Руководство по инсталляции серверных компонент системы ROBIN

Выпуск 1.3.3.5834.0rc1

ROBIN RPA Team

дек. 24, 2020

Содержание:

Глава 1

Сервер приложений WildFly

1.1 Настройка Datasource сервера WildFly

Глава 2

Информационная безопасность

2.1 Создание и установка сертификатов сервера приложений WildFly (JBoss EAP)

2.1.1 Сертификат сервера

Установить сертификат сервера можно двумя способами:

- Сформировать запрос на хосте, где установлен сервер приложений, подписать запрос в удостоверяющем центре, установить сертификат на хост.
- Установить готовый сертификат. В этом случае, приватный ключ не генерируется на сервере а импортируется вместе с сертификатом.

Для примера приведем обе процедуры, с использованием в качестве удостоверячего центра, частный УЦ от майкрософт, из состава Microsoft Windows Server 2012R.

Для манипулирования хранилищами и сертификатами на Linux хосте, будет использована утилита командной строки keytool.

Создание запроса на сертификат на сервере приложений

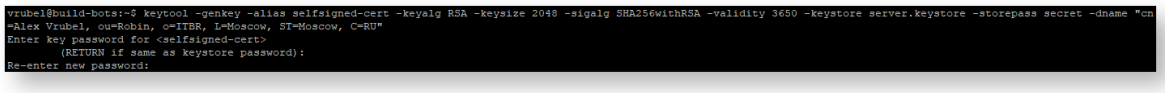
Спецификой создания запроса, который будет обрабатываться на УЦ от майкрософта, является то, что запрос должен быть обязательно подписан сертификатом (хотя бы самоподписанным сертификатом). Подпись запроса просто с использованием ключа, без сертификата, не будет принята в УЦ от майкрософта.

Поэтому, на первом шаге, генерируем самоподписанный серитификат.

Генерация ключевой пары

```
keytool -genkey -alias selfsigned-cert -keyalg RSA -keysize 2048 -sigalg SHA256withRSA -
↪validity 3650 -keystore server.keystore -storepass secret -dname "cn=Alex Vrubel, ou=Robin,␣
↪o=ITBR, L=Moscow, ST=Moscow, C=RU"
```

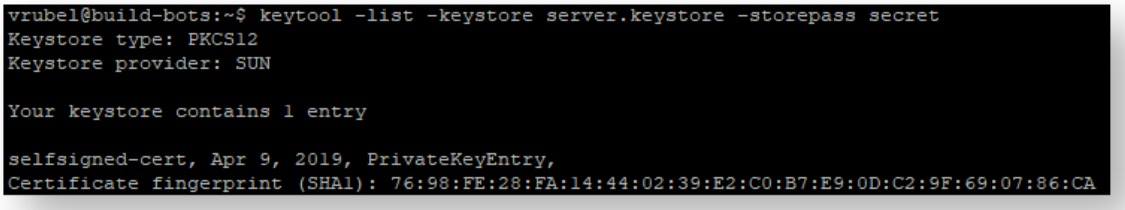
При генерации будет спрошен пароль, которым будет защищен доступ к приватному ключу сертификата.



Просмотреть список объектов в хранилище

```
keytool -list -keystore server.keystore -storepass secret
```

Видно, что объекту будет присвоен алиас



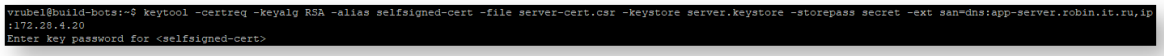
Генерация запроса на сертификат

В качестве алиаса, надо указать алиас, под которым в хранилище размещен ключ, которым подписывается создаваемый запрос

Генерация подписанного запроса на сертификат

```
keytool -certreq -keyalg RSA -alias selfsigned-cert -file server-cert.csr -keystore server.
↪keystore -storepass secret -ext san=dns:app-server.robin.it.ru,ip:172.28.4.20
```

В процессе запроса будет запрошен пароль к приватному ключу сертификата, которым подписывается запрос.



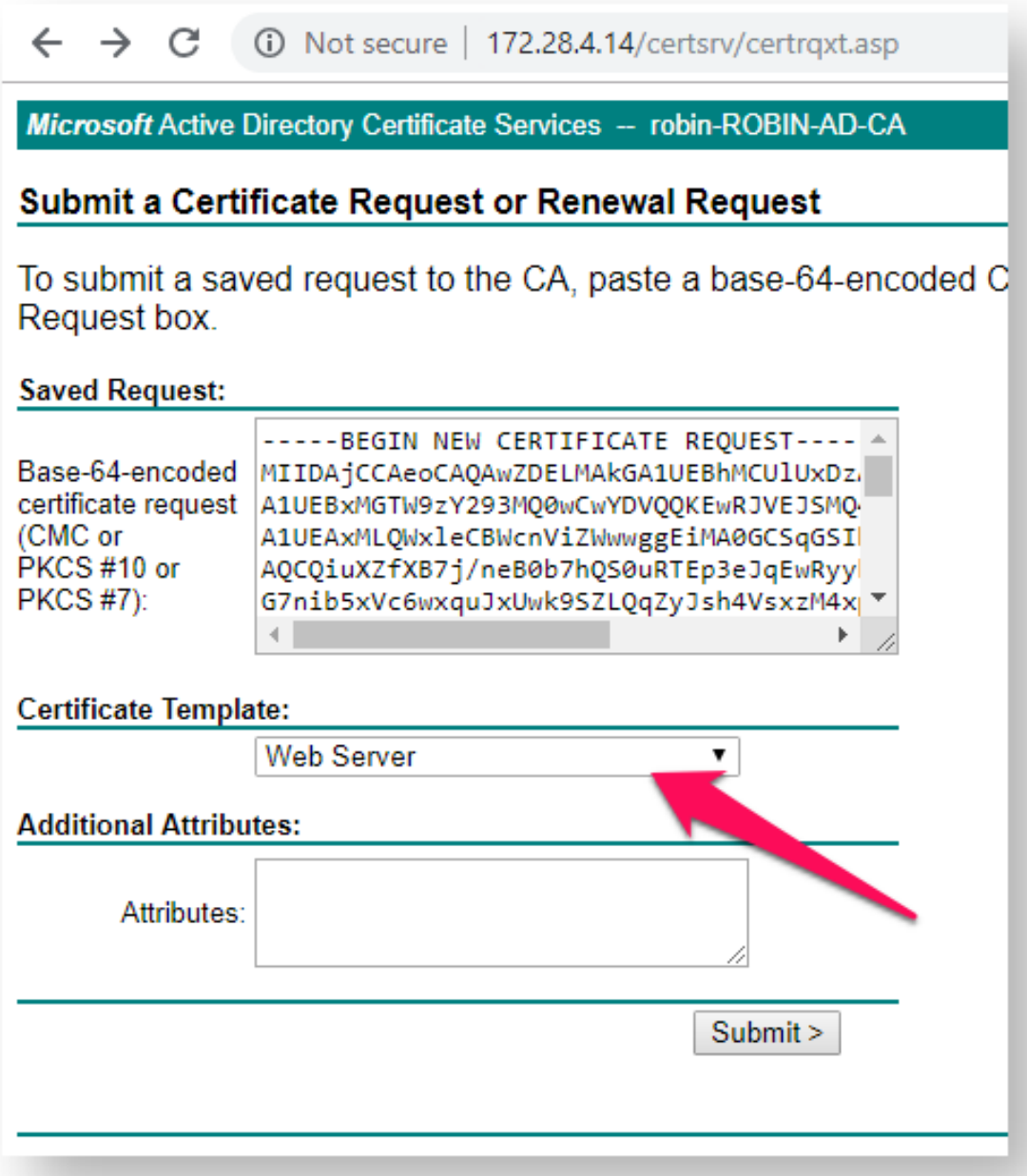
Обратите внимание, что добавлено расширение сертификата **subject alternative name (SAN)**, которое позволяет (обязательное требования для сертификатов серверов) указать доменное имя и/или IP адрес, по которому происходит обращение к серверу (для которого создается сертификат). В качестве доменного имени можно указать localhost.

