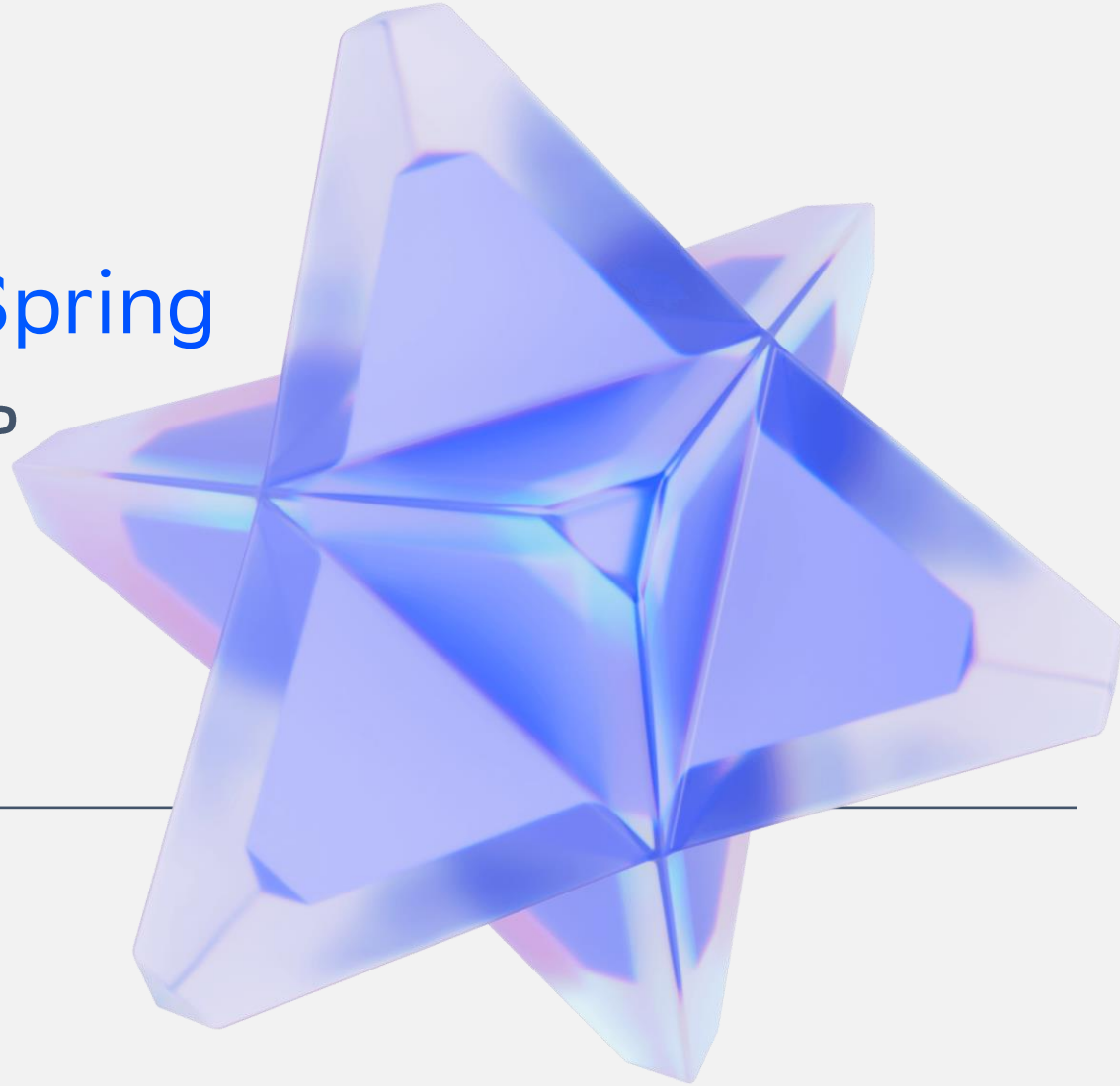
SBER TECH

Секреты в Java-сервисах на Spring
Откуда взять и как обновлять «на горячую»?



Андрей Чернов, Java архитектор в СберТехе

chernovaf@mail.ru

chernovaf

О себе

Андрей Чернов

17 лет опыта разработки
Начинал на C/C++

Кандидатская диссертация по индексам в БД

SBER TECH

8 лет в СберТехе
Сейчас Java-архитектор

План

1. Каждому сервису нужны секреты
    1.1. Что такое секреты и зачем они нужны?
    1.2. Секреты нужны любому современному сервису
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
4. Подводим итоги

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
    - 2.1. Получение секретов из HashiCorp Vault
    - 2.2. Примеры секретов для конкретного сервиса на Java
3. Применение секретов в сервисах на Java
4. Подводим итоги

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
4. Подводим итоги

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java

    3.1. Применение секретов при старте сервиса
    3.2. Hot reload секретов при их изменении
4. Подводим итоги

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
4. Подводим итоги

План

1. Каждому сервису нужны секреты
    1.1. Что такое секреты и зачем они нужны?
    1.2. Секреты нужны любому современному сервису
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
4. Подводим итоги

Что такое секреты?

Параметры

нужные для работы сервиса

Не простые

представляют интерес для злоумышленников

В чем суть секретности?

Возможна атака

- вывод из строя
- кража конфиденциального
- и т. п.

В чем суть секретности?

Возможна атака

- вывод из строя
- кража конфиденциального
- и т. п.

Чревато потерей

- денег
- нервов
- времени
- репутации

В чем суть секретности?

Возможна атака

- вывод из строя
- кража конфиденциального
- и т. п.

Чревато потерей

- денег
- нервов
- времени
- репутации

Нужно продумать
безопасную работу с секретами

Секреты на примере Platform V SessionsData

Детали на Joker22

доклад про архитектуру
и историю создания:

Секреты на примере Platform V SessionsData

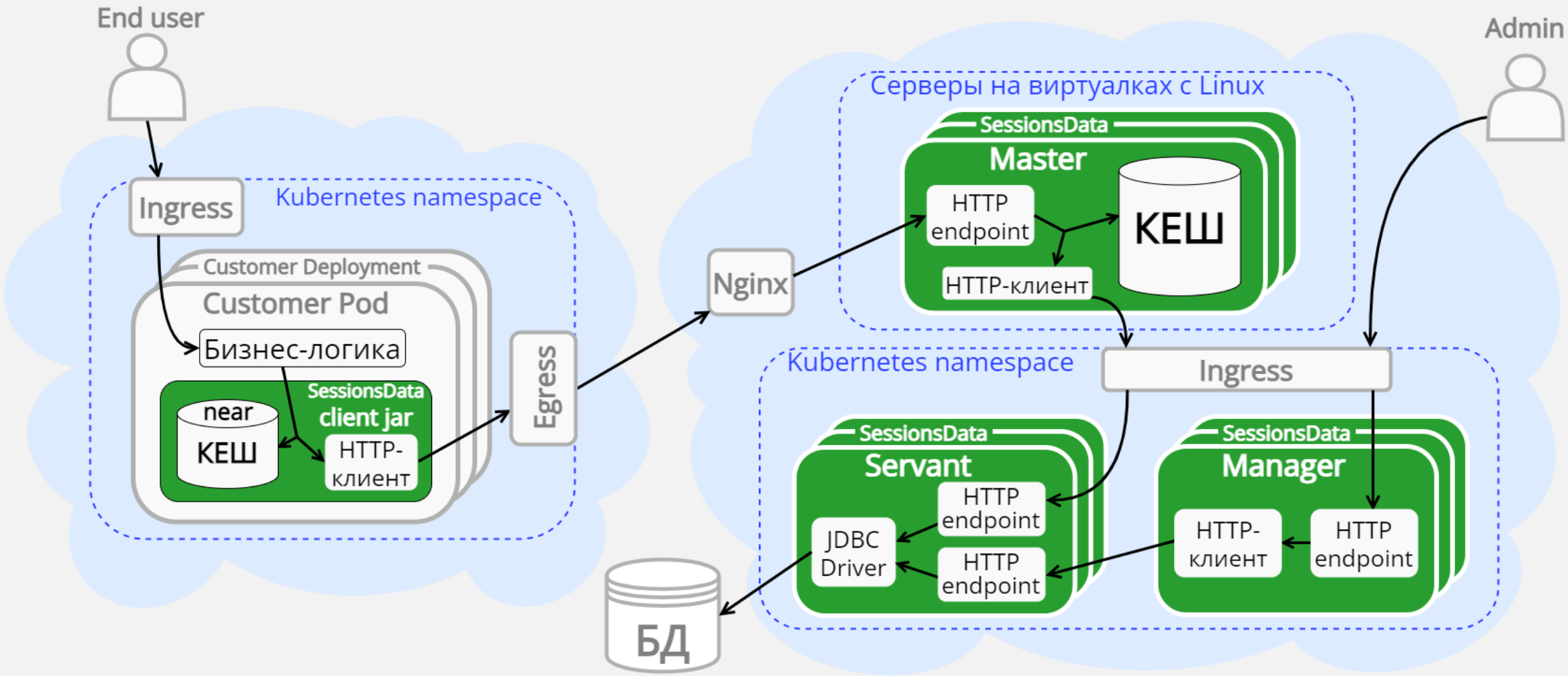
Детали на Joker22

доклад про архитектуру
и историю создания:

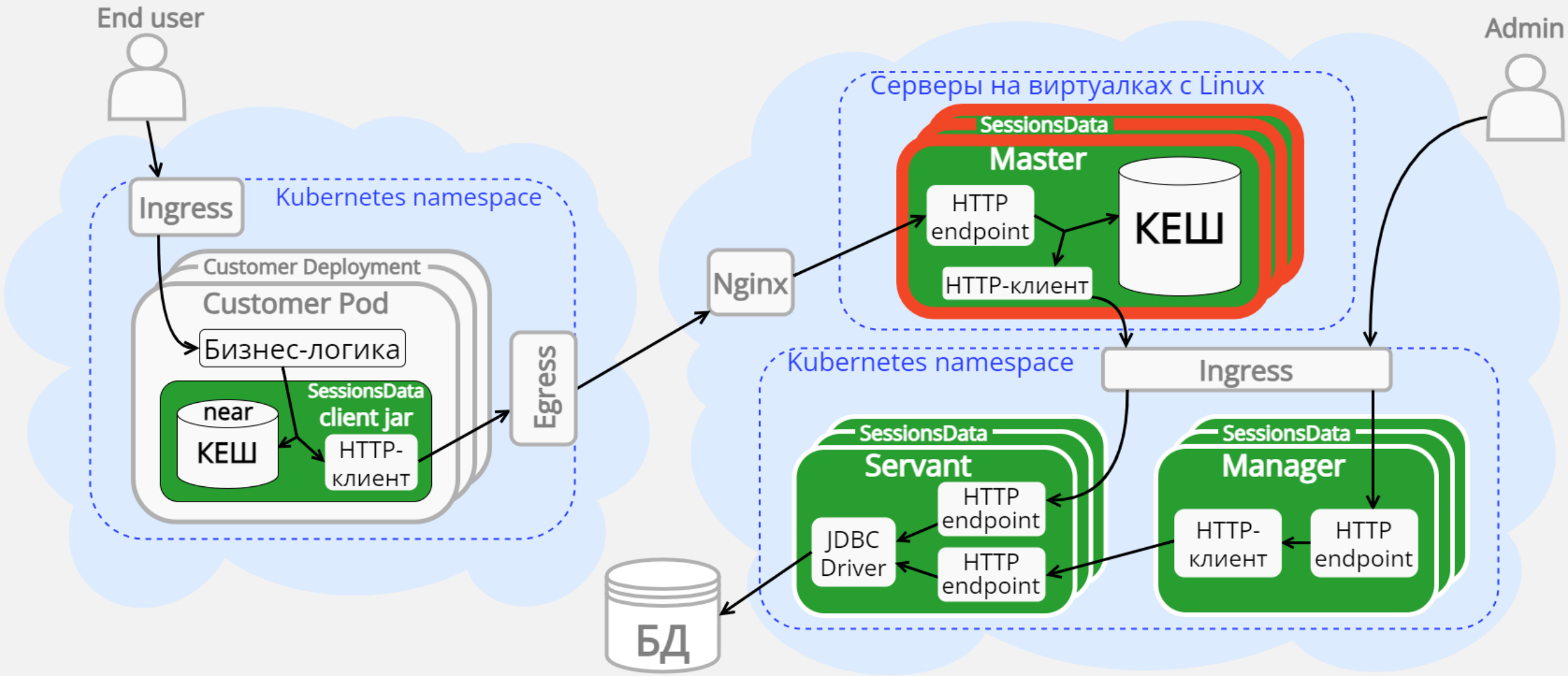
Что за сервис

распределенный in-memory
кеш для сессионных данных:

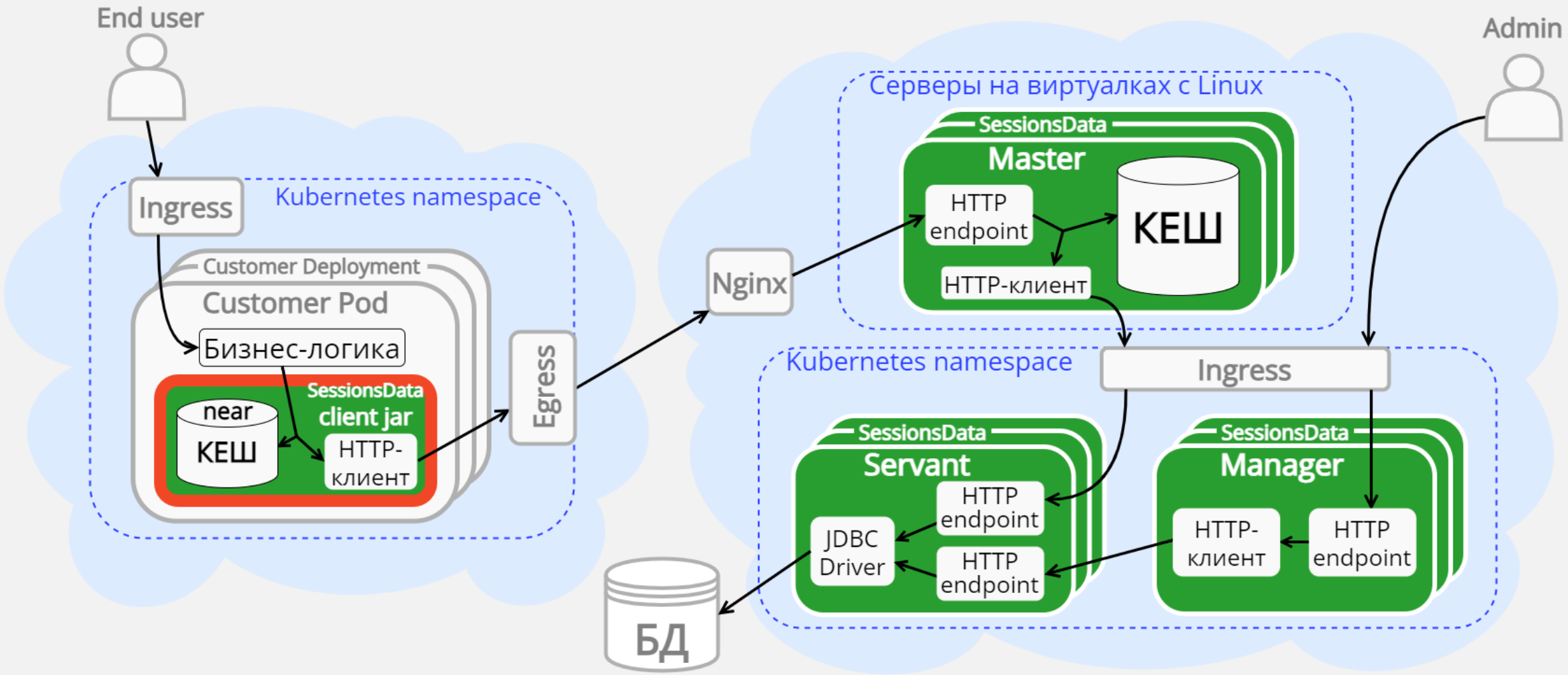
Архитектура Platform V SessionsData



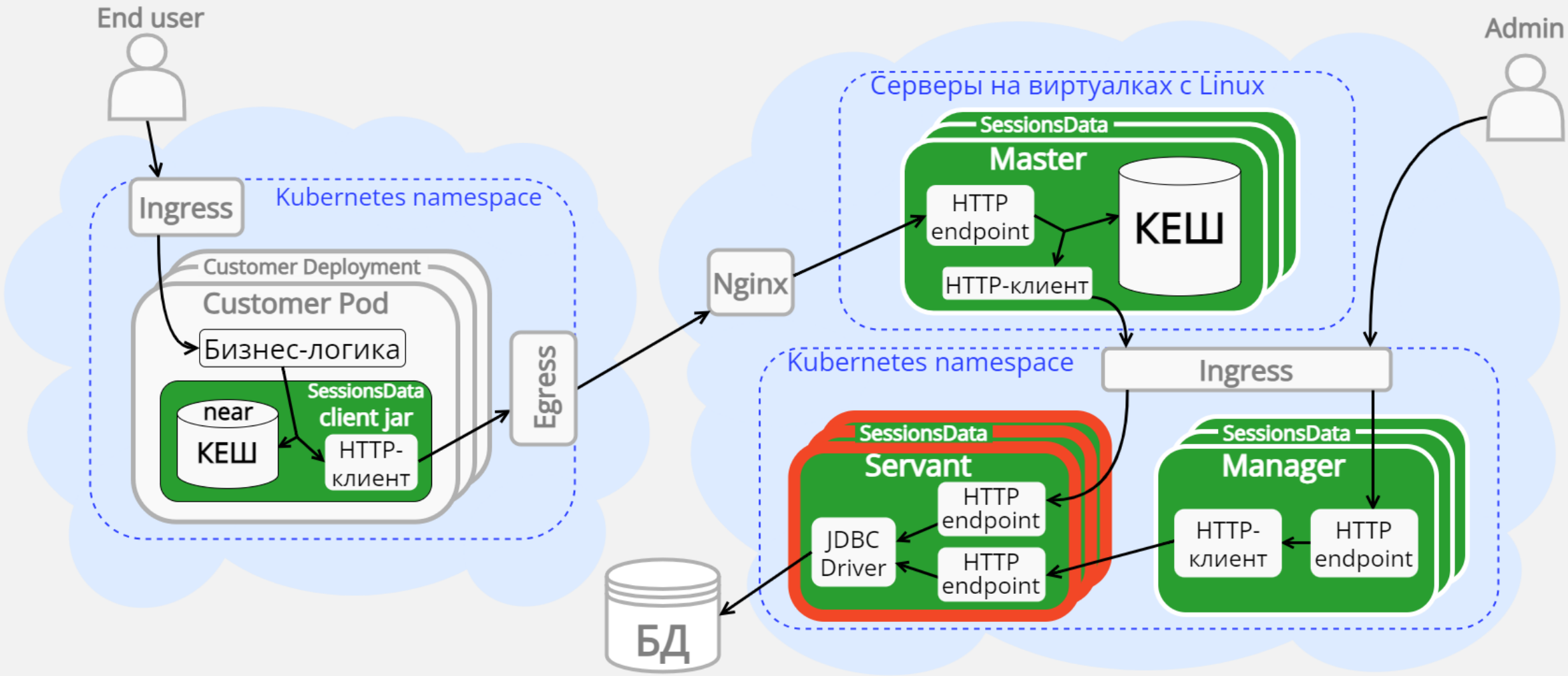
Архитектура Platform V SessionsData



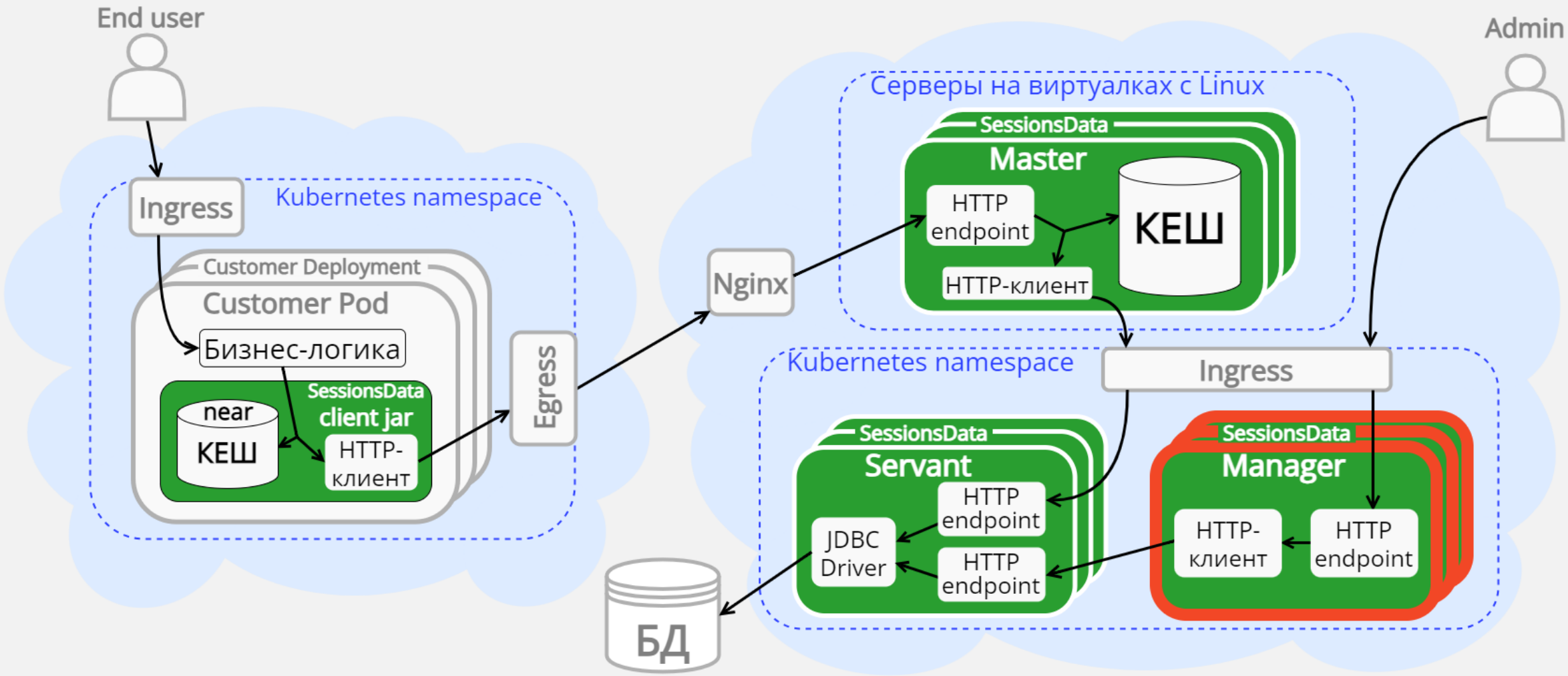
Архитектура Platform V SessionsData



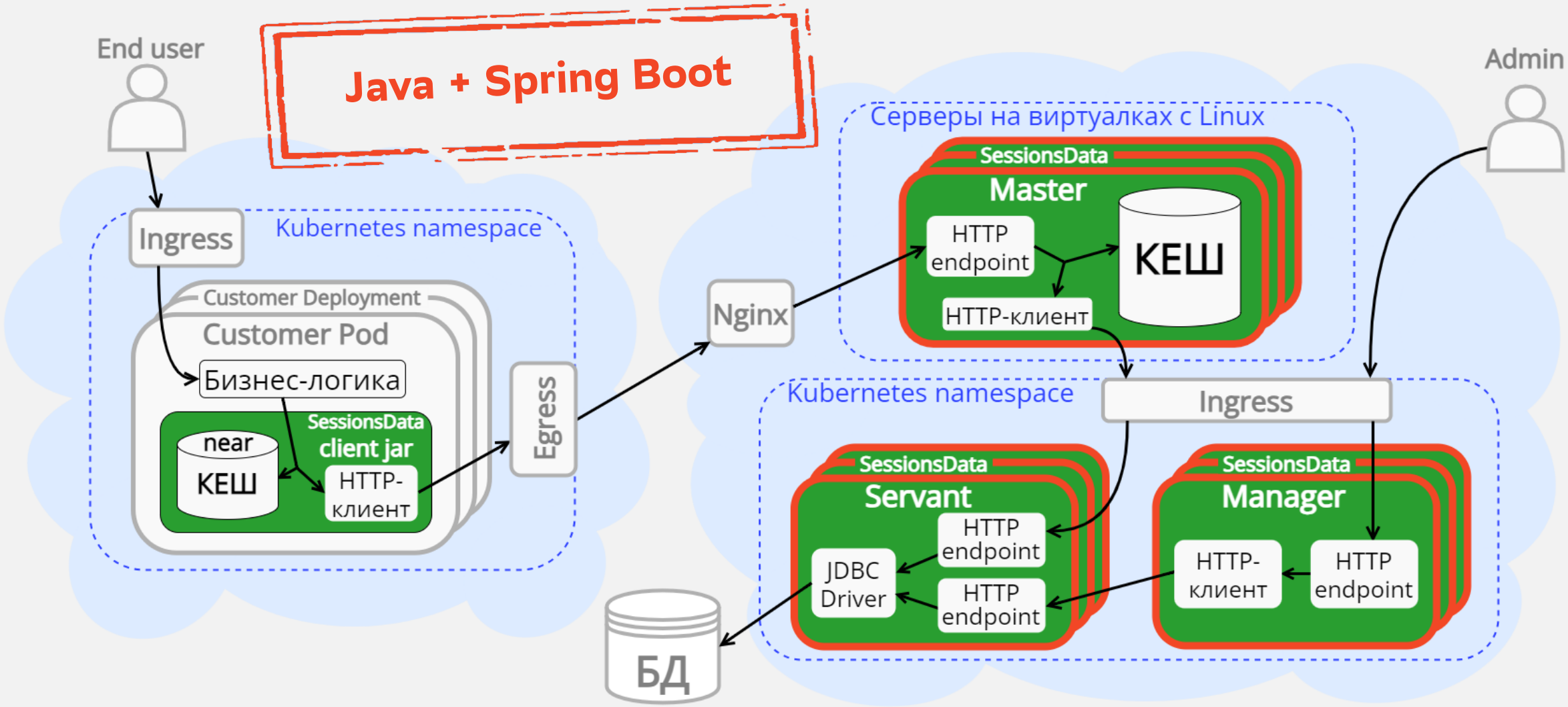
Архитектура Platform V SessionsData



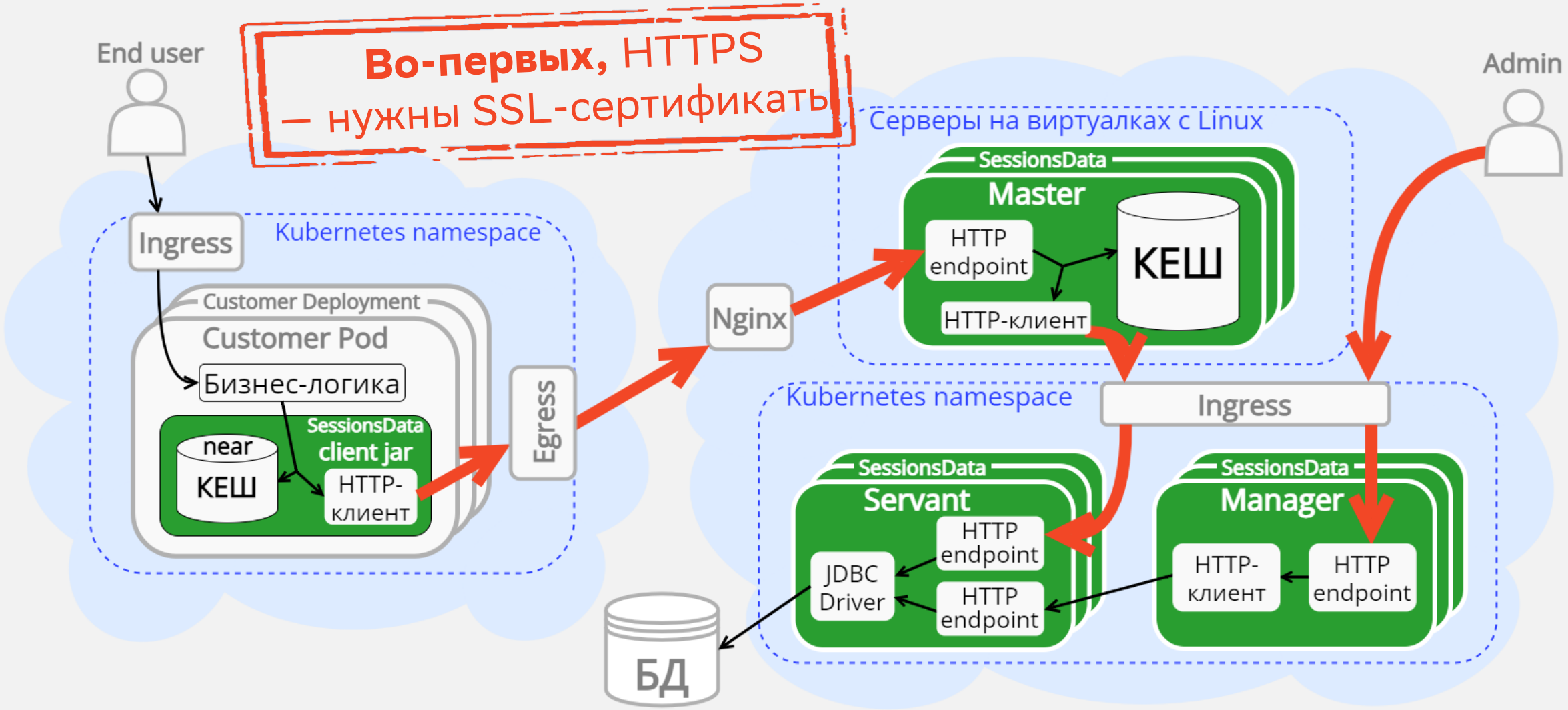
Архитектура Platform V SessionsData



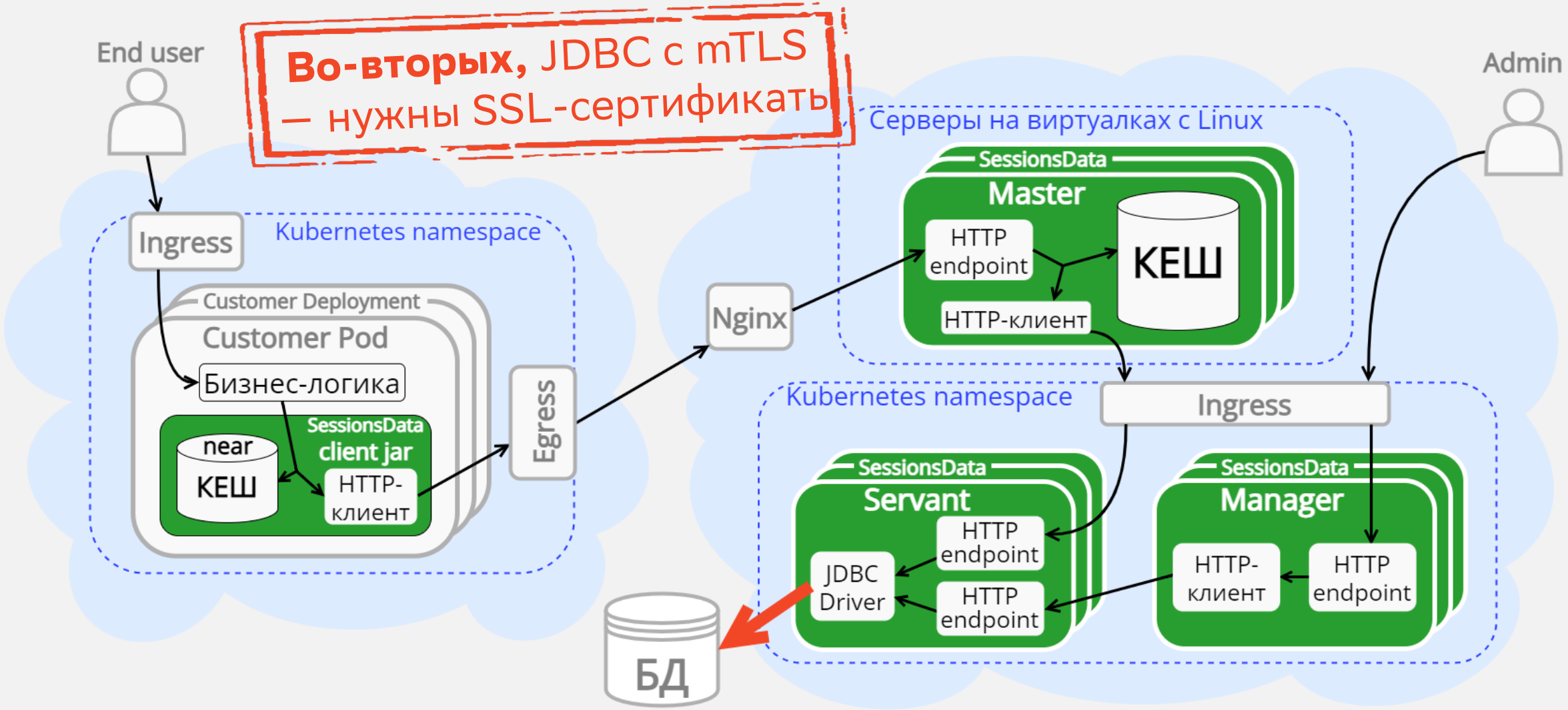
Архитектура Platform V SessionsData



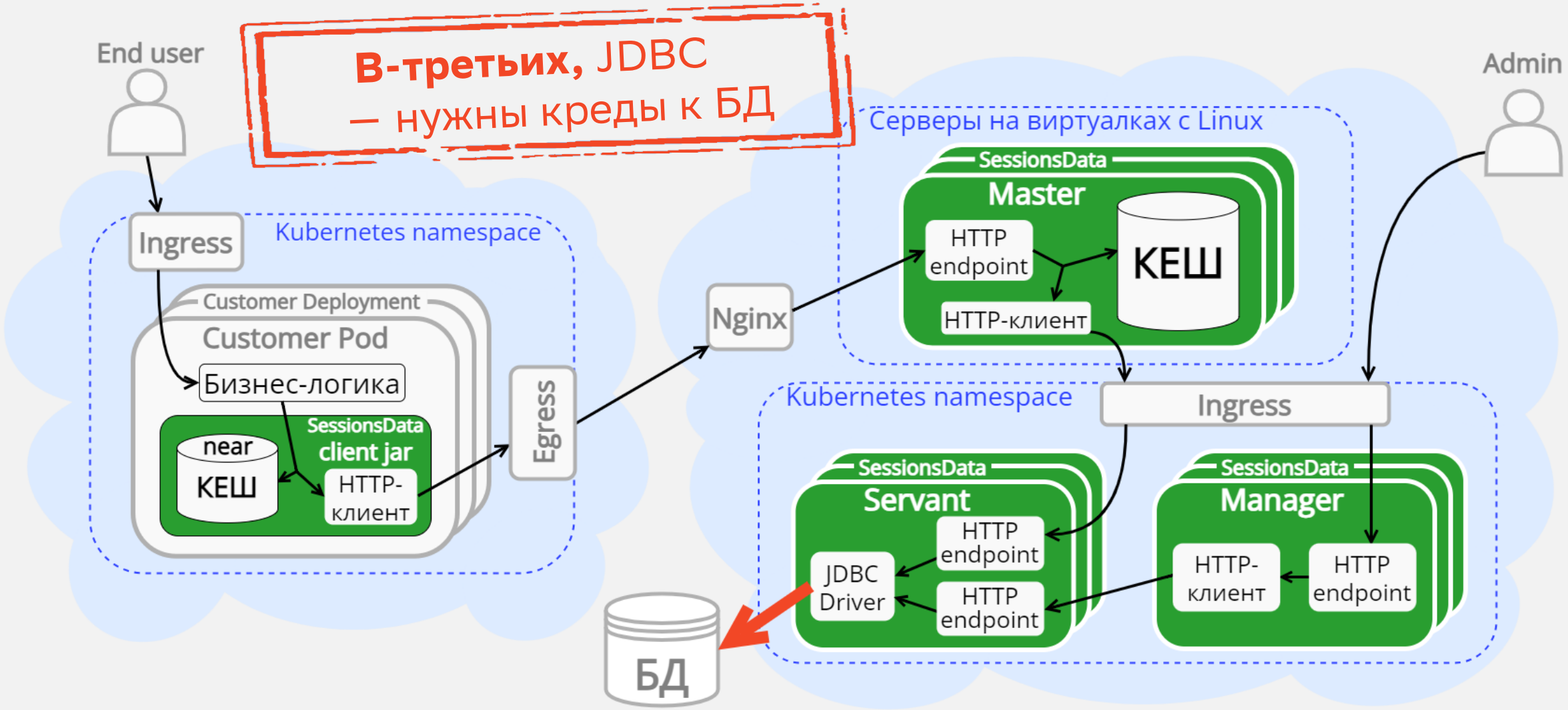
Какие секреты и зачем нужны?



Какие секреты и зачем нужны?



Какие секреты и зачем нужны?



План

1. Каждому сервису нужны секреты
    1.1. Что такое секреты и зачем они нужны?
    1.2. Секреты нужны любому современному сервису
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
4. Подводим итоги

Безопасные интеграции -> Mutual TLS -> нужны секреты

HTTPS

безопасные взаимодействия
с аутентификацией
и шифрованием трафика

JDBC

безопасные соединения
с базой данных

Безопасные интеграции -> Mutual TLS -> нужны секреты

HTTPS

безопасные взаимодействия
с аутентификацией
и шифрованием трафика

JDBC

безопасные соединения
с базой данных

Prometheus

безопасный pull
метрик через
/metrics endpoint

Безопасные интеграции -> Mutual TLS -> нужны секреты

HTTPS

безопасные взаимодействия
с аутентификацией
и шифрованием трафика

JDBC

безопасные соединения
с базой данных

Prometheus

безопасный pull
метрик через
/metrics endpoint

