 'amount': 10, 'fee': 10000000, 'type': 4, 'version': 3, 'atomicBadge': { 'trustedSender':
      ↪ '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP' }, 'attachment': '', 'sender':
      ↪ '3Mv79dyPX2cvLtRXn1MDDWiCZMBrkW9d97c', 'feeAssetId': None, 'proofs': [
      ↪ 'XQ7iAqkarmm14AATc2Y9cR3Z9WnirSH4kH6RUL4QdT82rEwsmWBbBfWrADLE9o4cp2VR39W6b3vdrwFgg1dX7m3'],
      ↪ 'assetId': None, 'recipient': '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'id':
      ↪ 'FZ59wAZnkFUqXjn61vvvj59fRa3cuS6nzuW3vqoRMsM5', 'timestamp': 1602857131666}, { 'senderPublicKey
      ↪ ': '56rV5kcR9SBsxQ9LtNrmp6V72S4BDkZUJaA6ujZswDneDmCtmeSG6UE2FQP1rPXdfpWQNunRw4aijGXxoK3o4puj',
      ↪ 'amount': 20, 'fee': 10000000, 'type': 4, 'version': 3, 'atomicBadge': { 'trustedSender':
      ↪ '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP' }, 'attachment': '', 'sender':
      ↪ '3MufokZsFzaf7heTV1yreUtm1uoJXPoFzdP', 'feeAssetId': None, 'proofs': [
      ↪ '5KaXUFan2JD6VsJeGNyBCxEwqCjUF1nASAzxnPZzBydXA5RjyXQGaL6N9MQ8GDNori1nXw5FsDLBqc3CPM3ezsk'},
      ↪ 'assetId': None, 'recipient': '3Mv79dyPX2cvLtRXn1MDDWiCZMBrkW9d97c', 'id':
      ↪ '8GTqE1cc6zTVxYgQxgHJWJitVsDFRc6GmU5FJcnp5gu2', 'timestamp': 1602857132314}
  ]
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```
],
'type': 120,
'version': 1}
```

**POST /transactions/broadcast**

Отправляет подписанную транзакцию в блокчейн.

**Запрос метода**

```
{
  "type": 10,
  "senderPublicKey": "G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
  "fee": 100000000,
  "timestamp": 1550591678479,
  "signature":
  ↪ "4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSsqxKWfEPJG",
  "alias": "dajzmj6gfuzmbfnhamsbuxivc"
}
```

**Ответ метода**

```
{
  "type": 10,
  "id": "9q7X84wFuVvKqRdDQeWbtBmpsHt9SXFbvPPtUuKBVxxr",
  "sender": "3MtrNP7AkTRuBhX4CBti6iT2lpQpEnmHtyw",
  "senderPublicKey": "G6h72icCSjdW2A89QWDb37hyXJoYKq3XuCUJY2joS3EU",
  "fee": 100000000,
  "timestamp": 1550591678479,
  "signature":
  ↪ "4gQyPXzJFEzMbsCd9u5n3B2WauEc4172ssyrXCL882oNa8NfNihnpKianHXrHWnZs1RzDLbQ9rcRYnSsqxKWfEPJG",
  "alias": "dajzmj6gfuzmbfnhamsbuxivc"
}
```

**POST /transactions/signAndBroadcast**

Подписывает и отправляет подписанную транзакцию в блокчейн.

**Запрос метода**

```
{
  "sender": "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "policyName": "Policy# 7777",
  "password": "sfgKYBFCF@#$fsdf()*%",
  "recipients": [
    "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
    "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
    "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T",
    "3NtNJV44wyxRXv2jyW3yXLxjJxvY1vR88TF",
    "3NxAoohUoLsAQvxBSqjE91WK3LwWGCjiiCxx"
  ],
  "fee": 15000000,
  "description": "Buy bitcoin by 1c",
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

"owners": [
  "3NkZd8Xd4KsuPiNVsuphRNCZE3SqJycqv8d",
  "3NotQaBygbSvYZW4ftJ2ZwLXex4rTHY1Qzn",
  "3Nm84ERiJqKfuqSYxzMAhaJXdj2ugA7Ve7T"
],
"type": 112
}

```

**Ответ метода**

```

{
  "senderPublicKey": "3X6Qb6p96dY4drVt3x4XyHKCRvree4QDqNZyDWHzjJ79",
  "policyName": "Policy for sponsored v1",
  "fee": 100000000,
  "description": "Privacy for sponsored",
  "owners": [
    "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwd9t"
  ],
  "type": 112,
  "version": 2,
  "sender": "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
  "feeAssetId": "G16FvJk9vabwxjQswh9CQAhbZzn3QrwqWjwnZB3qNVox",
  "proofs": [
    "3vDVjp6UJeN9ahtNcQWt5WDVqC9KqEsrr9HTToHfoXFd1HtVwnUPPtJKM8tAsCtby81XYQReLj33hLEZ8qbGA3V"
  ],
  "recipients": [
    "3JSaKNX94deXJkywQwTFgbigTxJa36TDVg3",
    "3JWDUsqyJEkVa1aivNPP8VCAa5zGuxiwd9t"
  ],
  "id": "EeymzQcM2LrsgGDFfxGn8DhahJbFYmorcBrEh8phv5S",
  "timestamp": 1585307711344
}

```

**Utils**


---

**Подсказка:** Правила формирования запросов к ноде приведены в разделе *Как использовать REST API*.

---

**POST /utils/hash/secure**

Возвращает secure (двойной) hash от заданного сообщения.

**Запрос метода:**

```
ridethewaves!
```

**Ответ метода:**

```

{
  "message": "ridethewaves!",

```

(continues on next page)

(продолжение с предыдущей страницы)

```
{
  "hash": "H6nsiifwYKYEx6YzYD7woP1XCn72RVvx6tC1zjjLXqsu"
}
```

**POST /utils/hash/fast**

Возвращает hash от заданного сообщения.

**Запрос метода:**

```
ridethewaves!
```

**Ответ метода:**

```
{
  "message": "ridethewaves!",
  "hash": "D J35yms chUFDmqCnDJewjcnVExVkWgX7mJDXhFy9X8oQ"
}
```

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

```
"script" - Base64 script
"complexity" - script complexity
"extraFee" - the fee for outgoing transactions set by the script
```

**Запрос метода:**

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

**Ответ метода:**

```
{
  "script":
  ↳ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34So cMRkRKFGzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
  ↳ ",
  "complexity": 11,
  "extraFee": 10001
}
```

или

**Запрос метода:**

```
x == 1
```

**Ответ метода:**

```
{
  "error": "Typecheck failed: A definition of 'x' is not found"
}
```

**POST /utils/script/estimate**

Декодирование base64 скрипта.

**Запрос метода:**

```
AQQAAAABeAAAAAAAAAAAAAQkAAAAAAAAACCQAAZAAAAAIFAAAAAXgAAAAAAAAAAAAEAAAAAAAAAAAAJdecYi
```

**Ответ метода:**

```
{
  "script":
  ↪ "3rbFDtbPwAvSp2vBvqGfGR9nRS1nBVnfuSCN3HxSZ7fVRpt3tuFG5JSmyTmvHPxYf34SocMRkRKFGzTtXXnnv7upRHXJzZrLSQo8tUW6yMtEiZ
  ↪ ",
  "scriptText": "FUNCTION_CALL(FunctionHeader(==,List(LONG, LONG)),List(CONST_LONG(1), CONST_
  ↪ LONG(2)),BOOLEAN)",
  "complexity": 11,
  "extraFee": 10001
}
```

**GET /utils/time**

Возвращает текущее время на нодe.

**Ответ метода:**

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


**POST /utils/reloadwallet**

Перезагружает keystore ноды. Выполняется, если новая ключевая пара была создана в keystore без перезапуска ноды.

**Ответ метода:**

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

**21.2.2 Методы REST API сервиса авторизации**

Подробно о работе с REST API можно почитать в этом разделе. Доступ к REST API сервиса авторизации осуществляется по протоколу HTTPS. Методы, закрытые авторизацией, отмечены значком .

## Способы авторизации

В зависимости от используемого метода авторизации указываются разные значения для получения доступа к REST API ноды.

### Available authorizations



#### OAuth2 Bearer (apiKey)

Name: Authorization

In: header

Value:

Authorize

Close

#### ApiKey or PrivacyApiKey (apiKey)

Name: X-API-Key

In: header

Value:

Authorize

Close

- OAuth2 Bearer (apiKey) значение **access** токена.
- ApiKey or PrivacyApiKey (apiKey) значение `apikeyhash` как для общего доступа к REST API ноды, так и для доступа к методам *privacy*.

## Авторизация по `apikeyhash`

Генерация значения `apikeyhash` выполняется при *конфигурации ноды*. Также получить значение поля `restarti.apikeyhash` можно при помощи метода `/utils/hash/secure` REST API ноды. Для подписания запросов ключем из keystore ноды в поле `password` запроса `POST /transaction/sign` требуется указания пароля доступа к keystore.

Пример запроса:

```
curl -X POST
--header 'Content-Type: application/json'
--header 'Accept: application/json'
--header 'X-API-Key: 1' -d '1' 'http://2.testnet-pos.com:6862/transactions/calculateFee'
```



## Авторизация по токenu

Если используется *сервис авторизации*, для доступа к нoде и другим сервисам клиент получает пару токенов **refresh** и **access**. Токены можно получить через REST API сервиса авторизации.

Для регистрации пользователя используется метод *POST /v1/user*. На вход передаются следующие параметры:

- **login** логин пользователя (электронный адрес пользователя). В качестве логина используется электронный адрес 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. Возможные варианты *license* и *voting*.

Только после регистрации пользователь получает токены авторизации.

Для получения и обновления токенов авторизации используются следующие методы:

1. *POST /v1/auth/login* получение токена авторизации с использованием логина и пароля. Этот метод предназначен для авторизации пользователей.
2. *POST /v1/auth/token* получение **refresh** и **access** токенов авторизации для сервисов и приложений. Метод не требует параметров и в ответ на вызов присылает значения токенов. Метод может быть использовать только администратором сервиса авторизации.
3. *POST /v1/auth/refresh* обновление **refresh** токена. На вход передаётся значение токена.

## Методы сервиса авторизации

### GET /status

Получение статуса сервиса авторизации.

#### Ответ метода