Результатом является файл запроса на выпуск сертификата **server-cert.csr**, контент которого выглядит примерно так

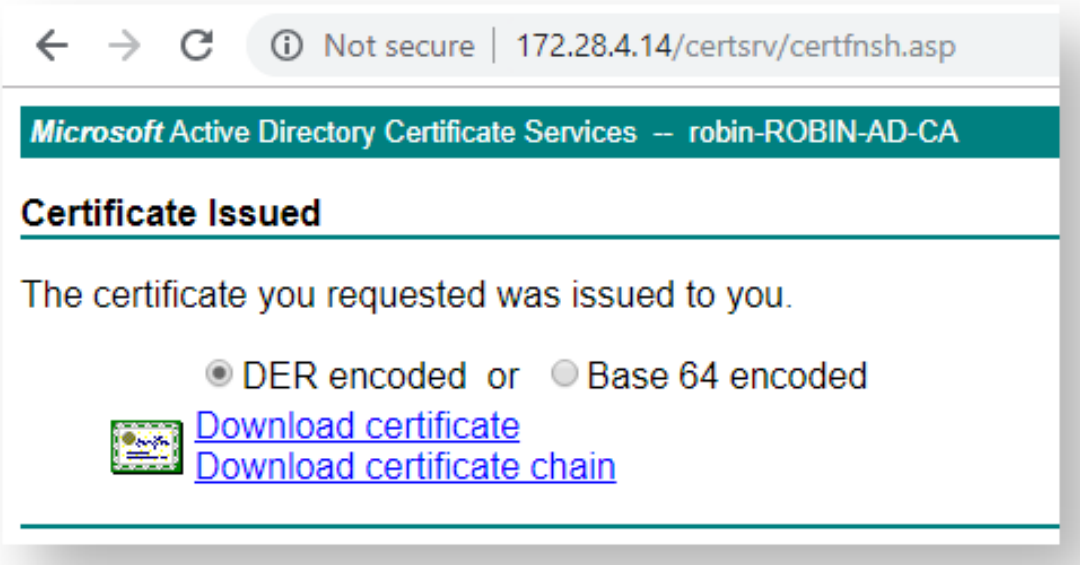
```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIDAjCCAeoCAQAwZDELMAkGA1UEBhMCU1UxDzANBgNVBAgTBk1vc2NvdzEPMA0G
A1UEBxMGTW9zY293MQ0wCwYDVQQKEwRJVEJSMQ4wDAYDVQQLEwVSb2JpbjEUMBIG
A1UEAxMLQWxleCBWcnViZWwwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQCQiuXZfXB7j/neB0b7hQS0uRTEp3eJqEwRyyksILV8Uj5rRuLyy2eQpujfvsBQ
G7nib5xVc6wxquJxUwk9SZLQqZyJsh4VsxzM4xpvrzS7cKCUEDLkAvzZga2iP9Wl
Iu9WiKY4rItmG/xojlxf2FfMXXVbpirc/iCTiIhQPuOlYY/P9aOmpiOmds8bE/HY
xCMgTtr+4/9Izu3dgA5Dn7EdgcIPmPP74dhswfjtgSiJN7PYJBLSnjRliZ+ryEvA
suU6umu7K3M2JIOxjMoTF4+pOvqRcYpuCTPILLOGu4iVXcIO6lwU8cNnAGel+0Od
upxq7c2qaRu0/iaIXeEcGBaDAgMBAAGgWTBXBgkqhkiG9w0BCQ4xSjBIMCcGA1Ud
EQQgMB6CFmFwcC1zZXJ2ZXIucm9iaW4uaXQucnWHBKwcBBQwHQYDVR0OBBYEFIh/
TEehOMZLWt0qHnKl5qzxglwkMA0GCSqGSIb3DQEBCwUAA4IBAQBADwv0zBPj59hz
cRHiAEcOu5yfN/MYZQXv6Cj8wkl7aCWN47LnfYc80rtHlG5PXkUJ2xWZRjM6Pvqo
pWbJQPdYubVfszeBQ8IOaSDZuVTMkj1CrIX8QAUV9Qd2LiOX+eGo4H4FTT3egJPm
C0PmnFIFWOd9KoNjwxGJfjnKkAJUab9xJewQ8uHQ3t4wEmtil6G5toepwl4qkPle
kZPVOD20WJiMFLJ9Xkv2hL4hHmsR26KjLZqGaYQDVl9VflJ2Zehm+blwlFekr/KV
2fqMlWnUJ2dmlj7k7g0xP/0HQNxaUKYapzWZrX67M+lUrAY6v6USS+ure9EIzFCY
rDZRzIE7
-----END NEW CERTIFICATE REQUEST-----
```

