

ПРАКТИЧЕСКАЯ РАБОТА

HTTPS и TLS в контейнерной инфраструктуре

Самоподписанный сертификат, локальный CA и TLS termination на Nginx Reverse Proxy

Направление: СА

Рабочая среда: Windows 11, Docker Desktop, Docker Compose

Продолжение предыдущей работы: используется инфраструктура Nginx + FileBrowser Quantum + Uptime Kuma. Изменять системный файл hosts студентам не требуется: для учебных DNS-имен в проверках используется параметр `curl --resolve`. Если преподаватель заранее настроил локальный DNS или hosts централизованно, те же адреса можно открывать и в браузере.

2026

1. Цель работы

В предыдущей практической работе несколько контейнерных сервисов были объединены общей Docker-сетью и опубликованы через Nginx Reverse Proxy. Соединение от клиента до reverse proxy при этом использовало обычный HTTP.

В этой работе та же инфраструктура переводится на HTTPS. Работа специально построена в два этапа. Сначала TLS настраивается непосредственно на одном веб-сервере, чтобы было видно, из каких файлов и параметров вообще складывается HTTPS. Затем создается локальный центр сертификации и TLS переносится на reverse proxy. Так становится понятна причина, по которой в реальной инфраструктуре сертификаты часто централизуют на входном прокси.

В результате:

- обычный веб-сайт будет временно переведен с HTTP на HTTPS непосредственно внутри своего контейнера;
- будет создан самоподписанный сертификат и разобрана причина предупреждения клиента;
- будет создан локальный центр сертификации (CA) и подробно разобрано, что именно в нем является доверенным, а что секретным;
- будет создан закрытый ключ сервиса, запрос CSR и серверный сертификат с несколькими DNS-именами в SAN;
- reverse proxy будет опубликован на портах 80 и 443;
- HTTP-запросы будут перенаправляться на HTTPS;
- Nginx Reverse Proxy будет выполнять TLS termination;
- внутренние сервисы останутся на HTTP внутри Docker-сети;
- будут проверены цепочка доверия, SAN, срок действия, issuer, SNI и типичные TLS-ошибки.

2. Повторение: HTTP, HTTPS и TLS

HTTP. протокол прикладного уровня для передачи веб-запросов и ответов. Сам по себе HTTP не шифрует данные.

HTTPS. HTTP, передаваемый внутри TLS-соединения. Шифрование работает между клиентом и точкой, на которой завершается TLS.

TLS. протокол защищенного транспортного соединения. Он обеспечивает шифрование, контроль целостности и проверку подлинности стороны по сертификату.

Сертификат X.509. публичный документ, который связывает открытый ключ с именами и другими сведениями о владельце. Сертификат не является секретом.

Закрытый ключ. секретная часть пары ключей. Сервер доказывает владение им во время TLS-handshake. Закрытый ключ не передается клиенту.

Самоподписанный сертификат. сертификат, подпись которого создана тем же закрытым ключом, которому соответствует его открытый ключ. Он может шифровать соединение, но по умолчанию не образует доверенную цепочку для клиента.

CA (Certificate Authority). центр сертификации. В простейшем учебном варианте это пара: закрытый ключ CA и публичный сертификат CA. Закрытым ключом CA подписываются другие сертификаты.

CSR (Certificate Signing Request). запрос на выпуск сертификата. CSR содержит открытый ключ и сведения о будущем сертификате, а также подпись владельца закрытого ключа. Закрытого ключа внутри CSR нет.

SAN (Subject Alternative Name). расширение сертификата со списком DNS-имен и/или IP-адресов, для которых сертификат действителен.

SNI (Server Name Indication). поле TLS-handshake, с помощью которого клиент сообщает серверу имя сайта до отправки обычного HTTP-запроса. Это позволяет нескольким HTTPS-сайтам работать на одном IP и порту 443.

TLS termination. схема, при которой TLS завершается на reverse proxy. Проxy расшифровывает HTTPS-запрос и передает его внутреннему сервису по другому соединению, в этой работе - по HTTP.

Главная мысль: шифрование и доверие - не одно и то же. Самоподписанный сертификат уже дает TLS-шифрование, но клиент не знает, почему этому сертификату следует доверять. CA добавляет отдельный корень доверия.

3. Исходная и конечная схемы

3.1. Исходная схема

```

Browser
|
| HTTP
v
Nginx Reverse Proxy
|
+----HTTP----> site:80
+----HTTP----> filebrowser:80
+----HTTP----> uptime-kuma:3001

```

3.2. Конечная схема

```

Browser / curl
|
| HTTP :80 ---> redirect ---> HTTPS :443
| TLS encrypted
v
+-----+
| Nginx Reverse Proxy |
| TLS termination    |
| certificate + key   |
+-----+
| Docker network sa-net
| HTTP
+----> site:80
+----> filebrowser:80
+----> uptime-kuma:3001

```

Шифрование в основной части работы действует от клиента до reverse proxy. Внутри Docker-сети reverse proxy начинает новое HTTP-соединение с backend. Это не означает, что браузерное TLS-соединение "продолжается" до приложения: на проху оно действительно завершается.

Не путать: Browser --HTTPS--> Proxy --HTTP--> Backend - TLS termination. Схема Proxy --HTTPS--> Backend также существует, но требует отдельной проверки backend-сертификатов и является другой задачей.

4. Перед началом работы

Необходимы:

- Windows 11 и запущенный Docker Desktop;
- рабочий каталог предыдущей практической работы;
- compose.yaml с сервисами site, filebrowser, uptime-kuma и proxy;
- рабочая Docker-сеть sa-net;
- свободные TCP-порты 80 и 443 на Windows для финального этапа.

Учебные имена site.sa.test, files.sa.test и status.sa.test используются для демонстрации SAN и SNI. Так как на учебных ПК нет необходимости изменять системный hosts, команды curl будут использовать --resolve. Этот параметр заставляет только конкретный запуск curl обратиться к указанному IP для выбранного имени и порта. Системные настройки Windows при этом не меняются.

Важно: если порт 80 или 443 уже занят IIS, другим веб-сервером, VPN-компонентом или контейнером, Docker не сможет его опубликовать. В конце работы есть диагностика этой ситуации.