```
{
  "status": "string",
  "version": "string",
  "commit": "string"
}
```

### POST /v1/user

Регистрация нового пользователя.

#### Запрос метода

```
{
  "username": "string",
  "password": "string",
  "locale": "string",
  "source": "string"
}
```

Если регистрация прошла успешно, в качестве ответа приходит код 201. В ином случае регистрация не состоялась.

### GET /v1/user/profile

Получение данных пользователя.

#### Ответ метода

```
{
  "id": "string",
  "name": "string",
  "locale": "en",
  "addresses": [
    "string"
  ],
  "roles": [
    "string"
  ]
}
```

### POST /v1/user/address

Получение адреса пользователя.

#### Запрос метода

```
{
  "address": "string",
  "type": "string"
}
```

#### Ответ метода

```
{
  "addressId": "string"
}
```

### GET /v1/user/address/exists

Проверка адреса электронной почты пользователя. В качестве параметра на вход метод принимает электронный адрес пользователя.

#### Ответ метода

```
{
  "exist": true
}
```

**POST /v1/user/password/restore**

Восстановление пароля доступа к аккаунту пользователя.

**Запрос метода**

```
{
  "email": "string",
  "source": "string"
}
```

**Ответ метода**

```
{
  "email": "string"
}
```

**POST /v1/user/password/reset**

Сброс пароля пользователя.

**Запрос метода**

```
{
  "token": "string",
  "password": "string"
}
```

**Ответ метода**

```
{
  "userId": "string"
}
```

**GET /v1/user/confirm/{code}**

Ввод кода подтверждения для восстановления пароля для доступа к аккаунту пользователя. На вход методу передаётся значение кода подтверждения.

**POST /v1/user/resendEmail**

Повторная отправка кода восстановления пароля на указанный электронный адрес.

**Запрос метода**

```
{
  "email": "string",
  "source": "string"
}
```

**Ответ метода**

```
{
  "email": "string"
}
```

## POST /v1/auth/login

Регистрация нового пользователя в сервисе авторизации.

### Запрос метода

```
{
  "username": "string",
  "password": "string",
  "locale": "string",
  "source": "string"
}
```

### Ответ метода

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

## POST /v1/auth/token

Регистрация внешних сервисов и приложений в сервисе авторизации. Метод не требует параметров запроса.

### Ответ метода

```
{
  "access_token": "string",
  "refresh_token": "string",
  "token_type": "string"
}
```

## POST /v1/auth/refresh

Получение нового refresh токена.

### Запрос метода

```
{
  "token": "string"
}
```

### Ответ метода

```
{
  "access_token": "string",
  "refresh_token": "string",
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"token_type": "string"
}
```

## GET /v1/auth/publicKey

Получение публичного ключа сервиса авторизации.

### Ответ метода

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEA7d90j/ZQTkkjf4UuMfUu
QIFDTYxYf6QBKMVJnq/wXyPYyV8HVfYFizCaEciv3CXmBH77sXnuTlrEtK7zHB
KvV870HmZuazjIgZVSk0n0Y7F8UUVNXnlzVD1dPs0GJ6orM41DnC1W65mCrP3bjn
fV4RbmykN/lk7McA6EsMcLEGbKkFhmeq2Nk4hn2CQvoTkupJUn0CP1dh04bq1lQ7
Ffj9K/FJq73wSXDoH+qqdRG9sfrtgrhtJHerruhv3456e0zyAcD08+sJUQFKY80B
SZMEndVzFS2ub9Q8e7BfcNXTmQPM4PhH05wuTqL32qt3uJBx20I4lu30ND44ZrDJ
BbVog73oPjRYXj+kTbwUZI66SP4aLcQ8sypQyLwqKk5DtLRozSN00IrupJJ/pwZs
9zPEggL91T0rirbEhG1f5U8/6XN8GVXX4iMk2fD8FHLfJuXCD70j4JC2iWfFDC6a
uUkwUfqfjJB8BzIHkncoq0ZbpideE2lTW1+svuEu/wyP5rNlyMiE/e/fZQqM2+o0
ch5Qow6HH35BrloCSZciutUcd1U7YPqESJ5tryy1xn9bsMb+On1ocZTtvec/ow4M
RmnJwm0j1nd+cc190KLG5/boeA+2zqWu0jCbWR9c0oCmgbhucZCHaHTBEAKDWcsC
VRz5qD6FPpePpTQDb6ss3bkCAwEAAQ==
-----END PUBLIC KEY-----
```

## 21.2.3 Методы REST API сервиса подготовки данных

### Транзакции

Набор методов, позволяющий выводить список транзакций по заданным условиям и фильтрам.

## GET /transactions

Возвращает список транзакций, соответствующий условиям поискового запроса и применённым фильтрам.

**Важно:** За один запрос через метод API **GET /transactions** возвращается не более 500 транзакций.

### Ответ метода:

```
[
{
  "id": "string",
  "type": 0,
  "height": 0,
  "fee": 0,
  "sender": "string",
  "senderPublicKey": "string",
  "signature": "string",
  "timestamp": 0,
  "version": 0
}
```

**GET /transactions/count**

Возвращает количество транзакций, соответствующих условиям поискового запроса и применённым фильтрам.

**Ответ метода:**

```
{
  "count": "string"
}
```

**GET /transactions/{id}**

Возвращает транзакцию по идентификатору {id}.

**Ответ метода:**

```
{
  "id": "string",
  "type": 0,
  "height": 0,
  "fee": 0,
  "sender": "string",
  "senderPublicKey": "string",
  "signature": "string",
  "timestamp": 0,
  "version": 0
}
```

**Наборы токенов**

Набор методов, позволяющий вывести информацию о доступных наборах токенов в блокчейне.

**GET /assets**

Возвращает список доступных в блокчейне наборов токенов (в виде транзакций выпуска токенов).

**Ответ метода:**

```
[
  {
    "index": 0,
    "id": "string",
    "name": "string",
    "description": "string",
    "reissuable": true,
    "quantity": 0,
    "decimals": 0
  }
]
```

**POST /assets/count**

Возвращает количество доступных в блокчейне наборов токенов.

**Ответ метода**

```
{
  "count": 0
}
```

**GET /assets/{id}**

Возвращает информацию о доступном наборе токенов по его {id}.

**Ответ метода**

```
{
  "index": 0,
  "id": "string",
  "name": "string",
  "description": "string",
  "reissuable": true,
  "quantity": 0,
  "decimals": 0
}
```

**Пользователи**

Набор методов, позволяющий вывести информацию об участниках блокчейна в соответствии с заданными условиями и фильтрами.

**GET /users**

Возвращает список пользователей, соответствующий условиям поискового запроса и применённым фильтрам.

**Ответ метода:**

```
[
  {
    "address": "string",
    "aliases": [
      "string"
    ],
    "registration_date": "string",
    "permissions": [
      "string"
    ]
  }
]
```

**GET /users/count**

Возвращает количество пользователей, удовлетворяющих установленным в запросе фильтрам.

**Ответ метода:**

```
{
  "count": 0
}
```

**GET /users/{userAddressOrAlias}**

Возвращает информацию о пользователе по его адресу или алиасу.

**Ответ метода:**

```
{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ]
}
```

**GET /users/contractid/{contractId}**

Возвращает список пользователей, вызывавших когдалибо смартконтракт с указанным {contractId}.

**Ответ метода:**

```
{
  "address": "string",
  "aliases": [
    "string"
  ],
  "registration_date": "string",
  "permissions": [
    "string"
  ]
}
```



**POST /users/byaddresses**

Возвращает список пользователей для заданного набора адресов.

**Ответ метода:**

```
[
  {
    "index": 0,
    "id": "string",
    "name": "string",
    "description": "string",
    "reissuable": true,
    "quantity": 0,
    "decimals": 0
  }
]
```

**Блоки****GET /blocks/at/{height}**

Возвращает блок на указанной высоте.

**Ответ метода:**

```
{
  "version": 0,
  "timestamp": 0,
  "reference": "string",
  "features": [
    0
  ],
  "generator": "string",
  "signature": "string",
  "blocksize": 0,
  "transactionsCount": 0,
  "fee": 0,
  "height": 0,
  "transactions": [
    {
      "id": "string",
      "type": 0,
      "height": 0,
      "fee": 0,
      "sender": "string",
      "senderPublicKey": "string",
      "signature": "string",
      "timestamp": 0,
      "version": 0
    }
  ]
}
```

## Смартконтракты

Набор методов, позволяющий вывести информацию о смартконтрактах Docker, загруженных в блокчейн.

### GET /contracts

Возвращает список смартконтрактов в блокчейне по заданным условиям и фильтрам.

**Ответ метода:**

```
[
  {
    "id": "string",
    "type": 0,
    "height": 0,
    "fee": 0,
    "sender": "string",
    "senderPublicKey": "string",
    "signature": "string",
    "timestamp": 0,
    "version": 0
  }
]
```

### GET /contracts/count

Возвращает количество смартконтрактов в блокчейне, соответствующих заданным условиям и фильтрам.

**Ответ метода:**

```
{
  "count": 0
}
```

### GET /contracts/id/{id}

Возвращает информацию о смартконтракте по его {id}.

**Примечание:** Для этого метода предусмотрен вывод стеята смартконтракта по эндпоинту /state. Для корректного вывода необходимо обработать параметры lastIndex и limit, чтобы ограничить количество выводимых записей. Пример: curl X GET "https://<youraddress>/dataServiceAddress/contracts/id/{id}/state?lastIndex=0&limit=50&q="

**Ответ метода:**

```
{
  "id": "string",
  "type": 0,
  "height": 0,
  "fee": 0,
  "sender": "string",
  "senderPublicKey": "string",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
"signature": "string",  
"timestamp": 0,  
"version": 0  
}
```

### GET /contracts/id/{id}/versions

Возвращает историю версий смартконтракта с заданным {id}.

**Ответ метода:**

```
[  
  {  
    "version": 0,  
    "image": "string",  
    "imageHash": "string",  
    "timestamp": "string"  
  }  
]
```

### GET /contacts/history/{id}/key/{key}

Возвращает историю изменений ключа смартконтракта по его {id} и {key}.

