109052, г. Москва, ул. Смирновская, д. 2, стр. 1, пом. 108А
https://web3tech.ru/confident
E-mail: support@web3tech.ru

Блокчейн-платформа Конфидент

Версия 1.9

Руководство по установке

Листов 44

2026 г.

Содержание

1 Основные технические данные и характеристики блокчейн-платформы Конфидент

1.1 Программно-аппаратные среды функционирования

Блокчейн-платформа Конфидент функционирует в следующих группах программно-аппаратных сред:

- Astra Linux Special Edition 1.6 (x64).

Примечание. При эксплуатации блокчейн-платформы Конфидент необходимо учитывать, что порядок и сроки эксплуатации операционных систем, в среде которых функционирует блокчейн-платформа Конфидент, определяются производителями операционных систем. Использование блокчейн-платформы Конфидент под управлением ОС, для которых не выпускаются обновления, не допускается.

Требования к характеристикам сервера развертывания блокчейн-платформы Конфидент

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

Примечание. «Xmx» – флаг, определяющий максимальный размер доступной для JVM памяти.

Требования к окружению для блокчейн-платформы Конфидент:

Eclipse Temurin Java SE 11 (64-bit) или OpenJDK 11.0.12-jre (64-bit) (необходимы для запуска генератора);
Для работы со смарт-контрактами необходимо ПО Docker CE версии 19.03.14.

Требования к окружению для узла (ноды):

OpenJDK 11.0.12-jre (64-bit);
Ядро, реализующее криптографические алгоритмы и протоколы, описано в документации, которая поставляется вместе с дистрибутивом платформы.

Примечание.

1. При эксплуатации блокчейн-платформы Конфидент необходимо учитывать, что порядок и сроки эксплуатации ОС, в среде которых функционирует Блокчейн-платформа Конфидент, определяются производителями ОС. Использование ОС, поддержка которых остановлена производителем, не допускается.
2. Необходимо использовать дистрибутивы указанных ОС, полученные у разработчика ОС, и их штатные репозитории с пакетами. Использование прочих сборок ОС не допускается.

2 Установка и удаление дистрибутива ПО блокчейн-платформы Конфидент

2.1 Установка ПО блокчейн-платформы Конфидент

Блокчейн-платформа Конфидент упакована в JAR-архив и не требует установки как таковой. Для разворачивания платформы необходима только Java-машина Java JRE версии 11 (64-bit).

Развертывание дистрибутива блокчейн-платформы Конфидент должно производиться пользователем, имеющим права администратора.

Важно: Администратор должен разворачивать узел только локально.

Предварительные шаги для разных режимов развёртывания блокчейн-платформы Конфидент, а также само развёртывание описаны ниже в разделе «Развертывание платформы Конфидент».

2.2 Обновление ПО блокчейн-платформы Конфидент

Для получения инструкций по обновлению блокчейн-платформы Конфидент свяжитесь со службой технической поддержки ООО «Веб3 Интегратор», перейдя по ссылке "Поддержка" на портале web3tech.ru.

2.3 Удаление ПО блокчейн-платформы Конфидент

Для удаления блокчейн-платформы Конфидент необходимо выполнить следующие шаги:

1. остановить процесс с помощью командной строки;
2. удалить jar-файлы платформы и библиотек ядра, реализующего криптографические алгоритмы и протоколы.

3 Развертывание блокчейн-платформы Конфидент

Блокчейн-платформу Конфидент можно развернуть в одном из следующих режимов:

- в ознакомительном (тестовом) режиме (Sandbox) – см. раздел «Развертывание платформы в ознакомительном режиме (Sandbox)» ниже ;
- в частной сети – см. раздел «Развертывание платформы в частной сети» ниже.

Важно: Перед развертыванием платформы необходимо провести проверку её целостности согласно разделу 5 «Контроль целостности Платформы «Конфидент» документа «643.19172778.2019662198-01 95. «Блокчейн-платформа Конфидент» Версия 1.9. Правила пользования», который поставляется вместе с дистрибутивом платформы.

Примечание: Настройка канала связи по протоколу TLS описана в разделе «Тонкая настройка платформы «Конфидент»: настройка TLS» ниже.

3.1 Развертывание блокчейн-платформы Конфидент в тестовом ознакомительном режиме (Sandbox)

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

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

- отправка транзакций;
- прием данных из блокчейна;
- установка и вызов смарт-контрактов;
- передача конфиденциальных данных между узлами;
- тестирование методов подписи и шифрования;

Взаимодействие с блокчейн-платформой Конфидент осуществляется через интерфейсы gRPC и REST API.

3.1.1 Установка блокчейн-платформы Конфидент

Перед началом установки убедитесь, что на вашей машине установлены Docker Engine и Docker Compose. Также ознакомьтесь с системными требованиями к блокчейн-платформе Конфидент, представленными в разделе «Программно-аппаратные среды функционирования» выше.

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

1. Создайте рабочую директорию и поместите в нее файл **docker-compose.yml**. Файл docker-compose.yml доступен для скачивания по адресу: https://web3tech.ru/assets/docker-compose.yml.
2. Откройте терминал и перейдите в директорию, содержащую файл docker-compose.yml.
   Запустите Docker-контейнер для развертывания блокчейн-платформы Конфидент:

   ```
   docker run --rm -ti -v $(pwd):/config-manager/output web3techru/config-manager:v1.9.0
   ```

   Дождитесь сообщения об окончании развертывания.

В результате будут созданы 3 узла с автоматически сгенерированными учетными данными. Информация об узлах доступна в файле **./credentials.txt**:

```
node-0
blockchain address:    3Nzi7jJYn1ek6mMvtKbPhehxMQarAz9YQvF
public key:            7cLSA5AnvZgiL8CnoffwxXPkpQhvviJC9eywBKSUsi58
keystore password:     OEtrVSL9gzjO87jYx-gIoQ
keypair password:      JInWk1kauuZDHGXFJ-rNXQ
API key:               we

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

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

3. Запустите готовую конфигурацию:

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

При успешном запуске узлов и сервисов отобразится сообщение:

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

После успешного запуска контейнеров интерфейс REST API узла располагается по адресу **127.0.0.1/node-0** или **localhost/node-0**.

Внимание: По умолчанию для локального nginx-сервера блокчейн-платформы Конфидент предоставляется порт **80:80**. Если на вашей ОС этот порт занят другим приложением, измените параметр **ports** секции **nginxproxy** в файле **docker-compose.yml**, выбрав доступный порт:

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

После этого интерфейс REST API узла будет доступен по адресу **127.0.0.1:81** или **localhost:81**.

4. Для остановки запущенных узлов выполните команду:

```
docker-compose down
```

3.1.2 Блокчейн-платформа Конфидент в ознакомительном режиме: устранение ошибок

1. **Ошибка** при запуске контейнера для развертывания Платформы «Конфидент»:

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

Причина: Повторный запуск контейнера.

Решение: Удалите рабочую директорию с файлами Платформы «Конфидент» и начните заново со скачивания файла *docker-compose.yml*.

2. **Ошибка** при запуске Платформы «Конфидент» после успешного развертывания:

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

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

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

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

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

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

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

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

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

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

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

3. **Ошибка** при запуске контейнеров:

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

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

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

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

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

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

После этого REST API интерфейс узла будет доступен по адресу **127.0.0.1:81** или **localhost:81**. Остальные сервисы будут доступны по адресам со своими прежними портами.

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

```
  SSL_ERROR_RX_RECORD_TOO_LONG
```

Причина: Вход на localhost по умолчанию выполняется через HTTPS, однако при развертывании Платформы «Конфидент» в ознакомительном режиме SSL не предусмотрено.

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

3.2 Развертывание блокчейн-платформы Конфидент в частной сети

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

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

Описание ключевой системы и инструкции по работе с ключами приведены в разделе «Ключевая система и ключевые носители» документа «643.19172778.2019662198-01 95. «Блокчейн-платформа Конфидент» Версия 1.9. Правила пользования», который поставляется вместе с дистрибутивом платформы.

3.2.1 Создание аккаунта узла (ноды)

Создайте аккаунты для каждого узла вашей будущей сети.

Аккаунт узла включает в себя адрес, ключевую пару и сертификат ключа проверки ЭП. Запрос на сертификат создается с использованием утилиты **GeneratePkiKeypair**, которая входит в пакет **generators**. Этот пакет входит в состав Платформы «Конфидент» и поставляется вместе с ней в файле generator-1.9.0.jar. В процессе создания запроса на сертификат генерируется адрес узла и ключевая пара (ключ проверки ЭП и ключ ЭП узла).

Важно: Исключена возможность генерации ключевых пар по удаленному подключению к узлу.

Примечание

Для генерации аккаунта узла необходимо наличие графического интерфейса в операционной системе и предварительно установленных компонент ядра, реализующего криптографические алгоритмы и протоколы (описано в документации, которая поставляется с дистрибутивом платформы).
Ключи ЭП Платформы «Конфидент» хранятся в ключевом контейнере, формат которого определяется входящим в её состав ядром, реализующим криптографические алгоритмы и протоколы (описано в документации, которая поставляется с дистрибутивом платформы).

Чтобы создать аккаунт узла, запустите утилиту GeneratePkiKeypair как описано ниже в разделе «GeneratePkiKeypair».
Утилита выполняет следующие действия:

- отображает в командной строке адрес и ключ проверки ЭП узла;
- записывает ключ ЭП узла в ключевой контейнер в хранилище ключей;
    - при создании пары ключей необходимо задать пароль для защиты ключевой пары узла:
        - при работе на ОС Ubuntu пароль запрашивается через консоль;

      В дальнейшем вы можете использовать этот пароль в ручном режиме при каждом старте вашего узла, либо задать глобальные переменные для запроса пароля при старте системы.
- выгружает запрос на сертификат в указанную директорию; запрос на сертификат содержит ранее сгенерированный ключ проверки ЭП и информацию, полученную из конфигурационного файла; в дальнейшем этот запрос необходимо самостоятельно отправить в удостоверяющий центр (УЦ) для получения сертификата;
- выводит лог, в конце которого отображается имя контейнера, в котором хранится ключ ЭП, и алиас (блокчейн адрес) узла.

После того как вы отправили в УЦ запрос и получили из УЦ сертификат, его необходимо вручную импортировать в контейнер с ключевой парой в хранилище ключей, как описано ниже в разделе «Импорт сертификата».

Примечание: Утилита GeneratePkiKeypair также используется для создания ключей для создания канала связи по протоколу TLS. За один запуск генератор создаёт одну ключевую пару.

3.2.1.1 GeneratePkiKeypair

Используйте утилиту GeneratePkiKeypair для создания ключевой пары узла и подписанного этой ключевой парой запроса на сертификат (CSR – Certificate Signing Request) стандарта X.509.

Создание запроса на сертификат выполняется с использованием ядра, реализующее криптографические алгоритмы и протоколы, входящего в состав Платформы «Конфидент», и описанного в документации, которая поставляется с дистрибутивом платформы.

Ключи ЭП Платформы «Конфидент» хранятся в ключевом контейнере, формат которого определяется входящим в её состав ядром, реализующим криптографические алгоритмы и протоколы (описано в документации, которая поставляется с дистрибутивом платформы).

Утилита GeneratePkiKeypair заполняет поля в созданном запросе на сертификат в соответствии с конфигурационным файлом, который указывается во входных параметрах утилиты.

Примечание: Утилита GeneratePkiKeypair также используется при генерации ключей для создания канала связи по протоколу TLS. За один запуск генератор создаёт одну ключевую пару.

Конфигурационный файл для GeneratePkiKeypair

До запуска генератора необходимо подготовить конфигурационный файл в формате HOCON `pki-keypair-generator.conf` с данными для запроса на сертификат. Ниже приведён пример этого файла:

```
pki-key-pair-generator {
    crypto-type = gost
    chain-id = T
    key-pair-algorithm = "GOST_EL_2012_256" # or GOST_DH_2012_256
    keystore-password = "avada-kedavra"
    out-dir = "/Users/dummy/cert_requests" # optional: execution dir if not set
    cert-request-content {
        CN = "Node-0"
        O = "Web3Tech"
        OU = "IT Business"
        C = "RU"
        S = "Moscow"
        L = "Moscow"
        extensions {
            key-usage = ["digitalSignature"]
            extended-key-usage = ["clientAuth"] # or ["serverAuth"]
            subject-alternative-name = "DNS:welocal.dev,DNS:localhost,IP:51.210.211.61,IP:127.0.0.1"
        }
    }
}
```

Заполните поля конфигурационного файла:

- `crypto-type` — режим работы; поддерживается только значение `gost` – поддержка алгоритмов ГОСТ криптографии;
- `chain-id` — идентифицирующий байт сети; допустимо передать только один символ без кавычек либо в двойных кавычках: латинскую букву (строчную или прописную), либо цифру; значение должно совпадать с параметром `blockchain.custom.address-scheme-character` из конфигурационного файла узла;
- `key-pair-algorithm` — алгоритм генерации ключевой пары; допустимые значения:
  - `GOST_EL_2012_256` — для ключей ЭП (ключей узла); в этом случае в поле `extensions.key-usage` должно быть указано значение `digitalSignature`;
  - `GOST_DH_2012_256` — для ключей обмена для создания канала связи по протоколу TLS; в этом случае в поле `extensions.key-usage` должно быть указано значение `keyEncipherment` или `dataEncipherment`;
- `keystore-password` — пароль хранилища ключей (keystore);

  Примечание: Этот же пароль потребуется в дальнейшем:
  – пароль должен быть указан в конфигурационном файле узла в параметре `wallet.password`;
  – при использовании PKI этот же пароль используется при подписании genesis-блока, когда в консоли выводится запрос ввода пароля хранилища ключей (сообщение «`enter keystore password`»);

- `out-dir` — директория, в которую будет выгружен запрос на сертификат; опциональное поле; если значение не указано, то запрос выгружается в директорию, из которой запускался генератор;
- `cert-request-content` — поля с информацией для создания запроса на сертификат:
  - `CN` — общее имя узла;
  - `O` — организация;
  - `OU` — подразделение;
  - `C` — страна;
  - `S` — штат или провинция;
  - `L` — наименование населенного пункта;
  - `extensions`
    - `key-usage` — допускаются следующие значения, указывающие на назначение ключа:
      - `digitalSignature` — укажите это значение для ключей ЭП (ключей узла);
      - `keyEncipherment` — укажите это значение при генерации ключей обмена для создания канала связи по протоколу TLS;
      - `dataEncipherment` — укажите это значение при генерации ключей обмена для создания канала связи по протоколу для TLS.
    - `extended-key-usage` — массив дополнительных значений типа enum, указывающих назначение ключа; опциональное поле; допустимые значения:
      - `ServerAuth`,
      - `ClientAuth`,
      - `CodeSigning`,
      - `EmailProtection`.
    - `subject-alternative-name` — альтернативное имя субъекта; в блокчейн-сети используется для задания дополнительных имен хостов в рамках протокола TLS; опциональное поле.

Запуск GeneratePkiKeypair

Скопируйте в директорию узла файл **generators.jar**. После этого утилиту GeneratePkiKeypair можно запустить из командной строки. В качестве аргумента утилите необходимо передать один параметр: путь до файла с конфигурацией:

```
java -cp "generator-1.9.0.jar:gostCrypto-5.0.42119-A/*"
com.wavesenterprise.generator.pki.PkiKeyPairGenerator pki-keypair-generator.conf
```

В процессе работы генератор запрашивает пароль ключевой пары; пример сообщения: "`Enter key entry '3MrfnwhPPmvJp5B4hiwUwqtSJBJGs9DuxWe' new password:`". Введите и подтвердите пароль. Этот же пароль потребуется в дальнейшем:

– при использовании PKI – при подписании genesis-блока, когда в консоли выводится запрос ввода пароля ключевой пары (сообщение «`enter` **`keypair`** `password`»);

– в файле `env` в поле `WE_NODE_OWNER_PASSWORD` узла необходимо указать этот же пароль:

```
WE_NODE_OWNER_PASSWORD=console # пароль из enter keypair password
WE_NODE_OWNER_PASSWORD_EMPTY=false
```

В процессе работы утилита выводит сообщения об успешном создании ключевой пары и запроса на сертификат, например:

```
2022-03-28 11:37:56,369 INFO [ioapp-compute-0] c.w.g.p.PkiKeyPairGenerator$ - Key pair successfully saved
2022-03-28 11:37:56,370 INFO [ioapp-compute-0] c.w.g.p.PkiKeyPairGenerator$ - Generating certificate request
2022-03-28 11:37:56,375 INFO [ioapp-compute-0] c.w.g.p.PkiKeyPairGenerator$ - Certificate request generated
2022-03-28 11:37:56,380 INFO [ioapp-compute-0] c.w.g.p.PkiKeyPairGenerator$ - Certificate request
successfully exported to
   '/Users/username/Documents/VSCodeProjects/web3techru/generator/jars/requests/3NAjk7wnh6VfE6esY9s94FGE1Z5QFDvPB3S.req'
2022-03-28 11:37:56,383 INFO [ioapp-compute-0] c.w.g.p.PkiKeyPairGenerator$ - Generated keypair:
Public key: 5GZrzce48Jv48Lq8Hn8PTqFDA1Yg2wWkAww1VXSonATfSub4mwo3YPymvqjkKYszxj4f794GqySz4deAZayroNnA
Alias (address for chain-id 'T'): 3NAjk7wnh6VfE6esY9s94FGE1Z5QFDvPB3S
```

Результат работы GeneratePkiKeypair

В результате работы генератор GeneratePkiKeypair

– выводит в командной строке
  – блокчейн адрес (алиас) узла,
  –ключ проверки ЭП узла в кодировке Base58,
  – имя контейнера (в конце лога генератора); оно потребуется для дальнейшей настройки.
– записывает ключ ЭП узла в ключевой контейнер в хранилище ключей по адресу /var/opt/cprocsp/keys/root/{username};
– выгружает запрос на сертификат (CSR) в указанную в конфигурационном файле `pki-keypair-generator.conf` директорию; запрос на сертификат содержит ранее сгенерированный ключ проверки ЭП и информацию, полученную из конфигурационного файла.

3.2.1.1 Удостоверяющий центр

Формирование и управление сертификатами ключей проверки ЭП производится УЦ.

В рамках одной блокчейн-сети используется один Удостоверяющий центр. Ключи проверки ЭП в виде запросов на сертификат направляются в УЦ. Полученные в УЦ сертификаты (в том числе корневой сертификат УЦ) ключей проверки ЭП помещаются в доверенные хранилища на СВТ, на которых развернут узел. Действия с ключами фиксируется в журналах, которые ведет администратор.

Корневой сертификат УЦ должен передаваться только доверенным образом, исключающим его подмену в процессе доставки из УЦ на СВТ, на которых развернут узел.

Установка сертификатов открытых ключей (ключей проверки ЭП) описана в разделе «Получение и импорт сертификата».

3.2.1.2 Получение и импорт сертификата

Для работы с удостоверяющим центром (далее – УЦ) необходимо получить его корневой сертификат и установить этот корневой сертификат на Платформу «Конфидент»:

1. Получите корневой сертификат УЦ доверенным способом.
2. Поместите корневой сертификат УЦ в хранилище сертификатов CAcerts.
3. Помимо этого, для создания genesis-блока необходимо добавить корневой сертификат в хранилище доверенных сертификатов JVM, как описано ниже в разделе «Подготовка к запуску генератора GenesisBlockGenerator».

После этого узел будет считать достоверными сертификаты, подписанные этим УЦ.

Далее необходимо передать запрос на сертификат, полученный в результате работы утилиты GeneratePkiKeypair, в УЦ, получить из УЦ сертификат и импортировать его в контейнер с соответствующей парой ключей. Для этого выполните следующие шаги:

1. Конвертируйте запрос на сертификат, полученный в результате работы генератора, в формат base64. Ниже приведён пример команды конвертации:

```
base64 ./jars/requests/3N5YTeLzpH18qrGRTVdL11DPkRPnM32aNVH.req
```

2. Отправьте запрос на сертификат в формате base64 в УЦ и получите выпущенный сертификат.
3. Конвертируйте этот сертификат в формат base64.
4. Импортируйте сертификат в контейнер с соответствующей парой ключей в хранилище ключей. Ниже приведён пример команды импорта:

```
/opt/cprocsp/bin/cryptcp -instcert -cont container_name cert.crt
```

   В этом примере
   `container_name` – имя контейнера, которое указано в конце лога генератора,
   `cert.cer` – полученный из УЦ файл сертификата.
5. Помимо этого, необходимо добавить сертификат в секцию `node.blockchain.custom.genesis.pki` конфигурационного файла узла для создания genesis-блока как описано ниже в разделе «Подготовка к запуску генератора GenesisBlockGenerator».

3.2.2 Настройка блокчейн-платформы Конфидент для работы в частной сети

Для конфигурации Платформы «Конфидент» используется файл `node.conf` – конфигурационный файл узла, определяющий принципы работы узла и набор опций.

Ниже приведено пошаговое руководство по ручной конфигурации отдельного узла для работы в частной сети. Если в вашей сети развернуто несколько узлов, для каждого из них требуется выполнить аналогичные шаги по конфигурации.

Шаг 1. Общая настройка Платформы «Конфидент»

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

Общая настройка Платформы «Конфидент» описана ниже в соответствующих разделах:

- «Общая настройка Платформы «Конфидент»: настройка режима работы»;
- «Общая настройка Платформы «Конфидент»: настройка консенсуса»;
- «Общая настройка Платформы «Конфидент»: настройка исполнения смарт-контрактов»;
- «Общая настройка Платформы «Конфидент»: настройка майнинга».

Шаг 2. Тонкая настройка Платформы «Конфидент»

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

Все необходимые параметры также располагаются в файле конфигурации узла **node.conf**.

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

- «Тонкая настройка Платформы «Конфидент»: настройка авторизации для gRPC и REST API»;
- «Тонкая настройка Платформы «Конфидент»: настройка инструментов gRPC и REST API»;
- «Тонкая настройка Платформы «Конфидент»: настройка TLS»;
- «Тонкая настройка Платформы «Конфидент»: настройка групп доступа к конфиденциальным данным»;
- «Тонкая настройка Платформы «Конфидент»: настройка механизма создания снимка данных»;
- «Тонкая настройка Платформы «Конфидент»: настройка узла (ноды) в режиме наблюдения».

Пример полного конфигурационного файла для настройки каждого узла приведен в разделе «Пример конфигурационного файла узла».

3.2.2.1 Общая настройка Платформы «Конфидент»: настройка режима работы

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

Детальная настройка криптографии описана в документации, которая поставляется вместе с дистрибутивом платформы.

3.2.2.2 Общая настройка Платформы «Конфидент»: настройка консенсуса

Платформа «Конфидент» поддерживает три алгоритма консенсуса: **PoS**, **PoA** и **CFT**.

Важно: Алгоритмы консенсуса **PoA** и **PoS** доступны только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы «Конфидент»: настройка режима работы» выше.

Настройки консенсуса располагаются в блоке `consensus` секции `blockchain`:

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

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

`type = "pos"` или закомментированный блок `consensus`

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

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

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

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

`type = "poa"`

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

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

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

type = "cft"

Основные параметры настройки алгоритма консенсуса **CFT** идентичны параметрам алгоритма консенсуса PoA:

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

По сравнению с PoA, для CFT предусмотрены следующие дополнительные параметры конфигурации, необходимых для валидации блоков в ходе раунда голосования:

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

При настройке CFT обратите внимание на следующие рекомендации:

- Параметр `sync-duration` должен быть отличен от нуля. Рекомендуется устанавливать значение от 1 до 5 секунд в зависимости от размера и сложности транзакций.
- Примерный расчет значения параметра `finalization-timeout`:

  (`round-duration` + `sync-duration`) / 2.

  Не рекомендуется занижать это значение для ускорения финализации: если майнер наберет необходимое число голосов ранее окончания этого времени, он сразу выпустит финализирующий микроблок.
- Если в сети присутствует большое количество майнеров, ограничьте количество валидаторов раунда параметром `max-validators`. Механизм выбора валидаторов обеспечит равномерную ротацию всех валидаторов по раундам. Слишком большое количество валидаторов может отрицательно повлиять на производительность сети. Рекомендуемый диапазон значений: от 5 до 10.
- Если сеть работает под постоянной нагрузкой, установите параметр `full-vote-set-timeout`. До истечения указанного периода времени майнер ждет полного набора голосов от валидаторов. Если валидатор сталкивается с какими-либо неполадками, сеть использует время `full-vote-set-timeout` для создания дополнительного временного промежутка, который позволяет отставшему валидатору завершить синхронизацию. Рекомендуемое значение: `sync-duration` * 2, не может превышать `sync-duration` + `finalization-timeout`.

3.2.2.3 Общая настройка Платформы «Конфидент»: настройка исполнения смарт-контрактов

Общее описание принципов работы смарт-контрактов представлено в разделе «Разработка и применение смарт-контрактов» документа «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы. Для работы со смарт-контрактами узел использует два типа соединения, для каждого из которых необходимо обеспечить защиту канала:

1. Соединение с docker-хостом – удалённой машиной, на которой запускаются смарт-контракты. На этой машине используется docker-библиотека, которая обращается на сокет по своим протоколам. Для неё можно включить опцию безопасного соединения, которое в этой документации обозначается как «docker-TLS». Соединение docker-TLS настраивается в секции `node.docker-engine.docker-tls` конфигурационного файла узла; эта настройка описана ниже в этом разделе;
2. Соединение, которое открывает запущенный смарт-контракт в сторону узла по протоколу gRPC. Допустимо использовать только методы из «Перечня функций gRPC интерфейса Платформы «Конфидент», доступных только смарт-контрактам», приведенного в документе «643.19172778.2019662198-01 95. «Блокчейн-платформа Конфидент» Версия 1.9. Правила пользования», который поставляется вместе с дистрибутивом платформы. Это подключение по API, так как точка подключения смарт-контракта к узлу такая же, как и для любого другого пользователя или приложения. Этот API настраивается в секции `node.api.grpc`, в частности для него необходимо обеспечить защиту канала TLS; эта настройка описана в разделе «Тонкая настройка платформы «Конфидент»: настройка TLS» ниже. Также пример такой настройки дан в разделе «Пример конфигурационного файла узла».

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

```
docker-engine {
     enable = yes
     integration-tests-mode-enable = no
     # docker-host = "unix:///var/run/docker.sock"
     execution-limits {
        startup-timeout = 10s
        timeout = 10s
        memory = 512
        memory-swap = 0
      }
     reuse-containers = yes
     remove-container-after = 10m
     allow-net-access = yes
     remote-registries = [
       {
        domain = "myregistry.com:5000"
        username = "user"
        password = "password"
       }
     ]
     check-registry-auth-on-startup = no
     # default-registry-domain = "registry.yourdomain.com"
     contract-execution-messages-cache {
        expire-after = 60m
        max-buffer-size = 10
        max-buffer-time = 100ms
        utx-cleanup-interval = 1m
        contract-error-quorum = 2
     }
     contract-auth-expires-in = 1m
```

```
      grpc-server {
        # host = "192.168.97.3"
        port = 6865
      }
      remove-container-on-fail = yes
      docker-tls {
        tls-verify = yes
        cert-path = "/node/certificates"
       }
      contracts-parallelism = 8
}
```

- `enable` – включение обработки транзакций для Docker-контрактов.
- `integration-tests-mode-enable` – режим тестирования Docker-контрактов. При включении этой опции смарт-контракты исполняются локально в контейнере.
- `docker-host` – адрес демона docker (опционально). Если это поле закомментировано, адрес демона для исполнения смарт-контрактов будет взят из системного окружения.
- `startup-timeout` – время, отводимое на создание контейнера контракта и его регистрацию в узле (в секундах).
- `timeout` – время, отводимое на выполнение контракта (в секундах).
- `memory` – ограничение по памяти для контейнера контракта (в мегабайтах).
- `memory-swap` – выделяемый объем виртуальной памяти для контейнера контракта (в мегабайтах).
- `reuse-containers` – использование одного контейнера для нескольких контрактов, использующих один и тот же Docker-образ. Включение опции – `yes`, отключение – `no`.
- `remove-container-after` – промежуток времени бездействия контейнера, по прошествии которого он будет удален.
- `allow-net-access` – разрешение доступа к сети.
- `remote-registries` – адреса Docker-репозиториев и настройки авторизации к ним.
- `check-registry-auth-on-startup` – проверка авторизации для Docker-репозиториев при запуске узла. Включение опции – `yes`, отключение – `no`.
- `default-registry-domain` – адрес Docker-репозитория по умолчанию (опционально). Этот параметр используется, если в имени образа контракта не указан репозиторий.
- `contract-execution-messages-cache` – секция настроек кэша со статусами исполнения транзакций по docker контрактам;
- `expire-after` – время хранения статуса смарт-контракта.
- `max-buffer-size` и `max-buffer-time` – настройки объема и времени хранения кэша статусов.
- `utx-cleanup-interval` – интервал, по прошествии которого невалидные транзакции (со статусом Error) удаляются из UTX-пула узла, который не является майнером. Значение по умолчанию – 1m.
- `contract-error-quorum` – минимальное количество полученных от разных узлов-майнеров сообщений, в которых статус транзакции по вызову смарт-контракта содержит бизнес-ошибку (Error); когда указанное в параметре количество сообщений получено, транзакция удаляется из UTX-пула узла, который не является майнером. Значение по умолчанию – 2.
- `contract-auth-expires-in` – время жизни токена авторизации, используемого смарт-контрактами для вызовов к узлу.
- `grpc-server` – секция настроек gRPC сервера для работы Docker-контрактов с gRPC API.
- `host` – сетевой адрес узла (опционально).
- `port` – порт gRPC-сервера. Укажите порт прослушивания gRPC-запросов, использующийся Платформой «Конфидент».
- `remove-container-on-fail` – удаление контейнера, если при его старте произошла ошибка. Включение опции – `yes`, отключение – `no`.

- `tls-verify` – флаг включения или отключения канала связи по протоколу TLS; если установлено значение `yes`, то выполняется поиск сертификатов в директории, указанной в параметре `certs-path`; если указано значение `no`, то поиск сертификатов не выполняется.

  Важно: При использовании алгоритмов ГОСТ криптографии взаимодействие должно осуществляться по каналу связи по протоколу TLS, то есть параметр `tls-verify` должен иметь значение `yes`.

- `certs-path` – путь до директории с сертификатами для TLS; по умолчанию параметр имеет значение {node.directory}/certificates.
- `contracts-parallelism` – параметр определяет количество параллельно выполняемых транзакций всех контейнеризированных смарт-контрактов. По умолчанию параметр имеет значение «8».

3.2.2.4 Общая настройка Платформы «Конфидент»: настройка майнинга

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

```
miner {
        enable = yes
        quorum = 2
        interval-after-last-block-then-generation-is-allowed = 10d
        no-quorum-mining-delay = 5s
        micro-block-interval = 5s
        min-micro-block-age = 3s
        max-transactions-in-micro-block = 500
        min-micro-block-age = 6 s
        pulling-buffer-size = 100
        utx-check-delay = 1s
}
```

- `enable` – активация опции майнинга. Включение – `yes`, отключение – `no`.
- `quorum` – необходимое количество узлов-майнеров для создания блока. Значение 0 позволит генерировать блоки оффлайн и используется только в тестовых целях в сетях с одним узлом. При указании этого значения необходимо учитывать, что собственный узел-майнер не суммируется со значением этого параметра, т. е. если вы указываете `quorum = 2`, то для майнинга нужно минимум 3 узла-майнера.
- `interval-after-last-block-then-generation-is-allowed` – создание блока только в том случае, если последний блок не старше указанного периода времени (в днях).
- `micro-block-interval` – интервал между микроблоками (в секундах).
- `min-micro-block-age` – минимальный возраст микроблока (в секундах).
- `max-transactions-in-micro-block` – максимальное количество транзакций в микроблоке.
- `pulling-buffer-size` – размер буфера транзакций. Чем выше значение параметра, тем дольше группируются транзакции.
- `utx-check-delay` – задержка проверки UTX-пула (есть ли в пуле транзакции или он пуст) майнером. По умолчанию используется значение 1 с. Значение параметра должно быть больше либо равно 100 мс.

Настройки майнинга зависят от планируемого в вашей сети размера транзакций.

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

- `micro-block-interval` – интервал между микроблоками. Значение указывается в секундах.
- `min-micro-block-age` – минимальный возраст микроблока. Значение указывается в секундах и не должно превышать значение параметра `micro-block-interval`.

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

3.2.2.5 Тонкая настройка Платформы «Конфидент»: настройка авторизации для gRPC и REST API

Авторизация необходима для обеспечения доступа к gRPC и REST API инструментам узла.

Для настройки авторизации предназначена секция `auth` конфигурационного файла узла.

Платформа «Конфидент» поддерживает авторизацию по `tls-whitelist`: необходимо, чтобы открытый ключ в клиентском TLS сертификате был равен одному из открытых ключей администраторов, перечисленных в разделе `node.api.auth` конфигурационного файла узла.

Авторизация обязательна для клиента блокчейн. Авторизация по `tls-whitelist` используется в узле по умолчанию.

Ниже приведён пример раздела секции `auth` конфигурационного файла узла:

```
auth {
     type = "tls-whitelist"

     # Public keys are expected in Base64 format
     admin-public-keys = [

           "MGYwHwYIKoUDBwEBAQEwEwYHKoUDAgIkAAYIKoUDBwEBAgIDQwAEQLh7lrv/ioWdnUkvX3NybRoeew1PPz1vMaa
           jxzpi1CYoWRlrPC9RjlV ul5PCMyAL2OuO4lgpcqBj1+y2cEjajXw="
     ]
 }
```

3.2.2.6 Тонкая настройка Платформы «Конфидент»: настройка инструментов gRPC и REST API узла

Параметры работы gRPC и REST API для каждого узла находятся в секции `api` конфигурационного файла:

```
     api {
       rest {
         # Enable/disable REST API
         enable = yes
         # Network address to bind to
         bind-address = "0.0.0.0"
         # Port to listen to REST API requests
         port = 6862
         # Enable/disable TLS for REST
         tls = yes
         # 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 = yes
         # Parameters for internal gRPC services. Recommended to be left as is.
```

```
        services {
         blockchain-events {
            max-connections = 5
            history-events-buffer {
                enable: false
                size-in-bytes: 50MB
            }
         }
        privacy-events {
            max-connections = 5
            history-events-buffer {
                enable: false
                size-in-bytes: 50MB
            }
        }
        contract-status-events {
            max-connections = 5
        }
    }
}
```

Блок `rest { }`

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

- `enable` – активация опции REST API на узле. Включение опции – «`yes`», отключение – «`no`».
- `bind-address` – сетевой адрес узла, на котором будет доступен REST API интерфейс.
- `port` – порт прослушивания REST API запросов.
- `tls` – включение/отключение TLS для REST API запросов. Включение – «`yes`», отключение – «`no`». Для включения требуется настройка TLS узла, описанная ниже в разделе «Тонкая настройка платформы «Конфидент»: настройка TLS».

  Важно: параметр по умолчанию имеет значение «`yes`», то есть TLS для REST API запросов включен. Использование значения «no» допускается только в тестовых целях.
- `cors` – поддержка кросс-доменных запросов к REST API. Включение опции – `yes`, отключение – `no`.
- `transactions-by-address-limit` – максимальное количество транзакций, возвращаемых методом `GET /transactions/address/{address}/limit/{limit}`. Подробное описание метода см. в разделе «Методы REST API» документа «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы.
- `distribution-address-limit` – максимальное количество адресов, указываемых в поле `limit` и возвращаемых методом `GET /assets/{assetId}/distribution/{height}/limit/{limit}`. Подробное описание метода см. в разделе «Методы REST API» документа «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика» , который поставляется вместе с дистрибутивом платформы.

Блок `grpc { }`

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

- `enable` – активация gRPC-интерфейса на узле.
- `bind-address` – сетевой адрес узла, на котором будет доступен gRPC-интерфейс.
- `port` – порт прослушивания gRPC запросов.
- `tls` – включение/отключение TLS для gRPC запросов. Включение – `yes`, отключение – `no`. Для включения требуется настройка TLS узла, описанная ниже в разделе «Тонкая настройка платформы «Конфидент»: настройка TLS».

  Важно: Параметр по умолчанию имеет значение `yes`, то есть TLS для gRPC запросы осуществляются по каналу связи по протоколу TLS. Использование значения `no` допускается только в тестовых целях.

Секция `services{ }` содержит настройки методов gRPC-интерфейса, собирающих данные из компонентов платформы (перечень сервисов см. в Таблице 1 документа «643.19172778.2019662198-01 95. «Блокчейн-платформа Конфидент» Версия 1.9. Правила пользования», который поставляется вместе с дистрибутивом платформы:

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

В этой секции рекомендуется использовать предустановленные параметры.

3.2.2.7 Тонкая настройка Платформы «Конфидент»: настройка TLS

Для работы со смарт-контрактами узел использует два типа соединения, для каждого из которых необходимо настроить защищенный канал связи по протоколу TLS: docker-TLS и подключение по API. Подробнее см. раздел «Общая настройка платформы «Конфидент»: настройка исполнения смарт-контрактов» выше.

Настроить защищенный канал связи по протоколу TLS для gRPC и REST API для каждого узла можно с помощью параметров работы gRPC и REST API в секции `api` конфигурационного файла узла. Для настройки защищенного канала связи по протоколу TLS используйте параметр `TLS` в блоке `rest` и в блоке `grpc`, которые описаны выше в разделе «Тонкая настройка платформы «Конфидент»: настройка инструментов gRPC и REST API узла».

Настройки, необходимые для работы с API по защищенному каналу связи по протоколу TLS, описаны в документации, которая поставляется вместе с дистрибутивом платформы.

3.2.2.7.1 Пример подготовки артефактов для TLS

Поскольку канал связи по протоколу TLS между узлами и клиентом и узлом является обязательным, то в рамках настройки инфраструктуры необходимо настроить параметры связи по протоколу TLS.

Для использования канала связи по протоколу TLS для API необходимо для каждого узла выполнить следующие шаги:

1. Создать клиентские ключевую пару и запрос на сертификат с помощью генератора GeneratePkiKeypair. Для этого необходимо:

   1.1 Подготовить конфигурационный файл утилиты GeneratePkiKeypair:

   - полю `extended-key-usage` задать значение `["clientAuth"];`
   - полю `key-usage` задать значение `["KeyEncipherment", "DataEncipherment"];`
   - в полях `CN` и `subject-alternative-name` указать соответствующие имена узлов и DNS или IP адрес.

   Структура конфигурационного файла генератора описана выше в разделе GeneratePkiKeypair.

   1.2 Запустить генератор, используя следующую команду:

   ```
   java -cp "generators-1.9.0-M1-38-d823c7f.jar:gostCrypto-5.0.42119-A/*"
   com.wavesenterprise.generator.pki.PkiKeyPairGenerator key_gen.conf
   ```

   где `key_gen.conf` – имя конфигурационного файла утилиты GeneratePkiKeypair.

   1.3 Ввести и подтвердить пароль хранилища ключей, когда генератор отобразит соответствующий запрос.

   1.4 Указать этот же пароль в конфигурационном файле узла в секции `tls` в поле `keystore-password`.

   В результате работы генератор GeneratePkiKeypair выгружает артефакты для построения канала связи по протоколу TLS:

   - запрос на клиентский сертификат (CSR) вида 3MqFWYnVcBndh3Nc8gz9Ar5LEjnFQFjZX5u.req – в указанную в конфигурационном файле генератора директорию;
   - файлы клиентской ключевой пары – в контейнер в хранилище ключей. В конце лога генератора указано имя контейнера; оно потребуется для дальнейшей настройки.

2. Отправить запрос на сертификат в УЦ и получить сертификат (клиентский сертификат).
3. Импортировать сертификат в контейнер с соответствующей парой ключей в хранилище ключей с помощью следующей команды:

   ```
   /opt/cprocsp/bin/cryptcp -instcert -cont client_container_name client_cert.crt
   ```

   `client_containter_name` – имя контейнера, которое указано в конце лога генератора,
   `client_cert.crt` – полученный из УЦ файл клиентского сертификата.

4. Создать серверные ключевую пару и запрос на сертификат с помощью генератора GeneratePkiKeypair аналогично тому, как создавались клиентские ключевая пара и запрос на сертификат:

   4.1 Подготовить конфигурационный файл утилиты GeneratePkiKeypair:

   - полю `extended-key-usage` задать значение `["serverAuth"];`

   В полях `CN` и `subject-alternative-name` уже указаны соответствующие имена узлов и DNS или IP адрес.

   Структура конфигурационного файла генератора описана выше в разделе GeneratePkiKeypair.

   4.2 Запустить генератор, используя следующую команду:

   ```
   java -cp "generators-1.9.0-M1-38-d823c7f.jar:gostCrypto-5.0.42119-A/*"
   com.wavesenterprise.generator.pki.PkiKeyPairGenerator key_gen.conf
   ```

   где `key_gen.conf` – имя конфигурационного файла утилиты GeneratePkiKeypair.

   4.3 Ввести и подтвердить тот же пароль хранилища ключей, что и для клиентской ключевой пары, когда генератор отобразит соответствующий запрос.

   В результате работы генератор GeneratePkiKeypair выгружает артефакты для TLS:
   - запрос на серверный сертификат (CSR) – в указанную в конфигурационном файле генератора директорию;
   - файлы серверной ключевой пары – в контейнер в хранилище ключей. В конце лога генератора указано имя контейнера; оно потребуется для дальнейшей настройки.

5. Отправить запрос на сертификат в УЦ и получить сертификат (серверный сертификат).

6. Импортировать серверный сертификат в контейнер с серверной парой ключей в хранилище ключей с помощью следующей команды:

   ```
   /opt/cprocsp/bin/cryptcp -instcert -cont server_container_name server_cert.crt
   ```

   `server_containter_name` – имя контейнера, которое указано в конце лога генератора,
   `server_cert_file` – полученный из УЦ файл сертификата.

7. Сформировать хранилище сертификатов TrustStore, то есть получить файл CertStore, и поместить в него корневой сертификат УЦ, клиентский и серверный сертификаты. Для этого используется утилита **keytool**.
   При первом запуске утилита создаёт новый TrustStore и записывает в него переданный ей сертификат. При последующих запусках утилита добавляет передаваемый ей сертификат в существующий TrustStore. За один запуск утилита добавляет один сертификат.
   Ниже представлен пример использования утилиты **keytool**.

   7.1 Создание хранилища сертификатов TrustStore и запись в него корневого сертификата УЦ:

   ```
   keytool -J-Dkeytool.compat=true -import -alias certnew -providername JCSP
   -storetype CertStore -keystore node_0_certstore -storepass
   certstore_password -file CA_certs/certnew
   -providerpath JCSP.jar:JCP.jar:ASN1P.jar:asn1rt.jar:forms_rt.jar:
   -providerclass ru.CryptoPro.JCSP.JCSP
   ```

   `alias` – любое уникальное в рамках хранилища сертификатов имя; может совпадать с именем файла сертификата;
   `file` – имя файла сертификата.

   7.2 Запись в то же хранилище клиентского сертификата:

   ```
   keytool -J-Dkeytool.compat=true -import
   -alias 3MqFWYnVcBndh3Nc8gz9Ar5LEjnFQFjZX5u -providername JCSP
   -storetype CertStore -keystore node_0_certstore
   -storepass certstore_password
   -file tls_certs/3MqFWYnVcBndh3Nc8gz9Ar5LEjnFQFjZX5u.crt
   -providerpath JCSP.jar:JCP.jar:ASN1P.jar:asn1rt.jar:forms_rt.jar:
   -providerclass ru.CryptoPro.JCSP.JCSP
   ```

7.3 Запись в то же хранилище серверного сертификата:

```
keytool -J-Dkeytool.compat=true -import
-alias 2LqFWYnVcBndh3Nc8gz9Ar5LEjnFQFjZX3j -providername JCSP
-storetype CertStore -keystore node_0_certstore
-storepass certstore_password
-file tls_certs/2LqFWYnVcBndh3Nc8gz9Ar5LEjnFQFjZX3j.crt
-providerpath JCSP.jar:JCP.jar:ASN1P.jar:asn1rt.jar:forms_rt.jar:
-providerclass ru.CryptoPro.JCSP.JCSP
```

8 Скопировать контейнеры с ключевыми парами к другим ключам узла в хранилище ключей, расположенное по адресу /var/opt/cprocsp/keys/root/{username} при помощи стандартных утилит Linux – `mv`, `cp` и т. п..

3.2.2.8 Тонкая настройка Платформы «Конфидент»: настройка групп доступа к конфиденциальным данным

Если вы используете API-методы **privacy** для управления конфиденциальными данными, настройте параметры доступа к этим данным в конфигурационном файле узла. Для этого предназначена секция `privacy`.

Методы privacy описаны в разделе «gRPC: работа с конфиденциальными данными» и «REST API: обмен конфиденциальными данными и получение информации о группах доступа» документа «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы.

Важно. API-методы **privacy** допустимо использовать только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы «Конфидент»: настройка режима работы» выше.

Ниже представлен пример настройки с использованием БД PostgreSQL:

```
privacy {
 replier {
   parallelism = 10
   stream-timeout = 1 minute
   stream-chunk-size = 1MiB
 }
 synchronizer {
   request-timeout = 2 minute
   init-retry-delay = 5 seconds
   inventory-stream-timeout = 15 seconds
   inventory-request-delay = 3 seconds
   inventory-timestamp-threshold = 10 minutes
   crawling-parallelism = 100
   max-attempt-count = 24
   lost-data-processing-delay = 10 minutes
   network-stream-buffer-size = 10
 }
 inventory-handler {
   max-buffer-time = 500ms
   max-buffer-size = 100
   max-cache-size = 100000
   expiration-time = 5m
   replier-parallelism = 10
 }
 cache {
   max-size = 100 expire-after = 10m
 }
 storage {
   vendor = postgres
   schema = "public"
   migration-dir = "db/migration"
   profile = "slick.jdbc.PostgresProfile$"
   upload-chunk-size = 1MiB
   jdbc-config {
     url = "jdbc:postgresql://postgres:5432/node-1"
     driver = "org.postgresql.Driver"
     user = postgres
     password = web3techru
     connectionPool = HikariCP
     connectionTimeout = 5000
     connectionTestQuery = "SELECT 1"
     queueSize = 10000
     numThreads = 20
   }
 }
 service {
   request-buffer-size = 10MiB
   meta-data-accumulation-timeout = 3s
 }
}
```

Выбор базы данных

Перед изменением конфигурационного файла узла выберите базу данных, которую планируете использовать для хранения конфиденциальных данных. Платформа «Конфидент» поддерживает взаимодействие с БД PostgreSQL и Amazon S3.

Важно. Выбор базы данных для хранения конфиденциальных данных доступен только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы «Конфидент»: настройка режима работы» выше.

PostgreSQL

Во время установки БД под управлением PostgreSQL вы создадите аккаунт для доступа к БД. Заданные при этом логин и пароль затем необходимо будет указать в конфигурационном файле узла в полях `user` и `password` блока `storage` секции `privacy`. Подробнее см. раздел «vendor = postgres» ниже.

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

Amazon S3

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

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

Блок `storage`

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

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

Важно: Если вы не используете API-методы **privacy** (подробное описание методов см. в документе «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы), в параметре `vendor` укажите значение `none` и закомментируйте или удалите остальные параметры в секции `privacy`.

vendor = postgres

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

```
storage {
  vendor = postgres
  schema = "public"
  migration-dir = "db/migration"
  profile = "slick.jdbc.PostgresProfile$"
  upload-chunk-size = 1MiB
  jdbc-config {
          url = "jdbc:postgresql://postgres:5432/node-1"
          driver = "org.postgresql.Driver"
          user = postgres
          password = web3techru
          connectionPool = HikariCP
          connectionTimeout = 5000
          connectionTestQuery = "SELECT 1"
          queueSize = 10000
          numThreads = 20
  }
 }
```

В блоке должны быть указаны следующие параметры:

- `schema` – используемая схема взаимодействия между элементами в рамках БД; по умолчанию применяется схема `public`; если в вашей БД предусмотрена иная схема, то укажите ее название;
- `migration-dir` – директория для миграции данных;
- `profile` – имя профиля для доступа к JDBC, заданное при установке JDBC (см. раздел «PostgreSQL» выше);
- `upload-chunk-size` – размер фрагмента данных, загружаемых с помощью REST API метода POST/privacy/sendLargeData или gRPC API метода SendLargeData; подробное описание методов см. в документе «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы;
- `url` – адрес БД PostgreSQL; подробнее см. раздел «Поле url» ниже;
- `driver` – имя драйвера JDBC, позволяющим Java-приложениям взаимодействовать с БД;
- `user` – имя пользователя для доступа к БД; укажите логин созданного вами аккаунта для доступа к БД под управлением PostgreSQL;
- `password` – пароль для доступа к БД; укажите пароль созданного вами аккаунта для доступа к БД под управлением PostgreSQL;
- `connectionPool` – имя пула соединений, по умолчанию `HikariCP`;
- `connectionTimeout` – время бездействия соединения до его разрыва (в миллисекундах);
- `connectionTestQuery` – тестовый запрос для проверки соединения с БД; для PostgreSQL рекомендуется отправлять запрос `SELECT 1`;
- `queueSize` – размер очереди запросов;
- `numThreads` – количество одновременных подключений к БД.

Поле `url`

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

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

, где

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

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

```
privacy { storage {
    ...
    url                                                                    =
    "jdbc:postgresql://yourpostgres.com:5432/privacy_node_0?user=user_privacy_node_0
    @company&password=7nZL7Jr41qOWUHz5qKdypA&sslmode=require"
    ...
    }
}
```

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

pgBouncer

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

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

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

vendor = s3

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

```
storage {
  vendor = s3
  url = "http://localhost:9000/"
  bucket = "privacy"
  region = "aws-global"
  access-key-id = "minio"
  secret-access-key = "minio123"
  path-style-access-enabled = true
  connection-timeout = 30s
  connection-acquisition-timeout = 10s
  max-concurrency = 200
  read-timeout = 0s
  upload-chunk-size = 5MiB
}
```

- `url` – адрес сервера Minio для хранения данных; по умолчанию, Minio использует порт 9000;
- `bucket` – имя таблицы БД S3 для хранения данных;
- `region` – название региона S3, значение параметра – `aws-global`;
- `access-key-id` – идентификатор ключа доступа к данным; укажите логин для доступа к данным, который вы задали в процессе установки сервера Minio (см. раздел «Amazon S3» выше);
- `secret-access-key` – ключ доступа к данным в хранилище S3; укажите пароль для доступа к данным, который вы задали в процессе установки сервера Minio (см. раздел «Amazon S3» выше);
- `path-style-access-enabled` = `true` – путь к таблице S3 – неизменяемый параметр;
- `connection-timeout` – период бездействия до разрыва соединения (в секундах);
- `connection-acquisition-timeout` – период бездействия при установлении соединения (в секундах);
- `max-concurrency` – максимальное число параллельных обращений к хранилищу;
- `read-timeout` – период бездействия при чтении данных (в секундах);
- `upload-chunk-size` – размер фрагмента данных, загружаемых с помощью REST API метода POST /privacy/sendLargeData или gRPC API метода SendLargeData; подробное описание методов см. в документе «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы.

Блок `replier`

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

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

В блоке должны быть указаны следующие параметры:

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

Блок `inventory-handler`

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

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

В блоке должны быть указаны следующие параметры:

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

Блок `cache`

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

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

Примечание: Большие файлы (файлы, загружаемые с помощью REST API метода POST/privacy/sendLargeData или gRPC API метода SendLargeData, подробное описанных в документе «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы) не подлежат кешированию.

В блоке должны быть указаны следующие параметры кэша:

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

Блок `synchronizer`

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

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

В блоке должны быть указаны следующие параметры:

- `request-timeout` – максимальное время ожидания ответа после запроса данных; значение по умолчанию – `2 minute`;
- `init-retry-delay` – пауза после неудачной попытки; с каждой попыткой задержка увеличивается на 4/3; значение по умолчанию – `5 seconds`;
- `inventory-stream-timeout` – максимальное время ожидания сетевого сообщения с инвентаризационной информацией (privacy inventory), т.е. подтверждения от конкретного узла, что у него есть определенные данные, и он может их предоставить для загрузки. По истечении этого таймаута узел опрашивает всех пиров (рассылает inventory-request), есть ли у них необходимые для загрузки данные; значение по умолчанию – `15 seconds`;
- `inventory-request-delay` – задержка после запроса инвентарных данных у пиров (inventory-request); значение по умолчанию – `3 seconds`;
- `inventory-timestamp-threshold` – параметр используется для принятия решения, отправлять ли PrivacyInventory сообщение при успешной синхронизации (загрузке) данных; значение по умолчанию – `10 minutes`;
- `crawling-parallelism` – максимальное количество параллельно выполняемых задач краулера – компонента, который собирает конфиденциальные данные у пиров; значение по умолчанию – `100`;
- `max-attempt-count` – количество попыток, которые предпримет краулер, прежде чем данные будут помечены как потерянные; значение по умолчанию – `24`;
- `lost-data-processing-delay` – задержка между попытками обработки очереди потерянных данных; значение по умолчанию – `10 minutes`;
- `network-stream-buffer-size` – максимальное количество фрагментов данных в буфере; когда указанное количество достигнуто, активируется обратное давление; значение по умолчанию – `10`.

Поле `inventory-timestamp-threshold`

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

Блок `service`

В блоке `service` секции `privacy` укажите параметры gRPC метода SendLargeData и REST метода POST /privacy/sendLargeData для отправки потока конфиденциальных данных. Подробное описание методов см. в документе «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы.

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

В блоке должны быть указаны следующие параметры:

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

3.2.2.9 Тонкая настройка Платформы «Конфидент»: настройка анкоринга

Важно: Анкоринг поддерживается только в тестовом режиме функционирования Платформы «Конфидент», то есть, когда в конфигурационном файле узла параметру `node.crypto.pki.mode` присвоено значение `TEST`. Подробнее об этом параметре см. раздел «Общая настройка платформы «Конфидент»: настройка режима работы» выше.

В приватном блокчейне транзакции обрабатываются определенным списком участников, каждый из которых заранее известен. Малое, по сравнению с публичной сетью, количество участников, блоков и транзакций в приватном блокчейне несёт угрозу подмены информации. Что, в свою очередь, создает риск перезаписи цепочки блоков - особенно в случае использования алгоритма консенсуса PoS, не имеющего защиты от таких ситуаций.

Для повышения доверия участников приватного блокчейна к размещенным в нём данным разработан механизм анкоринга. Анкоринг позволяет проверить данные на неизменность. Гарантия неизменности достигается публикацией данных из приватного блокчейна в более крупную сеть `Targetnet`, где подмена данных менее вероятна из-за большего количества участников и блоков. Из приватной сети публикуются подписи блоков и высота блокчейна. Взаимная связность двух и более сетей повышает их устойчивость, поскольку для подлога или изменения данных в результате long-range атаки необходимо атаковать все связанные сети.

Если вы планируете использовать анкоринг данных из вашей сети в более крупную сеть, настройте параметры передачи данных в блоке `anchoring` конфигурационного файла узла. В терминологии конфигурационного файла, `targetnet` – это блокчейн, в который ваш узел будет выполнять транзакции анкоринга из текущей сети.

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

   targetnet-authorization {
      type = " api-key"
      api-key-hash = "G3PZAsY6EA8esgpKxB2UYTQJZJPzc14gLnNbm2xvcDf6"
      privacy-api-key-hash = "G3PZAsY6EA8esgpKxB2UYTQJZJPzc14gLnNbm2xvcDf6"
   }
   targetnet-scheme-byte = "V"
   targetnet-node-address = "https:// node.targetnet.com:6862/NodeAddress" targetnet-node-
   recipient-address = ""
   targetnet-private-key-password = ""
   wallet {
      file = "node-1_targetnet-wallet.dat"
      password = "small"
   }
   targetnet-fee = 10000000
   sidechain-fee = 5000000
 }
```