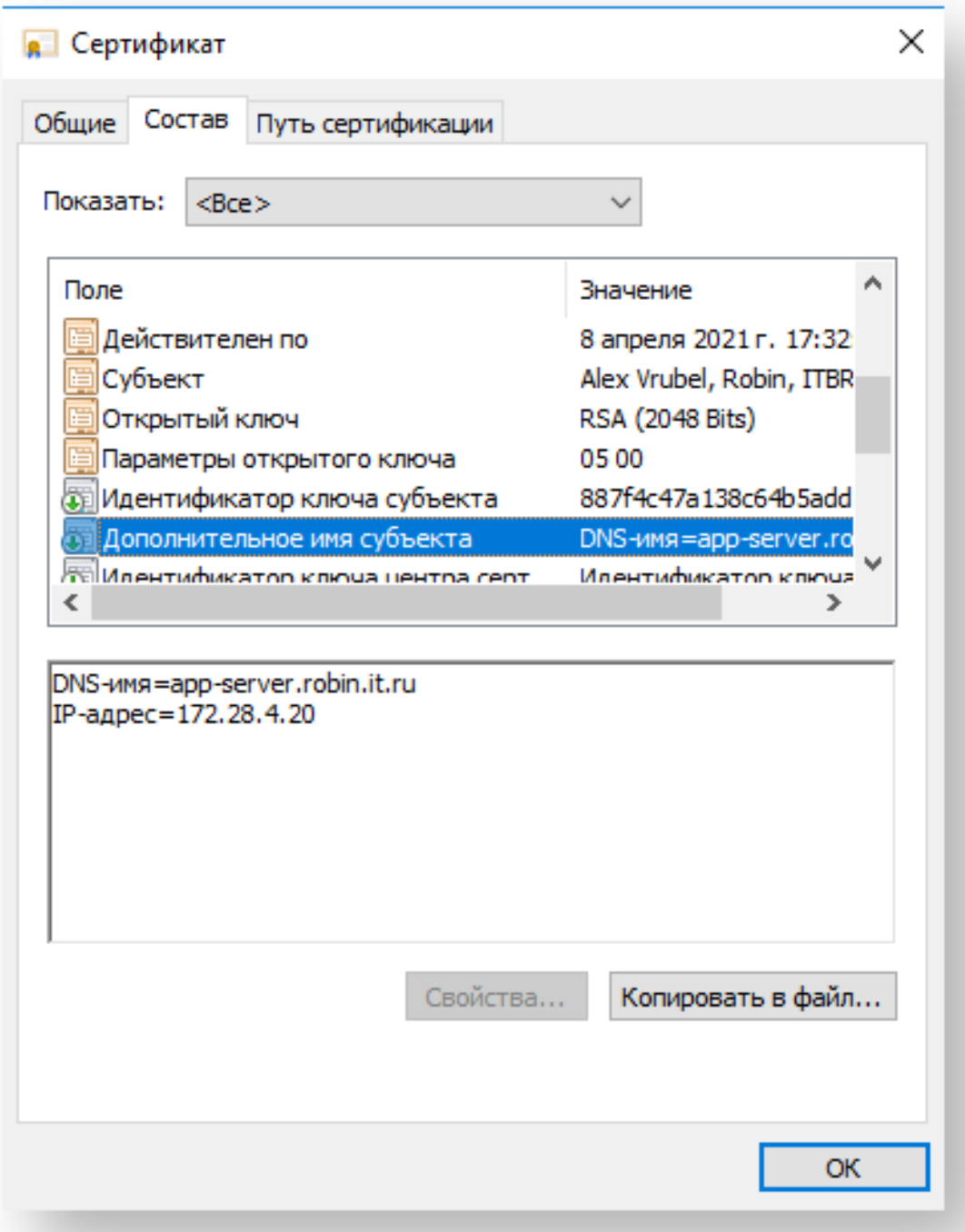
Выпуск сертификата в удостоверяющем центре

Далее, выпускаем сертификат в УЦ. В качестве примера приведены скриншоты с УЦ, установленного хосте тестового контролера домена. У пользователя должен быть доступ к серверу.



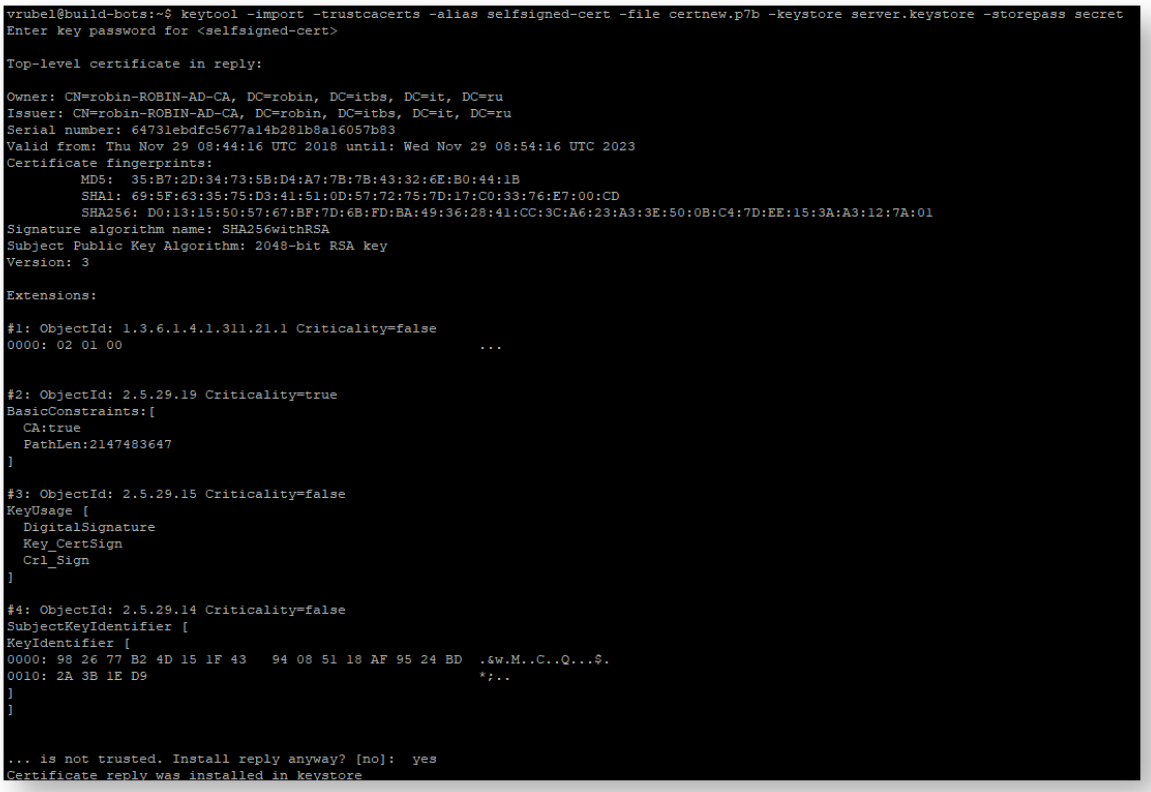
Как результат выпуска сертификата, можем скачать файл выпущенного сертификата или цепочку сертификатов для выпущенного сертификата.