5. Подготовка файлов проекта

Через Проводник Windows создайте в каталоге проекта папку certs. Конфигурационные файлы создавайте обычным текстовым редактором. Команды PowerShell для создания папок и файлов не нужны.

После подготовки структура проекта должна включать:

```
docker-sa-infra\
|-- compose.yaml
|-- certs\
|-- site\
|   |-- index.html
|   |-- https.conf      # временно, для первого этапа
|-- shared\
|-- filebrowser-data\
|-- proxy\
|   |-- default.conf
```

Почему отдельная папка certs: она монтируется в служебный контейнер OpenSSL. Все создаваемые там ключи, CSR и сертификаты сразу появляются в папке Windows и сохраняются после удаления служебного контейнера.

6. OpenSSL как служебный контейнер

OpenSSL не требуется устанавливать в Windows. Мы используем контейнер только как инструмент командной строки. Это важная идея Docker: контейнер может быть не постоянно работающим сервером, а одноразовым набором утилит.

Добавьте в `compose.yaml` сервис:

```
openssl:
  image: alpine/openssl:3.5.8
  profiles:
    - tools
  volumes:
    - ./certs:/certs
  working_dir: /certs
  networks:
    - sa-net
```

Compose profile. механизм, позволяющий описать вспомогательный сервис в том же `compose.yaml`, но не запускать его при обычном `docker compose up -d`.

Логика записи: каталог `./certs` на Windows монтируется как `/certs` в контейнере, а `working_dir` делает `/certs` текущим рабочим каталогом. Поэтому команда OpenSSL может писать просто `site-self.crt` вместо полного `/certs/site-self.crt`.

6.1. Как читать команды из этой методички

```
docker compose run --rm openssl req ...
```

Эта строка состоит из двух уровней: Docker Compose запускает инструмент, а OpenSSL выполняет криптографическую операцию.

Фрагмент	Что означает	Зачем нужен
<code>docker compose run</code>	создать одноразовый контейнер для выбранного сервиса и выполнить в нем команду	не нужно постоянно держать OpenSSL-контейнер запущенным
<code>--rm</code>	удалить контейнер после завершения команды	не накапливать остановленные служебные контейнеры
<code>openssl</code>	имя сервиса из <code>compose.yaml</code>	Compose знает образ, volume и <code>working_dir</code> этого инструмента
<code>req / x509 / verify / s_client</code>	подкоманда OpenSSL	каждая подкоманда решает отдельный класс задач
остальные параметры	опции выбранной подкоманды OpenSSL	задают алгоритм ключа, файлы, срок, имена, проверку и т.д.

Откуда берутся параметры: OpenSSL устроен как набор подкоманд. Их официальный синтаксис можно посмотреть в документации `openssl-req`, `openssl-x509`, `openssl-verify`, `openssl-s_client` или прямо выполнить, например: `docker compose run --rm openssl req -help`. Поэтому длинная команда ниже - не специальная "команда Docker", а обычный OpenSSL CLI, запущенный внутри контейнера.

Проверьте версию:

```
docker compose run --rm openssl version
```

Здесь version - простейшая подкоманда OpenSSL. Если она отработала, значит образ доступен, bind mount не мешает запуску и служебный сервис можно использовать дальше.

7. Этап 1. HTTPS непосредственно на обычном веб-сервере

Сначала HTTPS настраивается без reverse proxy. Это позволяет увидеть минимальный набор: закрытый ключ, сертификат и TLS-конфигурация веб-сервера. После этого станет понятнее, что именно reverse proxy централизует.

7.1. Создание самоподписанного сертификата

Выполните из каталога проекта одной строкой:

```
docker compose run --rm openssl req -x509 -newkey rsa:2048 -noenc -days 365 -sha256 -keyout site-self.key -out site-self.crt -subj "/CN=site.sa.test" -addext "subjectAltName=DNS:site.sa.test"
```

Команда использует подкоманду openssl req. Обычно req создает CSR, но параметр -x509 переключает ее в режим выпуска самоподписанного сертификата. Это прямо предусмотрено OpenSSL для тестовых сертификатов и, например, корневых CA.

Фрагмент	Что означает	Зачем нужен
req	работа с PKCS#10 запросами и создание сертификата в режиме -x509	одной командой получить ключ и тестовый сертификат
-x509	вывести сертификат X.509 вместо CSR	получить самоподписанный сертификат непосредственно сейчас
-newkey rsa:2048	создать новый RSA-ключ длиной 2048 бит	не использовать заранее подготовленный ключ
-noenc	не шифровать создаваемый private key паролем	Nginx должен читать ключ автоматически без интерактивного ввода; это удобно для учебной среды
-days 365	срок действия сертификата - 365 дней	ограничить период, в котором сертификат считается действительным
-sha256	использовать SHA-256 при формировании подписи	явно задать современный digest для RSA-подписи
-keyout site-self.key	имя файла закрытого ключа	ключ будет использовать Nginx для TLS
-out site-self.crt	имя выходного сертификата	его Nginx отправляет клиенту
-subj "/CN=site.sa.test"	задать Subject без интерактивных вопросов	команда работает одинаково у всей группы
-addext "subjectAltName=DNS:site.sa.test"	добавить расширение SAN	современная проверка имени сервера опирается на SAN

Почему -noenc: в старых примерах часто встречается -nodes. В OpenSSL 3.x этот вариант считается устаревшим; современный параметр называется -noenc. Он делает ключ незашифрованным на диске. Для реального CA это плохая практика, но для временной учебной лаборатории позволяет не вводить пароль при каждом запуске Nginx.

В папке certs должны появиться site-self.key и site-self.crt. Первый файл секретный, второй публичный.

```
site-self.key -> private key (секрет)
site-self.crt -> certificate (можно передавать клиенту)
```

Что произошло криптографически: OpenSSL сгенерировал пару RSA-ключей. Открытая часть попала в сертификат. Затем сертификат был подписан тем же закрытым ключом site-self.key. Поэтому Subject и Issuer у него совпадают.

7.2. Просмотр сертификата

```
docker compose run --rm openssl x509 -in site-self.crt -noout -subject -issuer -dates -ext subjectAltName
```

Здесь подкоманда x509 не выпускает новый сертификат, а читает существующий X.509-файл и выводит выбранные поля.