Параметры анкоринга

- `enable` – включение или отключение анкоринга (`yes` / `no`);
- `height-range` – интервал блоков, по прошествии которого узел приватного блокчейна отправляет в Targetnet транзакции для анкоринга;
- `height-above` – число блоков в Targetnet, по прошествии которого узел приватного блокчейна создаёт подтверждающую анкоринг транзакцию с данными первой транзакции. Рекомендуется устанавливать значение, не превышающее максимальную величину отката блоков в Targetnet (`max-rollback`);

- `threshold` – число блоков, которое отнимается от текущей высоты приватного блокчейна. В транзакцию для анкоринга, отправляемую в Targetnet, попадёт информация из блока на высоте `current-height` – `threshold`. Если устанавливается значение 0, в транзакцию анкоринга записывается значение блока на текущей высоте блокчейна. Рекомендуется устанавливать значение, близкое к максимальной величине отката в приватном блокчейне (`max-rollback`);
- `tx-mining-check-delay` – время ожидания между проверками доступности транзакции для анкоринга в Targetnet;
- `tx-mining-check-count` – максимальное количество проверок доступности транзакции для анкоринга в Targetnet, по выполнении которых транзакция считается не поступившей в сеть.

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

Параметры авторизации при использовании анкоринга

- `type` – тип авторизации при использовании анкоринга; доступно одно значение:
  - `api-key` – авторизация по api-key-hash;

В случае выбора авторизации по `api-key-hash`, достаточно указать значение ключа в параметре `api-key`.

Параметры для доступа Targetnet

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

- `targetnet-scheme-byte` – байт сети Targetnet;
- `targetnet-node-address` – полный сетевой адрес узла 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`; например: `http://node.web3techservices.com:6862/NodeAddress`;
- `targetnet-node-recipient-address` – адрес узла в сети Targetnet, на который будут записываться транзакции для анкоринга, подписанные ключевой парой данного адреса;
- `targetnet-private-key-password` – пароль от закрытого ключа узла для подписи транзакций анкоринга.

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

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

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

Параметры комиссий

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

Как работает анкоринг на Платформе «Конфидент»

После того как вы настроили анкоринг в конфигурационном файле ноды приватного блокчейна (как описано выше в разделе «Тонкая настройка Платформы «Конфидент»: настройка анкоринга»), выполняются следующие шаги:

1. Узел через заданный интервал блоков `height-range` фиксирует информацию о блоке на высоте `current-height - threshold` в виде транзакции в Targetnet. В качестве такой транзакции используется транзакция 12 Data Transaction со списком пар полей «ключ - значение», описание которых приведено в разделе «Структура транзакции для анкоринга» ниже; описание транзакции 12 приведено в разделе «12 Data Transaction» документа «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы.
2. После отправки транзакции нода получает её высоту в Targetnet.
3. Узел выполняет проверку высоты блокчейна в Targetnet каждые 30 секунд, пока высота не достигнет значения, равного высоты созданной транзакции + `height-above`.
4. При достижении этой высоты блокчейна Targetnet и подтверждения наличия первой транзакции в блокчейне, узел Targetnet создаёт вторую транзакцию с данными для анкоринга уже в приватном блокчейне.

Структура транзакции для анкоринга

Транзакция для отправки в Targetnet содержит следующие поля:

- `height` – высота сохраняемого блока из приватного блокчейна;
- `signature` – подпись сохраняемого блока из приватного блокчейна.

Транзакция, создаваемая в приватном блокчейне, содержит следующие поля:

- `height` – высота сохраняемого блока из приватного блокчейна;
- `signature` – подпись сохраняемого блока из приватного блокчейна;
- `targetnet-tx-id` – идентификатор транзакции для анкоринга в Targetnet;
- `targetnet-tx-timestamp` – дата и время создания транзакции для анкоринга в Targetnet.

3.2.2.10 Тонкая настройка Платформы «Конфидент»: настройка механизма создания снимка данных

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

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

Подробное описание механизма создания снимка данных на Платформе «Конфидент» см. в разделе «Механизм создания снимка данных» документа «643.19172778.2019662198-01 96. «Блокчейн-платформа Конфидент» Версия 1.9. Руководство разработчика», который поставляется вместе с дистрибутивом платформы.

В блоке `node.consensual-snapshot` настраиваются следующие параметры:

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

3.2.2.11 Тонкая настройка Платформы «Конфидент»: настройка узла (ноды) в режиме наблюдения

Узел блокчейна может быть настроен для работы в режиме наблюдения.
В этом режиме узел работает следующим образом:

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

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

Конфигурация

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

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

- `default` – стандартный режим работы узла;
- `watcher` – режим наблюдения.

3.2.3 Получение лицензии для работы в частной сети

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

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

Для обсуждения деталей вашей лицензии свяжитесь с отделом продаж ООО «Веб3 Интегратор» по электронной почте: sales@web3tech.ru.

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

Перед развертыванием ознакомьтесь с системными требованиями к блокчейн-платформе.

3.2.4 Подписание genesis-блока и запуск сети

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

Genesis-блок подписывается утилитой **GenesisBlockGenerator**, входящей в пакет **generators**. Этот пакет входит в состав Платформы «Конфидент» и поставляется вместе с ней.

В рамках PKI в генезис-конфигурацию входят сертификаты пользователей, а также корневые доверенные сертификаты (или их идентификаторы). Эта информация не попадает в транзакции или сам блок; она необходима для функционирования PKI.

Генератор выполняет ряд проверок, которые описаны подробнее ниже в разделе «Запуск генератора GenesisBlockGenerator». Затем генератор формирует genesis-блок и подписывает его ключом пользователя, а также вносит изменения в блок `genesis` секции `node.blockchain.custom` конфигурационного файла узла, что также детально описано ниже в разделе «Запуск генератора GenesisBlockGenerator».

3.2.4.1 Подготовка к запуску генератора GenesisBlockGenerator

До создания genesis-блока подготовьте конфигурационный файл узла:

- задайте параметру `node.crypto.type` значение "`gost`";
- задайте параметру `node.crypto.pki.mode` одно из следующих значений: "`on`" или "`test`"; значение "`test`" допустимо только в режиме тестирования Платформы «Конфидент». Подробнее об этом параметре см. раздел «Общая настройка платформы «Конфидент»: настройка режима работы» выше.
- добавьте ключ проверки ЭП, которым будет проверяться подпись genesis-блока (ЭП создается на соответственном ключе ЭП), в секцию `network-participants` и там же присвойте ему роль `permissioner`;

  Примечание: Ключ ЭП, которым будет подписываться genesis-блок, уже создан генератором **GeneratePkiKeypair**. Ключ должен находиться локально в хранилище ключей машины, с которой запускается генератор.

- добавьте в конфигурационный файл узла идентификаторы корневых доверенных сертификатов – в качестве идентификаторов используются отпечатки (`fingerprint`) сертификатов; укажите сертификаты участников сети в формате DER, закодированные при помощи Base64 в текст; для этого в разделе `node.blockchain.custom.genesis.pki` заполните следующие параметры:
  - `trusted-root-fingerprints` – массив Base64 строк, перечисляющих отпечатки доверенных корневых сертификатов, которые должны находиться в trust-store JVM. Ниже приведен пример получения отпечатка корневого сертификата:

    ```
    openssl x509 -inform der -fingerprint -sha1 -in ./certificates/crypto_cert.cer -noout
    SHA1 Fingerprint=CD:32:1B:87:FD:AB:B5:03:82:9F:88:DB:68:D8:93:B5:9A:7C:5D:D3
    ```

    Полученный отпечаток необходимо сконвертировать в формат Base64.
  - `certificates` – массив строк в формате Base64, содержащих тела сертификатов в формате DER (бинарной кодировке).

Также необходимо сконфигурировать среду запуска генератора GenesisBlockGenerator:

- добавить в хранилище доверенных сертификатов (TrustStore) JVM корневые сертификаты, которые будут использоваться в качестве доверенных при валидации цепочек сертификатов в блокчейн сети.
  Например, используйте для этого встроенную в JVM утилиту `keytool` с параметром `keystore`.

  Пример вызова утилиты:

```
keytool -import -trustcacerts -alias %CERT_ALIAS% -noprompt -storepass
'changeit' -keystore
'%<your_path_to_jvm_home_folder>%/Contents/Home/lib/security/cacerts' -file
certificates/cert-to-add.cer
```

3.2.4.2 Запуск генератора GenesisBlockGenerator

В качестве параметров генератор GenesisBlockGenerator принимает следующие данные:

- путь до конфигурационного файла узла, который требуется подписать,
- алиас (адрес) узла, которым будет подписываться genesis-блок (адрес, которому вы присвоили роль `permissioner`).

Пример запуска генератора GenesisBlockGenerator в командной строке:

```
java -cp "generators-x.x.x.jar::./java-csp-5.0.R2/*"
   com.wavesenterprise.generator.GeneratorLauncher GenesisBlockGenerator
   ./node_alone.conf 3N1uZiamcpuTnRASi7L17vM8xhbC292UNgU
```

Генератор выполняет проверки следующих сущностей и условий:

- конфигурационный файл узла,
- ключи проверки ЭП и роли пользователя,
- наличие ключевой пары в хранилище ключей,
- наличие указанных доверенных корневых сертификатов из конфигурационного файла в TrustStore JVM,
- для каждого указанного промежуточного или пользовательского сертификата (в кодировке Base64) генератор строит цепочку и проверяет её валидность; для этого генератор скачивает все списки отозванных сертификатов (CRL) по URL-адресам точек распространения CRL (CDP), которые находятся в полях `extensions/CRL Distribution Points` сертификатов; генератор подписывает genesis-блок, если все сертификаты валидны, и не подписывает, если хотя бы один сертификат отозван,
- для каждого ключа проверки ЭП, указанного в конфигурационном файле, передан соответствующий ему сертификат.

Затем генератор формирует генезис-блок и подписывает его ключом ЭП узла, а также вносит следующие изменения в блок `genesis` секции `node.blockchain.custom` конфигурационного файла узла:

- в поле `public-key` указывает ключ проверки ЭП узла,
- в поле `signature` – подпись,
- в поле `block-timestamp` – локальное время в момент подписания,
- в новое поле `crls` – массив списков отозванных сертификатов (CRL) в формате Base64, для каждого из которых указан открытый ключ и точка распространения CRL (CDP).