Not secure | 172.28.4.14/certsrv/certfnsh.asp
Microsoft Active Directory Certificate Services -- robin-ROBIN-AD-CA
Certificate Issued
The certificate you requested was issued to you.
DER encoded or Base 64 encoded
Download certificate
Download certificate chain



Импорт выпущенного сертификата в хранилище

```
keytool -import -trustcacerts -alias selfsigned-cert -file certnew.p7b -keystore server.
↪keystore -storepass secret
```



Установка на сервер приложений готового сертификата

Импорт сертификата из pfx

Можно импортировать сертификат с ключом из PFX файла.

```
keytool -importkeystore -srckeystore app-server.robin.it.ru.pfx -srcstoretype pkcs12 -
↪deststoretype JKS
```

При этом алиас будет установлен в какое то дикое значение. Но мы можем поменять имя алиаса и установить новый пароль для алиаса (читай - для доступа к приватному ключу сертификата), используя полезные команды, которые описаны тут

https://blog.blundellapps.co.uk/tut-change-alias-passwords-of-your-android-keystore/

Регистрация сертификатов на сервере приложения

Копируем хранилище, содержащее серверный сертификат, в файловую структуру сервера приложений

```
cp server.keystore $JBOSS_HOME/standalone/configuration
```

Далее конфигурируем подсистему безопасности Elytron, как описано в оригинальной статье

```
batch

/subsystem=elytron/key-store=demoKeyStore:add(path=server.keystore,relative-to=jboss.server.
↪config.dir, credential-reference={clear-text=secret},type=JKS)

/subsystem=elytron/key-manager=demoKeyManager:add(key-store=demoKeyStore,credential-reference=
↪{clear-text=Qwerty123})
```

```
/subsystem=elytron/server-ssl-context=demoSSLContext:add(key-manager=demoKeyManager,protocols=[
↪"TLSv1.2"])

# This is only needed if WildFly uses by default the Legacy security realm (по умолчанию это␣
↪так)
/subsystem=undertow/server=default-server/https-listener=https:undefine-
↪attribute(name=security-realm)

/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
↪context,value=demoSSLContext)

run-batch

reload
```

Обратите внимание на **Qwerty123** - это пароль от приватного ключа сертификата, что лежит в хранилище server.keystore, а **secret** это пароль от самого кейстора.

Оринигал инструкции

https://support.embotics.com/support/solutions/articles/8000035243-generating-and-installing-an-ssl-certificate-with-a

2.1.2 Сертификаты клиента

Для аутентификации сервером клиентов по сертификату, необходимо клиентские сертификаты (без приватного ключа) разместить в хранилище на сервере, и зарегестрировать данное хранилище в качестве хранилища с доверенными сертификатами. Получить клиентские сертификаты можно двумя способами - 1. сгенерировать их на сервере и отдать пользователю (в нашем случае клиентскому приложению) с приватным ключом, 2. импортировать уже существующий сертификат в хранилище.

Генерация клиентского сертификата на сервере

Пример генерации клиентского сертификата на сервере

```
keytool -genkey -keystore client.keystore -storepass secret -validity 365 -keyalg RSA -keysize␣
↪2048 -storetype pkcs12 -dname "cn=Sergey Garnov, ou=Robin, o=ITBR, L=Moscow, ST=Moscow, C=RU"
keytool -exportcert -keystore client.keystore -storetype pkcs12 -storepass secret -keypass␣
↪secret -file client.crt
keytool -import -file client.crt -alias Garnov -keystore client.truststore -storepass secret
keytool -importkeystore -srckeystore client.keystore -srcstorepass secret -destkeystore␣
↪GarnovCert.p12 -srcstoretype PKCS12 -deststoretype PKCS12 -deststorepass secret
```

Импорт сертификата клиента

```
keytool -import -file vrubel.cer -alias quickstartUser -keystore client.truststore -storepass␣
↪secret
```

TODO: проверить с вновь выпущенными сертификатами

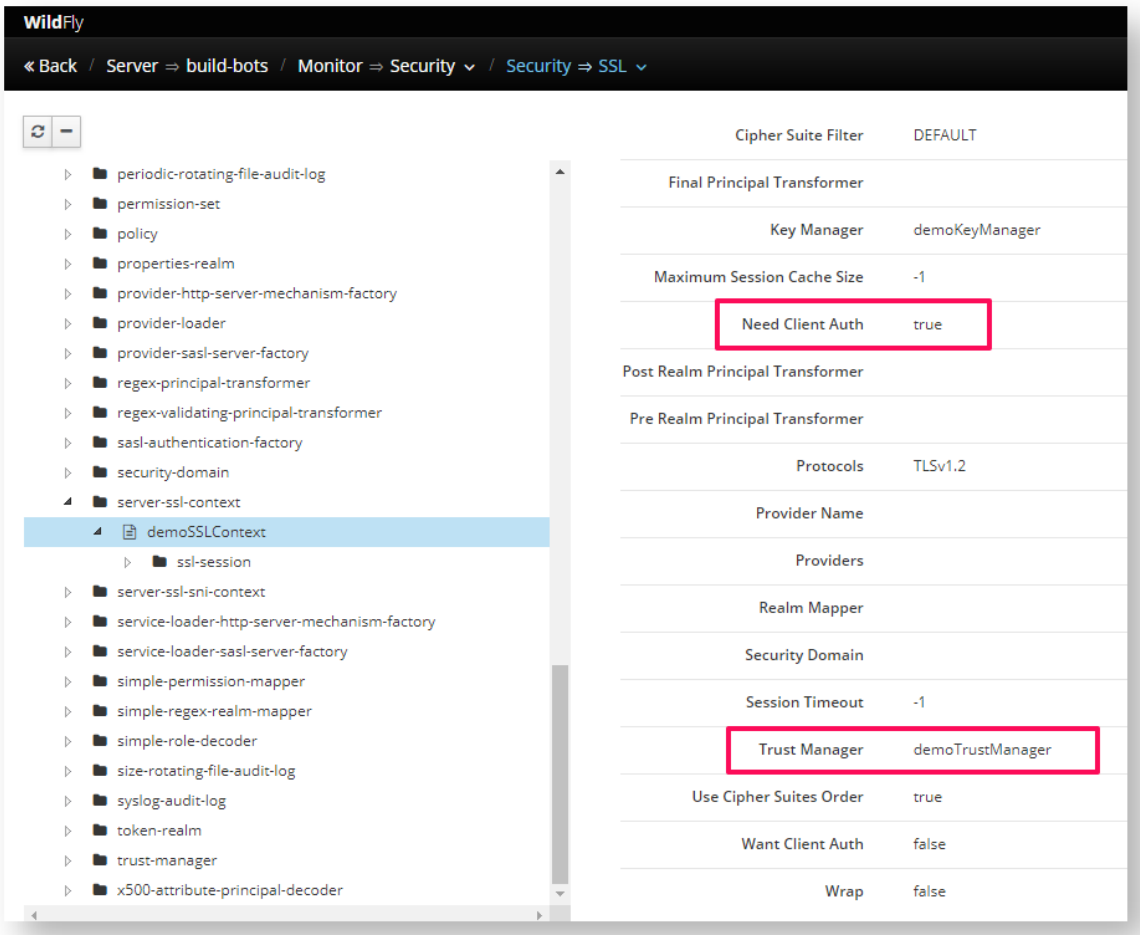
Включения режима двусторонней аутентификации