Фрагмент	Что означает	Зачем нужен
x509	подкоманда обработки X.509 сертификатов	просмотр и преобразование сертификата
-in site-self.crt	входной сертификат	указать, какой файл читать
-noout	не печатать PEM-код сертификата	оставить только нужную диагностическую

Фрагмент	Что означает	Зачем нужен
	целиком	информацию
-subject	показать владельца сертификата	увидеть CN/Subject
-issuer	показать издателя	для self-signed он совпадает с Subject
-dates	показать notBefore и notAfter	проверить срок действия
-ext subjectAltName	показать расширение SAN	убедиться, что имя site.sa.test действительно включено

Откуда взята команда: это типичное диагностическое использование openssl x509. Те же поля можно вывести через openssl x509 -text, но выбор отдельных параметров делает вывод короче и удобнее для лабораторной.

7.3. Конфигурация Nginx для прямого HTTPS

Создайте site\https.conf:

```
server {
    listen 80;
    server_name site.sa.test;
    return 308 https://$host:8443$request_uri;
}

server {
    listen 443 ssl;
    server_name site.sa.test;

    ssl_certificate /etc/nginx/certs/site-self.crt;
    ssl_certificate_key /etc/nginx/certs/site-self.key;

    root /usr/share/nginx/html;
    index index.html;
}
```

Логика конфигурации: первый server-блок слушает обычный HTTP и ничего не обслуживает сам - он отвечает перенаправлением 308. Второй блок слушает HTTPS и указывает Nginx, какой сертификат отправлять клиенту и каким закрытым ключом доказывать владение этим сертификатом.

Фрагмент	Что означает	Зачем нужен
listen 443 ssl	слушать TCP 443 как HTTPS/TLS	включить TLS для этого server-блока
ssl_certificate	путь к публичному сертификату внутри контейнера	Nginx отправляет сертификат клиенту
ssl_certificate_key	путь к соответствующему private key	Nginx использует ключ во время TLS-handshake
return 308 ...	вернуть HTTP redirect	перевести клиента с HTTP на HTTPS, сохранив путь запроса
\$host	имя, с которым клиент обратился к серверу	сформировать корректный адрес перенаправления
\$request_uri	исходный путь и query string	не потерять /page?x=1 при redirect

Почему сертификат и ключ должны составлять пару: сертификат содержит открытый ключ. Nginx должен иметь именно соответствующий ему закрытый ключ. Если подключить чужой key, nginx -t или запуск Nginx завершится ошибкой key values mismatch.

7.4. Временное изменение сервиса site

```
site:
  image: nginx:stable-alpine
  ports:
    - "8081:80"
    - "8443:443"
  volumes:
    - ./site:/usr/share/nginx/html:ro
    - ./site/https.conf:/etc/nginx/conf.d/default.conf:ro
    - ./certs:/etc/nginx/certs:ro
  restart: unless-stopped
  networks:
    - sa-net
```

Публикация 8443:443 означает: браузер или curl в Windows подключается к порту 8443, Docker Desktop передает соединение на порт 443 контейнера, где Nginx выполняет TLS. Нестандартный 8443 используется только временно, чтобы не путать этот этап с финальным reverse proxy на 443.

Сертификаты подключаются read-only (:ro), потому что веб-сервер должен читать их, а не изменять.

```
docker compose up -d site
docker compose exec site nginx -t
```

Первая команда применяет конфигурацию сервиса site. Вторая запускает nginx -t внутри уже работающего контейнера и проверяет синтаксис конфигурации и доступность файлов сертификата/ключа.

7.5. Проверка

Так как системный hosts изменять не требуется, используйте curl --resolve:

```
curl.exe -I --resolve site.sa.test:8081:127.0.0.1 http://site.sa.test:8081/
curl.exe -vk --resolve site.sa.test:8443:127.0.0.1 https://site.sa.test:8443/
```

Фрагмент	Что означает	Зачем нужен
--resolve site.sa.test:8443:127.0.0.1	для этого запуска считать, что site.sa.test на порту 8443 имеет IP 127.0.0.1	получить DNS-подобное поведение без изменения Windows hosts
-I	запросить только HTTP-заголовки	удобно увидеть код 308 и Location
-v	подробный диагностический вывод	увидеть TLS-handshake, сертификат и HTTP обмен
-k	не прерывать запрос из-за недоверенного сертификата	самоподписанный сертификат пока не имеет доверенного CA

Важно: -k не "включает HTTPS" и не исправляет сертификат. Он лишь отключает проверку доверия и имени для данного запуска curl. Поэтому -k пригоден для диагностики, но не является способом нормальной эксплуатации.

8. Почему неудобно настраивать HTTPS в каждом сервисе

Теперь видно, сколько деталей потребовалось только для одного Nginx: private key, certificate, volume, listen 443 ssl, redirect, срок действия и диагностика. Если переносить TLS внутрь каждого приложения, эту работу придется повторять для FileBrowser и Uptime Kuma, причем у разных приложений параметры TLS отличаются.

```
site          -> certificate + key + TLS config
filebrowser   -> certificate + key + TLS config
uptime-kuma   -> certificate + key + TLS config
```

Reverse proxy позволяет сосредоточить внешний TLS в одном месте:

```

      one place for certificates
          |
          v
Browser -- HTTPS --> Nginx Reverse Proxy -- HTTP --> internal services

```

Практический смысл: централизация уменьшает количество компонентов, которым нужен доступ к закрытым ключам, и дает одну точку для обновления сертификатов, redirect с 80 на 443 и TLS-политики.

9. Этап 2. Создание локального центра сертификации

Самоподписанный серверный сертификат смешивает две роли: он одновременно является сертификатом сервера и сам себе издатель. Для изучения нормальной цепочки доверия разделим роли. Создадим локальный root CA, а затем этим CA подпишем отдельный серверный сертификат.

```
ca.key (секретный ключ CA)
|
+-- создает подпись --> ca.crt (публичный корневой сертификат)
|
+-- подписывает ----> services.crt (сертификат сервера)
```

Ключевая идея: доверяют не файлу services.crt сам по себе, а издателю, который его подписал. Если клиент заранее доверяет ca.crt, он может проверить цифровую подпись server certificate и построить цепочку до известного корня.