Ниже приведён пример раздела `node.blockchain.custom.genesis` конфигурационного файла узла после работы генератора GenesisBlockGenerator:

```
genesis {
        pki {
        trusted-root-fingerprints = ["knweHrKzl8vF0RcS4uO77ch2vX4="]
         certificates = [
          "MIIFETCCBL6gAwIBAgITfAAGKtupa5ZvGCAREQABAAYq2zAKBggqhQMHAQEDAjCCAQoxGDAWBgUqhQNkARINMTIzNDU2Nzg5MDEyMzEaMBgGCCqFAwOBAwEB
           EgwwMDEyMzQlNjc4OTAxLzAtBgNVBAkMJtGD0LsuINCh0YPRidGR0LLRgdC60LjQuSDQstCw0Lsg0LQuIDE4MQswCQYDVQQGEwJSVTEZMBcGA1UECAwQ0LMuI
           NCc0L7RgdC60LLQsDEVMBMGA1UEBwwM0JzQvtGB0LrQstCwMSUwIwYDVQQKDBzQntCe0J4gItCa0KDQmNCf0KLQni3Qn3Cg0J4iMTswOQYDVQQDDDLQotC10Y
           HRgtC+0LLRi9C5INCj0KYg0J7QntCeICLQmtCg0JjQn9Ci0J4t0J/QoNCeIjAeFw0yMjA3MjcxMzA3MzJaFw0yMjEwMjcxMzE3MzJaMHQxCzAJBgNVBAYTAlJ
           VMQowCAYDVQQIEwFhMQowCAYDVQQHEwFiMRkwFwYDVQQKExBXYXZlcyBFbnRlcnByaXNlMRQwEgYDVQQLEwtJVCBCdXNpbmVzczEcMBoGA1UEAwwTRUxfbm9k
           ZV8wX3Bhc3N3b3JkczBmMB8GCCqFAwcBAQEBMBMGByqFAwICIwEGCCqFAwcBAQICA0MABEAWLt/Pz/y04uJSuIaDD/dcu0xj8wlTF7EA2+ue0RMYEQzgv6mrS
           1JG5z5DdDnllILS7rTTdscEoTycF2IfaQ5Zo4ICiDCCAoQwDgYDVR0PAQH/BAQDAgeAMBMGA1UdJQQMMAoGCCsGAQUFBwMBMC0GA1UdEQQmMCSCC3dlbG9jYW
           wuZGV2gglsb2NhbGhvc3SHBDPS0z2HBH8AAAEwHQYDVR0OBBYEFLvz65pv/mvEYzLXRAV6eQu2/33sMB8GA1UdIwQYMBaAFJuFXvuB3E1ZB1Fjz77f2ix/yUQ
           8MIIBDwYDVR0fBIIBBjCCAQIwgf+ggfyggfmGgbVodHRwOi8vdGVzdGdvc3QyMDEyLmNyeXB0b3Byby5ydS9DZXJ0RW5yb2xsLyEwNDIyITA0MzUhMDQ0MSEw
           NDQyITA0M2UhMDQzMiEwNDRiITA0MzklMjAhMDQyMyEwNDI2JTIwITA0MWUhMDQxZSEwNDFlJTIwITAwMjIhMDQxYSEwNDIwITA0MTghMDQxZiEwNDIyITA0M
           WUtITA0MWYhMDQyMCEwNDFlITAwMjIoMSkuY3Jshj9odHRwOi8vdGVzdGdvc3QyMDEyLmNyeXB0b3Byby5ydS9DZXJ0RW5yb2xsL3Rlc3Rnb3N0MjAxMigxKS
           5jcmwwgdoGCCsGAQUFBwEBBIHNMIHKMEQGCCsGAQUFBzAChjhodHRwOi8vdGVzdGdvc3QyMDEyLmNyeXB0b3Byby5ydS9DZXJ0RW5yb2xsL3Jvb3QyMDE4LmN
           ydDA/BggrBgEFBQcwAYYzaHR0cDovL3Rlc3Rnb3N0MjAxMi5jcnlwdG9wcm8ucnUvb2NzcDIwMTJnL29jc3Auc3JmMEEGCCsGAQUFBzABhjVodHRwOi8vdGVz
```

```
            dGdvc3QyMDEyLmNyeXB0b3Byby5ydS9vY3NwMjAxMmdzdC9vY3NwLnNyZjAKBggqhQMHAQEDAgNBALCDXGMTdhUwruxrXLV6LqebZRKDuDWpFeYT6A7BkABxt
            0uQrpbmyZ/Gaz3eNQ++9XtQaxjEIUvw7bjn26qgqI4=",
            "MIIE6DCCBJWgAwIBAgITfAAGS4CU1edyw4+SgwABAAZLgDAKBggqhQMHAQEDAjCCAQoxGDAWBgUqhQNkARINMTIzNDU2Nzg5MDEyMzEaMBgGCCqFAwOBAwE
            BEgwwMDEyMzQ1Njc4OTAxLzAtBgNVBAkMJtGD0LsuINCh0YPRidGR0LLRgdC60LjQuSDQstCw0Lsg0LQuIDE4MQswCQYDVQQGEwJSVTEZMBcGA1UECAwQ0LMu
            INCc0L7RgdC60LLQsDEVMBMGA1UEBwwM0JzQvtGB0LrQstCwMSUwIwYDVQQKDBzQntCe0J4gItCa0KDQmNCf0KLQni3Qn9Cg0J4iMTswOQYDVQQDDDLQotC10
            YHRgtC+0LLRi9C5INCj0KYg0J7QntCeICLQmtCg0JjQn9Ci0J4t0J/QoNCeIjAeFw0yMjA4MTIwODE0MDZaFw0yMjExMTIwODI0MDZaMGcxCzAJBgNVBAYTAl
            JVMQowCAYDVQQIEwFCMQowCAYDVQQHEwFBMRkwFwYDVQQKExBXYXZlcyBFbnRlcnByaXNlMRQwEgYDVQQLEwtJVCBCdXNpbmVzczEPMA0GA1UEAxMGTm9kZS0
            zMGYwHwYIKoUDBwEBAQEwEwYHKoUDAgIjAQYIKoUDBwEBAgIDQwAEQH9sEK7lAgNEi9+mY+20OWYbGlgnC5e4+Rz589N2W9rbiwrRIKr3ng3fo3upiKBtkFew
            Aykpwhz7U8s5eDMdtRCjggJsMIICaDAOBgNVHQ8BAf8EBAMCB4AwEwYDVR0lBAwwCgYIKwYBBQUHAwEwEQYDVR0RBAowCIIGbm9kZS0zMB0GA1UdDgQWBBTbD
            Mr79oDlISLk6uGq5b77t2DUSjAfBgNVHSMEGDAWgBSbhV77gdxNWQdRY8++39osf8lEPDCCAQ8GA1UdHwSCAQYwggECMIH/oIH8oIH5hoG1aHR0cDovL3Rlc3
            Rnb3N0MjAxMi5jcnlwdG9wcm8ucnUvQ2VydEVucm9sbC8hMDQyMiEwNDM1ITA0NDEhMDQ0MiEwNDNlITA0MzIhMDQ0YiEwNDM5JTIwITA0MjMhMDQyNiUyMCE
            wNDFlITA0MWUhMDQxZSUyMCEwMDIyITA0MWEhMDQyMCEwNDE4ITA0MWYhMDQyMiEwNDFlLSEwNDFmITA0MjAhMDQxZSEwMDIyKDEpLmNybIY/aHR0cDovL3Rl
            c3Rnb3N0MjAxMi5jcnlwdG9wcm8ucnUvQ2VydEVucm9sbC90ZXN0Z29zdDIwMTIoMSkuY3JsMIHaBggrBgEFBQcBAQSBzTCByjBEBggrBgEFBQcwAoY4aHR0c
            DovL3Rlc3Rnb3N0MjAxMi5jcnlwdG9wcm8ucnUvQ2VydEVucm9sbC9yb290MjAxOC5jcnQwPwYIKwYBBQUHMAGGM2h0dHA6Ly90ZXN0Z29zdDIwMTIuY3J5cH
            RvcHJvLnJlL29jc3AyMDEyZy9vY3NwLnNyZjBBBggrBgEFBQcwAYY1aHR0cDovL3Rlc3Rnb3N0MjAxMi5jcnlwdG9wcm8ucnUvb2NzcDIwMTJnc3Qvb2NzcC5
            zcmYwCgYIKoUDBwEBAwIDQQB9nl53V5KYsCqn6tFoaBWHW8CroaVlYWbQP7/iNkGXCd5KD8Ue6GZOww7VtlQl+nQuT6KvHuINJQlG3W/DrDeg"
            ]
            crls = [
              { publicKeyBase58 = "...", cdp = "http://example.com/", crl = "MIICwTCCAm4CAQEwCgYIKoUDBw
                EBAwIwggEKMRgwFgYFKoUDZAESDTEyMzQ1Njc4OTAxMjMxGjAYBggqhQMDgQMBARIMMDAxMjM0NTY3ODkwMS8wLQYDVQQJDCbRg9C7LiDQodGD0YnR
                kdCy0YHQutC40Lkg0LLQsNC7INC0LiAxODELMAkGA1UEBhMCUlUxGTAXBgNVBAgMENCzLiDQnNC+0YHQutCy0LAxFTATBgNVBAcMDNCc0L7RgdC60L
                LQsDElMCMGA1UECgwc0J7QntCeICLQmtCg0JjQn9Ci0J4t0J/QoNCeIjE7MDkGA1UEAwwy0KLQtdGB0YLQvtCy0YvQuSDQo9CmINCe0J7QniAi0JrQ
                oNCY0J/QotCeLdCf0KDQniIXDTIyMDgxMjExMTMwNVoXDTIyMDgxMzEzNTcwNVowgcwwJAITfAAGS4CU1edyw4+SgwABAAZLgBcNMjIwODEyMTEyMj
                U2WjAkAhN8AAT8Hj6QH75DAJacAAEABPweFw0yMjA0MDEwNzQwMTRaMCQCE3wABPrkedSz18eUu40AAQAE+uQXDTIyMDMzMTEzNTMwNVowMgITfAAC
                mXPmMRxspgYBrgABAAKZcxcNMjEwNTI2MTAxMDAwWjAMMAoGA1UdFQQDCgEFMCQCE3wAAfsy9FZzqkOn1PQAAQAB+zIXDTIxMDIwMzA1NDAyNlqgYj
                BgMB8GA1UdIwQYMBaAFJuFXvuB3E1ZB1Fjz77f2ix/yUQ8MBIGCSsGAQQBgjcVAQQFAgMBAAEwCwYDVR0UBAQCAgbGMBwGCSsGAQQBgjcVBAQPFw0y
                MjA4MTMxMTIzMDVaMAoGCCqFAwcBAQMCA0EAnQbYPqSbXxeBJYM+ln20fJBPXPcMmfid35088XjX0VhnNg4lxOGP8bloIQrU89NtS5+WI12c02IZFX
                8w5xSumA=="}
              {...}
            ]
          }
          average-block-delay: 40s
          block-timestamp: 1660306624184
          initial-base-target: 10000
          initial-balance: 17500000 WEST
          genesis-public-key-base-58: "SiyNYM988LpcqqJxbT2YVZa3gDAMgXXcock8h9WrVfD1R1rF2pZApvoGV5h2DTN2e1grFYmzgv94CkGNmKKsTuz"
          signature: "3omB9mpFq8fboSLR6Kd7shmoCuEj6uhoxrKGe7mMfunmCBQGfdpPtmovsQ62JjjKTN1hA7tY9exbTkf843xxmvub"
          transactions = [
            {recipient: "3N2gYbus4cHQJeEGJ6J2WfHyEPWnT4KjJ9e", amount: 12500000 WEST},
            {recipient: "3MtAfmSb4fzhMQGHH1UqFXDmzouZp3uVyv3", amount: 3000000 WEST},
            {recipient: "3MrfnwhPPmvJp5B4hiwUwqtSJBJGs9DuxWe", amount: 1000000 WEST},
            {recipient: "3N8MMJJPfDKfgZDHYFAcbLwkfb3KydKup3F", amount: 1000000 WEST}
          ]
          network-participants = [
            {public-key: "SiyNYM988LpcqqJxbT2YVZa3gDAMgXXcock8h9WrVfD1R1rF2pZApvoGV5h2DTN2e1grFYmzgv94CkGNmKKsTuz", roles:
                [permissioner, miner, connection_manager, contract_developer, issuer]},
            {public-key: "th4Pybsugj5JZkE245vKkGxPeCwEYDC4FBDeXXQFZGQST1xqB5udHcuNxSpk9fUj5X7A6LpV4h3ET82kVgowrSW", roles:
                [permissioner, miner, connection_manager, contract_developer, issuer]},
            {public-key: "37TqA4mUSG8ptc5hEPcdZikd1fE3H5EQbVcKh1NQeVevwmHS9Ci5RpvRd6eTz2XSxDaXRCReTtDWfHRjLLaXC89e", roles:
                [permissioner, miner, connection_manager, contract_developer, issuer]}
            {public-key: "3Ym3kFEFoBVTLgp5S7PRmzYW4kKYTD9UbDtu3uGHPgmUTmndrwamSCEDfMC7RDXxr8DSgd2za3io8rrvNxxxBTWw", roles:
                [permissioner, miner, connection_manager, contract_developer, issuer]}

          ]
        }
      }
    }
```