Создание хранилища доверенных клиентских сертификатов

```
/subsystem=elytron/key-store=demoTrustStore:add(path=client.truststore,relative-to=jboss.
↪server.config.dir,type=JKS,credential-reference={clear-text=secret})

/subsystem=elytron/trust-manager=demoTrustManager:add(key-store=demoTrustStore)
```

Пример включения обязательной проверки клиентского сертификата, через интерфейс настройки сервера.



Решение проблемы невозможности найти библиотеки libssl, при переходе на использование openssl

Установить пакеты

libcrypto++6

libssl1.1

запускать с указанием места расположения libssl

```
./standalone.sh -b=172.28.4.20 -bmanagement=172.28.4.20 -Dorg.wildfly.openssl.path=/usr/lib/
↪x86_64-linux-gnu/
```

Импорт сертификата клиента в формате PFX

Пароли ключа и контейнера (целевых) должны совпадать

Пример импорта сертификата в формате PFX в хранилище

```
keytool -importkeystore -srckeystore vrubel.pfx -srcstorepass Qwerty123 -destkeystore client.
↪keystore -srcstoretype PKCS12 -deststoretype PKCS12 -deststorepass Qwerty123 -destkeypass␣
↪Qwerty123
```

2.2 Настройка PostgreSQL на аутентификацию клиентов только по сертификату

2.2.1 Оригинальная документация по теме

Главными отправными точками являются страницы документации PostgreSQL:

17.9. Secure TCP/IP Connections with SSL

18.3.2. Security and Authentication

И вспомогательные:

19.3.9. Certificate Authentication

19.2. User Name Maps

19.1. The pg_hba.conf File

20.12. Certificate Authentication

2.2.2 Введение

Для включения режима аутентификации клиентов по сертификатам, потребуется произвести следующие изменения в настройках сервера postgresql:

- Разместить на сервере файлы: сертификата сервера, ключа сертификата сервера, доверенный корневой сертификат (для валидации клиентских сертификатов)
- Внести изменения в файлы конфигурации: **postgresql.conf**, **pg_hba.conf**, **pg_ident.conf**

2.2.3 Установка сертификатов на сервер

Необходимо разместить в каталоге **pgdata** трех файла:

- **postgresql.crt**
  Сертификат сервера
- **postgresql.key**
  Ключ сертификата сервера (без шифрования контента)
- **robin_CA.crt**
  Доверенный корневой сертификат (для валидации клиентских сертификатов)

Внести изменения в файлы конфигурации:

- **postgresql.conf**
  Укажем размещений файлов сертификатов и разрешим их использование
- **pg_hba.conf**

Включим требование на обязательное использование клиентом сертификата при аутентификации

- **pg_ident.conf**
  Смапируем имена клиентов на логины БД

Подготовка файлов сертификатов

Создание файла серверного сертификата

Для подготовки серверного сертификата воспользуемся процедурой описанной в разделе «Сертификат сервера» страницы *Создание и установка сертификатов сервера приложений WildFly (JBoss EAP)*, включая момент скачивания файла сертификата в формате CER, со страницы УЦ. Переименуем скачанный файл в postgresql.cer. Теперь необходимо преобразовать формат файла в CRT. Для этого воспользуемся утилитой openssl:

Конвертация формата файла сертификата CER → CRT

```
openssl x509 -inform DER -in postgresql.cer -out postgresql.crt
```

Таким образом мы получили первый из необходимых файлов - **postgresql.crt**

Создание файла приватного ключа серверного сертификата

Теперь необходимо получить незашифрованный приватный ключ. В данный момент он размещен в хранилище - файле server.keystore. Для его извлечения так же воспользуемся утилитой openssl (в две операции):

1) Экспортируем приватный ключ из хранилища в файл формата PKCS12

```
keytool -v -importkeystore -srckeystore server.keystore -storepass secret -srcalias␣
↪selfsigned-cert -destkeystore postgresql.p12 -deststoretype PKCS12
```

2) Извлечем приватный ключ из файла PKCS12 в незашифрованном виде

```
openssl pkcs12 -in postgresql.p12 -nocerts -nodes -out postgresql.key
```

Таким образом мы получили второй из необходимых файлов - **postgresql.key**

Создание файла клиентского корневого доверенного сертификата

Третий файл скачиваем с частного Центра сертификации по ссылке: http://172.28.4.14/certsrv/certcarc.asp

Далее конвертируем формат файла сертификата (CER → CRT):

```
openssl x509 -inform DER -in robin_CA.cer -out robin_CA.crt
```