**Ответ метода:**

```
{  
  "total": 777,  
  "data": [  
    {  
      "key": "some_key",  
      "type": "integer",  
      "value": "777",  
      "timestamp": 1559347200000,  
      "height": 14024  
    }  
  ]  
}
```

### GET /contracts/senderscount

Возвращает количество уникальных участников, отправляющих транзакции 104 на вызов смартконтрактов.

**Ответ метода:**

```
{  
  "count": 777  
}
```

**GET /contracts/calls**

Возвращает список транзакций 104 на вызов смартконтрактов с их параметрами и результатами.

**Ответ метода:**

```
[
  {
    "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"
      }
    ]
  }
]
```

## Группы доступа

Набор методов, позволяющий вывести информацию о группах доступа и нодах в блокчейне.

### GET /privacy/groups

Возвращает список групп доступа в блокчейне.

**Ответ метода:**

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

### GET /privacy/groups/count

Возвращает количество групп доступа в блокчейне.

**Ответ метода:**

```
{
  "count": 0
}
```

### GET /privacy/groups/{address}

Возвращает список групп доступа, в которые входит заданный {address}.

**Ответ метода:**

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

**GET /privacy/groups/byrecipient/{address}**

Возвращает список групп доступа, в которых заданный {address} фигурирует как получатель данных.

**Ответ метода:**

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

**GET /privacy/groups/{address}/count**

Возвращает количество групп доступа, в которые входит заданный {address}.

**Ответ метода:**

```
{
  "count": 0
}
```

**GET /privacy/groups/id/{id}**

Возвращает информацию о группе доступа по ее {id}.

**Ответ метода:**

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

**GET /privacy/groups/id/{id}/history**

Возвращает историю изменений группы доступа по ее {id} (в виде списка отправленных транзакций 112114 с их описанием).

**Ответ метода:**

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

**GET /privacy/groups/id/{id}/history/count**

Возвращает количество отправленных транзакций 112114 для внесения изменений в группу доступа с указанным {id}.

**Ответ метода:**

```
{
  "id": "string",
  "name": 0,
  "description": "string",
  "createdAt": "string"
}
```

**GET /privacy/nodes**

Возвращает список доступных нод в блокчейне.

**Ответ метода:**

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

**GET /privacy/nodes/count**

Возвращает количество доступных нод в блокчейне.

**Ответ метода:**

```
{
  "count": 0
}
```

**GET /privacy/nodes/publicKey/{targetPublicKey}**

Возвращает информацию о ноде по ее публичному ключу {targetPublicKey}.

**Ответ метода:**

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

**GET /privacy/nodes/address/{address}**

Возвращает информацию о ноде по ее адресу {address}.

**Ответ метода:**

```
[
  {
    "id": "string",
    "name": 0,
    "description": "string",
    "createdAt": "string"
  }
]
```

**Транзакции с данными****GET /api/v1/txIds/{key}**

Возвращает список идентификаторов транзакций с данными, содержащих указанный ключ {key}.

**Ответ метода:**

```
[
  {
    "id": "string"
  }
]
```

**GET /api/v1/txIds/{key}/{value}**

Возвращает список идентификаторов транзакций с данными, содержащих указанный ключ {key} и значение {value}.

**Ответ метода:**

```
[
  {
    "id": "string"
  }
]
```

**GET /api/v1/txData/{key}**

Возвращает тела транзакций с данными, содержащие указанный ключ {key}.

**Ответ метода:**

```
[
  {
    "id": "string",
    "type": "string",
    "height": 0,

```

(continues on next page)



(продолжение с предыдущей страницы)

```

    "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}**

Возвращает тела транзакций с данными, содержащие указанный ключ {key} и значение {value}.

**Ответ метода:**

```

[
  {
    "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
  }
]

```

**Функции лизинга****GET /leasing/calc**

Возвращает сумму выплат за лизинг токенов в указанном интервале высот блоков.

**Ответ метода:**

```

{
  "payouts": [
    {
      "leaser": "3P1EiJnPxFxGyhN9sucXfB2rhQ1ws4cmuS5",
      "payout": 234689
    }
  ],
  "totalSum": 4400000,
  "totalBlocks": 1600
}

```

## Вспомогательные функции сервиса подготовки данных

### GET /info

Возвращает информацию о сервисе подготовки данных.

**Ответ метода:**

```
{
  "version": "string",
  "buildId": "string",
  "gitCommit": "string"
}
```

### GET /status

Возвращает информацию о состоянии сервиса подготовки данных.

**Ответ метода:**

```
{
  "status": "string"
}
```

## Функции статистики и мониторинга

### GET /stats/transactions

Возвращает информацию о проведенных транзакциях за указанный временной промежуток.

**Ответ метода:**

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "transactions": [
        {
          "type": 104,
          "count": 100
        }
      ]
    }
  ]
}
```

### GET /stats/contracts

Возвращает информацию о транзакциях вызова смартконтрактов за указанный временной промежуток.

**Ответ метода:**

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "transactions": [
        {
          "type": 104,
          "count": 100
        }
      ]
    }
  ]
}
```

### GET /stats/tokens

Возвращает информацию об обороте токенов в блокчейне за указанный временной промежуток.

**Ответ метода:**

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "sum": "12000.001"
    }
  ]
}
```

### GET /stats/addressesactive

Возвращает адреса, которые были активными в указанный временной промежуток.

**Ответ метода:**

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "senders": "12",
      "recipients": "12"
    }
  ]
}
```

### GET /stats/addressestop

Возвращает адреса, которые были наиболее активными отправителями или получателями в указанный временной промежуток.

**Ответ метода:**

```
{
  "aggregation": "day",
  "data": [
    {
      "date": "2020-03-01T00:00:00.000Z",
      "senders": "12",
      "recipients": "12"
    }
  ]
}
```

### GET /stats/nodestop

Возвращает адреса нод, которые создали наибольшее количество блоков в указанный временной промежуток.

**Ответ метода:**

```
{
  "limit": "10",
  "data": [
    {
      "generator": "3NdPsjaFC7NeioGVF6X4J5A8FVaxdtKvAba",
      "count": "120",
      "node_name": "Genesis Node #5"
    }
  ]
}
```

### GET /stats/contractcalls

Возвращает список смартконтрактов, вызванных наибольшее количество раз в указанный временной промежуток.

**Ответ метода:**

```
{
  "limit": "5",
  "data": [
    {
      "contract_id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "count": "120",
      "contract_name": "oracle_contract",
      "last_call": "60.321"
    }
  ]
}
```

### GET /stats/contractlastcalls

Возвращает список последних вызовов смартконтрактов по их id и названию.

**Ответ метода:**

```
{
  "limit": "5",
  "data": [
    {
      "contract_id": "Cm9MDf7vpETuzUCsr1n2MVHsEGk4rz3aJp1Ua2UbWBq1",
      "contract_name": "oracle_contract",
      "last_call": "60.321"
    }
  ]
}
```

### GET /stats/contracttypes

Возвращает список смартконтрактов блокчейна по именам образов и их хэшам.

**Ответ метода:**

```
{
  "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

Возвращает информацию о сети.

**Ответ метода:**

```
{
  "tps": "5",
  "blockAvgSize": "341.391",
  "senders": "50",
  "nodes": "50",
  "blocks": "500000"
}
```

## Функции анкоринга

### GET /anchoring/rounds

Возвращает список транзакций в соответствии с заданными условиями и фильтрами.

**Ответ метода:**

```
[
  {
    "height": 0,
    "sideChainTxIds": [
      "string"
    ],
    "mainNetTxIds": [
      "string"
    ],
    "status": "string",
    "errorCode": 0
  }
]
```

### GET /anchoring/round/at/{height}

Возвращает информацию о раунде анкоринга на указанной высоте блоков {height}.

**Ответ метода:**

```
{
  "height": 0,
  "sideChainTxIds": [
    "string"
  ],
  "mainNetTxIds": [
    "string"
  ],
  "status": "string",
  "errorCode": 0
}
```

### GET /anchoring/info

Возвращает информацию об анкоринге в блокчейне.

**Ответ метода:**

```
{
  "height": 0,
  "sideChainTxIds": [
    "string"
  ],
  "mainNetTxIds": [
    "string"
  ],
  "status": "string",

```

(continues on next page)

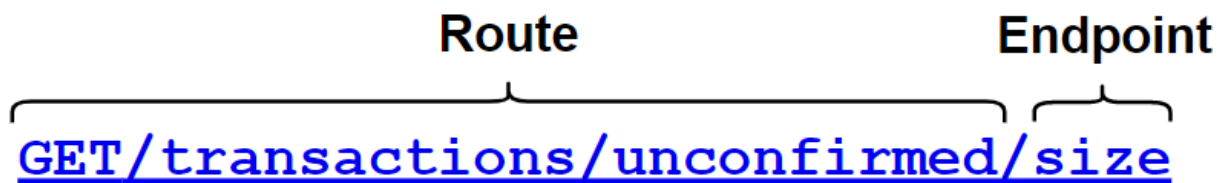
(продолжение с предыдущей страницы)

```
"errorCode": 0  
}
```

#### 21.2.4 Как использовать REST API

Все вызовы методов API — это GET, POST или DELETE https-запросы к URL `https://yournetwork.com/nodeN/apidocs/swagger.json` с набором параметров. В интерфейсе Swagger выбираются нужные группы запросов и далее маршруты с точками доступа. Маршрут в Swagger это URL к http-методу, а точка доступа (endpoint) — конечная часть маршрута, само обращение к методу. Пример:

URL к HTTP-методу



Для запросов, требующих нижеперечисленных действий, необходима обязательная авторизация по `apikeyhash`. Тип авторизации устанавливается в конфигурационном файле ноды. Если выбран тип авторизации по `apikeyhash`, то при авторизации необходимо указывать значение секретной фразы, `hash` которой указан в конфигурационном файле ноды (поле `restapi.apikeyhash`).

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

При авторизации по токenu в соответствующем поле указывается значение **access** токена. Если выбрана авторизация по токenu, в таком случае закрыты все методы REST API для доступа к нодe.