Kafka

безопасные
соединения
с брокерами

Безопасные интеграции -> Mutual TLS -> нужны секреты

HTTPS

безопасные взаимодействия
с аутентификацией
и шифрованием трафика

JDBC

безопасные соединения
с базой данных

Prometheus

безопасный pull
метрик через
/metrics endpoint

Kafka

безопасные
соединения
с брокерами

Redis

безопасные
соединения
с кластером

Безопасные интеграции -> Mutual TLS -> нужны секреты

HTTPS

безопасные взаимодействия
с аутентификацией
и шифрованием т

JDBC

безопасные соединения
с базой данных

Не бывает
сервисов без секретов

Prometheus

безопасный pull
метрик через
/metrics endpoint

Kafka

безопасные
соединения
с брокерами

Redis

безопасные
соединения
с кластером

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
    - 2.1. Получение секретов из HashiCorp Vault
    - 2.2. Примеры секретов для конкретного сервиса на Java
3. Применение секретов в сервисах на Java
4. Подводим итоги

Существует много провайдеров секретов

HashiCorp Vault

Azure Key Vault

AWS Secrets Manager

StarVault

AKEYLESS

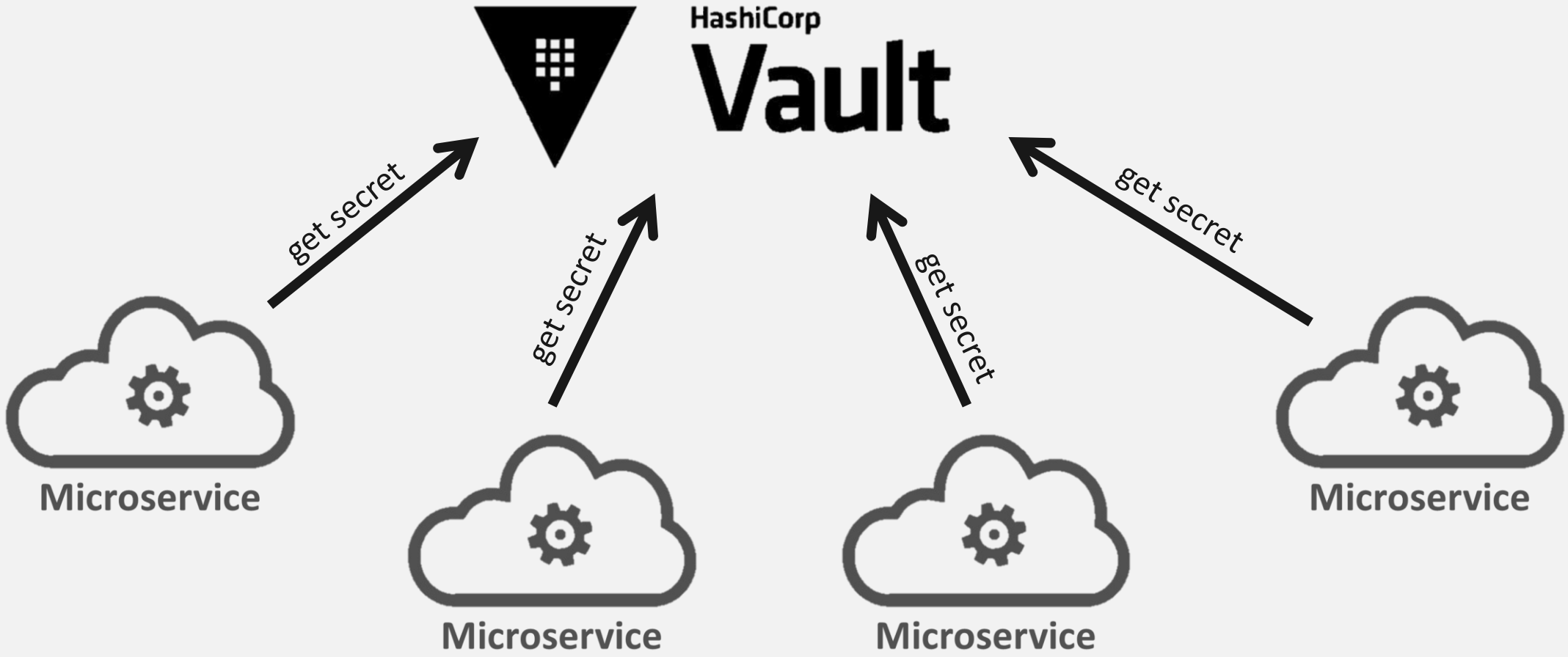
Mozilla Sops

Hypervault

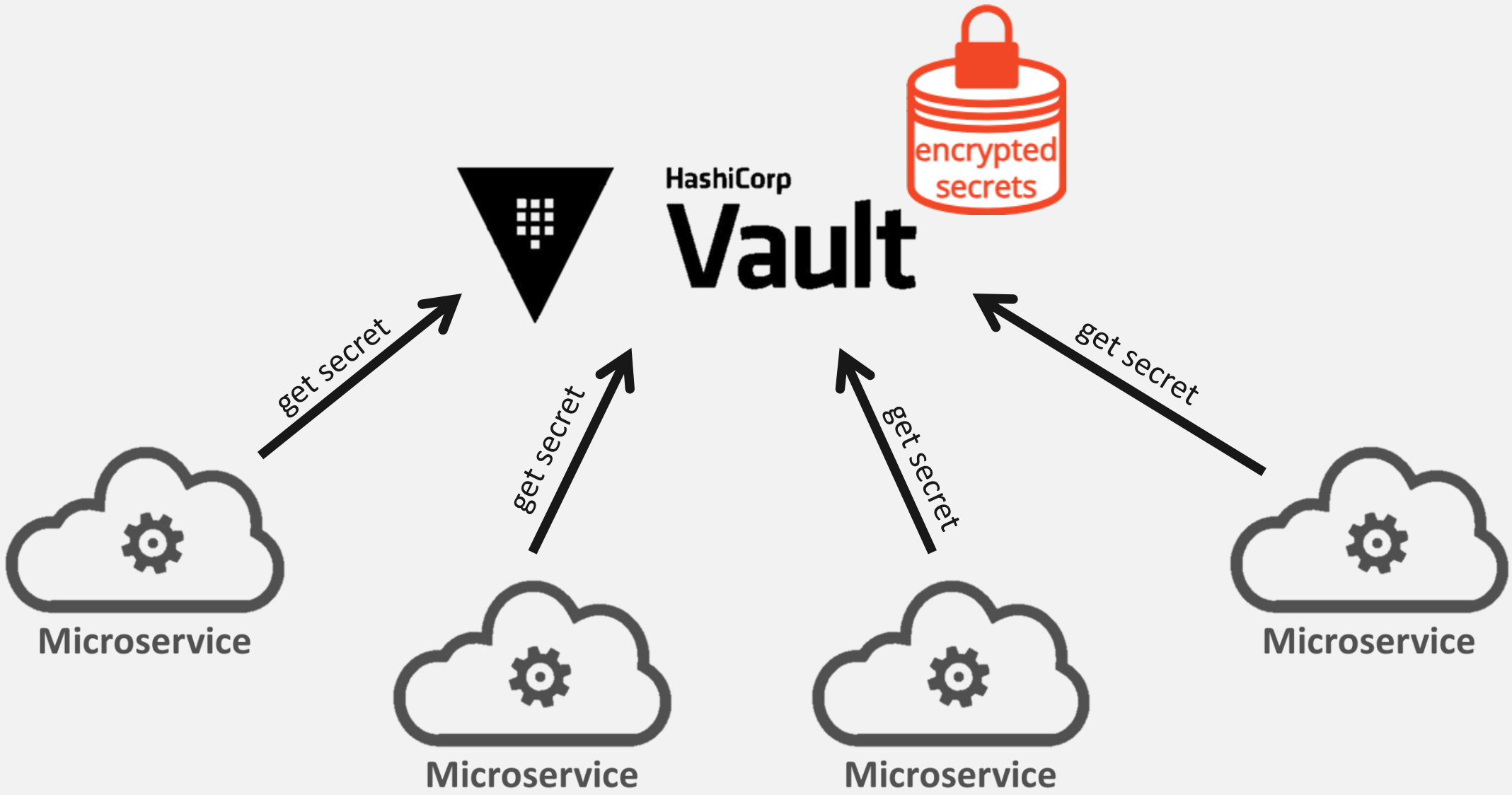
План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
    - 2.1. Получение секретов из HashiCorp Vault
        - 2.1.1. Способы доставки секретов из HashiCorp Vault в сервис
    - 2.2. Примеры секретов для конкретного сервиса на Java
3. Применение секретов в сервисах на Java
4. Подводим итоги