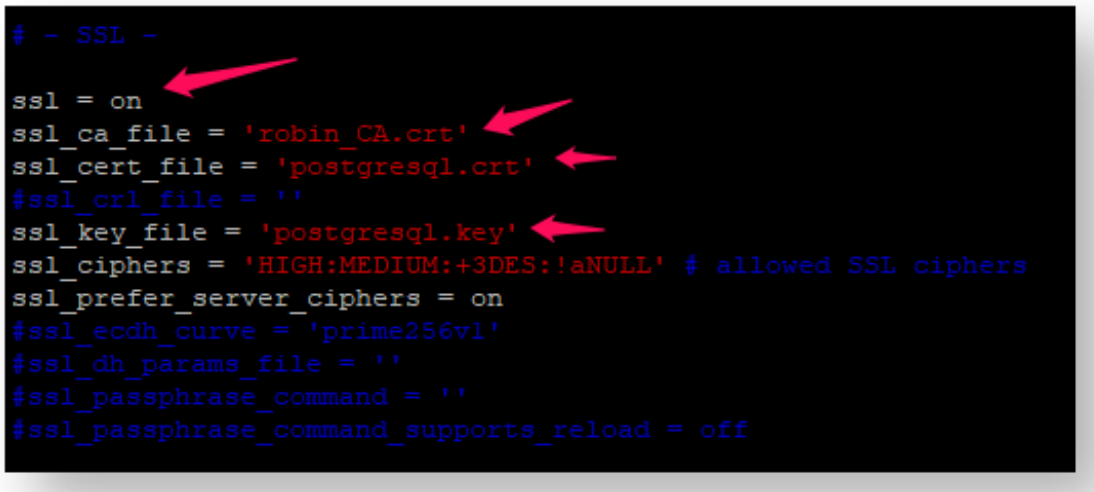
Таким образом мы получили третий и последний из необходимых файлов - **robin_CA.key**

Теперь, разместив все полученные файлы в директории pgdata, перейдем к правке конфигурационных файлов сервера БД.

Внесение правок в файлы конфигурации

postgresql.conf (Укажем размещений файлов сертификатов и разрешим их использование)

В секции SSL, необходимо ключить использование SSL и указать имена файлов сертификатов и приватный ключ



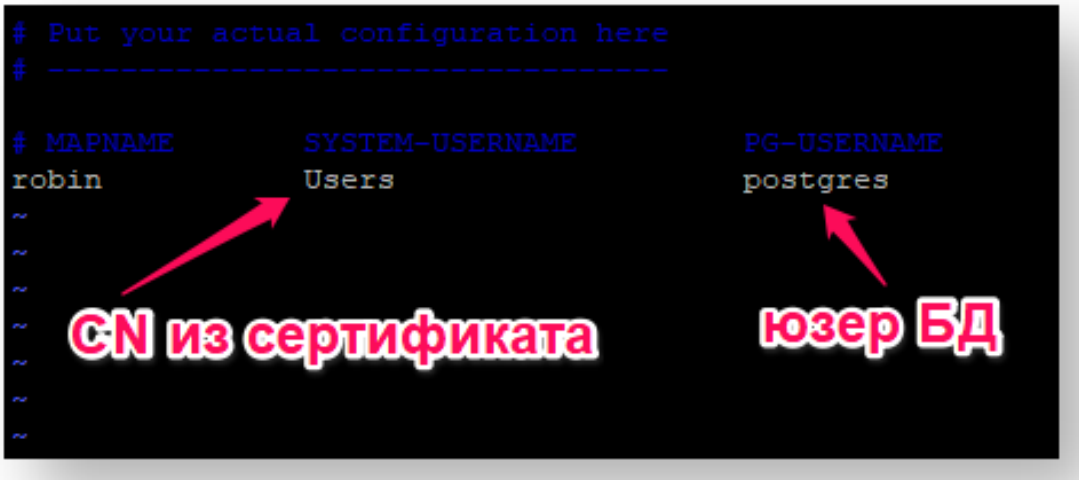
pg_hba.conf (Включим требование на обязательное использование клиентом сертификата при аутентификации)

Отключим аутентификацию по паролю (закомментировав соответствующую строку)

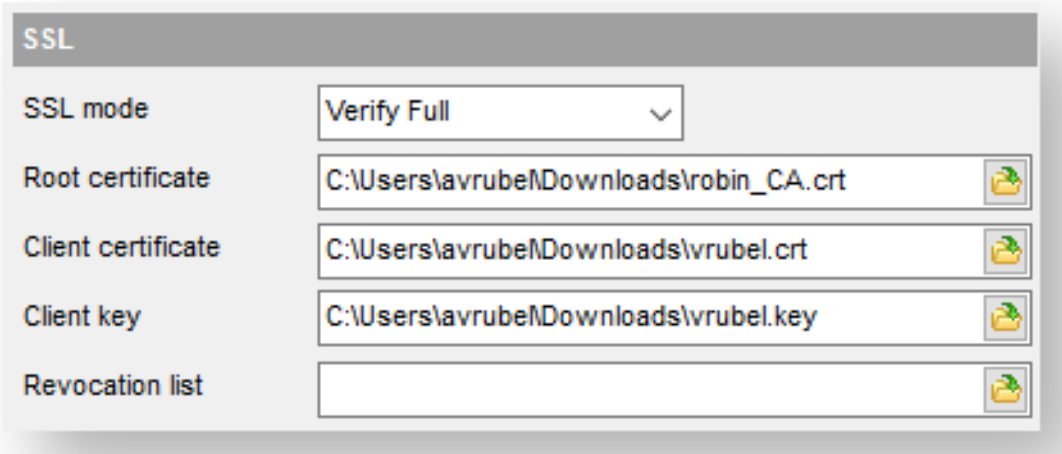
И включим аутентификацию по сертификату, добавив строку с ключом hostssl. В методе аутентификации указано: «cert» - аутентификация по сертификату, «map=robin» - использовать карту мапирования «robin» (из файла pg_ident.conf), «clientcert=1» - требовать от клиента предоставить клиентский сертификат.

pg_ident.conf (Мапинг имени клиента на логин БД)

Пример, приведенный на скриншоте, не очень удачный. Используется имя группы, вместо логина юзера. Но так получилось, потому что в сертификате было указано два поля CN, в одном было имя группы, в другом логи пользователя. Сервер подхватил имя группы, поэтому пришлось прописать сюда его. Что может быть удобно в некоторых сценариях организации коннекта к БД.



2.2.4 Пример использования клиентом сертификата для аутентификации



Клиентский сертификат получаем аналогично сертификату сервера.

Создаем самоподписанный сертификат, подписываем его в УЦ (шаблон сертификата - User), конвертируем в crt.

Приватный ключ получаем из соответствующего самоподписанного сертификата.

Указание доменного имени при коннекте

Т.к. в CN серверного сертификата указано конкретное доменное имя, клиент должен обязательно использовать данное доменное имя при установлении соединения

Предоставление файлов клиентского сертификата и ключа

Как видно на скриншоте, при настройке коннекта на клиенте, указываются пути к файлам сертификата в формате CRT и к файлу с незашифрованным приватным ключом.