9.1. Создание CA

Выполните:

```
docker compose run --rm openssl req -x509 -newkey rsa:4096 -noenc -days 3650 -sha256 -keyout ca.key -out ca.crt -subj "/CN=SA Local CA" -addext "basicConstraints=critical,CA:TRUE" -addext "keyUsage=critical,keyCertSign,cRLSign" -addext "subjectKeyIdentifier=hash"
```

На первый взгляд команда похожа на создание site-self.crt. Это нормально: корневой CA тоже является самоподписанным сертификатом. Отличие не в самом факте self-signing, а в расширениях, которые прямо указывают, что этот сертификат имеет право выступать CA и подписывать другие сертификаты.

Фрагмент	Что означает	Зачем нужен
rsa:4096	создать RSA private key 4096 бит для CA	учебно отделить долговечный ключ CA от 2048-битного ключа сервера
-days 3650	срок корневого сертификата около 10 лет	корень обычно живет дольше выпускаемых им серверных сертификатов
-keyout ca.key	закрытый ключ центра сертификации	именно им будут подписываться будущие сертификаты; это самый чувствительный файл лаборатории
-out ca.crt	публичный root certificate	его можно передавать клиентам как trust anchor
-subj "/CN=SA Local CA"	имя локального CA	позднее это имя будет видно в поле Issuer серверного сертификата
basicConstraints=critical,CA:TRUE	обозначить сертификат как CA	клиент не должен принимать обычный серверный сертификат как центр сертификации
keyUsage=critical,keyCertSign,cRLSign	разрешить ключу CA подписывать сертификаты и списки отзыва	зафиксировать назначение ключа CA
subjectKeyIdentifier=hash	добавить идентификатор открытого ключа CA	помочь связывать сертификаты с ключами и Authority Key Identifier

Почему basicConstraints и keyUsage помечены critical: critical означает, что программа, которая не понимает смысл расширения, не должна молча игнорировать его. Для ограничений CA это полезно: назначение сертификата влияет на безопасность цепочки.

Почему CA key без пароля только в лаборатории: здесь -ноенс выбран, чтобы одинаково выполнять команды на учебных ПК без интерактивных запросов. В реальной инфраструктуре root CA private key обычно защищают сильнее: паролем, отдельным хранилищем, HSM и часто держат offline.

Проверьте сертификат CA:

```
docker compose run --rm openssl x509 -in ca.crt -noout -subject -issuer -dates -ext basicConstraints,keyUsage,subjectKeyIdentifier
```

Ожидается: Subject и Issuer совпадают, basicConstraints содержит CA:TRUE, а keyUsage разрешает Certificate Sign. Совпадение Subject/Issuer здесь нормально: корневой CA является точкой начала цепочки и подписывает собственный сертификат сам.

9.2. Что именно нужно хранить и кому это можно отдавать

Фрагмент	Что означает	Зачем нужен
ca.crt	публичный сертификат корневого CA	его передают клиентам, которые должны доверять сертификатам этого CA
ca.key	закрытый ключ корневого CA	никому не передается; его компрометация позволяет злоумышленнику выпускать "доверенные" сертификаты от имени CA
services.crt	будущий публичный сертификат reverse проху	отправляется клиенту при TLS-handshake
services.key	будущий закрытый ключ reverse проху	нужен только reverse проху, не должен попадать клиентам

10. Сертификат для трех имен

Reverse proxy будет обслуживать site.sa.test, files.sa.test и status.sa.test на одном IP и порту 443. Один X.509 сертификат может подходить сразу нескольким именам, если каждое из них перечислено в SAN.

Создайте certs\server.ext:

```
subjectAltName=DNS:site.sa.test,DNS:files.sa.test,DNS:status.sa.test
basicConstraints=critical,CA:FALSE
keyUsage=critical,digitalSignature,keyEncipherment
extendedKeyUsage=serverAuth
subjectKeyIdentifier=hash
authorityKeyIdentifier=keyid,issuer
```

Это не отдельная программа и не shell-скрипт. server.ext - фрагмент конфигурации X.509 extensions. Команда openssl x509 ниже прочитает его через -extfile и добавит указанные расширения в выпускаемый сертификат.

Фрагмент	Что означает	Зачем нужен
subjectAltName=...	три допустимых DNS-имени	проверка имени клиента должна найти запрошенное имя в SAN
basicConstraints=critical,CA:FALSE	это конечный сертификат, не CA	не позволять использовать серверный сертификат как центр сертификации
keyUsage=...	разрешенные низкоуровневые криптографические операции	ограничить назначение ключа сервера
extendedKeyUsage=serverAuth	назначение сертификата - аутентификация TLS-сервера	клиент может проверить, что сертификат выпущен для server authentication
subjectKeyIdentifier=hash	идентификатор открытого ключа этого сертификата	служебная связь ключа с сертификатом
authorityKeyIdentifier=keyid,issuer	идентификатор ключа и issuer CA	помочь построить связь с ca.crt

10.1. Создание закрытого ключа и CSR

CSR. подписанный запрос на сертификат. Он доказывает, что запрос сформирован владельцем соответствующего private key, но сам еще не является сертификатом.

```
docker compose run --rm openssl req -new -newkey rsa:2048 -noenc -sha256 -keyout services.key
-out services.csr -subj "/CN=site.sa.test"
```

В этой команде -x509 уже нет. Поэтому req создает не сертификат, а именно CSR. Получаются два разных файла: services.key остается у сервера, services.csr можно передать CA.

Фрагмент	Что означает	Зачем нужен
-new	создать новый CSR	перейти в режим запроса сертификата
-newkey rsa:2048	одновременно создать новый RSA private key	CSR должен содержать соответствующий ему public key
-keyout services.key	сохранить private key сервиса	позднее его использует Nginx
-out services.csr	сохранить PKCS#10 request	этот файл будет входом для операции подписания CA
-subj "/CN=site.sa.test"	Subject запроса	задать его без интерактивного диалога

Почему SAN пока в server.ext, а не в CSR: в этой учебной схеме CA сам задает расширения выпускаемого сертификата при подписании через -extfile. Это наглядно показывает, что CA принимает решение, какие свойства попадут в конечный сертификат. CSR не должен автоматически диктовать CA все расширения.

Проверьте сам CSR:

```
docker compose run --rm openssl req -in services.csr -noout -subject -pubkey -verify
```

Параметр -verify проверяет внутреннюю подпись CSR. Он не проверяет доверие CA - CA еще не участвовал. Проверяется только то, что запрос корректно подписан private key, соответствующим публичному ключу внутри CSR.