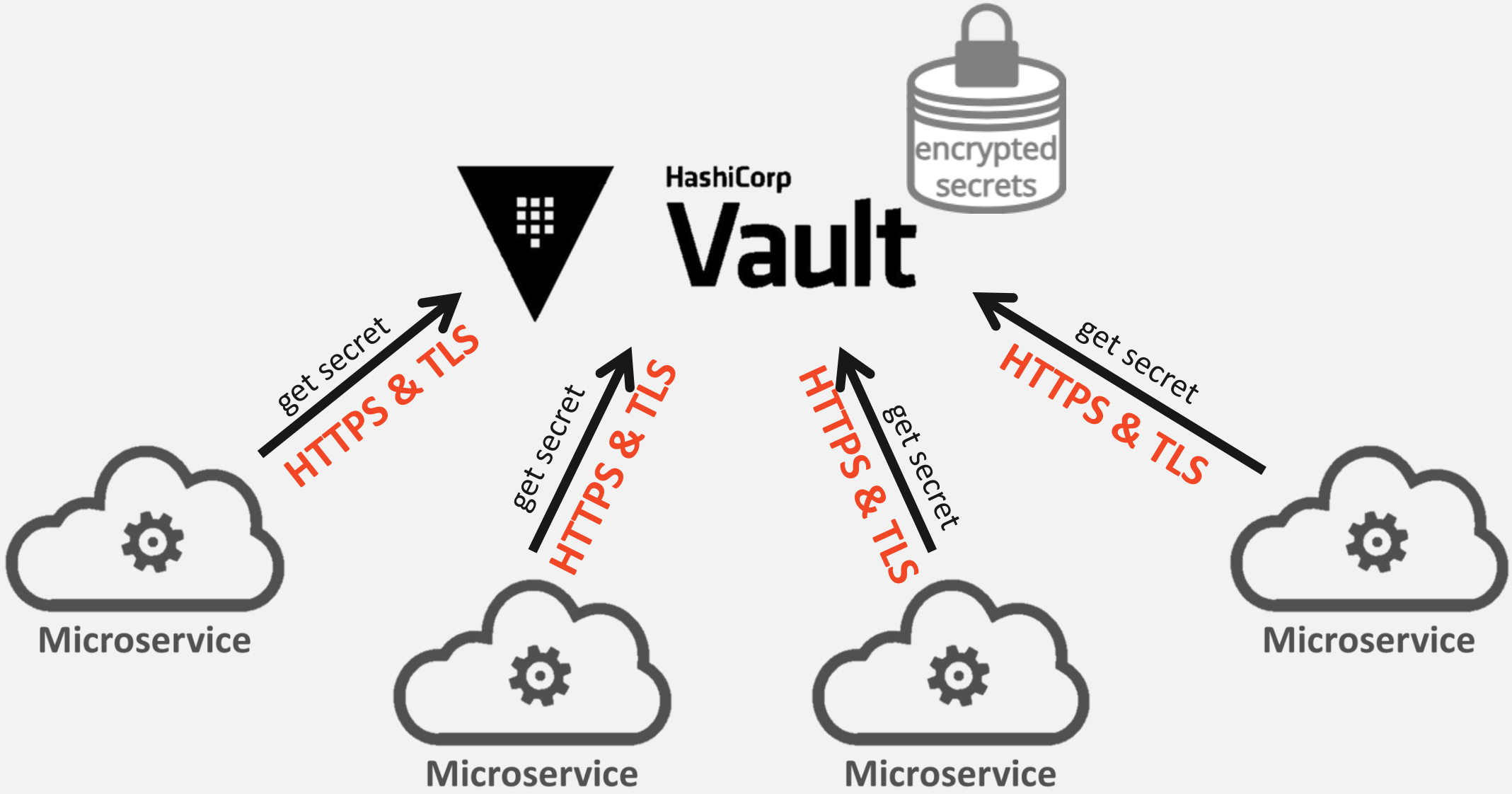
Получение секретов из HashiCorp Vault



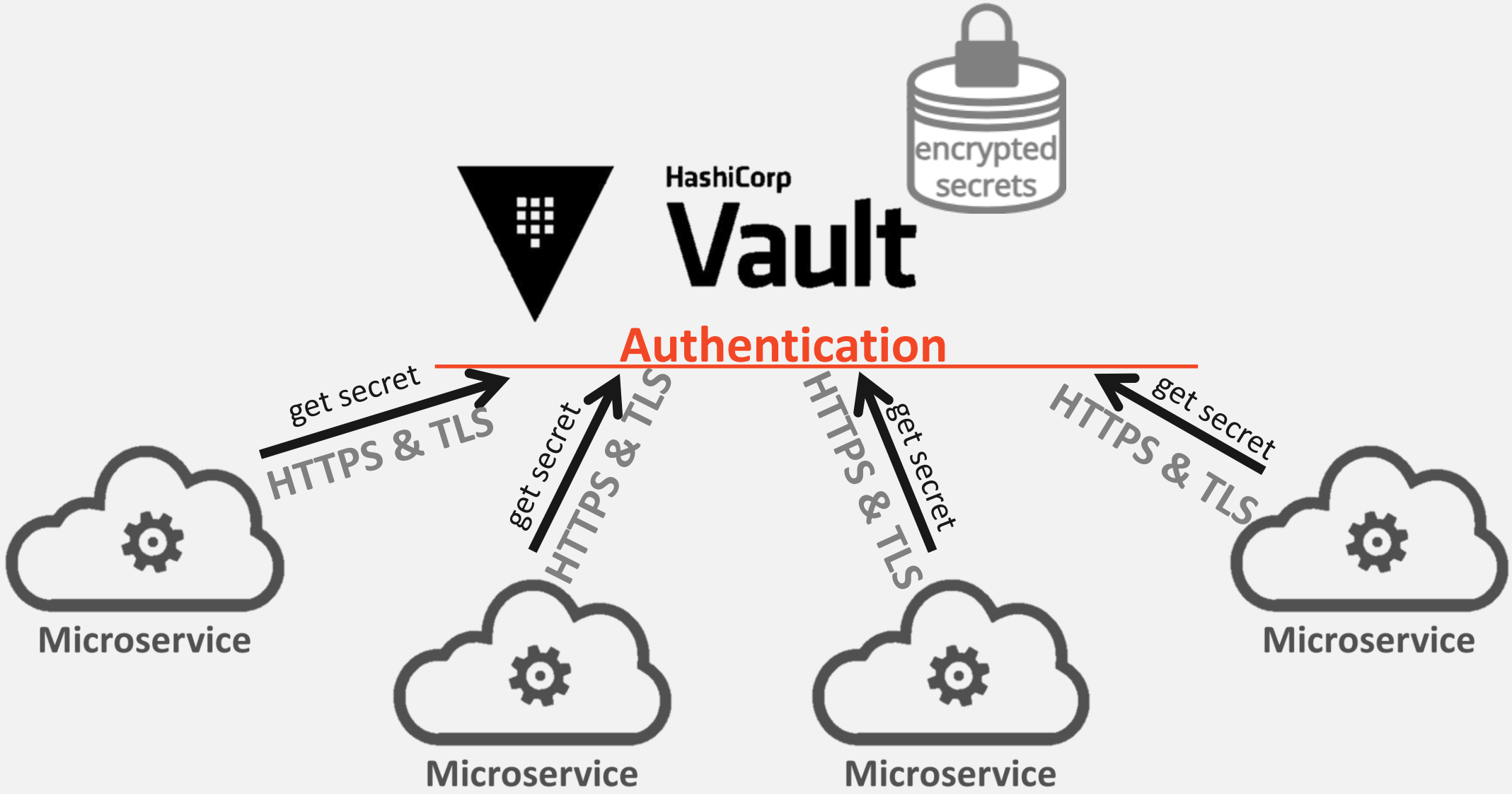
Получение секретов из HashiCorp Vault



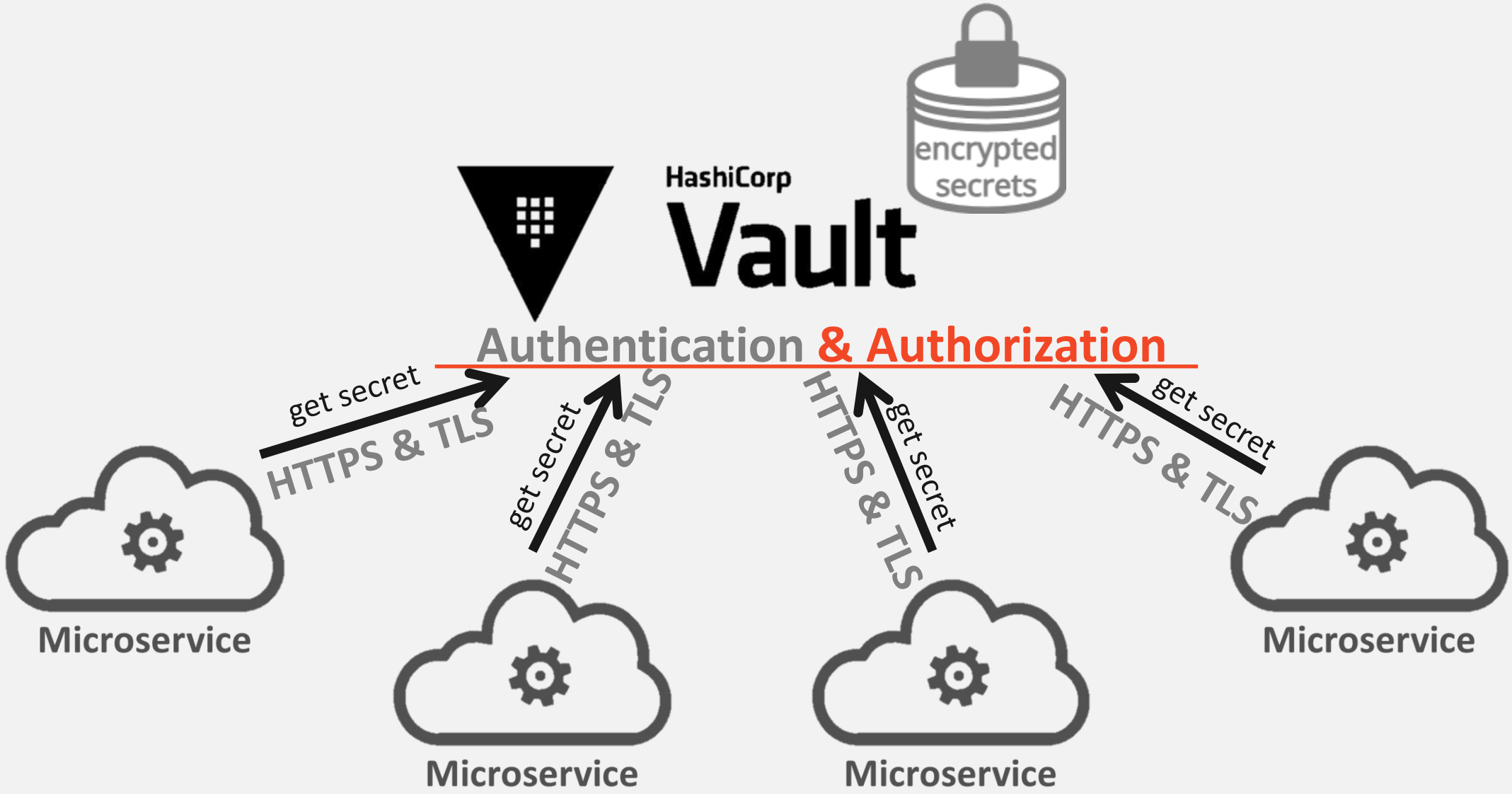
Получение секретов из HashiCorp Vault



Получение секретов из HashiCorp Vault



Получение секретов из HashiCorp Vault



План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
    - 2.1. Получение секретов из HashiCorp Vault
        - 2.1.1. Способы доставки секретов из HashiCorp Vault в сервис
    - 2.2. Примеры секретов для конкретного сервиса на Java
3. Применение секретов в сервисах на Java
4. Подводим итоги

Способы доставки секретов из HashiCorp Vault

1. **Использование Vault Agent sidecar**

   Sidecar приносит секреты из Vault и сохраняет их в emptyDir, который общий с основным контейнером в pod

Способы доставки секретов из HashiCorp Vault

1. Использование Vault Agent sidecar
2. **Использование External Secrets Operator**

   Оператор приносит секреты из Vault и сохраняет их в kind: Secret

   Сервис монтирует файлы из kind: Secret в свой контейнер

   См. https://external-secrets.io/v0.9.13/

Способы доставки секретов из HashiCorp Vault

1. Использование Vault Agent sidecar
2. Использование External Secrets Operator
3. **Прямая интеграция с HashiCorp Vault по HTTPS**
   Сервис сам приносит секреты из Vault
   В Platform V SessionsData master-сервис на VM приносит себе файлы с секретами

Способы доставки секретов из HashiCorp Vault

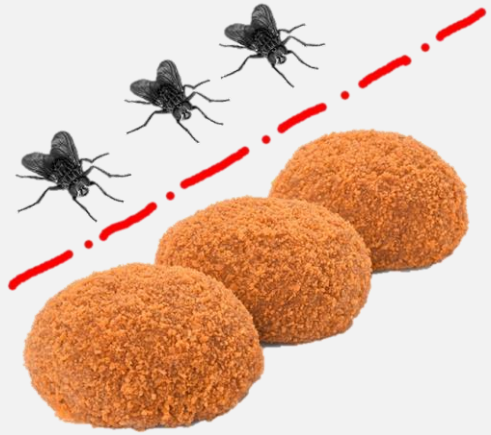
1. Использование Vault Agent sidecar
2. Использование External Secrets Operator
3. Прямая интеграция с HashiCorp Vault по HTTPS

Итого: секреты для сервиса — это файлы

Способы доставки секретов из HashiCorp Vault

1. Использование Vault Agent sidecar
2. Использование External Secrets Operator
3. Прямая интеграция с HashiCorp Vault по HTTPS

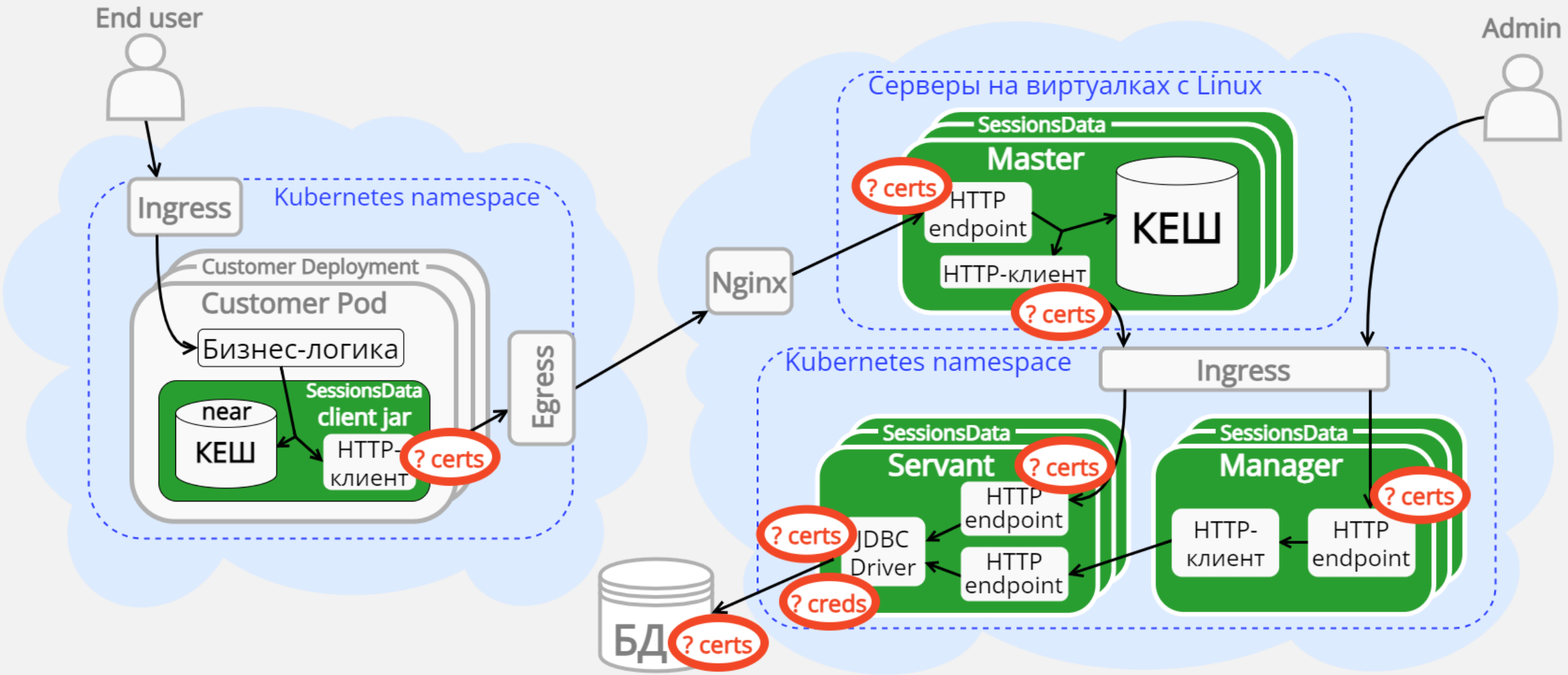
Итого: секреты для сервиса — это файлы



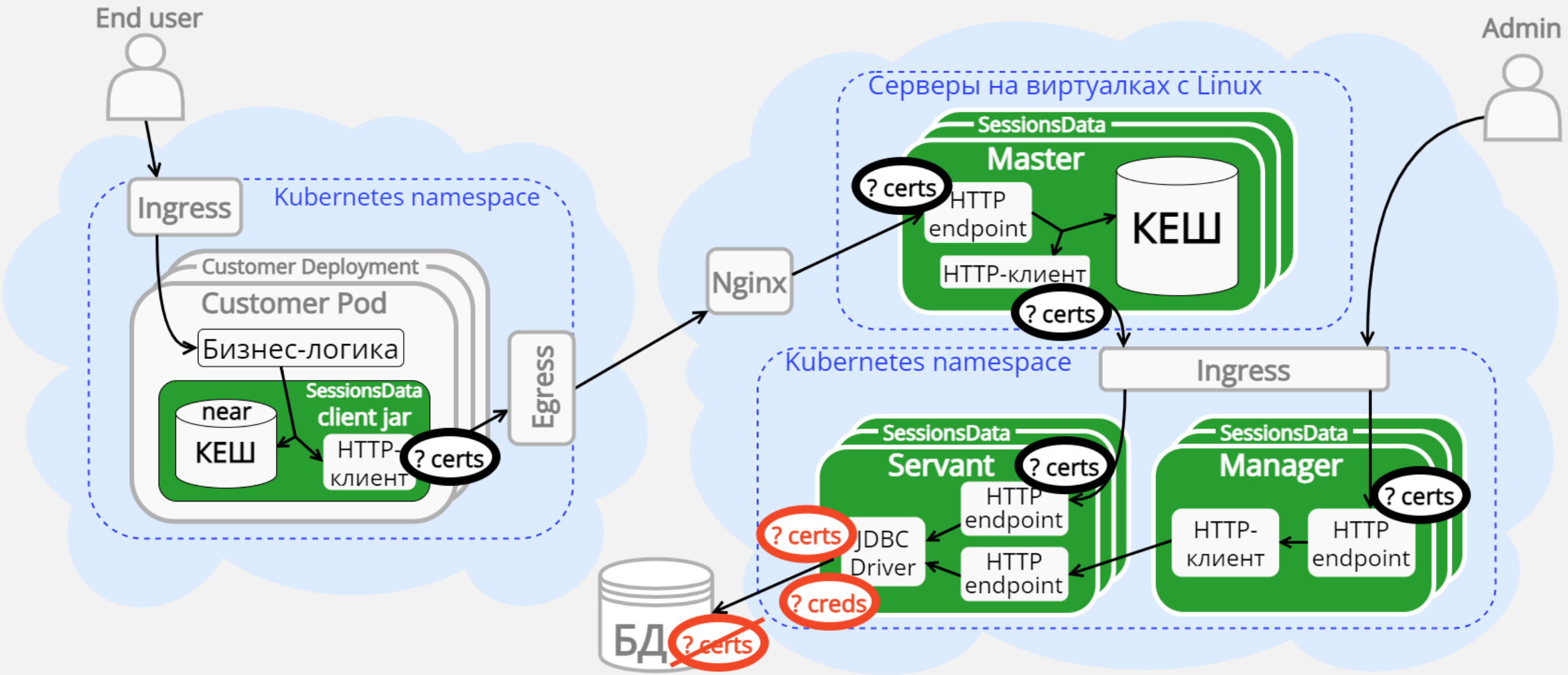
План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
    2.1. Получение секретов из HashiCorp Vault
    2.2. Примеры секретов для конкретного сервиса на Java
3. Применение секретов в сервисах на Java
4. Подводим итоги