Предоставление файла доверенного корневого серверного сертификата

Как видно на скриншоте, при настройке коннекта на клиенте, указывается путь к доверенному корневому сертификату сервера, который будет использован для валидации серверного сертификата.

2.2.5 Полезные команды OpenSSL

Конвертация формата файла сертификата

```
openssl x509 -inform DER -in postgresql.cer -out postgresql.crt
```

Экспорт приватного ключа в PKCS12

```
keytool -v -importkeystore -srckeystore server.keystore -storepass secret -srcalias selfsigned-
↪cert -destkeystore postgresql.p12 -deststoretype PKCS12
```

Извлечение приватного ключа в незашифрованном виде

```
openssl pkcs12 -in postgresql.p12 -nocerts -nodes -out postgresql.key
```

2.3 Настройка Wildfly для подключения к БД по SSL

2.3.1 Введение

Перед настройкой защищенного соединения сначала необходимо настроить БД на работу по SSL и проверку клиентских сертификатов. Описание на странице *Настройка PostgreSQL на аутентификацию клиентов только по сертификату*.

В статье описано как получить необходимые сертификаты и ключи: **postgresql.crt**, **postgresql.key**, **pg-client.crt**, **pg-client.key**.

Сервер приложений Wildfly для подключения будет использовать клиентские сертификат и ключ, полученные при настройке БД.

2.3.2 Создание клиентских trustore и keystore

Wildfly будет использовать truststore для валидации сертификата БД, а keystore для хранения клиентского ключа и сертификата.

1) С помощью утилиты keytool создаем wildfly.truststore и добавляем в него доверенный сертификат postgresql.crt:

```
keytool -import -file postgresql.crt -alias postgresql -keystore wildfly.truststore -storepass␣
↪secret
```

Будет спрошено доверять ли сертификату: `Trust this certificate? [no]: yes`

2) С помощью OpenSSL создаем контейнер pkcs12 для клиентского ключа и сертификата:

```
openssl pkcs12 -export -in pg-client.crt -inkey pg-client.key -out pg-client.p12 -name pg-
↪client
```

3) Создаем wildfly.keystore из полученного pkcs12:

```
keytool -importkeystore -deststorepass secret -destkeystore wildfly.keystore -srckeystore pg-
↪client.p12 -srcstoretype PKCS12 -srcstorepass secret -alias pg-client
```

4) размещаем полученные truststore и keystore в нужном месте на хосте

2.3.3 Конфигурация datasource

Настройка datasource описана на странице Настройка Datasource

Для включения SSL в *connection-url* добавляются соответствующие параметры:

```
<connection-url>jdbc:postgresql://localhost:5432/robin?currentSchema=orchestrator&amp;ssl=true&
↪amp;sslmode=verify-full&amp;sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory</connection-
↪url>
```

`ssl=true` - использовать SSL соединение с БД

`sslmode=verify-full` - шифрование данных, аутентификация по сертификату

`sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory` - использовать keystore и truststore JVM для установления защищенного соединения (см. *Переменные среды JVM*)

Исправить существующие datasource можно через веб консоль Wildfly, в файле standalone.xml или с помощью Wildfly CLI:

```
/subsystem=datasources/data-source=OrchestratorDS/:write-attribute(name=connection-url,value=
↪"jdbc:postgresql://localhost:5432/robin?currentSchema=orchestrator&ssl=true&sslmode=verify-
↪full&sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory")

/subsystem=datasources/data-source=QuartzDS/:write-attribute(name=connection-url,value=
↪"jdbc:postgresql://localhost:5432/robin?currentSchema=quartz&ssl=true&sslmode=verify-full&
↪sslfactory=org.postgresql.ssl.DefaultJavaSSLFactory")
```

2.3.4 Переменные среды JVM

Нужно добавить переменные JAVA_OPTS сервера Wildfle и указать путь к полученным клиентским хранилищам.

Для Linux (файл JBOSS_HOME/bin/standalone.conf):

```
JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.keyStore=C:\wildfly-16.0.0.
↪Final\standalone\configuration\wildfly.keystore -Djavax.net.ssl.keyStorePassword=secret"

JAVA_OPTS="$JAVA_OPTS -Djavax.net.ssl.trustStore=C:\wildfly-16.0.0.
↪Final\standalone\configuration\wildfly.truststore -Djavax.net.ssl.trustStorePassword=secret"
```

Для Windows (файл JBOSS_HOME/bin/standalone.conf.bat) :

```
set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.keyStore=C:\wildfly-16.0.0.
↪Final\standalone\configuration\wildfly.keystore -Djavax.net.ssl.keyStorePassword=secret"

set "JAVA_OPTS=%JAVA_OPTS% -Djavax.net.ssl.trustStore=C:\wildfly-16.0.0.
↪Final\standalone\configuration\wildfly.truststore -Djavax.net.ssl.trustStorePassword=secret"
```

2.3.5 Полезные ссылки

Пример настройки защищенного соединения между WildFly и БД: https://mirocupak.com/secure-database-connection-with-wildfly/

Настройка SSL на WildFly: http://www.mastertheboss.com/jboss-server/jboss-security/complete-tutorial-for-configuring-ssl-https-on-wildfly

Документация postgresql по ssl: https://www.postgresql.org/docs/11/libpq-ssl.html

Параметры JDBC драйвера postgresql: https://jdbc.postgresql.org/documentation/head/connect.html