10.2. Подпись сертификата локальным CA

```
docker compose run --rm openssl x509 -req -in services.csr -CA ca.crt -CAkey ca.key
-CAcreateserial -out services.crt -days 365 -sha256 -extfile server.ext
```

Теперь появляется настоящая связь "издатель -> сервер". OpenSSL читает CSR, берет открытый ключ сервиса, добавляет расширения из server.ext и подписывает получившийся сертификат private key центра сертификации ca.key.

Фрагмент	Что означает	Зачем нужен
x509 -req	читать на входе PKCS#10 CSR и выпустить X.509 сертификат	превратить запрос services.csr в сертификат
-in services.csr	входной CSR	взять Subject и public key будущего сервера
-CA ca.crt	сертификат издателя	его Subject станет Issuer нового сертификата
-CAkey ca.key	private key издателя	им создается цифровая подпись сертификата
-CAcreateserial	создать файл серийного номера CA, если его еще нет	каждому выпущенному сертификату нужен serial number
-out services.crt	выходной серверный сертификат	его будет использовать reverse proxy
-days 365	срок действия server certificate	серверный сертификат живет меньше root CA
-sha256	digest для подписи	сформировать RSA/SHA-256 подпись
-extfile server.ext	прочитать X.509 extensions из файла	добавить SAN, serverAuth, CA:FALSE и идентификаторы ключей

Почему здесь используется openssl x509, а не полноценный openssl ca: режим -CA/-CAkey у openssl x509 фактически работает как небольшой "micro-CA" и удобен для лаборатории. Он подписывает отдельный CSR, но не ведет полноценную базу выданных/отозванных сертификатов. Для настоящего корпоративного CA нужны учет, политики, отзыв и автоматизация.

Что такое файл ca.srl: при -CAcreateserial OpenSSL хранит номер, использованный CA. Serial number является частью сертификата и должен отличать один сертификат от другого у данного issuer. В отчете содержимое ca.srl обычно неинтересно, но важно понимать, почему файл появился.

10.3. Проверка сертификата

```
docker compose run --rm openssl x509 -in services.crt -noout -subject -issuer -serial -dates -ext
subjectAltName,basicConstraints,extendedKeyUsage
docker compose run --rm openssl verify -CAfile ca.crt services.crt
```

Первая команда читает поля конечного сертификата. Вторая строит цепочку доверия, используя ca.crt как доверенный корень. Ожидается services.crt: OK.

Фрагмент	Что означает	Зачем нужен
verify	подкоманда проверки цепочки сертификатов	отделить проверку доверия от простого просмотра файла
-CAfile ca.crt	использовать указанный CA как trust anchor	не зависеть от системного хранилища Windows
services.crt	проверяемый сертификат	убедиться, что подпись и цепочка сходятся

Что именно означает OK: OpenSSL смог построить и проверить цепочку services.crt -> ca.crt для заданного контекста проверки. Это не означает автоматически, что любое DNS-имя подойдет: имя отдельно проверяется по SAN клиентом при подключении.

11. Возврат внутренних сервисов к HTTP

После демонстрации прямого HTTPS сертификат больше не нужен контейнеру site. Он снова становится обычным внутренним HTTP-сервисом:

```
site:
  image: nginx:stable-alpine
  volumes:
    - ./site:/usr/share/nginx/html:ro
  restart: unless-stopped
  networks:
    - sa-net
```

Публикации 8081 и 8443 удаляются. Контейнер site остается доступен другим участникам сети sa-net как http://site:80. Это повторяет главный принцип предыдущей практики: внутреннему сервису не нужен host port, если к нему обращается другой контейнер той же Docker-сети.

12. Настройка Reverse Proxy на порты 80 и 443

Теперь services.crt и services.key подключаются только к reverse proxy. Порт 80 служит для redirect, а порт 443 - для TLS и проксирования к backend.

Замените proxy/default.conf:

```
server {
    listen 80;
    server_name site.sa.test files.sa.test status.sa.test;
    return 308 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name site.sa.test;
    ssl_certificate /etc/nginx/certs/services.crt;
    ssl_certificate_key /etc/nginx/certs/services.key;

    location / {
        proxy_pass http://site:80;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
    }
}

server {
    listen 443 ssl;
    server_name files.sa.test;
    ssl_certificate /etc/nginx/certs/services.crt;
    ssl_certificate_key /etc/nginx/certs/services.key;

    location / {
        proxy_pass http://filebrowser:80;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
    }
}

server {
    listen 443 ssl;
    server_name status.sa.test;
    ssl_certificate /etc/nginx/certs/services.crt;
    ssl_certificate_key /etc/nginx/certs/services.key;

    location / {
        proxy_pass http://uptime-kuma:3001;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto https;
    }
}
```

Три HTTPS server-блока используют одну и ту же пару services.crt/services.key. Это возможно, потому что SAN сертификата содержит все три имени. Nginx выбирает блок по имени клиента, а TLS дает единый защищенный вход.

Фрагмент	Что означает	Зачем нужен
server_name	какому имени соответствует server-блок	на одном IP/443 выбрать нужный backend
proxy_pass http://site:80	создать новое HTTP-соединение от proxy к service name site	это TLS termination: наружу HTTPS, внутри HTTP
proxy_set_header Host \$host	передать исходное имя сайта backend	приложение видит, к какому имени обратился клиент
X-Real-IP / X-Forwarded-For	передать сведения о клиентском IP	backend видит клиента за reverse proxy

Фрагмент	Что означает	Зачем нужен
X-Forwarded-Proto https	сообщить backend, что внешний клиент использовал HTTPS	приложение может корректно формировать ссылки/redirect
Upgrade / Connection	явно передать заголовки переключения протокола	нужно WebSocket-соединению Uptime Kuma через reverse proxy

Почему proxy_pass остается http://: TLS уже завершился на Nginx proxy. После расшифровки Nginx формирует новый запрос к backend. В этой лаборатории внутренняя Docker-сеть считается доверенной учебной зоной, поэтому второй TLS-слой не добавляется.