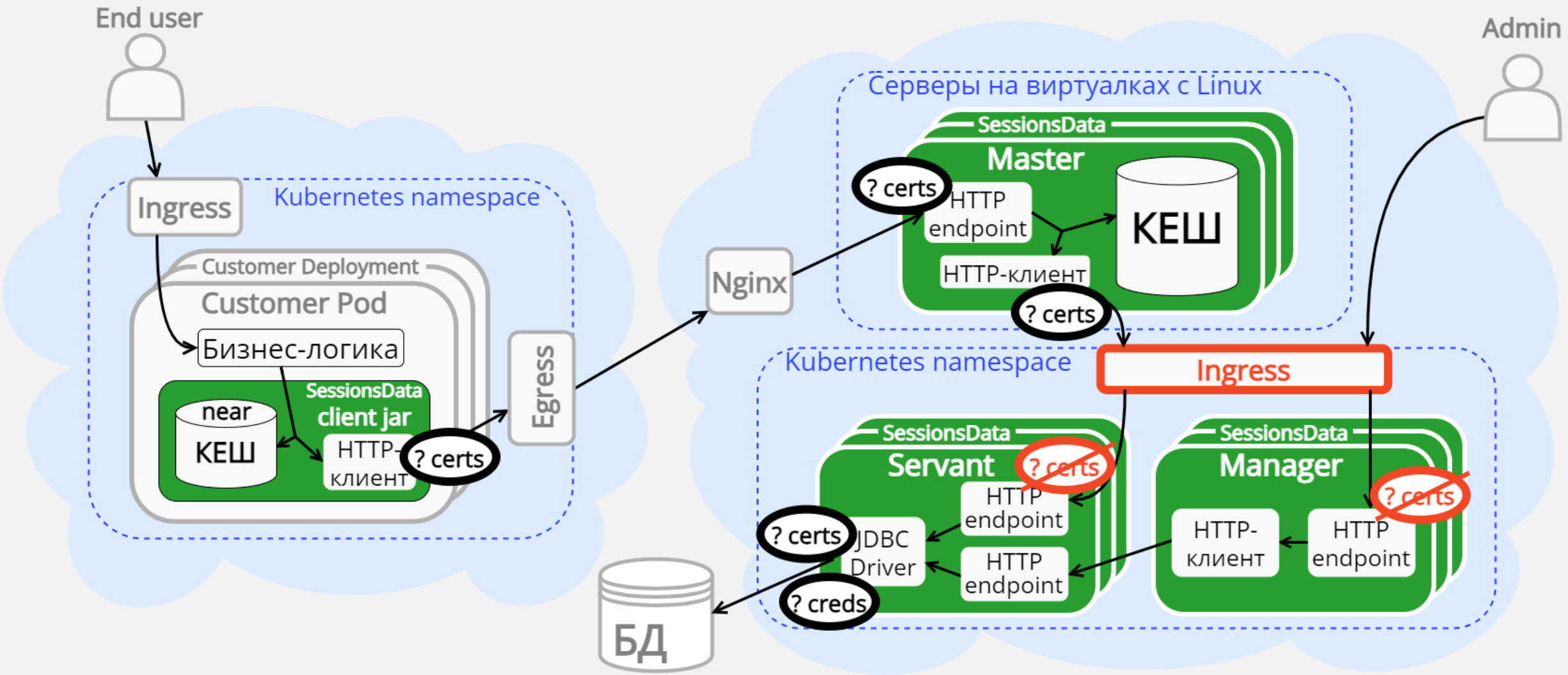
Нужны SSL-сертификаты и креды БД



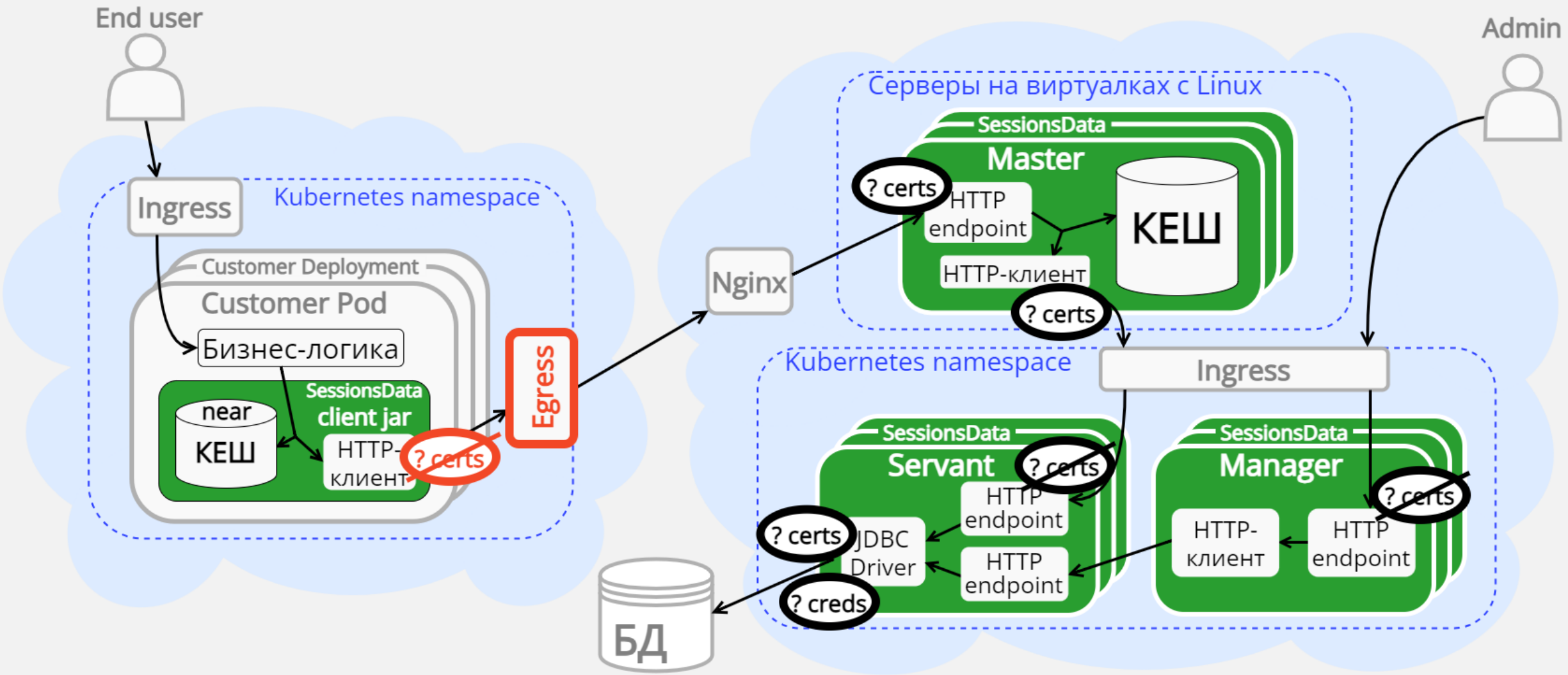
Для БД нужны только клиентские SSL-сертификаты и креды



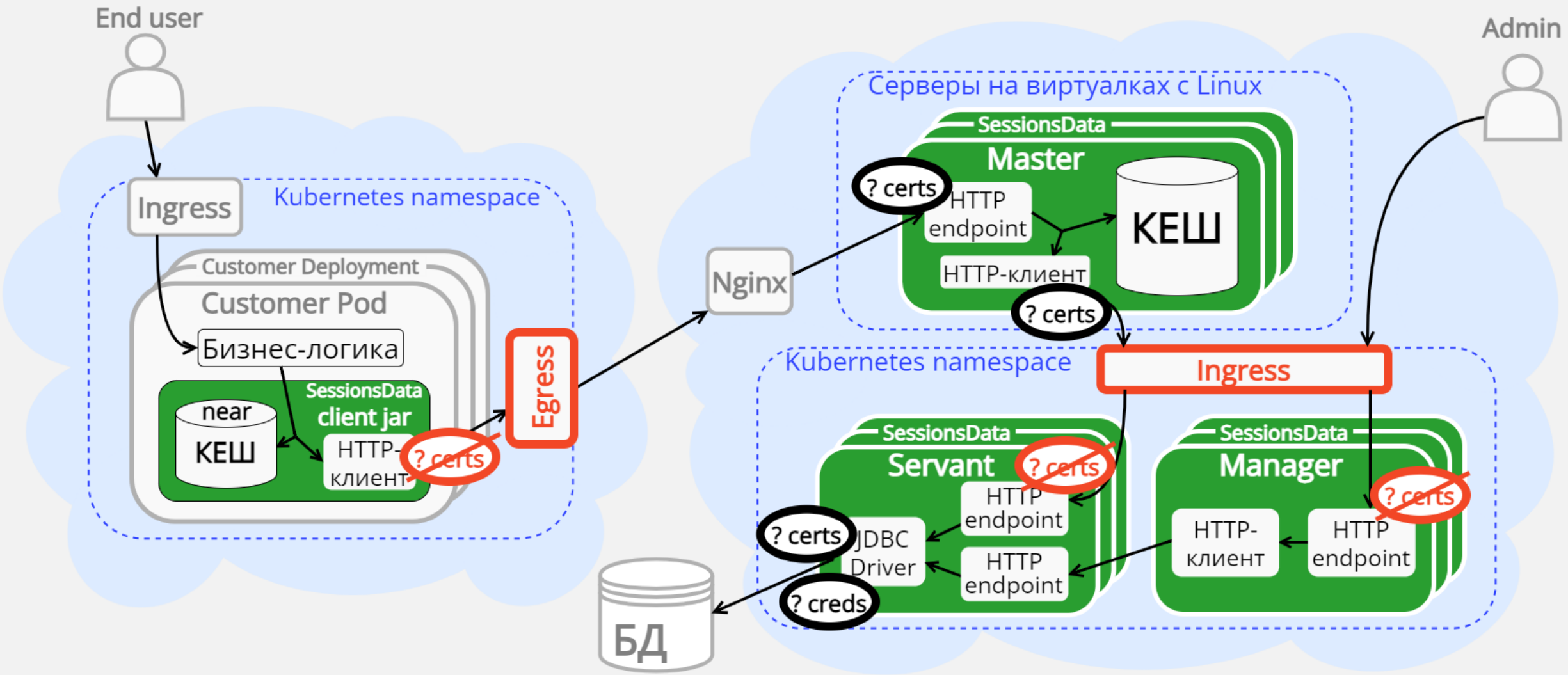
Из-за Ingress в Java не нужны серверные SSL-сертификаты



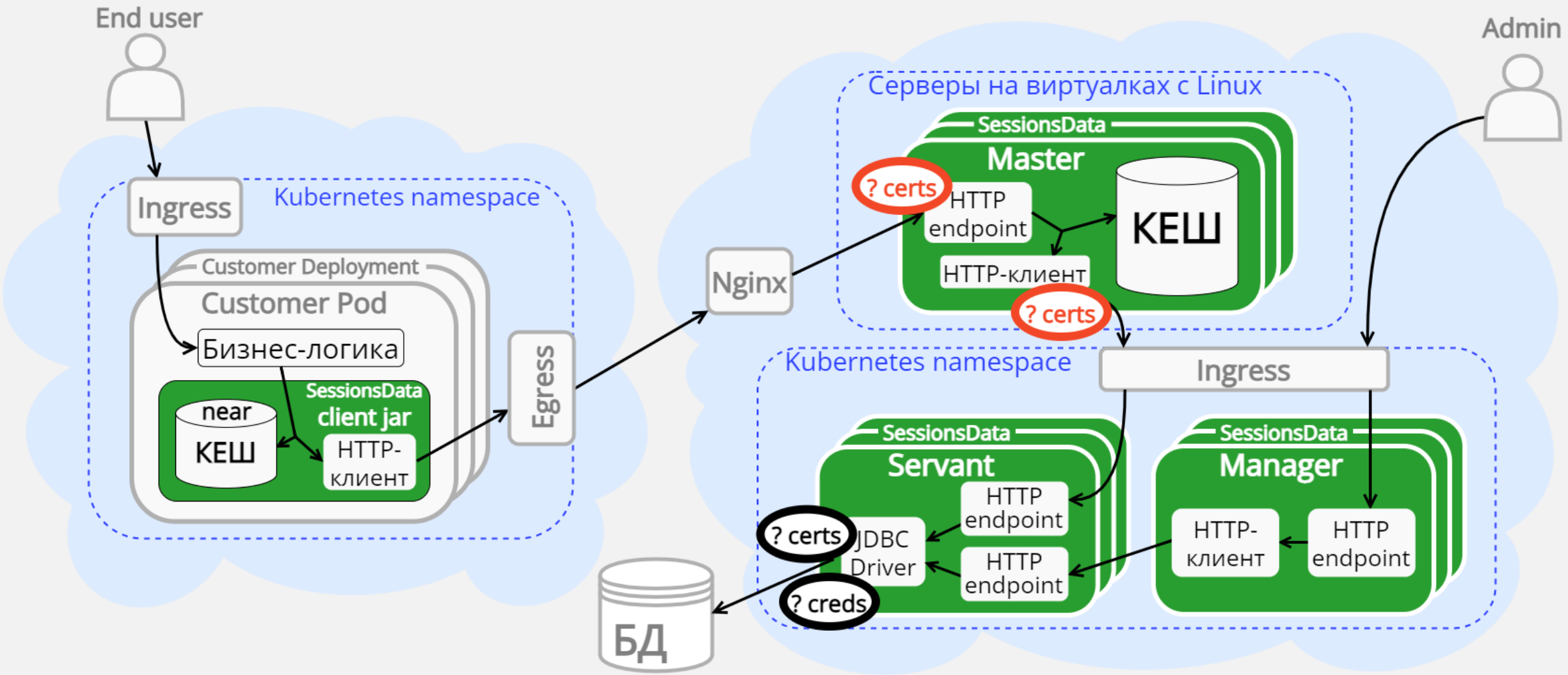
Из-за Egress в Java не нужны клиентские SSL-сертификаты