## 22.1 Подготовка к работе

Для начала работы с контейнеризованными смартконтрактами необходимо настроить возможность их исполнения.

Исполнение смартконтрактов настраивается в конфигурационном файле ноды. Также существует возможность настроить используемый образ для отдельного смартконтракта при помощи транзакции *103 CreateContractTransaction*.

### 22.1.1 Настройка исполнения Dockerконтрактов в конфигурационном файле ноды

Этот метод позволяет гибко настроить исполнение всех Dockerконтрактов для ноды. Для этого в файле `node.conf` предусмотрен раздел `dockerengine`, содержащий следующие параметры конфигурации:

- `enable` – включение обработки транзакций для Dockerконтрактов.
- `integrationtestsmodeenable` – режим тестирования Dockerконтрактов. При включении этой опции смартконтракты исполняются локально в контейнере.
- `dockerhost` – адрес демона `docker` (опционально).
- `noderestartapi` – путь до REST API ноды (опционально).
- `startuptimeout` – время, отводимое на создание контейнера gRPCконтракта и его регистрацию в ноде (в секундах).
- `timeout` – время, отводимое на выполнение контракта (в секундах).
- `memory` – ограничение по памяти для контейнера контракта (в мегабайтах).
- `memoryswap` – выделяемый объем виртуальной памяти для контейнера контракта (в мегабайтах).
- `reusecontainers` – использование одного контейнера для нескольких контрактов, использующих один и тот же Dockerобраз.



- `removecontainerafter` – промежуток времени бездействия контейнера, по прошествии которого он будет удален.
- `allownetaccess` – разрешение доступа к сети.
- `remoteregistries` – адреса Dockerрепозитория и настройки авторизации к ним.
- `checkregistryauthonstartup` – проверка авторизации для Dockerрепозитория при запуске ноды.
- `defaultregistrydomain` – адрес Dockerрепозитория по умолчанию (опционально). Этот параметр используется, если в имени образа контракта не указан репозиторий.
- `contractexecutionmessagescache` – настройки кэша со статусами исполнения транзакций по docker контрактам;
- `expireafter` – время хранения статуса смартконтракта.
- `maxbufferize` и `maxbuffertime` – настройки объема и времени хранения кэша статусов.
- `contractauthexpiresin` – время жизни токена авторизации, используемого смартконтрактами для вызовов к ноды.
- `grpcserver` – секция настроек gRPC сервера для работы Dockerконтрактов с gRPC API.
- `host` – сетевой адрес ноды (опционально).
- `port` – порт gRPCсервера.
- `akkahttpsettings` – секция настроек фреймворка Akka HTTP, используемого для gRPCсервера.
- `removecontaineronfail` – удаление контейнера, если при его старте произошла ошибка.

Блок параметров `remoteregistries` может включать адреса нескольких репозитория. При использовании этого параметра для доступа к каждому репозиторию необходимо указать его адрес, а также используемое имя пользователя и пароль. Для обращения к конкретному репозиторию из указанного списка адрес репозитория указывается в имени Dockerобраза.

Параметр `defaultregistrydomain`, используется в случаях, когда в имени Dockerобраза не указывается адрес репозитория. Если этот параметр используется, и при этом в имени Dockerобраза указан адрес репозитория, смартконтракт обращается к репозиторию, указанному в имени образа.

**Подсказка:** Если путь до репозитория указывается при помощи транзакции `103 CreateContractTransaction`, этот путь имеет приоритет перед параметрами конфигурации ноды `remoteregistries` и `defaultregistrydomain`.

### Пример секции `dockerengine` конфигурационного файла ноды

В приведенном примере рассмотрен вариант настройки исполнения Dockerконтрактов с указанием репозитория и его настроек авторизации (заполнен блок `remoteregistries`, параметр `defaultregistrydomain` закомментирован). Также используется стандартный демон Docker и RESTAPI ноды (закомментированы параметры `dockerhost` и `noderestapi`). Включено удаление контейнера в случае ошибки при его старте (параметр `removecontaineronfail`) для поиска ошибок при работе со смартконтрактами.

```
docker-engine {
  enable = yes
  integration-tests-mode-enable = no
  # docker-host = "unix:///var/run/docker.sock"
  # node-rest-api = "https://restapi.clientservice.com/"
  execution-limits {
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    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
  # default-registry-domain = "registry.wavesenterprise.com"
  contract-execution-messages-cache {
    expire-after = 60m
    max-buffer-size = 10
    max-buffer-time = 100ms
  }
  contract-auth-expires-in = 1m
  grpc-server {
    # host = "192.168.97.3"
    port = 6865
    akka-http-settings {
      akka {
        http.server.idle-timeout = infinite
        http.client.idle-timeout = infinite
        http.host-connection-pool.idle-timeout = infinite
        http.host-connection-pool.client.idle-timeout = infinite
      }
    }
  }
  remove-container-on-fail = yes
}

```

### 22.1.2 Настройка образа для отдельного Dockerконтракта при помощи транзакции 103 CreateContractTransaction

Транзакция *103 CreateContractTransaction* применяется для создания Dockerконтракта. При создании этой транзакции существует возможность задать образ для исполнения смартконтракта, создающегося при помощи этой транзакции. Для этого предусмотрены следующие параметры:

- `image` имя Dockerобраза, к которому обращается создаваемый смартконтракт.
- `imagehash` хэшсумма используемого Dockerобраза.
- `contractname` название смартконтракта.
- `password` пароль для доступа к Dockerрепозиторию (опционально).

**Подсказка:** Если в поле `image` указано только имя образа, смартконтракт обращается по адресу, указанному в параметрах конфигурации ноды `remoteregistries` или `defaultregistrydomain` и находит на этом адресе Dockerобраз, указанный в поле `image` транзакции. Также в этом поле может указываться

полный адрес образа: в этом случае для исполнения создаваемого смартконтракта используется образ, находящийся по этому адресу, а соответствующие параметры конфигурации ноды не применяются.

### Пример настройки исполнения Dockerконтракта при помощи транзакции 103 CreateContractTransaction

В приведенном примере рассмотрен вариант транзакции, которая создает Dockerконтракт из отдельного образа, полный адрес которого указан в поле `image`. Соответственно, для этого смартконтракта не будут применяться адреса, указываемые в конфигурационном файле ноды.

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yecRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
  "image": "customregistry.com:5000/stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "params": [],
  "height": 1619
}
```

## 22.2 Смартконтракты Docker с использованием gRPC

Помимо использования REST API смартконтракт может работать с нодой через фреймворк gRPC. Общее техническое описание особенностей реализации контрактов приведено в разделе *Смартконтракты Docker*. Перед написанием смартконтракта необходимо выполнить подготовительные действия, приведенные в разделе *Как использовать фреймворк gRPC*

В следующем разделе рассмотрим пример создания смартконтракта на Python, который выполняет операцию инкремента (увеличение заданного числа на единицу).

### 22.2.1 Описание работы смартконтракта

В нашем примере транзакция 103 для создания контракта инициализирует начальное состояние контракта, сохраняя в нем числовой ключ `sum` со значением 0:

```
{
  "key": "sum",
  "type": "integer",
  "value": 0
}
```

Каждая следующая транзакция вызова 104 увеличивает значение ключа `sum` на единицу (т.е. `sum = sum + 1`).

Как работает смартконтракт после вызова:

1. После старта программы выполняется проверка на наличие переменных окружения. Переменные окружения, используемые контрактом:

- `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 ноды.

Значения переменных `NODE` и `NODE_PORT` берутся из конфигурационного файла ноды секции `dockerengine.grpcserver`. Остальные переменные генерируются нодой и передаются в контейнер при создании смарт контракта.

2. Используя значения переменных окружения `NODE` и `NODE_PORT`, контракт создает gRPC-подключение с нодой.
3. Далее вызывается потоковый метод `Connect` gRPC сервиса `ContractService` (см. `contract.proto` файл). 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 создания (тип транзакции – `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 ноды (метод `CommitExecutionSuccess` сервиса `ContractService`), т.е. происходит обновление состояния контракта.

### 22.2.2 Создание смартконтракта

1. Скачайте и установите Docker for Developers (<https://www.docker.com/get-started>) для вашей операционной системы.
2. Подготовьте образ контракта. В папке с контрактом должны быть следующие файлы:
  - `src/contract.py`
  - `Dockerfile`
  - `run.sh`
  - `src/protobuf/contract.proto`
  - `src/protobuf/common.proto`

- src/protobuf/common\_pb2.py
- src/protobuf/contract\_pb2.py
- src/protobuf/contract\_pb2\_grpc.py

Файлы src/protobuf/common\_pb2.py, src/protobuf/contract\_pb2.py, src/protobuf/contract\_pb2\_grpc.py должны быть сгенерированы gRPCкомпилятором из protobuf файлов contract.proto и common.proto. Описание процедуры генерации программных файлов из protobuf файлов вы можете найти на [официальной странице gRPC](#).

**Важно:** После компиляции файлов необходимо поменять import директиву в сгенерированных файлах:

- в файле contract\_pb2.py должно быть `import protobuf.common_pb2 as common__pb2;`
- в файле contract\_pb2\_grpc.py должно быть `import protobuf.contract_pb2 as contract__pb2.`

3. Если вы хотите, чтобы транзакции с вызовом вашего контракта могли обрабатываться одновременно, то необходимо в самом коде контракта передать параметр `asyncfactor`. Контракт передаёт значение параметра `asyncfactor` в составе gRPCсообщения `ConnectionRequest`:

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

Значение параметра `asyncfactor` может быть как заранее установленное в интервале от 1 до 999, так и динамически вычисляемое. Вы можете устанавливать фиксированное значение этого параметра как константу, однако рекомендуется устанавливать вычисляемое значение данного параметра. Например, контракт может запросить количество свободных ядер и передать это число в качестве значения параметра `asyncfactor`. Это число будет использоваться для параллельной обработки транзакций с контрактом. Если параметр `asyncfactor` не будет определён, то по умолчанию все транзакции с контрактом будут обрабатываться последовательно.

Обратите внимание, что не все средства разработки могут поддерживать параллельную обработку кода контракта. Также логика кода контракта должна учитывать специфику параллельного исполнения контракта. Подробнее о параллельной обработке контрактов можно почитать в разделе *Параллельное исполнение контрактов*.

4. Установите образ в Docker репозиторий образов. Если используете локальный репозиторий, выполните в терминале следующие команды:

```
docker run -d -p 5000:5000 --name registry registry:2
cd contracts/grpc-increment-contract
docker build -t grpc-increment-contract .
docker image tag grpc-increment-contract localhost:5000/grpc-increment-contract
docker start registry
docker push localhost:5000/grpc-increment-contract
```

5. Для получения информации о смартконтракте используйте команду `docker inspect`:

```
docker inspect 57c2c2d2643d
[
{
```

(continues on next page)

(продолжение с предыдущей страницы)

```

"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,

```

**Важно:** Идентификатор Docker образа контракта Id является значением поля `imageHash` для использования в `CreateContractTransaction` транзакциях с созданным смартконтрактом.

- Подпишите транзакцию 103 на создание смартконтракта. В нашем примере транзакция подписывается ключом, сохраненным в `keystore` ноды. Описание REST API ноды и правила формирования транзакций приведены в разделе REST API.

Пример запроса для транзакции создания смартконтракта:

```

{
  "fee": 100000000,
  "image": "localhost:5000/grpc-increment-contract",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "grpc-increment-contract",
  "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2",
  "password": "",
  "params": [],
  "type": 103,
  "version": 2,
}

```

Пример curl-запроса:

```

curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
  header 'X-Contract-API-Token' -d '{ \
    "fee": 100000000, \
    "image": "localhost:5000/grpc-increment-contract", \
    "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
    "contractName": "grpc-increment-contract", \
    "sender": "3PudkbvjV1nPj1TkuuRahh4sGdgfr4YAUUV2", \
    "password": "", \
    "params": [], \
    "type": 103, \
    "version": 2 \
  }' 'http://localhost:6862/transactions/sign'

```

Пример ответа:

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJmVT2M",
  "fee": 100000000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yeChRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 2,
  "image": "localhost:5000/grpc-increment-contract",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "grpc-increment-contract",
  "params": [],
  "height": 1619
}
```

7. Отправьте подписанную транзакцию в блокчейн. Ответ от метода *sign* необходимо передать на вход для метода *broadcast*.

Пример запроса на отправку транзакции создания смартконтракта в блокчейн:

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky",
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJmVT2M",
  "fee": 500000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yeChRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 1,
  "image": "stateful-increment-contract:latest",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "stateful-increment-contract",
  "params": [],
  "height": 1619
}
```

Пример curl-запроса:

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --
↪header 'X-Contract-API-Token' -d '{ \
  "type": 103, \
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLLLZVj4Ky", \
  "sender": "3N3YTj1tNwn8XUJ8ptGKbPuEFNa9GFnhqew", \
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJmVT2M", \
  "fee": 100000000, \
  "timestamp": 1550591678479, \
  "proofs": [
    ↪ "yeChRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxfj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ], \
  "version": 2, \
  "image": "localhost:5000/grpc-increment-contract", \
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65", \
  "contractName": "grpc-increment-contract", \
  "params": [], \
  "height": 1619 \
}' 'http://localhost:6862/transactions/broadcast'
```

Пример ответа:

```
{
  "type": 103,
  "id": "ULcq9R7PvUB2yPMrmBdxoTi3bcRmQPT3JDLlLZVj4Ky",
  "sender": "3N3YTjtNwn8XUJ8ptGKbPuEFNa9GFnhqew",
  "senderPublicKey": "3kW7vy6nPC59BXM67n5N56rhhAv38Dws5skqDsJMVT2M",
  "fee": 100000000,
  "timestamp": 1550591678479,
  "proofs": [
    ↪ "yeCRFZm9iBLyDy93bDVaNo1PR5Qkkic7196GAgUt9TNH1cnQphq4yGQQ8Fxj4BYA4TaqYVw5qxtWzGMPQyVeKYv" ],
  "version": 2,
  "image": "localhost:5000/grpc-increment-contract",
  "imageHash": "7d3b915c82930dd79591aab040657338f64e5d8b842abe2d73d5c8f828584b65",
  "contractName": "grpc-increment-contract",
  "params": [],
  "height": 1619
}
```

Сравните идентификатор транзакции в обеих операциях (поле `id`) и убедитесь, что транзакция с инициализацией контракта размещена в блокчейне.

### 22.2.3 Вызов смартконтракта

1. Подпишите транзакцию *104* на вызов смартконтракта.

Пример запроса для транзакции вызова смартконтракта:

```
{
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
  "fee": 15000000,
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "password": "",
  "type": 104,
  "version": 2,
  "contractVersion": 1,
  "params": []
}
```

2. Отправьте подписанную транзакцию в блокчейн. Ответ от метода *sign* необходимо передать на вход для метода *broadcast*.

**Примечание:** Параметры транзакции *104* (блоки, добавляемые в раздел `params`) поддерживают 4 типа данных: *string*, *integer*, *boolean*, *binary*. Пример использования этих типов данных при оформлении вызова смартконтракта приведен ниже.

Пример запроса на отправку транзакции вызова смартконтракта в блокчейн:

```
{
  "type": 104,
  "id": "9fBrL2n5TN473g1gNfoZqaAqAsAJCuHRHYxZpLexL3VP",
  "sender": "3PKyW5FSn4fmdrLcUnDMRHVyoDBxybRgP58",
  "senderPublicKey": "2YvzcVLrqlCqouVrFZynjfoTEuPNV9GrdaunpgdWXLsq",
  "fee": 15000000,
  "timestamp": 1549365736923,
```

(continues on next page)



(продолжение с предыдущей страницы)

```

    "proofs": [
      "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
    ],
    "version": 1,
    "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2",
    "params": [ {
      "type" : "string",
      "value" : "data",
      "key" : "action"
    }, {
      "type" : "integer",
      "value" : 3,
      "key" : "number"
    }, {
      "type" : "boolean",
      "value" : true,
      "key" : "isPositive"
    }, {
      "type" : "binary",
      "value" : "base64:daaa",
      "key" : "code"
    } ]
  }
}

```

Пример curl-запроса:

```

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": 15000000, \
  "timestamp": 1549365736923, \
  "proofs": [ \
    "2q4cTBhDkEDkFxr7iYaHPAv1dzaKo5rDaTxPF5VHryyYTXxTPvN9Wb3YrsDYixKiUPXBnAyXzEcnKPFRCW9xVp4v"
  ↳\
  ], \
  "version": 1, \
  "contractId": "2sqPS2VAKmK77FoNakw1VtDTCbDSa7nqh5wTXvJeYGo2", \
  "params": [] \
}' 'http://localhost:6862/transactions/broadcast'

```

Пример ответа:

```

[
  {
    "key": "sum",
    "type": "integer",
    "value": 2
  }
]

```

Получите результат выполнения смартконтракта по его идентификатору.

## 22.2.4 Примеры файлов

### Листинг run.sh:

```
#!/bin/sh

eval $SET_ENV_CMD
python contract.py
```

### Листинг 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"]
```

### Листинг смартконтракта на Python:

```
import grpc
import os
import sys

from protobuf import common_pb2, contract_pb2, contract_pb2_grpc

CreateContractTransactionType = 103
CallContractTransactionType = 104

AUTH_METADATA_KEY = "authorization"

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

    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)
```

(continues on next page)

(продолжение с предыдущей страницы)

```

elif contract_transaction.type == CallContractTransactionType:
    self.__handle_call_transaction(contract_transaction_response)
else:
    print("Error: unknown transaction type '{}'".format(contract_transaction.type),
    ↪file=sys.stderr)

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

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

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

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

```

(continues on next page)

(продолжение с предыдущей страницы)

```

    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)

```

**Листинг contract.proto:**

```

syntax = "proto3";
package wavesenterprise;

option java_multiple_files = true;
option java_package = "com.wavesplatform.protobuf.service";
option csharp_namespace = "WavesEnterprise";

import "google/protobuf/wrappers.proto";
import "common.proto";

service ContractService {

    rpc Connect (ConnectionRequest) returns (stream ContractTransactionResponse);

    rpc CommitExecutionSuccess (ExecutionSuccessRequest) returns (CommitExecutionResponse);

    rpc CommitExecutionError (ExecutionErrorRequest) returns (CommitExecutionResponse);

    rpc GetContractKeys (ContractKeysRequest) returns (ContractKeysResponse);

    rpc GetContractKey (ContractKeyRequest) returns (ContractKeyResponse);
}

message ConnectionRequest {
    string connection_id = 1;
}

message ContractTransactionResponse {
    ContractTransaction transaction = 1;
    string auth_token = 2;
}

message ContractTransaction {
    string id = 1;
    int32 type = 2;
    string sender = 3;
    string sender_public_key = 4;
    string contract_id = 5;
    repeated DataEntry params = 6;
    int64 fee = 7;
    int32 version = 8;
}

```

(continues on next page)

(продолжение с предыдущей страницы)

```
bytes proofs = 9;
int64 timestamp = 10;
AssetId fee_asset_id = 11;

oneof data {
  CreateContractTransactionData create_data = 20;
  CallContractTransactionData call_data = 21;
}

message CreateContractTransactionData {
  string image = 1;
  string image_hash = 2;
  string contract_name = 3;
}

message CallContractTransactionData {
  int32 contract_version = 1;
}

message ExecutionSuccessRequest {
  string tx_id = 1;
  repeated DataEntry results = 2;
}

message ExecutionErrorRequest {
  string tx_id = 1;
  string message = 2;
}

message CommitExecutionResponse {
}

message ContractKeysRequest {
  string contract_id = 1;
  google.protobuf.Int32Value limit = 2;
  google.protobuf.Int32Value offset = 3;
  google.protobuf.StringValue matches = 4;
  KeysFilter keys_filter = 5;
}

message KeysFilter {
  repeated string keys = 1;
}

message ContractKeysResponse {
  repeated DataEntry entries = 1;
}

message ContractKeyRequest {
  string contract_id = 1;
  string key = 2;
}

message ContractKeyResponse {
  DataEntry entry = 1;
}
```

(continues on next page)

(продолжение с предыдущей страницы)