13. Финальная конфигурация проху в Compose

```
proxy:
  image: nginx:stable-alpine
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - ./proxy/default.conf:/etc/nginx/conf.d/default.conf:ro
    - ./certs/services.crt:/etc/nginx/certs/services.crt:ro
    - ./certs/services.key:/etc/nginx/certs/services.key:ro
  depends_on:
    - site
    - filebrowser
    - uptime-kuma
  restart: unless-stopped
  networks:
    - sa-net
```

Порт 80 и 443 публикуются только у проху. Внутренние контейнеры доступны по Docker DNS и внутренним портам. Сертификат и ключ монтируются read-only только в проху.

Принцип минимального доступа: site, FileBrowser и Uptime Kuma не нуждаются в services.key, поэтому не должны его видеть. Секретный ключ подключается только к компоненту, который фактически завершает TLS.

14. Финальный compose.yaml

```
services:
  site:
    image: nginx:stable-alpine
    volumes:
      - ./site:/usr/share/nginx/html:ro
    restart: unless-stopped
    networks: [sa-net]

  filebrowser:
    image: gtstef/filebrowser:stable
    volumes:
      - ./shared:/folder
      - ./filebrowser-data:/home/filebrowser/data
    restart: unless-stopped
    networks: [sa-net]

  uptime-kuma:
    image: louislam/uptime-kuma:2
    volumes:
      - uptime-kuma-data:/app/data
    restart: unless-stopped
    networks: [sa-net]

  proxy:
    image: nginx:stable-alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./proxy/default.conf:/etc/nginx/conf.d/default.conf:ro
      - ./certs/services.crt:/etc/nginx/certs/services.crt:ro
```

```

- ./certs/services.key:/etc/nginx/certs/services.key:ro
depends_on:
- site
- filebrowser
- uptime-kuma
restart: unless-stopped
networks: [sa-net]

openssl:
  image: alpine/openssl:3.5.8
  profiles: [tools]
  volumes:
  - ./certs:/certs
  working_dir: /certs
  networks: [sa-net]

networks:
  sa-net:

volumes:
  uptime-kuma-data:

```

Почему openssl не запускается вместе со всеми: profiles: [tools] исключает служебный сервис из обычного docker compose up -d. При этом явная команда docker compose run openssl ... может запустить именно его для разовой операции.

15. Запуск и проверка Nginx

```

docker compose config
docker compose up -d
docker compose ps
docker compose exec proxy nginx -t

```

Фрагмент	Что означает	Зачем нужен
docker compose config	разобрать YAML и показать итоговую Compose-модель	поймать ошибки структуры до запуска
docker compose up -d	создать/обновить сервисы и запустить в фоне	применить финальную конфигурацию
docker compose ps	показать состояние и опубликованные порты	убедиться, что 80/443 принадлежат только проху
docker compose exec proxy nginx -t	запустить проверку Nginx в работающем проху	проверить синтаксис и доступность cert/key

В docker compose ps у проху должны быть опубликованы 80 и 443. У site, filebrowser и uptime-kuma host ports отсутствуют.

16. Проверка HTTP -> HTTPS

Без изменения hosts выполните:

```

curl.exe -I --resolve site.sa.test:80:127.0.0.1 http://site.sa.test/
curl.exe -I --resolve files.sa.test:80:127.0.0.1 http://files.sa.test/
curl.exe -I --resolve status.sa.test:80:127.0.0.1 http://status.sa.test/

```

Ожидается код 308 и Location: https://... . В этом месте TLS еще не начинается: клиент приходит по HTTP на порт 80 и получает обычный HTTP-ответ "перейди на HTTPS".

HTTP redirect. ответ 3xx с заголовком Location. Сервер не пересылает тот же TCP-сеанс на 443, а просит клиента выполнить новый запрос по другому URL.

17. Проверка HTTPS и сертификата

17.1. Проверка без доверия CA

```

curl.exe -vk --resolve site.sa.test:443:127.0.0.1 https://site.sa.test/

```

Параметр `--resolve` одновременно дает curl IP-адрес и сохраняет правильное имя `site.sa.test`. Поэтому имя участвует и в HTTP Host, и в SNI во время TLS-handshake. `-k` временно отключает строгую проверку, чтобы увидеть, что сервер вообще отвечает по TLS.

17.2. Проверка с явным указанием локального CA

```
curl.exe --cacert certs/ca.crt --resolve site.sa.test:443:127.0.0.1 https://site.sa.test/
curl.exe --cacert certs/ca.crt --resolve files.sa.test:443:127.0.0.1 https://files.sa.test/
curl.exe --cacert certs/ca.crt --resolve status.sa.test:443:127.0.0.1 https://status.sa.test/
```

Теперь `-k` отсутствует. Вместо отключения проверки клиент получает конкретный доверенный корень через `--cacert`. Это принципиально другой подход: curl должен проверить подпись сертификата, срок, назначение и соответствие имени.

Фрагмент	Что означает	Зачем нужен
<code>--cacert certs/ca.crt</code>	добавить конкретный CA-файл в доверие для этого запуска curl	проверить цепочку без изменения Windows certificate store
<code>--resolve name:443:127.0.0.1</code>	локально сопоставить имя с IP для текущей команды	не требовать прав администратора на hosts
<code>https://site.sa.test/</code>	URL содержит проверяемое DNS-имя	клиент сравнивает это имя с SAN сертификата

Проверка имени важна: если использовать URL `https://127.0.0.1/`, сертификат с `DNS:site.sa.test` не подходит для IP `127.0.0.1`. Криптографическая подпись может быть полностью правильной, но identity check все равно должен завершиться ошибкой.

17.3. Проверка сертификата через OpenSSL s_client

```
docker compose run --rm openssl s_client -connect proxy:443 -servername site.sa.test -CAfile /certs/ca.crt -verify_return_error
```

`s_client` - диагностический TLS-клиент OpenSSL. Он удобен тем, что показывает детали handshake, версию TLS, cipher suite и цепочку сертификатов.

Фрагмент	Что означает	Зачем нужен
<code>s_client</code>	запустить тестовый TLS-клиент	исследовать handshake напрямую
<code>-connect proxy:443</code>	соединиться с сервисом проху внутри Docker-сети	не нужен опубликованный адрес Windows
<code>-servername site.sa.test</code>	отправить SNI	Nginx выбирает server-блок как при реальном обращении к этому имени
<code>-CAfile /certs/ca.crt</code>	использовать наш root CA	проверить цепочку
<code>-verify_return_error</code>	считать ошибку проверки фатальной	не продолжать handshake как будто проверка была успешной

В выводе найдите subject, issuer SA Local CA, negotiated TLS version, cipher и Verify return code: 0 (ok).