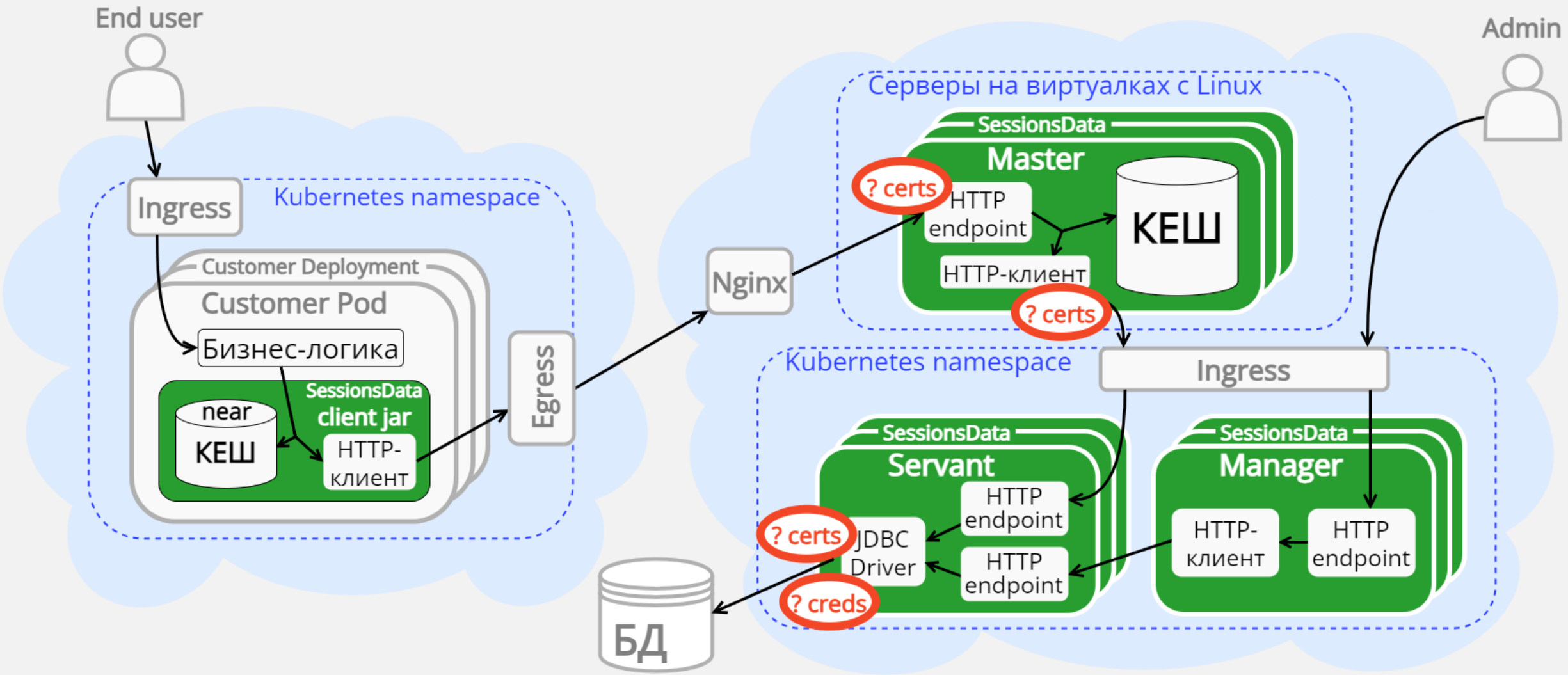
В k8s не нужны серверные и клиентские SSL-сертификаты



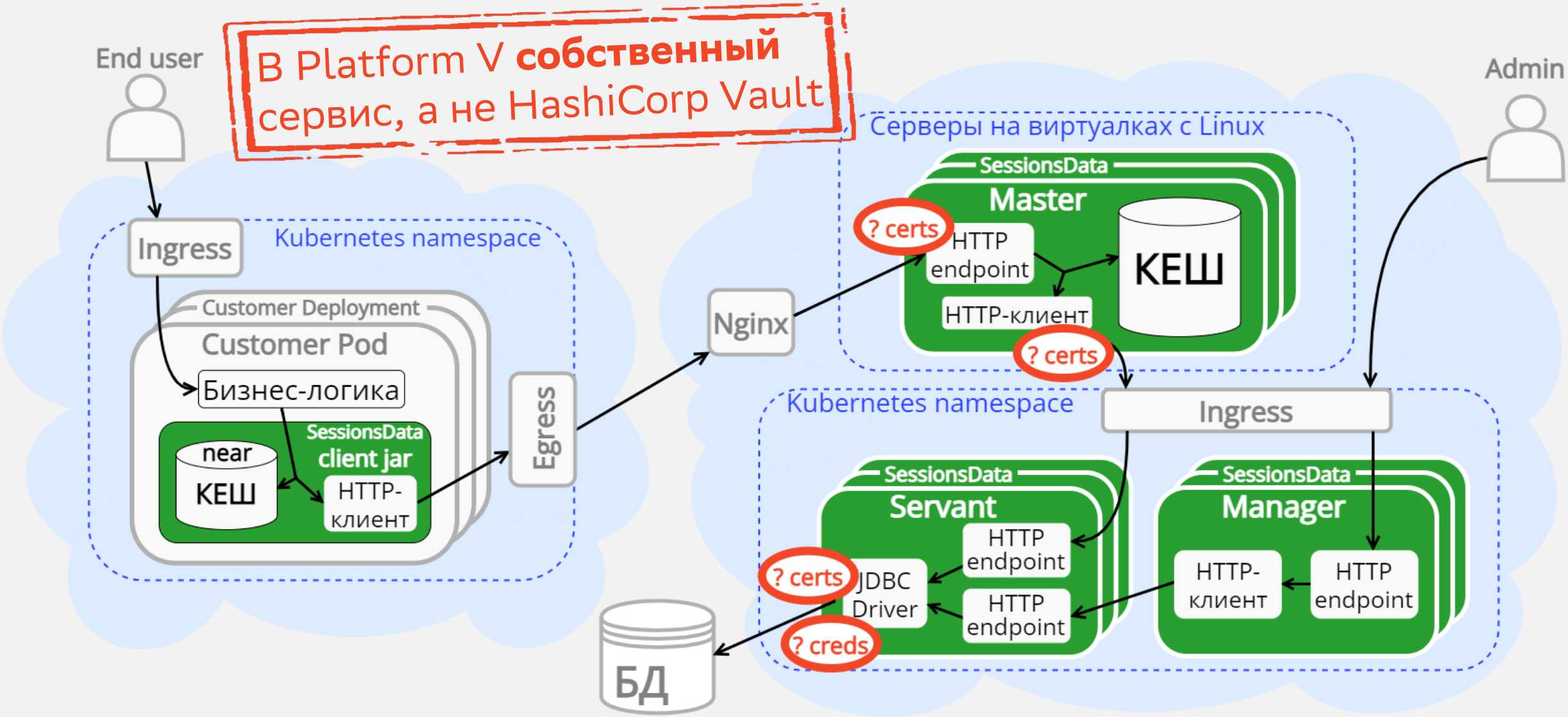
На VM нужны серверные и клиентские SSL-сертификаты



В Platform V SessionsData секреты нужны в master и servant



В Platform V SessionsData секреты нужны в master и servant



План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
4. Подводим итоги

Какие секреты нужно применить в Java?

Файлы:

1. Серверные SSL-сертификаты для HTTPS
2. Клиентские SSL-сертификаты для HTTPS
3. Клиентские SSL-сертификаты для JDBC
4. Креды БД

Какие секреты нужно применить в Java?

Файлы:

1. Серверные SSL-сертификаты для HTTPS
2. Клиентские SSL-сертификаты для HTTPS
3. Клиентские SSL-сертификаты для JDBC
4. Креды БД

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
        - 3.1.1. Применение серверных SSL-сертификатов
        - 3.1.2. Применение клиентских SSL-сертификатов
        - 3.1.3. Применение SSL-сертификатов для соединений с БД
        - 3.1.4. Применение кредов БД
    - 3.2. Hot reload секретов при их изменении
4. Подводим итоги

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
        - 3.1.1. Применение серверных SSL-сертификатов
        - 3.1.2. Применение клиентских SSL-сертификатов
        - 3.1.3. Применение SSL-сертификатов для соединений с БД
        - 3.1.4. Применение кредов БД
    - 3.2. Hot reload секретов при их изменении
4. Подводим итоги

Серверные SSL-сертификаты в application.yaml

```
server:
  ssl:
  ✔ key-store: ${secrets.server.keyStorePath}
    key-store-type: PKCS12
    key-alias: ...
    key-store-password: ${secrets.server.keyStorePasswordPath}
    key-password: ${secrets.server.keyPasswordPath}
  ✔ trust-store: ${secrets.server.trustStorePath}
    trust-store-type: PKCS12
    trust-store-password: ${secrets.server.trustStorePasswordPath}
    client-auth: need
```

Серверные SSL-сертификаты в application.yaml

```
server:
  ssl:
✅ key-store: ${secrets.server.keyStorePath}
   key-store-type: PKCS12
   key-alias: ...
❌ key-store-password: ${secrets.server.keyStorePasswordPath}
❌ key-password: ${secrets.server.keyPasswordPath}
✅ trust-store: ${secrets.server.trustStorePath}
   trust-store-type: PKCS12
❌ trust-store-password: ${secrets.server.trustStorePasswordPath}
   client-auth: need
```

Нужны строки, а не файлы

Пароли из файлов применяем с помощью customizer

```
@Bean
public WebServerFactoryCustomizer<TomcatServletWebServerFactory>
            tomcatCustomizer(SecretFilesProperties secretProps )
{
    return ( TomcatServletWebServerFactory factory ) -> {
        Ssl sslSettings = factory.getSsl();
        sslSettings.setKeyStorePassword(
            stringFromFile( secretProps.getServerKeyStorePasswordPath() ) );
        sslSettings.setKeyPassword(
            stringFromFile( secretProps.getServerKeyPasswordPath() ) );
        sslSettings.setTrustStorePassword(
            stringFromFile( secretProps.getServerTrustStorePasswordPath() ) );
    };
}
```

Best practice

```
@Bean
public WebServerFactoryCustomizer<TomcatServletWebServerFactory>
            tomcatCustomizer(SecretFilesProperties secretProps )
{
    return ( TomcatServletWebServerFactory factory ) -> {
        Ssl sslSettings = factory.getSsl();
        sslSettings.setKeyStorePassword(
            stringFromFile( secretProps.getServerKeyStorePasswordPath() ) );
        sslSettings.setKeyPassword(
            stringFromFile( secretProps.getServerKeyPasswordPath() ) );
        sslSettings.setTrustStorePassword(
            stringFromFile( secretProps.getServerTrustStorePasswordPath() ) );
    };
}
```

char[] предпочтительнее String
при работе с паролями

Best practice

```
@Bean
public WebServerFactoryCustomizer<TomcatServletWebServerFactory>
            tomcatCustomizer(SecretFilesProperties secretProps )
{
    return ( TomcatServletWebServerFactory factory ) -> {
        Ssl sslSettings = factory.getSsl();
        sslSettings.setKeyStorePassword(
            stringFromFile( secretProps.getServerKeyStorePasswordPath() ) );
        sslSettings.setKeyPassword(
            stringFromFile( secretProps.getServerKeyPasswordPath() ) );
        sslSettings.setTrustStorePassword(
            stringFromFile( secretProps.getServerTrustStorePasswordPath() ) );
    };
}
```

Однако, Spring просит String

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
        - 3.1.1. Применение серверных SSL-сертификатов
        - 3.1.2. Применение клиентских SSL-сертификатов
        - 3.1.3. Применение SSL-сертификатов для соединений с БД
        - 3.1.4. Применение кредов БД
    - 3.2. Hot reload секретов при их изменении
4. Подводим итоги

Из клиентских SSL-сертификатов собираем SSLContext

```
public SSLContext createSslContext( SecretFilesProperties secretProps )
            throws GeneralSecurityException, IOException {
    SSLContextBuilder sslContextBuilder = SSLContextBuilder.create();
    sslContextBuilder.setKeyStoreType( "PKCS12" );

    loadKeyStore( sslContextBuilder,
        Files.newInputStream( secretProps.getClientKeyStorePath() ),
        charsFromFile( secretProps.getClientKeyStorePasswordPath() ) );
    loadTrustStore( sslContextBuilder,
        Files.newInputStream( secretProps.getClientTrustStorePath() ),
        charsFromFile( secretProps.getClientTrustStorePasswordPath() ));

    return sslContextBuilder.build();
}
```

Best practice

```
public SSLContext createSslContext( SecretFilesProperties secretProps )
            throws GeneralSecurityException, IOException {
    SSLContextBuilder sslContextBuilder = SSLContextBuilder.create();
    sslContextBuilder.setKeyStoreType( "PKCS12" );

    loadKeyStore( ... );
    loadTrustStore( ... );
    return sslContextBuilder.build();
}

private void loadKeyStore( SSLContextBuilder builder,InputStream keyStoreStm,
                           char[] keyStorePass ) {
    KeyStore keyStore = KeyStore.getInstance( "PKCS12" );
    keyStore.load( keyStoreStm, keyStorePass );
    builder.loadKeyMaterial( keyStore, keyStorePass, <aliasStrategy> );
    Arrays.fill( keyStorePass, ( char )0 );
}
```

Задаем SSLContext для Jersey client

```
ClientBuilder clientBuilder =
    ClientBuilder.newBuilder().withConfig(...);
clientBuilder.sslContext( createSslContext( secretProps ) );
Client jerseyClient = clientBuilder.build();
```

Jersey client готов

```
ClientBuilder clientBuilder =
    ClientBuilder.newBuilder().withConfig(...);
clientBuilder.sslContext( createSslContext( secretProps ) );
Client jerseyClient = clientBuilder.build();
```

Client готов к отправке
HTTPS-запросов с mTLS

План

SSL-сертификаты указываются в JDBC URL

```
jdbc:postgresql://<db-host>:<db-port>/<db-name>
    ?targetServerType=master&prepareThreshold=0
    &ssl=true&sslmode=verify-full
    &sslcert=/etc/config/ssl/postgresql/tls.crt
    &sslkey=/etc/config/ssl/postgresql/tls.key
    &sslrootcert=/etc/config/ssl/ca/root.crt
```

SSL-сертификаты указываются в JDBC URL

```
jdbc:postgresql://<db-host>:<db-port>/<db-name>
    ?targetServerType=master&prepareThreshold=0
    &ssl=true&sslmode=verify-full
    &sslcert=/etc/config/ssl/postgresql/tls.crt
    &sslkey=/etc/config/ssl/postgresql/tls.key
    &sslrootcert=/etc/config/ssl/ca/root.crt
```

PEM-формат — не нужны
пароли от stores

Задаем URL с сертификатами для JDBC data source

```
@Bean
public HikariDataSource postresqlDataSource(
            SecretFilesProperties secretProps ) {
    var config = new HikariConfig();
    config.setJdbcUrl( secretProps.getJdbcUrlWithCertPaths() );
    // ...
    return new HikariDataSource( config );
}
```

Задаем URL с сертификатами для JDBC data source

```
@Bean
public HikariDataSource postresqlDataSource(
            SecretFilesProperties secretProps ) {
    var config = new HikariConfig();
    config.setJdbcUrl( secretProps.getJdbcUrlWithCertPaths() );
    // ...
    return new HikariDataSource( config );
}
```

DataSource готов к работе
с использованием mTLS

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
        - 3.1.1. Применение серверных SSL-сертификатов
        - 3.1.2. Применение клиентских SSL-сертификатов
        - 3.1.3. Применение SSL-сертификатов для соединений с БД
        - 3.1.4. Применение кредов БД
    - 3.2. Hot reload секретов при их изменении
4. Подводим итоги

Задаем креды из файлов для JDBC data source

```
@Bean
public HikariDataSource postresqlDataSource(
            SecretFilesProperties secretProps ) {
   var config = new HikariConfig();
   config.setUsername( stringFromFile( secretProps.getDbUsernamePath() ) );
   config.setPassword( stringFromFile( secretProps.getDbPasswordPath() ) );
   // ...
   return new HikariDataSource( config );
}
```

Best practice

```
@Bean
public HikariDataSource postresqlDataSource(
            SecretFilesProperties secretProps ) {
   var config = new HikariConfig();
   config.setUsername( stringFromFile( secretProps.getDbUsernamePath() ) );
   config.setPassword( stringFromFile( secretProps.getDbPasswordPath() ) );
   // ...
   return new HikariDataSource( config );
}
```

char[] предпочтительнее String
при работе с паролями

Best practice

```
@Bean
public HikariDataSource postresqlDataSource(
            SecretFilesProperties secretProps ) {
   var config = new HikariConfig();
   config.setUsername( stringFromFile( secretProps.getDbUsernamePath() ) );
   config.setPassword( stringFromFile( secretProps.getDbPasswordPath() ) );
   // ...
   return new HikariDataSource( config );
}
```