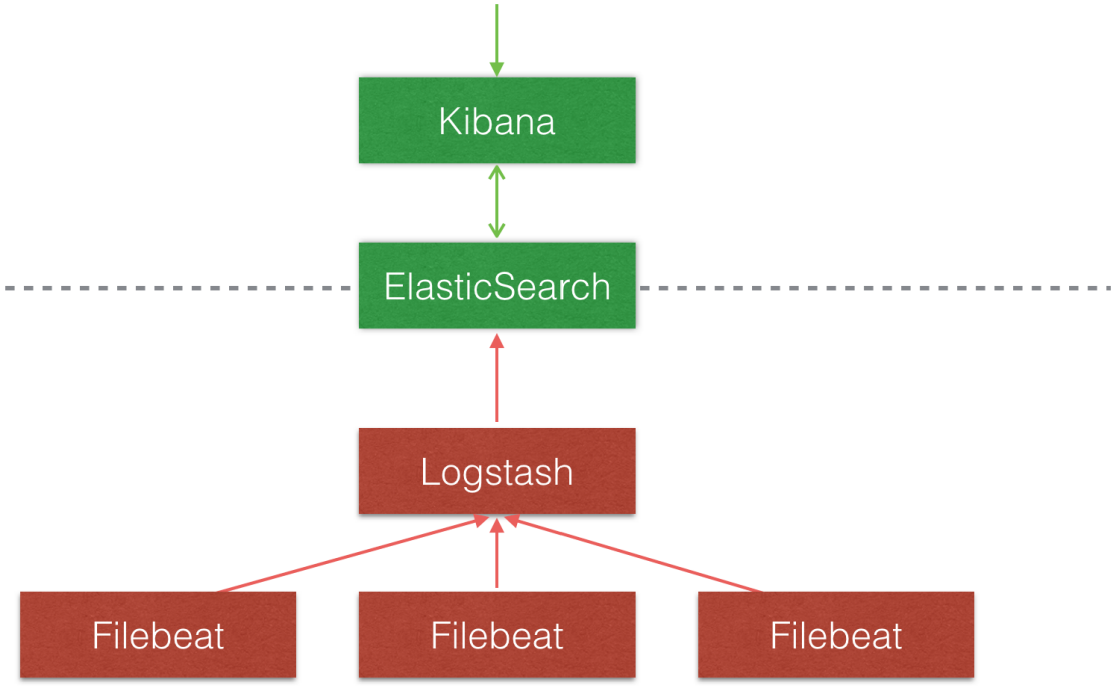
Глава 3

Логгирование

3.1 Хранение логов (Elastic Stack)

Для централизованного сбора и анализа логов системы используются инструменты Elastic Stack.

Схема сбора логов:



Filebeat — служба, считывающая логи из файлов

Logstash — приложение первичной обработки данных

Elasticsearch — база данных с поисковым движком

Kibana — web-интерфейс

3.1.1 Конфигурация Logstash, Elasticsearch, Kibana

На данный момент приложения запущены на сервере 172.28.1.57

http://172.28.1.57:9200 - API Elasticsearch

http://172.28.1.57:5601 - Kibana

Приложения запускаются сервисом Docker Compose.

Структура каталогов и файлов:

```
/opt
    └── elk
        ├── docker-compose.yml
        ├── postgresql-42.2.6.jar
        ├── data
        └── settings
            ├── logstash.yml
            ├── pipelines.yml
            ├── filebeat.config
            ├── postgres.config
            ├── .logstash_jdbc_last_run
            ├── startup.options
            ├── jvm.options
            └── log4j2.properties
```

Для запуска на новом сервере:

1) установить Docker Compose

2) создать папки, скопировать файлы и выдать права доступа к ним

`docker-compose.yml` - файл конфигурации Docker Compose

`postgresql-42.2.6.jar` - jdbc драйвер postgresql, необходим для плагина импорта данных из базы Оркестратора

/data - каталог данных Elasticsearch

Создать каталог, установить принадлежность группе 1000 и выдать полные права:

```
sudo mkdir data

sudo chmod g+rwx data

sudo chgrp 1000 data
```

В файле **sysctl.conf** прописать параметр ядра vm.max_map_count=262144

```
$ grep vm.max_map_count /etc/sysctl.conf vm.max_map_count=262144
```

Информация по запуску Elasticsearch в докере: https://www.elastic.co/guide/en/elasticsearch/reference/7.3/docker.html

/**settings** - содержит конфигурационные файлы Logstash:

`logstash.yml` logstash.yml - конфигурация Logstash

`pipelines.yml` - конфигурация пайпланов

`filebeat.config` - конфигурация пайплайна сбора данных из лог-файлов

`postgres.config` - конфигурация пайплайна импорта данных из базы Оркестратора

`.logstash_jdbc_last_run` - файл, содержащий дату последнего изменения базы Оркестратора (создастся автоматически)

`startup.options` - стартовые параметры запуска

`jvm.options` - конфигурация JVM

`log4j2.properties` - параметры логгирования

Подробнее о конфигурации Logstash: https://www.elastic.co/guide/en/logstash/current/configuration.html

---

Примечание

При запуске на новом сервере нужно исправить адрес коннекта к базе в файле postgres.config

---

3.1.2 Служба Filebeat

На данный момент служба запущена на машине с Win10 - 172.28.0.134

Агенты Filebeat нужно запускать на тех машинах, с которых планируется собирать логи.

На Win10 агент запускается следующим образом:

1) скачать zip-архив с официального сайта, распаковать в нужную директорию

2) установить filebeat как службу

Выполнить в PowerShell:

```
PS > cd 'C:\Program Files\Filebeat'
PS C:\Program Files\Filebeat> .\install-service-filebeat.ps1
```

Предварительно исправить пути в файле install-service-filebeat.ps1

3) исправить конфигурационный файл filebeat.yml

В нем нужно указать директории, откуда считывать логи, паттерн начала новой записи, указать logstash в выходном канале.

Пример файла: `filebeat.yml`

4) запустить службу через «Панель управления»

Подробная инструкция по запуску и конфигурации: https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-installation.html

3.1.3 Kibana - инструмент визуализации данных

http://172.28.1.120:5601

Чтобы визуализировать сохраненные данные, нужно указать шаблон индекса.

Для этого нужно в кибане открыть вкладку Management и добавить Index Pattern вида *filebeat-7.2.0-**

После этого во вкладке *Discover* можно просматривать сохраненные записи.

В дальнейшем эти данные можно визуализировать в виде диаграмм и объединять их в *dashboards*.

Перенос сохраненных визуализаций и дашбордов из одного экземпляра кибаны в другой:

Экспорт: https://www.elastic.co/guide/en/kibana/7.x/dashboard-api-export.html

Импорт: https://www.elastic.co/guide/en/kibana/7.x/dashboard-import-api.html

После импорта данных нужно будет исправить идентификаторы Index Pattern у всех визуализаций. Веб интерфейс сам предложит поменять Index Pattern при открытии визуализации.

Узнать идентификаторы Index Pattern-ов можно запросом:

```
GET .kibana/_search?q=type:index-pattern&size=100
```

3.1.4 Рекомендации для сохранения логов в файлы

1. важные для сбора лог-файлы нужно хранить в конкретной директории;
2. другие логи нужно вынести в другую папку или их расширение должно отличаться;
3. расширение лог-файлов не должно удаляться или изменяться;
4. время в логах писать по UTC.