18. Доверие сертификату в Windows

Чтобы браузер полностью доверял `services.crt` без предупреждений, Windows/браузер должен доверять корневому `ca.crt`. На учебных ПК это может быть запрещено политиками, и установка доверенного CA не является обязательной частью работы.

Основной безопасный способ проверки в лаборатории - `curl.exe --cacert certs/ca.crt`. Он создает доверие только для конкретной команды и не меняет системные настройки.

Если преподаватель разрешил импорт: импортируется только `ca.crt`. Нельзя импортировать или распространять `ca.key`. После учебной работы локальный CA не следует оставлять доверенным без необходимости.

19. Как один порт 443 обслуживает три сайта

Все три учебных имени могут указывать на один IP `127.0.0.1` и один порт `443`. Nginx различает их благодаря SNI, а после TLS - также Host header.

```

site.sa.test    ---\
files.sa.test   ----+--> 127.0.0.1:443 --> Nginx
status.sa.test  ---/

```

Проверьте это без hosts тремя командами --resolve из раздела 17.2. Во всех случаях IP и порт одинаковые; меняется только имя. Именно имя уходит в SNI, и Nginx выбирает соответствующий server_name.

Почему один сертификат подходит всем: SAN services.crt перечисляет все три DNS-имени. SNI выбирает конфигурацию сервера, а SAN отвечает на другой вопрос: имеет ли предъявленный сертификат право представлять это имя.

20. Обновление мониторинга Uptime Kuma

Uptime Kuma находится внутри Docker-сети. Внутренние мониторы http://site:80 и http://filebrowser:80 можно оставить. Они показывают работоспособность backend независимо от внешнего TLS.

Получаются два уровня диагностики:

- внутренний monitor отвечает на вопрос: жив ли backend внутри sa-net;
- curl с --resolve и --cacert отвечает на вопрос: работает ли вся внешняя цепочка порт 443 -> TLS -> SNI -> reverse proxy -> backend.

Диагностическая логика: если Uptime Kuma показывает backend UP, но внешняя HTTPS-проверка падает, проблема вероятнее находится в проху, сертификате, порте 443, SNI или клиентской проверке доверия/имени.

21. Что именно защищает HTTPS в этой схеме

Участок	Протокол	Состояние
Клиент -> Reverse Proxy	HTTPS/TLS	зашифрован
Reverse Proxy -> site	HTTP	не шифруется внутри sa-net
Reverse Proxy -> FileBrowser	HTTP	не шифруется внутри sa-net
Reverse Proxy -> Uptime Kuma	HTTP + WebSocket	не шифруется внутри sa-net

Для локального Docker Desktop такая схема учебно оправдана: цель - понять termination и централизованный TLS. В распределенной инфраструктуре backend может находиться на другом хосте, и тогда внутренний участок также может защищаться TLS, VPN или mTLS.

22. Типичные неисправности и диагностика

Симптом	Вероятная причина	Проверка	Исправление
Docker не публикует 80/443	порт уже занят	docker compose up; netstat -ano findstr :80	остановить конфликтующий сервис или выбрать другой host port
Nginx не запускается	неверный путь к cert/key	docker compose logs proxy; nginx -t	проверить volume и имена файлов
key values mismatch	cert и key из разных пар	nginx -t	подключить правильный services.key
unknown authority	клиент не доверяет SA Local CA	-k работает, --cacert нет/не указан	использовать правильный ca.crt
certificate name mismatch	имя URL отсутствует в SAN	openssl x509 ... -ext subjectAltName	перевыпустить cert с нужным SAN
502 Bad Gateway	backend недоступен или проху_pass неверен	docker compose ps/logs; DNS имя сервиса	исправить backend или проху_pass
HTTPS есть, HTTP redirect нет	ошибка server-блока listen 80	curl -I --resolve ...	исправить listen 80 / return 308
status.sa.test не работает	WebSocket headers не передаются	логи проху и Kuma	проверить Upgrade/Connection

23. Учебные неисправности

Выполняйте по одной неисправности. После каждой верните рабочее состояние и объясните, на каком уровне возникла проблема: DNS/имя, TLS, reverse proxy или backend.

1. В proxy/default.conf измените путь services.crt на несуществующий файл. Выполните docker compose exec проху nginx -t, прочитайте ошибку и восстановите путь.
2. В server-блоке site.sa.test измените проху_pass на https://site:80. Объясните, почему внешний TLS-handshake проходит, но Nginx не может установить корректное TLS-соединение с backend, который слушает обычный HTTP.
3. Остановите filebrowser командой docker compose stop filebrowser. Проверьте curl --cacert --resolve для files.sa.test. Объясните, почему TLS может установиться успешно, а HTTP-ответом будет ошибка проху.

4. Выполните `curl.exe --cacert certs/ca.crt https://127.0.0.1/` и сравните с запросом через `--resolve site.sa.test`. Объясните роль SAN.
5. Временно замените `services.key` другим RSA-ключом. Выполните `nginx -t` и найдите сообщение о несовпадении сертификата и ключа.

24. Самостоятельное расширение

Выполните не менее трех пунктов:

1. Добавьте четвертое имя `app.sa.test` в SAN и четвертый `server`-блок Nginx. Для проверки используйте `curl --resolve`, не изменяя `hosts`.
2. Создайте отдельный серверный сертификат сроком действия 7 дней и найдите `notAfter`.
3. Сравните `curl.exe -vk` и `curl.exe --cacert certs/ca.crt`. Объясните, какая проверка отключена в первом случае и выполняется во втором.
4. Найдите в `openssl s_client` согласованную версию TLS и cipher suite.
5. Остановите `site`, но оставьте `proxu`. Объясните, почему TLS-handshake остается успешным, а получение страницы заканчивается ошибкой `backend`.
6. Проверьте, что внутренние `site`, `filebrowser` и `uptime-kuma` не имеют опубликованных Windows-портов.
7. Сформулируйте, какие файлы из `certs` нужно перенести вместе с `reverse proxu` на другой компьютер и какие из них являются секретными.
8. Объясните, почему перенос `sa.key` на `reverse proxu` не нужен и ухудшает безопасность.