```

}

message AssetId {
    string value = 1;
}

```

**Листинг common.proto:**

```

syntax = "proto3";
package wavesenterprise;

option java_multiple_files = true;
option java_package = "com.wavesplatform.protobuf.common";
option csharp_namespace = "WavesEnterprise";

message DataEntry {
    string key = 1;
    oneof value {
        int64 int_value = 10;
        bool bool_value = 11;
        bytes binary_value = 12;
        string string_value = 13;
    }
}

```

## 22.3 Сервисы gRPC, используемые смартконтрактом

Общее описание работы смартконтрактов с использованием gRPC приведено в разделе *Смартконтракты Docker с использованием gRPC*

Protobuf файлы, необходимые для работы со смартконтрактами посредством gRPC, вы можете скачать со страницы проекта в [GitHub](#). Список protobuf файлов следующий:

- address.proto методы для работы с адресами.
- common.proto общий файл для работы других protobuf файлов.
- crypto.proto методы для работы с шифрованием данных.
- permission.proto методы для работы с выдачей разрешений.
- pki.proto методы работы с PKI.
- privacy.proto методы работы с приватными данными.
- util.proto методы для служебных утилит.
- contract.proto методы для работы с контрактами.

Каждый protobuf файл (кроме common.proto) содержит набор небольших блоков, включающих набор полей «ключзначение». Список таких блоков для каждого файла приведен ниже.

**address.proto**

- GetAddresses получение всех адресов участников, ключевые пары которых хранятся в keystore ноды.
- GetAddressData получение всех данных, записанных на аккаунт адресата {address}.

**contract.proto**

- **Connect** подключение контракта к ноду.
- **CommitExecutionSuccess** получение результата успешного исполнения контракта и отправка результатов на ноду.
- **CommitExecutionError** получение ошибки исполнения контракта и отправка результатов на ноду.
- **GetContractKeys** получение результата исполнения контракта по его идентификатору (id транзакции создания контракта).
- **GetContractKey** получение результата исполнения контракта по его идентификатору (id транзакции создания контракта) и ключу {key}.

#### **crypto.proto**

- **EncryptSeparate** шифрование данных отдельно для каждого получателя уникальным ключом.
- **EncryptCommon** шифрование данных единым ключом СЕК для всех получателей, СЕК оборачивается уникальными КЕК для каждого получателя.
- **Decrypt** расшифровка данных. Расшифровка доступна в случае, если ключ получателя сообщения находится в keystore ноды.

#### **permission.proto**

- **GetPermissions** получение списка всех ролей на указанный адрес, действительных на текущий момент.
- **GetPermissionsForAddresses** получение списка всех ролей, действительных на текущий момент, на указанный диапазон адресов.

#### **pki.proto**

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

#### **privacy.proto**

- **GetPolicyRecipients** получение адресов всех участников, записанных в группу {policyid}.
- **GetPolicyOwners** получение адресов всех владельцев, записанных в группу {policyid}.
- **GetPolicyHashes** получение массива идентификационных хешей, которые записаны в привязке к {policyid}.
- **GetPolicyItemData** получение пакета конфиденциальных данных по идентификационному хешу.
- **GetPolicyItemInfo** получение метаданных для пакета конфиденциальных данных по идентификационному хешу.

#### **util.proto**

- **GetNodeTime** получение текущего времени на ноду.

## 22.4 Методы REST API, доступные смартконтракту

---

**Важно:** Методы REST API для смартконтрактов Docker постепенно выводятся из эксплуатации.

---

Смартконтракты на базе контейнеров Docker могут использовать *REST API ноды*. Общее руководство по созданию смартконтрактов при помощи REST API приведено в статье Смартконтракты Docker с использованием REST API ноды

Разработчикам Docker смартконтрактов доступны не все методы REST API. Ниже приведен список методов REST API ноды, которые смартконтракт может использовать прямо из Docker контейнера.

### Методы Addresses

- *GET /addresses*
- *GET /addresses/publicKey/{publicKey}*
- *GET /addresses/balance/{address}*
- *GET /addresses/data/{address}*
- *GET /addresses/data/{address}/{key}*

### Методы Crypto

- *POST /crypto/encryptCommon*
- *POST /crypto/encryptSeparate*
- *POST /crypto/decrypt*

### Методы Privacy

- *GET /privacy/{policyid}/getData/{policyitemhash}*
- *GET /privacy/{policyid}/getInfo/{policyitemhash}*
- *GET /privacy/{policyid}/hashes*
- *GET /privacy/{policyid}/recipients*

### Методы Transactions

- *GET /transactions/info/{id}*
- *GET /transactions/address/{address}/limit/{limit}*

### Методы Contracts

Для улучшения производительности смартконтракт может использовать методы *Contracts* по выделенному маршруту */internal/contracts/*, которые полностью идентичны обычным методам *Contracts*.

- *GET /internal/contracts/{contractId}/{key}*
- *GET /internal/contracts/executedtxfor/{id}*
- *GET /internal/contracts/{contractId}*
- *GET /internal/contracts*

### Методы PKI

- *PKI /verify*



### 22.4.1 Авторизация Docker смартконтракта

Для работы с *REST API* ноды смартконтракту необходима авторизация. Чтобы Dockerконтракт корректно работал с методами API, выполняются следующие действия:

1. В переменных окружения Dockerконтракта должны быть определены следующие переменные:
  - `NODE_API` URLадрес к *REST API* ноды.
  - `API_TOKEN` токен авторизации для Dockerконтракта.
  - `COMMAND` команды для создания и вызова Dockerконтракта.
  - `TX` транзакция, необходимая Dockerконтракту для работы (коды *103 107*).
2. Разработчик Dockerконтракта присваивает значение переменной `API_TOKEN` заголовку запроса `XContractApiToken`. В переменную `API_TOKEN` нода прописывает *JWT* токен авторизации при создании и выполнении контракта.
3. Код контракта должен передавать полученный токен в заголовке запроса (`XContractApiToken`) при каждом обращении к API ноды.

---

## Управление ролями участников

---

Список возможных ролей в блокчейнплатформе приведен в разделе «Авторизация участников».

---

**Важно:** Обязательным условием для изменения полномочий участников (добавления или удаления ролей) является наличие приватного ключа участника с ролью «permissioner» в keystore ноды, с которой осуществляется запрос.

---

### 23.1 Вариант №1: через REST API

Управление полномочиями участника выполняется путем подписания (метод sign) и рассылки (метод broadcast) permissionтранзакций через *REST API* ноды.

Объект запроса для метода sign:

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

Поля запроса:

- type тип транзакции для управления полномочиями участников (type = 102);
- sender адрес участника с полномочиями на выпуск permissionтранзакций;
- proofs подпись транзакции;

- `target` адрес участника, для которого требуется установить или удалить полномочия;
- `role` полномочия участника, которые требуется установить или удалить. Возможные значения: «miner», «issuer», «dex», «permissioner», «blacklister», «banned», «contract\_developer», «connection\_manager»;
- `opType` тип операции «add» (добавить полномочия) или «remove» (удалить полномочия);
- `dueTimestamp` дата действия permission в формате timestamp. Поле является опциональным.

Полученный ответ от ноды передается методу broadcast.

## 23.2 Вариант №2: через утилиту Generators

Утилита Generators позволяет автоматизировать процесс управления ролями участников.

Пример запуска из консоли:

```
java -jar generators.jar GrantRolesApp [configfile]
```

Пример конфига:

```
permission-granter {
waves-crypto = no
chain-id = T
account = {
  addresses = [
    "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w"
  ]
  storage = "${user.home}"/node/keystore.dat"
  password = "some string as password"
}
send-to = [
  "devnet-aws-fr-2.we.wavesnodes.com:6864"
]
grants = [
  {
    address: "3N2cQFfUDzG2iujBrFTnD2TAsCNohDxYu8w"
    assigns = [
      {
        permission = "miner",
        operation = "add",
        due-timestamp = 1527698744623
      },
      {
        permission = "issuer",
        operation = "add",
        due-timestamp = 1527699744623
      },
      {
        permission = "blacklister",
        operation = "add"
      },
      {
        permission = "permissioner",
        operation = "remove"
      }
    ]
  }
]
```

(continues on next page)

(продолжение с предыдущей страницы)