3.2.4.3 Запуск блокчейн-платформы Конфидент

После подписания genesis-блока Платформа «Конфидент» полностью настроена и готова для запуска сети. Убедитесь, что выполнены следующие условия:

- конфигурационный файл узла `node.conf` лежит в той же директории, что и дистрибутив Платформы «Конфидент»;
- конфигурационный файл взят под контроль целостности;
- библиотеки ядра, реализующего криптографические алгоритмы и протоколы (описано в документации, которая поставляется с дистрибутивом платформы), распакованы в директорию `lib`;
- проведен контроль целостности согласно эксплуатационной документации ядра, реализующего криптографические алгоритмы и протоколы (описано в документации, которая поставляется с дистрибутивом платформы).

Затем для разворачивания Платформы «Конфидент» вызовите команду:

```
java -cp "confident-1.9.0.jar:lib/*" com.wavesenterprise.Application node.conf,
```

где `confident-1.9.0.jar` – имя файла дистрибутива Платформы «Конфидент».

Примечание: если конфигурационный файл узла лежит не в той же директории, что и дистрибутив, укажите полный путь до него вместо имени файла node.conf.

3.3 Пример конфигурационного файла узла

В этом разделе приведён пример конфигурационного файла узла node.conf. В этом примере конфигурации:

- используется алгоритм консенсуса CFT;
- включена защита канала связи по протоколу TLS с использованием ГОСТ (отключение допускается только в тестовых целях);
- запущены инструменты gRPC и REST API с использованием канала связи по протоколу TLS с использованием ГОСТ;
- включена авторизация по tls-whitelist для gRPC и REST API;
- включено исполнение смарт-контрактов;
- настроена функция периодического удаления невалидных транзакций из UTX-пула участника блокчейна, который не является майнером;
- настроена задержка проверки UTX-пула (есть ли в пуле транзакции или он пуст) майнером.

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

Каждая секция снабжена дополнительным комментарием.

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

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

  # Node "home" and data directories to store the state
  # Default: ${user.home}"/node"
  directory = " /FILL/ "

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

  # Crypto settings
  crypto {
    type = GOST

    pki {
      mode = ON
      # At least one of the OIDs is required to be listed in EKU of user's certificates to pass
verification
      required-oids = [ /FILL/ ]
      crl-checks-enabled = true
    }
  }

  # NTP settings
  ntp {
    # Defaults: ["0.pool.ntp.org", "1.pool.ntp.org", "2.pool.ntp.org", "3.pool.ntp.org"]
    # servers = [/FILL/]

    # Socket timeout for synchonization request.
    request-timeout = 10 seconds

    # Time between synchronization requests.
    expiration-timeout = 1 minute

    # Maximum time without synchronization. Required for PoA consensus.
    fatal-timeout = 1 minute
  }

  # keystore access password
  wallet.password = " /FILL/ "
  }
```

```
  # Blockchain settings
  blockchain {

    type = CUSTOM
    # Must be configured for specific network topology and load
    consensus {
      type = CFT
      round-duration = 10s
      sync-duration = 2s
      ban-duration-blocks = 20
      warnings-for-ban = 5
      max-bans-percentage = 30
      max-validators = 5
    }

    custom {
      # Network byte, used to distinguish addresses from different networks
      address-scheme-character = "T"
      functionality {
        feature-check-blocks-period = 10000
        blocks-for-feature-activation = 6600
        pre-activated-features = {
          2 = 0
          3 = 0
          4 = 0
          5 = 0
          6 = 0
          7 = 0
          9 = 0
          10 = 0
          100 = 0
          101 = 0
          119 = 0
          120 = 0
          130 = 0
          140 = 0
          160 = 0
          162 = 0
          173 = 0
          180 = 0
          190 = 0
        }
      }

      genesis {
        pki {
          # SHA1 fingerprints of trusted roots
          trusted-root-fingerprints = [/FILL/]
          # Certificates of all participants of the genesis block
          certificates = [/FILL/]
          # Initial CRLs
          crls = [{publicKeyBase58 = " /FILL/ ", cdp = " /FILL/ ", crl = "/FILL/"}]
        }
        average-block-delay: 40s
        # Will be reset by GenesisBlockGenerator
        block-timestamp: 1662019967398
        initial-base-target: 10000
        initial-balance: 16500000 WEST
       # Will be reset by GenesisBlockGenerator
        genesis-public-key-base-58: " /FILL/ "
        # Well be reset by GenesisBlockGenerator
        signature: " /FILL/ "
        # Initial distribution of initial coins
        transactions = [
          {recipient: " /FILL/ ", amount: /FILL/}
        ]
```

```
      # Initial network participants and role distribution among them
      network-participants = [
        {public-key: " /FILL/ ", roles: [/FILL/]}
      ]
    }
  }
}

miner {
  enable = yes
  # Important: use quorum = 0 only for testing purposes, while running a single-node network;
  # In other cases always set quorum > 0
  quorum = 2
  interval-after-last-block-then-generation-is-allowed = 10d
  micro-block-interval = 1500ms
  min-micro-block-age = 0ms
  max-transactions-in-micro-block = 500
  utx-check-delay = 1000ms
}

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 = GOST
  keystore-type = "HDIMAGE"
  keystore-password = " /FILL/ "
  truststore-type = "CertStore"
  truststore-path = " /FILL/ "
  truststore-password = " /FILL/ "
}

# P2P Network settings
network {
  # Network address
  bind-address = "0.0.0.0"
  # Port number
  port = 6864
  # Enable/disable network TLS
  tls = true

  # ENUM: regular or watcher
  mode = regular

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

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

  break-idle-connections-timeout = 3m
}
```

```
  # 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 = true
    }

    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 = true
    }

    auth {
        type = "tls-whitelist"

        # Public keys are expected in Base64 format
        admin-public-keys = [
        "MGYwHwYIKoUDBwEBAQEwEwYHKoUDAgIkAAYIKoUDBwEBAgIDQwAEQLh7lrv/ioWdnUkvX3NybRoeew1PPz1vMaajxzpi1CYoWRl
rPC9 RjlVul5PCMyAL2OuO4lgpcqBj1+y2cEjajXw="
        ]
    }
  }


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

    # For starting contracts in a local docker
    use-node-docker-host = no

    # default-registry-domain = "registry.yourdomain.com"
    # Basic auth credentials for docker host
    #docker-auth {
    #  username = "some user"
    #  password = "some password"
    #}

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

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

    # Execution settings
    execution-limits {
```

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

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

4 Состав и назначение компонент блокчейн-платформы Конфидент

Каждый узел блокчейна – это самостоятельный участник сети, имеющий ПО для работы в ней. Узел состоит из следующих компонентов:

- **Сервисы консенсуса и криптографические библиотеки** (Consensus and cryptolibraries) – компоненты, отвечающие за механизм достижения согласия между узлами, а также за криптографические алгоритмы.

- **API-интерфейсы узла (ноды)** (Node API) – интерфейсы gRPC и REST API узла, позволяющие получать данные из блокчейна, подписывать и отправлять транзакции, отправлять конфиденциальные данные, создавать и выполнять смарт-контракты и др.

- **Пул неподтвержденных транзакций** (Unconfirmed transaction pool, UTX pool) – компонент, обеспечивающий хранение неподтвержденных транзакций до момента их проверки и отправки в блокчейн.

- **Майнер** (Miner) – компонент, отвечающий за формирование блоков транзакций для записи в блокчейн, а также за взаимодействие со смарт-контрактами.

- **Хранилище ключей** (Key store) - хранилище ключевых пар узла и пользователей. Все ключи защищены паролем.

- **Сетевой слой** (Network layer) – слой логики, обеспечивающий взаимодействие узлов на прикладном уровне по сетевому протоколу поверх TCP.

- **Хранилище узла (ноды)** (Node storage) – компонент системы на базе RockDB, обеспечивающий хранение пар ключ-значение для полного набора проверенных и подтверждённых транзакций и блоков, а также текущего состояния блокчейна.

- **Логика валидации** (Validation logic) – слой логики, содержащий такие правила проверки транзакций, как базовая проверка подписи и расширенная проверка по сценарию.

- **Конфигурация** (Config) – конфигурационные параметры узла, задаваемые в файле node-name.conf.

Для каждого узла предусмотрен набор дополнительных сервисов:

- **Дата-краулер** – сервис извлечения данных с узла и загрузки извлечённых данных в дата-сервис.

- **Генератор** – сервис генерации ключевых пар для новых аккаунтов и подписанного этой ключевой парой запроса на сертификат.

- **Сервис мониторинга** – внешний сервис мониторинга, использующий базу данных InfluxDB для хранения временных рядов с данными и метриками приложения.

Перечень всех файлов Платформы «Конфидент», которые обеспечивают ее функционал, приведён в разделе «6 Требования по обеспечению контроля целостности» ниже.