25. Контрольные вопросы

1. Чем HTTP отличается от HTTPS?
2. Какую роль выполняет TLS?
3. Что хранится в серверном сертификате?
4. Почему закрытый ключ нельзя передавать вместе с отчетом?
5. Что означает самоподписанный сертификат?
6. Почему браузер/клиент не доверяет самоподписанному сертификату автоматически?
7. Что такое CA?
8. Чем `sa.crt` отличается от `sa.key`?
9. Почему корневой `sa.crt` самоподписанный?
10. Что такое CSR и содержит ли он private key?
11. Для чего нужен SAN?
12. Почему CN недостаточно считать единственным источником имени сервера?
13. Что такое SNI?
14. Как три имени используют один IP и один порт 443?
15. Что означает `listen 443 ssl` в Nginx?
16. Для чего используются `ssl_certificate` и `ssl_certificate_key`?
17. Что означает TLS termination?
18. Почему `proxu_pass` в финальной конфигурации использует `http://`?
19. Для чего порт 80 оставлен после перехода на HTTPS?
20. Что делает `return 308 https://$host$request_uri`?
21. Почему `curl -k` не является нормальным способом эксплуатации HTTPS?
22. Для чего используется `--cacert`?
23. Для чего используется `curl --resolve` в этой лаборатории?

24. Что проверяет openssl verify -CAfile ca.crt services.crt?
25. Что делает параметр -CAkey при выпуске сертификата?
26. Для чего нужен server.ext?
27. Что означает CA:TRUE и CA:FALSE?
28. Что обычно означает 502 после успешного TLS-handshake?
29. Почему сертификат может быть правильно подписан CA, но не подходить конкретному DNS-имени?
30. Какие преимущества дает централизованное хранение server certificate/private key на reverse proxy?

26. Требования к отчету

Отчет должен содержать:

- тему и цель работы;
- схему итоговой инфраструктуры с указанием места TLS termination;
- фрагмент compose.yaml с сервисами proxy и openssl;
- финальный proxy/default.conf;
- скриншот docker compose ps с портами 80 и 443 только у reverse proxy;
- вывод проверки ca.crt, где видны CA:TRUE и key usage;
- вывод проверки services.crt с Subject, Issuer, сроком действия и SAN;
- вывод openssl verify -CAfile ca.crt services.crt;
- результат HTTP redirect через curl --resolve;
- результат HTTPS-проверки через curl --cacert и --resolve;
- фрагмент openssl s_client с Verify return code: 0 (ok);
- описание одной намеренно созданной неисправности и способа ее устранения;
- ответы на контрольные вопросы.

В отчет не включать: ca.key и services.key. Закрытые ключи не являются материалом отчета. Если требуется показать, что файл существует, достаточно имени файла или вывода списка каталога без содержимого.

27. Завершение работы

Остановить контейнеры без удаления постоянных данных:

```
docker compose down
```

Сертификаты, ключи и CSR находятся в каталоге certs на Windows и docker compose down их не удаляет, потому что это bind mount обычной папки проекта.

Учебные ключи: не используйте созданные в этой работе CA и server key для реальных интернет-сервисов. В производственной инфраструктуре сертификаты обычно выпускаются доверенным публичным или корпоративным CA, а выпуск и обновление часто автоматизируются.

28. Итоговый конспект

Понятие	Кратко
HTTPS	HTTP внутри TLS-соединения.
TLS	Шифрование, целостность и проверка подлинности.
Certificate	Публичный X.509 документ с public key и именами.
Private key	Секрет; подтверждает владение ключевой парой.
Self-signed	Сертификат подписан соответствующим собственным ключом.
CA	Издатель сертификатов и корень доверия.
ca.crt	Публичный сертификат CA; может распространяться.
ca.key	Секретный ключ CA; подписывает сертификаты.
CSR	Запрос на сертификат с public key и подписью владельца key.
SAN	Список имен/IP, для которых сертификат действителен.
SNI	Передача имени сайта во время TLS-handshake.
TLS termination	HTTPS завершается на reverse proxy.
--resolve	Локальное сопоставление имени и IP только для одного запуска curl.
--cacert	Указать доверенный CA только для данного запуска curl.

Port 80	HTTP и redirect на HTTPS.
Port 443	Стандартный HTTPS.
502	Прoxy доступен, но backend не отвечает корректно.

29. Используемые источники

Команды и параметры в методичке составлены из официального синтаксиса соответствующих инструментов. Это важно: при изменении версии не следует искать готовую "магическую строку" - нужно открыть справку нужной подкоманды и проверить параметры.

- OpenSSL Documentation. openssl-req: <https://docs.openssl.org/3.5/man1/openssl-req/>
- OpenSSL Documentation. openssl-x509: <https://docs.openssl.org/3.5/man1/openssl-x509/>
- OpenSSL Documentation. openssl-verify: <https://docs.openssl.org/3.5/man1/openssl-verify/>
- OpenSSL Documentation. openssl-s_client: https://docs.openssl.org/3.5/man1/openssl-s_client/
- OpenSSL Documentation. X.509 V3 extensions: https://docs.openssl.org/3.5/man5/x509v3_config/
- OpenSSL Documentation. openssl-ca: <https://docs.openssl.org/3.5/man1/openssl-ca/>
- NGINX Documentation. Configuring HTTPS servers: https://nginx.org/en/docs/http/configuring_https_servers.html
- NGINX Documentation. ngx_http_proxy_module: https://nginx.org/en/docs/http/ngx_http_proxy_module.html
- NGINX Documentation. WebSocket proxying: <https://nginx.org/en/docs/http/websocket.html>
- Docker Docs. docker compose run: <https://docs.docker.com/reference/cli/docker/compose/run/>
- Docker Docs. Using profiles with Compose: <https://docs.docker.com/compose/how-tos/profiles/>
- Docker Docs. Networking in Compose: <https://docs.docker.com/compose/how-tos/networking/>

Актуальность: методичка переработана в октябре 2026 года. Для OpenSSL-команд используется синтаксис ветки 3.5; вместо устаревшего `-nodes` используется `-noenc`. Служебный контейнер OpenSSL остается изолированным инструментом и не является постоянно работающей частью инфраструктуры.