Однако, Hikari просит String

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах
4. Подводим итоги

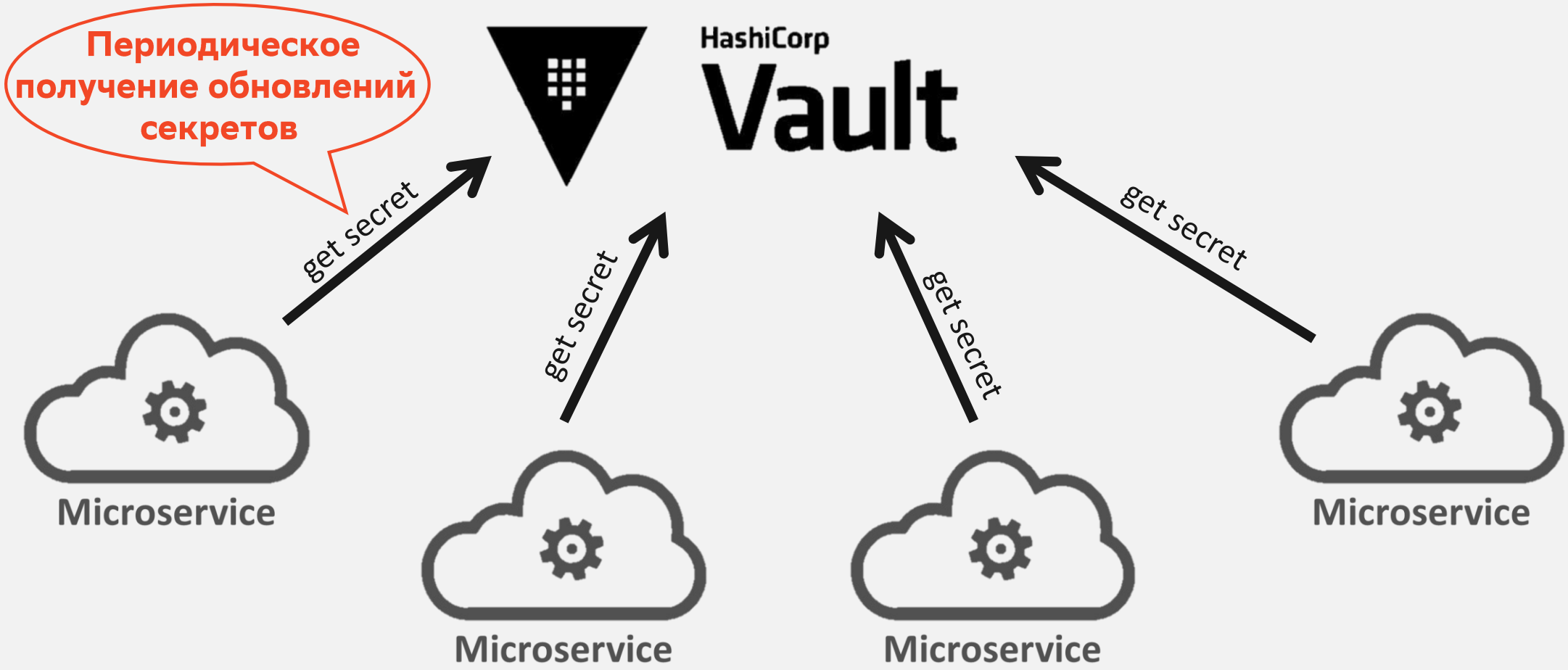
Секреты могут меняться

1. Периодическая ротация SSL-сертификатов
2. Компрометация кредов БД, требующая их замены
3. И т. п.

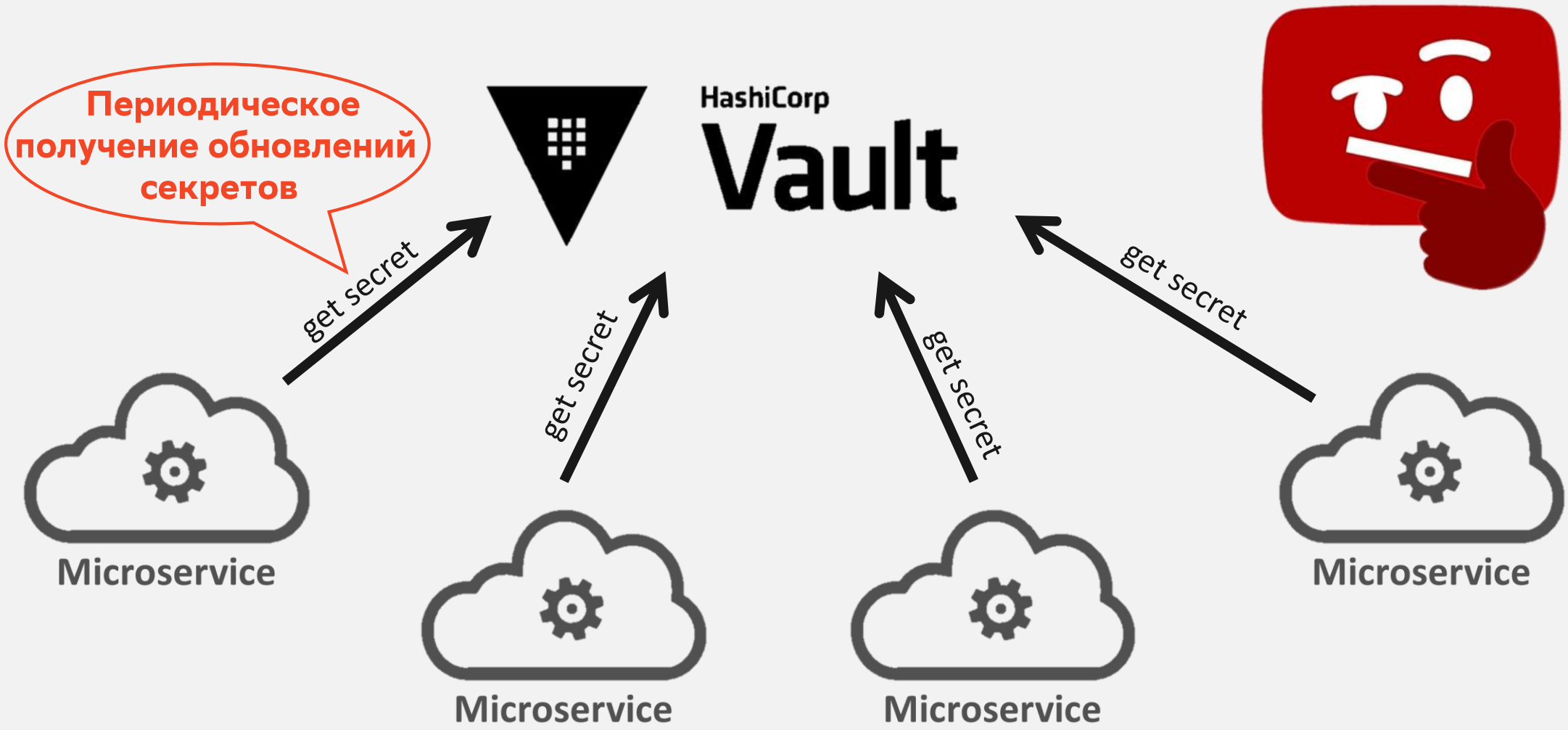
Секреты могут меняться

1. Периодическая ротация SSL-сертификатов
2. Компрометация кредов БД, требующая их замены
3. И т. п.

Нужно применять обновления файлов с секретами



Нужно применять обновления файлов с секретами



План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах
4. Подводим итоги

Обновление секретов через рестарт

Просто и надежно.

Мы были вынуждены отказаться.

Почему?

Почему не RollingUpdate в k8s?

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: application-name
spec:
  replicas: ...
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: ...
      maxSurge: ...
```

Почему не RollingUpdate в k8s?

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: application-name
spec:
  replicas: ...
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: ...
      maxSurge: ...
```

Вариант 1:

- Число pods **временно уменьшится**
- Увеличится нагрузка
- Может пострадать качество

Почему не RollingUpdate в k8s?

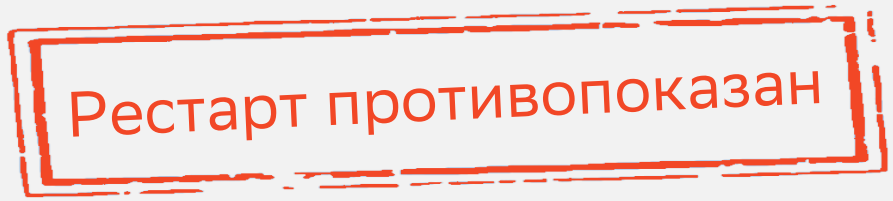
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: application-name
spec:
  replicas: ...
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: ...
      maxSurge: ...
```

Вариант 2:

- Число pods **временно увеличится**
- Бывают секреты на много сервисов
- Их изменение приведет к увеличению потребления железа в кластере

А как рестартовать сервисы на виртуалках?

- Нет RollingUpdate из коробки
- В SessionsData master-хранилище – stateful-сервис



Решено — hot reload секретов

- Не останавливая сервисы
- Не рестартуя сервисы
- Не снимая нагрузку с сервисов

Решено — hot reload секретов

- Не останавливая сервисы
- Не рестартуя сервисы
- Не снимая нагрузку с сервисов

Challenge — придется найти способ, как обновить секреты «на горячую».

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
            - 3.2.2.1. @RefreshScope не подходит для reload секретов
            - 3.2.2.2. Reload SSL bundles имеет недостатки для использования в production
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах
4. Подводим итоги

Обновление секретов стандартными средствами Spring

- @RefreshScope из Spring Cloud
- SSL bundles из Spring Boot 3.2+

На практике не подошло.

Почему?

План

@RefreshScope пересоздает beans при изменении env

```
@RefreshScope
@Bean
public Client httpClient( SecretFilesProperties secretProps ) {
    ClientBuilder clientBuilder =
        ClientBuilder.newBuilder().withConfig(...);
    clientBuilder.sslContext( createSslContext( secretProps ) );
    return clientBuilder.build();
}
```

@RefreshScope пересоздает beans при изменении env

```
@RefreshScope
@Bean
public Client httpClient( SecretFilesProperties secretProps ) {
    ClientBuilder clientBuilder =
        ClientBuilder.newBuilder().withConfig(...);
    clientBuilder.sslContext( createSslContext( secretProps ) );
    return clientBuilder.build();
}
```

После изменения файлов с секретами:

- либо `curl http://<host>:<port>/actuator/refresh --request POST`
- либо `applicationContext.publishEvent(new RefreshEvent(this, <event>, <eventName>))`

@RefreshScope пересоздает beans при изменении env

```
@RefreshScope
@Bean
public Client httpClient( SecretFilesProperties secretProps ) {
    ClientBuilder clientBuilder =
        ClientBuilder.newBuilder().withConfig(...);
    clientBuilder.sslContext( createSslContext( secretProps ) );
    return clientBuilder.build();
}
```

После изменения файлов с секретами:

- либо `curl http://<host>:<port>/actuator/refresh --request POST`
- либо `applicationContext.publishEvent(new RefreshEvent(this, <event>, <eventName>))`

Следующее обращение к bean пересоздаст его

@RefreshScope подошел не везде

- ✅ Клиентские SSL-сертификаты для Jersey client
- ❌ Серверные SSL-сертификаты для Tomcat
- ❌ Клиентские SSL-сертификаты и креды для HikariDataSource

@RefreshScope не подошел для Tomcat

1. Нет подходящего bean в Spring Boot

```
// @RefreshScope
// 2. Применить аннотацию невозможно – Spring Boot start error:
// Unable to start ServletWebServerApplicationContext
// due to multiple ServletWebServerFactory beans :
// scopedTarget.tomcatServletWebServerFactory,tomcatServletWebServerFactory
@Bean
public TomcatServletWebServerFactory tomcatFactory(
            SecretFilesProperties secretProps ) {
    var factory = new TomcatServletWebServerFactory();
    // customize using secretProps
    return factory;
}
```

** Свежий Spring Boot 3.2.5, свежий Tomcat 10.1.20*

@RefreshScope не подошел для Tomcat

```
@RefreshScope
@Bean
public TomcatServletWebServerFactory tomcatFactory(
            SecretFilesProperties secretProps ) {
    var factory = new TomcatServletWebServerFactory();
    // customize using secretProps
    return factory;
}
```

Все равно бы не вышло:
bean после старта не используется

@RefreshScope не подошел для HikariDataSource

```
@RefreshScope
@Bean
public HikariDataSource postresqlDataSource(
            SecretFilesProperties secretProps ) {
   var config = new HikariConfig();
   config.setJdbcUrl( secretProps.getJdbcUrlWithCertPaths() );
   config.setUsername( stringFromFile( secretProps.getDbUsernamePath() ) );
   config.setPassword( stringFromFile( secretProps.getDbPasswordPath() ) );
   // ...
   return new HikariDataSource( config );
}
```

Рвутся текущие соединения с БД: потоки получают `PSQLException`

** Свежий HikaryCP 5.0.1*

Почему @RefreshScope рвет коннекты к БД

1. `HikariDataSource` реализует `AutoClosable`

Почему @RefreshScope рвет коннекты к БД

1. `HikariDataSource` реализует `AutoClosable`
2. `close()` вызывает `HikariPool.shutdown()`, обрывающий коннекты

Почему @RefreshScope рвет коннекты к БД

1. `HikariDataSource` реализует `AutoClosable`
2. `close()` вызывает `HikariPool.shutdown()`, обрывающий коннекты
3. `RefreshScope` разрушает предыдущий экземпляр через `close()` при пересоздании `AutoClosable` bean

Почему @RefreshScope рвет коннекты к БД

1. `HikariDataSource` реализует `AutoClosable`
2. `close()` вызывает `HikariPool.shutdown()`, обрывающий коннекты
3. `RefreshScope` разрушает предыдущий экземпляр через `close()` при пересоздании `AutoClosable` bean

Недопустимо рвать коннекты под нагрузкой

Решено — не используем @RefreshScope

❌ Обновление SSL-сертификатов для Tomcat

❌ Обновление SSL-сертификатов и кредов для HikariDataSource

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
            - 3.2.2.1. @RefreshScope не подходит для reload секретов
            - 3.2.2.2. Reload SSL bundles имеет недостатки для использования в production
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах
4. Подводим итоги

SSL bundles делают hot reload сертификатов

```yaml
spring:
  ssl:
    bundle:
      jks:
        server-bundle:  # имя bundle
          key:
            alias: server
            password: <password value>
          keystore:
            type: PKCS12
            location: ${secrets.server.keyStorePath}
            password: <password value>
          truststore:
            type: PKCS12
            location: ${secrets.server.trustStorePasswordPath}
            password: <password value>
          reloadOnUpdate: true  # включает hot reload
```

SSL bundles удобно использовать

Для Tomcat

```
server:
  ssl:
    # вместо указания
    # сертификатов,
    # паролей и пр.
    bundle: server-bundle
    client-auth: need
```

Для HTTP-клиента