```

    ]
  }
]
txs-per-bucket = 10
}

```

Поле «duetimestamp» ограничивает время действия роли; Поля nodes, roles обязательные.

Если у ноды уже задана какая-либо из ролей, которая задана в конфиге, то ситуация обрабатывается в соответствии с правилами:

Текущее состояние ноды	Состояние полученное из транзакции	Результат обработки
Роль не назначена	Новая роль	Success назначается роль
Назначена роль без dueDate	Роль с dueDate	Проверка dueDate, если меньше текущей, то IncorrectDatetime, иначе Success назначается роль с dueDate
Назначена роль с dueDate	Роль с dueDate	Проверка dueDate, если меньше текущей, то IncorrectDatetime, иначе Success обновление dueDate
Назначена роль с dueDate	Роль без dueDate	Success назначается роль без dueDate
Назначена роль с/без dueDate	Удаление роли	Проверка адреса ноды, если <> адресу генезиса, то Success удаляется роль

---

## Подключение участников к сети

---

Момент *запуска* первой ноды можно считать началом создания новой блокчейнсети. Разворачивать блокчейнсеть вы можете со старта всего одной ноды, добавляя новые ноды по мере необходимости.

- *Подключите* новую ноду к уже существующей сети.
- *Удалите* лишние ноды из сети.

### 24.1 Подключение новой ноды к существующей сети

Новые ноды можно добавлять в сеть в любой момент времени. Настройка конфигурационных файлов новой ноды описана в подразделе *Установка и запуск платформы Waves Enterprise*. Выполните все указанные в приведённом подразделе действия и *запустите* ноду. Далее выполняются следующие действия:

1. Пользователь новой ноды передаёт публичный ключ и описание ноды администратору сети.
2. Администратор сети (нода с ролью «Connectionmanager») использует полученные публичный ключ и описание ноды при создании транзакции *111 RegisterNode* с параметром "opType": "add".
3. Транзакция попадает в блок и далее в стейты нод участников сети. Вследствие транзакции среди сохраняемых данных каждый участник сети хранит обязательно публичный ключ и адрес новой ноды.
4. При необходимости администратор сети может добавить новой ноде дополнительные роли при помощи транзакции *102 Permit*.
5. Пользователь *запускает* ноду.
6. После запуска нода отправляет *handshake*сообщение со своим публичным ключом участникам из списка «peers» своего конфигурационного файла.
7. Участники сети сравнивают публичный ключ из *handshake*сообщения и ключ из транзакции *111 RegisterNode*, отправленной администратором сети. Если проверка успешна, участник сети обновляет свою БД и рассылает в сеть сообщение *Peers Message*.
8. Успешно подключившись, новая нода выполняет синхронизацию с сетью и получает таблицу адресов участников сети.

## 24.2 Удаление ноды

1. Администратор сети создает транзакцию *111 RegisterNode* с параметром "opType": "remove" для удаления ноды из сети, в которую помещается её публичный ключ.
2. Транзакция с удалением ноды вместе с остальными попадает в блок, и её принимают другие ноды.
3. После принятия транзакции ноды находят в своем стейте публичный ключ, указанный в транзакции *111 RegisterNode*, и удаляют его из стеята.
4. Далее ноды удаляют сетевой адрес ноды с ключом, указанным в транзакции *111 RegisterNode*, из списка `network.knownpeers` конфигурационного файла ноды.

---

## Обмен конфиденциальными данными

---

Перед тем, как обмениваться конфиденциальными данными, вам нужно создать группы доступа. Используя транзакции, вы можете *добавлять* или *изменять* группы доступа к конфиденциальным данным.

### 25.1 Создание группы доступа к конфиденциальным данным

Группу доступа к конфиденциальным данным может создать любой участник сети. Перед созданием группы необходимо определиться с кругом участников сети, которые будут получать конфиденциальные данные. Далее любой из участников выполняет следующие действия:

1. Участник сети, который будет владельцем группы доступа, создаёт транзакцию *112 CreatePolicy* со следующими основными параметрами:
  - **sender** публичный ключ создателя группы доступа.
  - **description** описание группы доступа.
  - **policyName** имя группы доступа.
  - **recipients** публичные ключи участников группы доступа, которые будут иметь право получать конфиденциальные данные.
  - **owners** публичные ключи владельцев группы доступа, которые, помимо доступа к данным, смогут изменять состав участников группы.
2. Транзакция с созданием группы доступа вместе с остальными попадает в блок, и её принимают другие ноды.
3. После принятия транзакции доступ к отправляемым в сеть конфиденциальным данным получают все участники, зарегистрированные в созданной группе доступа.

## 25.2 Изменение группы доступа

Изменять группы доступа могут только их владельцы. Выполняются следующие действия для изменения списка участников в группе доступа:

1. Владелец группы доступа создаёт транзакцию *113 UpdatePolicy* со следующими основными параметрами:
  - **policyId** идентификатор группы доступа;
  - **sender** публичный ключ владельца группы доступа;
  - **opType** опция добавления (add) или удаления (remove) участников группы;
  - **recipients** публичные ключи участников группы доступа, которые добавляются или удаляются из группы доступа;
  - **owners** публичные ключи владельцев группы доступа, которые добавляются или удаляются из группы доступа.
2. Транзакция с изменением группы доступа вместе с остальными попадает в блок, и её принимают другие ноды.
3. После принятия транзакции в сети обновляется информация об участниках изменённой группы.

## 25.3 Процесс обмена конфиденциальными данными

---

**Важно:** Через метод API *POST /privacy/sendData* можно отправить данные размером не более 20МБ.

---

1. Пользователь отправляет данные в сеть, используя инструмент API *POST /privacy/sendData* (параметры API: отправитель, пароль, ID группы, тип данных, информация о данных, данные и хеш).
2. Участники группы доступа используют инструмент *GET /privacy/{policyId}/getData/{policyItemHash}* для получения информации о данных и их последующей загрузки.

Выполните следующие действия для создания значений в поля `data` и `hash`:

1. Переведите байтовую последовательность данных в кодировку **Base64**.
2. Результат преобразования данных поместите в поле "data": "29sCt...RgdC60LL" запроса API *POST /privacy/sendData*.
3. В поле "hash": "9wetTB...SU2zr1Uh" укажите хешсумму от данных по алгоритму **SHA256**. Результат хеширования нужно указывать в кодировке **Base58**.
4. Отправьте данные в сеть, нажав кнопку *Try it out!*.
5. В результате отправки данных в сеть нода автоматически сформирует транзакцию *114 PolicyDataHash*.



---

## Операции шифрования данных

---

Для операций шифрования/расшифрования данных применяются симметричные ключи СЕК и КЕК. СЕК (Content Encryption Key) используется для шифрования текстовых данных, КЕК (Key Encryption Key) используется для шифрования СЕК. Ключ СЕК формируется блокчейн-узлом случайным образом с применением соответствующих алгоритмов хеширования. Ключ КЕК формируется нодой на базе алгоритма ДиффиХелмана, используя публичные и приватные ключи отправителя и получателей, и применяется для шифрования ключа СЕК.

Симметричный ключ СЕК недоступен для прочтения и не отображается в процессе шифрования. От отправителя к получателю он передается в зашифрованном виде (*wrappedKey*) по открытым каналам связи вместе с зашифрованным сообщением. Одним из таких каналов может являться запись в блокчейн *DataTransaction* или стейт смартконтракта. Ключ КЕК от отправителя к получателю не передается, он восстанавливается получателем на основе своего закрытого ключа и известного публичного ключа отправителя (алгоритм *DiffieHellman key exchange*).

Процесс шифрования/расшифрования данных включает в себя следующие действия:

1. Для шифрования данных для каждого получателя отдельно используется метод *POST /crypto/encryptSeparate*. Параметры в объекте запроса:
  - *sender* адрес отправителя;
  - *password* пароль от ключевой пары отправителя, создаваемый вместе с аккаунтом;
  - *encryptionText* текст для шифрования;
  - *recipientsPublicKeys* массив публичных ключей получателей.
2. Для шифрования данных для всех получателей единым ключом СЕК используется метод *POST /crypto/encryptCommon*.
3. Для расшифровывания данных используется метод *POST /crypto/decrypt*. Параметры в объекте запроса:
  - *recipient* адрес получателя;
  - *password* пароль от ключевой пары получателя, создаваемый вместе с аккаунтом;

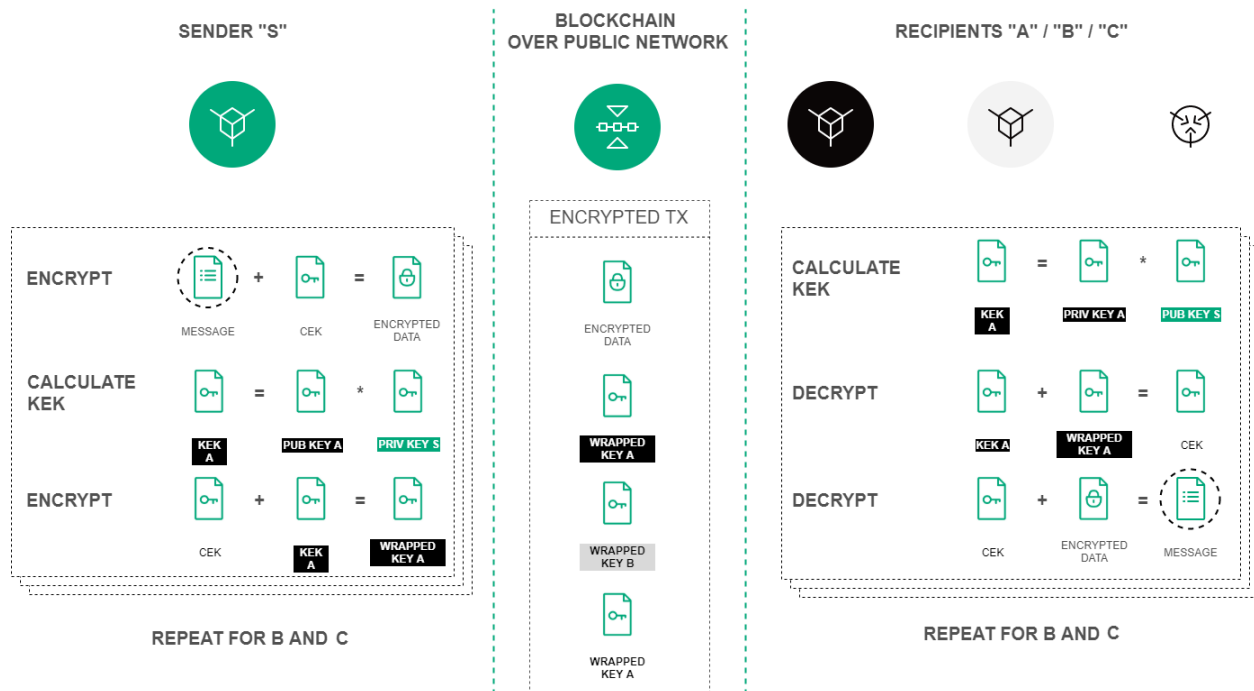


Рис. 1: Процедура шифрования текстовых данных на базе алгоритма ДиффиХелмана

- `encryptedText` зашифрованный текст;
- `wrappedKey` обернутый ключ, полученный при шифровании данных;
- `senderPublicKey` публичный ключ отправителя.

JavaScript SDK

---

Пакет JavaScript SDK предназначен для разработчиков приложений, интегрируемых с платформой Waves Enterprise. С помощью JavaScript SDK пользователи разработанных приложений подписывают и отправляют все типы транзакций в блокчейнсеть.

JavaScript SDK использует методы REST API для работы с блокчейном. В отличие от прямого взаимодействия с блокчейном через традиционный инструмент REST API приложение подписывает транзакцию локально в браузере или в среде Node.js, не обращаясь к ноде. Подписанная транзакция отправляется сразу в сеть. Такой способ работы с блокчейном более удобен и эффективен для разработки сторонних сервисов и приложений, взаимодействующих с блокчейнсетью. Данные передаются и принимаются в формате *json* по HTTPS-протоколу.

JavaScript SDK работает одновременно и в браузере, и в среде Node.js. Вебприложения используют вариант работы в браузере, бэкендприложения — работу в среде Node.js. Если вы хотите работать без браузера, то для использования JavaScript SDK потребуется установка LTS версии [Node.js](#). Сам пакет JavaScript SDK и инструкцию по установке и инициализации вы найдёте на [странице](#) в GitHub. Общая схема работы JavaScript SDK представлена ниже.



## 27.1 Состав JavaScript SDK

JavaScript SDK включает в себя исходные материалы для работы, а также два вспомогательных компонента:

- **transactionsfactory** компонент предназначен для корректной сериализации в байты всех типов транзакций для последующей подписи приватным ключом.
- **signaturegenerator** компонент, подписывающий транзакции и поддерживающий ГОСТ и Waves криптографию.

## 27.2 Криптографические методы ноды в JavaScript SDK

Для реализации криптографических алгоритмов SDK поддерживает *криптографические методы REST API ноды*:

- `crypto/encryptCommon`
- `crypto/encryptSeparate`
- `crypto/decrypt`

## 27.3 Авторизация в блокчейне через JavaScript SDK

Пользователь приложения использует стандартные средства авторизации в блокчейне Waves Enterprise. Подробнее о методах авторизации вы можете почитать в разделе *Методы авторизации*.

JavaScript SDK поддерживает авторизацию в браузере и в среде Node.js. Для авторизации в браузере используется интерфейс [Fetch API](#). Для Node.js применяется HTTPклиент [Axios](#).

## 27.4 Подписание и отправка транзакций в блокчейн

Приложение может создавать, подписывать и отправлять в блокчейн любой транзакции. Список всех транзакций вы можете посмотреть в разделе *Транзакции*.

1. Создание транзакции инициируется приложением.
2. Все поля транзакции сериализуются в байты и подписываются приватным ключом пользователя прямо в браузере или в среде Node.js.
3. JavaScript SDK отправляет транзакцию в блокчейн, используя соответствующий HTTP POST запрос.
4. Приложение получает ответ в виде хэша транзакции на выполненный HTTP POST запрос.

Для получения ID транзакции до её отправки воспользуйтесь отдельным методом:

```
const txHash = await Waves.API.Node.transactions.getTxId('transfer', txBody, { publicKey })
```

## 27.5 Создание seedфразы

Приложение работает с seedфразой в следующих вариантах:

- Создаёт фразу в совершенно случайном порядке.
- Создаёт зашифрованную фразу с указанным паролем.
- Расшифровывает фразу при помощи указанного пароля.

**Аккаунт**

Набор данных о пользователе, содержащийся в БД и использующийся для его идентификации

**Алиас**

Логин пользователя, связанный с его адресом в результате транзакции, по итогам которой в БД записывается сопоставление алиасадрес, и можно в последующих транзакциях указывать алиас

**Анонимная сеть**

Блокчейн открытого типа, к которой может подключиться любой участник на правах анонимности

**Атомикконтейнер**

Транзакция, помещающая другие транзакции в контейнер, которые выполняются атомарно: либо выполнятся все, либо не выполнится ни одна из них. В качестве атомарной транзакции выступает *120* транзакция

**Блокчейн**

Децентрализованный, распределённый и общедоступный цифровой реестр, записывающий информацию таким образом, что любая соответствующая запись не может быть изменена задним числом без внесения изменений во все последующие блоки

**Генезис блок**

Первый блок в блокчейне, содержащий специальные генезис транзакции, распределяющие первоначальный баланс и разрешения

**Группа доступа**

Таблица в стейте ноды, содержащая список участников сети, которые могут обмениваться конфиденциальными данными в рамках этой группы

**Криптовалюта**

Цифровая монета, основанная на криптографических алгоритмах и обращающаяся на децентрализованных платформах, построенных по технологии блокчейн

**Консенсус**

Способ согласования об информационном событии о транзакциях и блоках в сети между участниками

### **Майнинг**

Процесс верификации и добавления в блокчейн транзакций

### **Мейннет**

Рабочая сеть, в которой происходят транзакции, выпуск и хранение токенов

### **Нода**

Компьютер с запущенным ПО ноды и включённый в блокчейн

### **Пир**

Сетевой адрес ноды

### **Приватный ключ**

Строковая комбинация символов для подписания транзакций и доступа к токенам, хранимая приватно. Приватный ключ неразрывно связан с публичным ключом

### **Публичная сеть**

Блокчейн закрытого типа, где каждый участник известен и зарегистрирован в сети

### **Публичный ключ**

Строковая комбинация символов, неразрывно связанная с приватным ключом. Публичный ключ прикладывается к транзакциями для подтверждения корректности подписи пользователя сделанной на закрытом ключе

### **Публичный адрес**

Публичный адрес представляет собой хеш публичного ключа и байт сети. В отличии от приватных ключей публичный адрес может использоваться где угодно в сети, как электронный адрес

### **Роллбек**

Откат уже созданного блока на повторный майнинг

### **Форк**

Образование новой ветки блокчейна

### **Сваггер**

Инструмент для работы с API

### **Секретная фраза**

Набор из 24 произвольно заданных слов для восстановления доступа к токенам

### **Смартаккаунт**

Аккаунт, к которому добавлены определённые свойства для создания и исполнения смартконтрактов

### **Смартассет**

Токен с привязанным к нему скриптом, при осуществлении каждой новой транзакции с таким токеном она будет подтверждаться сначала скриптом, потом уже блокчейном

### **Смартконтракт**

Программный код, предназначенный для облегчения, прямого выполнения или обеспечения выполнения соглашения между участниками

### **Стейт**

Полная история транзакций, хранящаяся во внутренней БД ноды

#### **Стейт контракта**

Все текущие данные смартконтракта, после каждого вызова *104* транзакции данные обновляются

#### **Стейт адреса**

Комплексная сущность, которая включает в себя сведения, связанные с какимлибо участником (блокчейн-адресом) балансы, информация о транзакций с данными (в формате ключзначение), данные смартконтрактов (в формате ключзначение) и т.д.

#### **Токен**

Расчётная единица, актив блокчейна, не являющийся криптовалютой и предназначенный для представления цифрового баланса, аналог акций компании

#### **Транзакция**

Операция, производимая участниками в сети, для инициации любых действий

#### **Участник**

Участник блокчейна, который направляет транзакции на подтверждение в сеть

#### **Хеш**

Уникальная конфигурация символов (буквы, цифры), результат исполнения хешфункции по заданному алгоритму над данными. Хеш однозначно идентифицирует объект

#### **Частная сеть**

Блокчейн закрытого типа, где все транзакции контролируются центральным органом

#### **Шлюз**

Приложение для перевода токенов из одного блокчейна в другой

#### **Эйдроп**

Процесс раздачи токенов пользователям blockchainкошельков на безвозмездной основе

#### **CFT (Crash Fault Tolerance)**

Алгоритм консенсуса на основе PoA, минимизирующий возникновение форков блокчейна при какойлибо неполадке со стороны одного или нескольких участников.

#### **PoS (ProofofStake)**

Алгоритм консенсуса на основе «доли», которая применяется для выбора ноды для следующего процесса проверки транзакций и создания блока

#### **LPoS**

Алгоритм консенсуса, который является улучшенной версией ProofofStake. Особенностью алгоритма является то, что он предоставляет возможность сдавать токены пользователя в лизинг

#### **PoA (ProofofAuthority)**

Алгоритм консенсуса в приватном блокчейне, при котором возможность проверки транзакций и создание новых блоков отводится более авторитетным узлам



---

### Что нового в Waves Enterprise

---

#### 29.1 1.5.0

Версия 1.5.0 является последней выпущенной версией и в этой справочной системе имеет тег *latest*.

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 Смартконтракты Docker с использованием REST API ноды убран из индекса

## 29.2 1.4.0

Добавлены следующие разделы:

- *Атомарные транзакции*
- *Работа в вебклиенте*
- *JavaScript SDK*

Изменены следующие разделы:

- *Архитектура*
- *Транзакции*
- *Настройка авторизации и REST API и gRPC интерфейсов ноды*
- *API инструменты ноды*
- *Обновление ноды*

## 29.3 1.3.1

Добавлены следующие разделы:

- *Параллельное исполнение контрактов*

Изменены следующие разделы:

- *Создание смартконтракта*
- *Настройка Docker*

## 29.4 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*

## 29.5 1.2.3

Изменены следующие разделы:

- *Смартконтракты Docker*
- *Описание основных параметров и секций конфигурационного файла ноды*
- *Настройка групп доступа к конфиденциальным данным*

## 29.6 1.2.2

Добавились следующие разделы:

- Методы REST API *Debug*
- Полное описание REST API на странице [Документация API](#)

Изменены следующие разделы:

- *Установка и запуск платформы Waves Enterprise*

## 29.7 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 *Смартконтракты Docker с использованием gRPC*
- Добавлен новый раздел *Сервисы gRPC, используемые смартконтрактом*

Изменены следующие разделы:

- *Установка и запуск платформы Waves Enterprise*
- Обновлен раздел *Криптография*. Часть информации была перенесена в *Операции шифрования данных*
- *Изменения в конфигурационном файле ноды*
- *Транзакции*

## 29.8 1.1.2

Изменены следующие разделы:

- Демоверсия
- *Изменения в конфигурационном файле ноды*
- Раздел *Установка ноды* преобразован в раздел «Установка и запуск платформы Waves Enterprise»
- *Подключение участников к сети*
- *Настройка анкоринга*
- *Настройка типа авторизации для доступа к REST API ноды*
- *Подключение ноды в сеть «Waves Enterprise Partnernet»*
- *Подключение ноды в сеть «Waves Enterprise Mainnet»*
- *Системные требования*

## 29.9 1.1.0

Добавились следующие разделы:

- *Методы API, доступные смартконтракту*
- Демоверсия
- *Изменения в конфигурационном файле ноды*

Изменены следующие разделы:

- Смартконтракты Docker
- Пример запуска контракта Docker
- *Установка ноды*
- Конфигурация и запуск дополнительных сервисов

## 29.10 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 *NTPсервер*
- Добавлена секция *auth* выбора типа авторизации в *REST API* секции