```
@Bean
public Client httpClient(SslBundles sslBundles){
  ClientBuilder clientBuilder =
      ClientBuilder.newBuilder()
                   .withConfig(...);
  SslBundle bundle =
      sslBundles.getBundle( "client-bundle" );
  clientBuilder.sslContext(
      bundle.createSslContext() );
  return clientBuilder.build();
}
```

Удобно создавать SSLContext

Reload SSL bundles не подошел по ряду причин

1. **Не работает в k8s, когда секреты монтируются из kind: Secret**
   External Secrets Operator может менять kind: Secret на лету
   Файлы с секретами — symlinks, которые остаются неизменными
   — нет реакции

Reload SSL bundles не подошел по ряду причин

1. Не работает в k8s, когда секреты монтируются из kind: Secret
2. **Не следит за изменениями паролей от keyStore и trustStore**
   В application.yaml пароли для bundles задаются строками
   Обновить пароли, взяв их из файлов, невозможно

Reload SSL bundles не подошел по ряду причин

1. Не работает в k8s, когда секреты монтируются из kind: Secret
2. Не следит за изменениями паролей от keyStore и trustStore
3. **Reload срабатывает при любых событиях изменения файлов**

   На практике было необходимо следить за конкретными событиями (e. g. delete only)

Reload SSL bundles не подошел по ряду причин

1. Не работает в k8s, когда секреты монтируются из kind: Secret
2. Не следит за изменениями паролей от keyStore и trustStore
3. Reload срабатывает при любых событиях изменения файлов
4. **Применение новых сертов через фиксированный timeout**
   По умолчанию через 10 секунд, но одновременно во всех pods/узлах
   На практике требовалось не одновременно

Reload SSL bundles не подошел по ряду причин

1. Не работает в k8s, когда секреты монтируются из kind: Secret
2. Не следит за изменениями паролей от keyStore и trustStore
3. Reload срабатывает при любых событиях изменения файлов
4. Применение новых сертов через фиксированный timeout

Сыровато для production

Reload SSL bundles не подошел по ряду причин

1. Не работает в k8s, когда секреты монтируются из kind: Secret
2. Не следит за изменениями паролей от keyStore и trustStore
3. Reload срабатывает при любых событиях изменения файлов
4. Применение новых сертов через фиксированный timeout

Сыровато для production

Не подходит для кредов БД

Решено — не используем SSL bundles

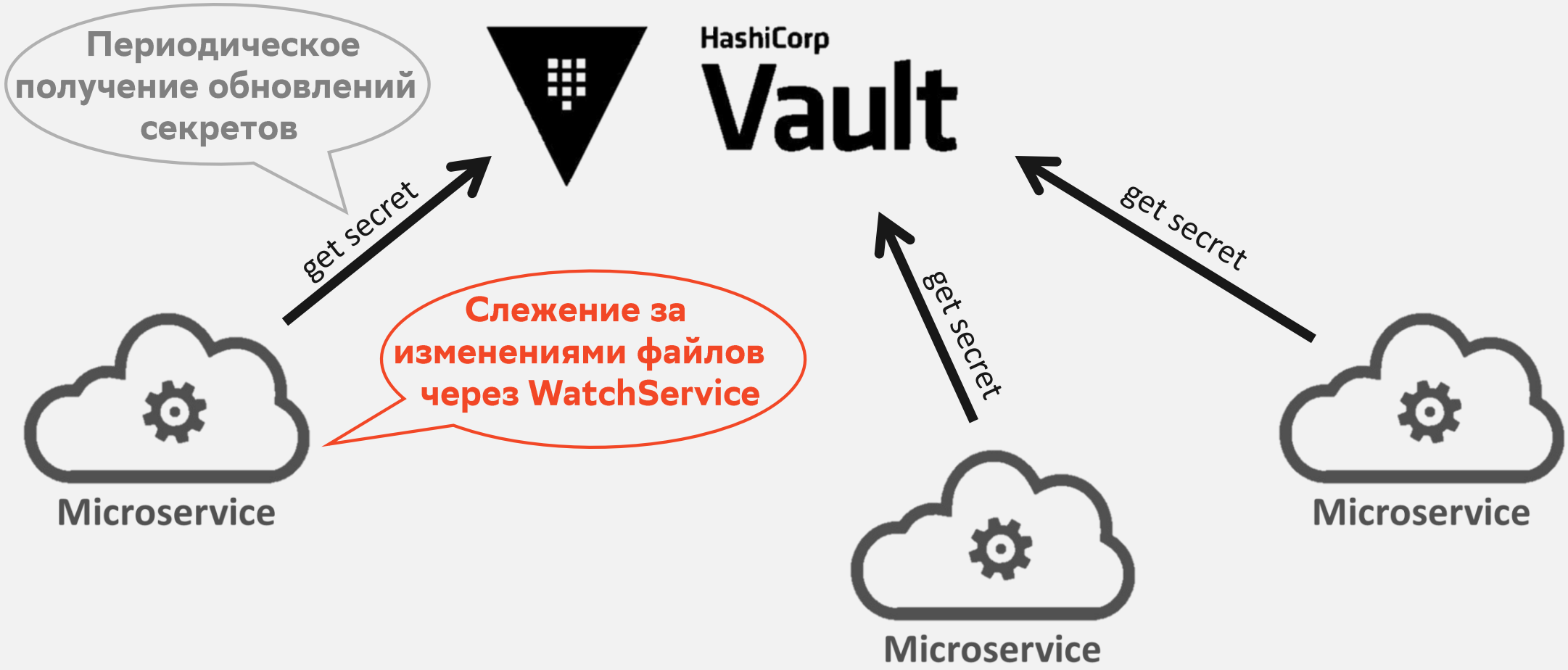
Пришлось реализовать **свой инструмент** для hot reload секретов:

- Java-сервисы на Spring Boot
- деплой хоть в k8s, хоть на виртуалки

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
            - 3.2.2.1. Если Vault Agent, то следим за созданием файлов
            - 3.2.2.2. Слежение за symlinks на файлы в Kubernetes
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах
4. Подводим итоги

Нужно следить за обновлениями файлов с секретами



Следим за файлами секретов через WatchService

```
WatchService watchService = FileSystems.getDefault().newWatchService();

Path secretsParentDir = watchedFilePath.getParent();
secretsParentDir.register( watchService,
        StandardWatchEventKinds.ENTRY_CREATE,
        StandardWatchEventKinds.ENTRY_MODIFY,
        StandardWatchEventKinds.ENTRY_DELETE );
```

Следим за файлами секретов через WatchService

```
WatchService watchService = FileSystems.getDefault().newWatchService();

Path secretsParentDir = watchedFilePath.getParent();
secretsParentDir.register( watchService, ... );

while (true) {                              // в отдельном потоке
  WatchKey key = watchService.take(); // засыпаем до изменения файлов
  if ( key != null ) {
    ...
}
```

Следим за файлами секретов через WatchService

```
WatchService watchService = FileSystems.getDefault().newWatchService();

Path secretsParentDir = watchedFilePath.getParent();
secretsParentDir.register( watchService, ... );

while (true) {                              // в отдельном потоке
  WatchKey key = watchService.take(); // засыпаем до изменения файлов
  if ( key != null ) {                      // секреты изменились!
    Path dirPath = ( Path )key.watchable();
    for ( WatchEvent<?> event : key.pollEvents() ) {
      Path filePath = dirPath.resolve((Path)event.context()).toAbsolutePath();
      WatchEvent.Kind<Path> eventKind = (WatchEvent.Kind<Path>)event.kind();
      // Читаем изменения из секретного файла filePath
    }
    key.reset();
  }
}
```

Следим за файлами секретов через WatchService

```
WatchService watchService = FileSystems.getDefault().newWatchService();

Path secretsParentDir = watchedFilePath.getParent();
secretsParentDir.register( watchService, ... );

while (true) {                                  // в отдельном потоке
  WatchKey key = watchService.take(); // засыпаем до изменения файлов
  if ( key != null ) {                          // секреты изменились!
    Path dirPath = ( Path )key.watchable();
    for ( WatchEvent<?> event : key.pollEvents() ) {
      Path filePath = dirPath.resolve((Path)event.context()).toAbsolutePath();
      WatchEvent.Kind<Path> eventKind = (WatchEvent.Kind<Path>)event.kind();
      // Читаем изменения из секретного файла filePath
    }
    key.reset();
  }
}
```

В SSL bundles так же

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
            - 3.2.2.1. Если Vault Agent, то следим за созданием файлов
            - 3.2.2.2. Слежение за symlinks на файлы в Kubernetes
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах
4. Подводим итоги

Как Vault Agent обновляет файл с секретом?

```
secrets/parent/dir/
├─ db-username.properties  // следим за этим файлом
└─ 497906923               // создает временный файл
```

Как Vault Agent обновляет файл с секретом?

```
secrets/parent/dir/
├─ db-username.properties  // следим за этим файлом
└─ 497906923               // наполняет новым содержимым
```

Как Vault Agent обновляет файл с секретом?

```
secrets/parent/dir/
├─ db-username.properties
└─ 497906923
```

Почему так?

Vault Agent обновляет файл с секретом атомарно

```
secrets/parent/dir/
├─ db-username.properties
└─ 497906923
```

Для атомарности изменения

** При прямой интеграции с Vault мы делаем так же*

Для файла с секретом есть только событие создания

```
secrets/parent/dir/
├─ db-username.properties
└─ 497906923
```

WatchService фиксирует только событие ENTRY_CREATE

```
secretsParentDir.register( watchService,
        StandardWatchEventKinds.ENTRY_CREATE );
```

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
            - 3.2.2.1. Если Vault Agent, то следим за созданием файлов
            - 3.2.2.2. Слежение за symlinks на файлы в Kubernetes
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах
4. Подводим итоги

Как External Secrets Operator обновляет файл с секретом?

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-with-files
stringData:
  db-username.properties: |
    db.username=<secret>
  db-password.properties: |
    db.password=<secret>
```

Обновляется на лету

```yaml
apiVersion: v1
kind: Pod
...
volumes:
- name: files
  secret:
    secretName: secret-with-files
containers:
- name: servant
  ...
  volumeMounts:
  - name: files
    mountPath: /secrets/parent/dir/
```

Как External Secrets Operator обновляет файл с секретом?

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: secret-with-files
stringData:
  db-username.properties: |
    db.username=<secret>
  db-password.properties: |
    db.password=<secret>
```

Обновляется на лету

```yaml
apiVersion: v1
kind: Pod
...
volumes:
- name: files
  secret:
    secretName: secret-with-files
containers:
- name: servant
  ...
  volumeMounts:
  - name: files
    mountPath: /secrets/parent/dir/
```

Как Kubernetes монтирует файлы из kind: Secret?

```
$ ls -la
total 0
drwxrwsrwx. … .
drwxr-sr-x. … ..
drwxr-sr-x. … ..2024_01_10_11_39_07.1287132292
lrwxrwxrwx. … ..data -> ..2024_01_10_11_39_07.1287132292
lrwxrwxrwx. … db-username.properties -> ..data/db-username.properties
lrwxrwxrwx. … db-password.properties -> ..data/db-password.properties
```

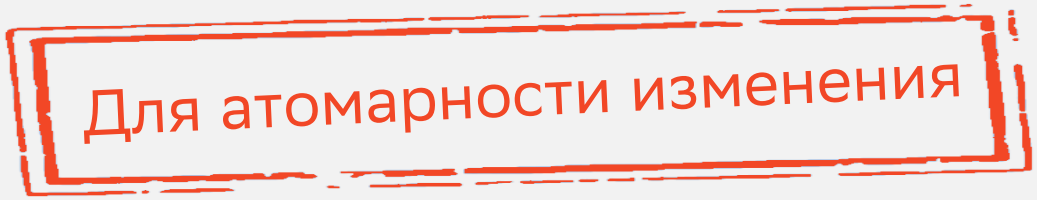
Как Kubernetes монтирует файлы из kind: Secret?

```
$ ls -la
total 0
drwxrwsrwx. … .
drwxr-sr-x. … ..
drwxr-sr-x. … ..2024_01_10_11_39_07.1287132292
lrwxrwxrwx. … ..data -> ..2024_01_10_11_39_07.1287132292
lrwxrwxrwx. … db-username.properties -> ..data/db-username.properties
lrwxrwxrwx. … db-password.properties -> ..data/db-password.properties
```

Почему так?

Kubernetes обновляет файлы с секретами атомарно

```
$ ls -la
total 0
drwxrwsrwx. … .
drwxr-sr-x. … ..
drwxr-sr-x. … ..2024_01_10_11_39_07.1287132292
lrwxrwxrwx. … ..data -> ..2024_01_10_11_39_07.1287132292
lrwxrwxrwx. … db-username.properties -> ..data/db-username.properties
lrwxrwxrwx. … db-password.properties -> ..data/db-password.properties
```



Как Kubernetes обновляет файлы с секретами атомарно?

```
secrets/parent/dir/
├─ ..2024_01_10_11_39_07.1287132292              // старый каталог
├─ ..2024_01_10_12_00_00.1287132292              // 1. новый каталог с файлами
├─ ..data -> ..2024_01_10_11_39_07.1287132292
├─ db-username.properties -> ..data/<filename>  // следим за этим
└─ db-password.properties -> ..data/<filename>  // и за этим
```

Как Kubernetes обновляет файлы с секретами атомарно?

```
secrets/parent/dir/
├─ ..2024_01_10_11_39_07.1287132292              // старый каталог
├─ ..2024_01_10_12_00_00.1287132292              // 1. новый каталог с файлами
├─ ..data -> ..2024_01_10_12_00_00.1287132292    // 2. атомарное изменение
├─ db-username.properties -> ..data/<filename>   // следим за этим
└─ db-password.properties -> ..data/<filename>   // и за этим
```

Как Kubernetes обновляет файлы с секретами атомарно?

```
secrets/parent/dir/
├─ ..2024_01_10_11_39_07.1287132292                  // 3. удаление старого каталога
├─ ..2024_01_10_12_00_00.1287132292                  // 1. новый каталог с файлами
├─ ..data -> ..2024_01_10_12_00_00.1287132292        // 2. атомарное изменение
├─ db-username.properties -> ..data/<filename>       // следим за этим
└─ db-password.properties -> ..data/<filename>       // и за этим
```

Что видит WatchService для секретных файлов в k8s?

```
secrets/parent/dir/
├─ ~~..2024_01_10_11_39_07.1287132292~~              // 3. удаление старого каталога
├─ ..2024_01_10_12_00_00.1287132292                  // 1. новый каталог с файлами
├─ ..data -> ..2024_01_10_12_00_00.1287132292        // 2. атомарное изменение
├─ db-username.properties -> ..data/<filename>       // следим за этим
└─ db-password.properties -> ..data/<filename>       // и за этим
```

А что видит WatchService?

- для symlinks на файлы — ничего
- для реальных файлов из старого каталога
  - 1 раз delete и тишина ...

Что видит WatchService для секретных файлов в k8s?

```
secrets/parent/dir/
├─ ~~..2024_01_10_11_39_07.1287132292~~                // 3. удаление старого каталога
├─ ..2024_01_10_12_00_00.1287132292                    // 1. новый каталог с файлами
├─ ..data -> ..2024_01_10_12_00_00.1287132292          // 2. атомарное изменение
├─ db-username.properties -> ..data/<filename>         // следим за этим
└─ db-password.properties -> ..data/<filename>         // и за этим
```

А что видит WatchService?

- для symlinks на файлы – ничего
- для реальных файлов из старого каталога
  - 1 раз delete и тишина ...

* https://ahmet.im/blog/kubernetes-inotify/

Подружили WatchService с обновлением секретов в k8s

```
WatchService watchService = FileSystems.getDefault().newWatchService();

Path realSecretFilePath = symlinkPath.toRealPath();
Path realSecretsParentDir = realSecretFilePath.getParent();
// Запоминаем связь realSecretFilePath -> symlinkPath
realSecretsParentDir.register( watchService,
        StandardWatchEventKinds.ENTRY_DELETE );
```

Подружили WatchService с обновлением секретов в k8s

```
WatchKey key = watchService.take();
if ( key != null ) {
  Path dirPath = ( Path )key.watchable();
  for ( WatchEvent<?> event : key.pollEvents() ) {
    Path realSecretFilePath = dirPath.resolve((Path)event.context()).toAbsolutePath();
    Path symlinkPath = ...;  // вспоминаем, как назывался наблюдаемый symlink
    if ( event.kind().equals( StandardWatchEventKinds.ENTRY_DELETE ) ) &&
①      Files.exists( symlinkPath ) &&  // symlink уже смотрит на другой файл
②      !symlinkPath.toRealPath().getParent().equals(realSecretFilePath.getParent()))
    {
      // Запоминаем новую связь symlinkPath.toRealPath() -> symlinkPath
③     symlinkPath.toRealPath().getParent().register( watchService,
              StandardWatchEventKinds.ENTRY_DELETE );
    }
    // Читаем изменения из symlinkPath
  }
  key.reset();
}
```

Подружили WatchService с обновлением секретов в k8s

Чего добились:

1. При обновлении kind: Secret получаем событие **ENTRY_DELETE**
2. Но читаем **новое содержимое** секрета, т. к. symlink обновлен
3. Событие ENTRY_DELETE приходит при каждом следующем обновлении

Подружили WatchService с обновлением секретов в k8s

Чего добились:

1. При обновлении kind: Secret получаем событие **ENTRY_DELETE**
2. Но читаем **новое содержимое** секрета, т. к. symlink обновлен
3. Событие ENTRY_DELETE приходит при каждом следующем обновлении

Подружили WatchService с обновлением секретов в k8s

Чего добились:

1. При обновлении kind: Secret получаем событие ENTRY_DELETE
2. Но читаем новое содержимое секрета, т. к. symlink обновлен
3. Событие ENTRY_DELETE приходит при каждом следующем обновлении

4. **Нюанс**: Если Vault Agent, то ловим ENTRY_CREATE,
   а если External Secrets Operator — ENTRY_DELETE
   Настраивается

Подружили WatchService с обновлением секретов в k8s

Отлаженное решение

https://gitverse.ru/chernov-af/file-watch

Работает и в k8s, и на VM

Подружили WatchService с обновлением секретов в k8s

Отлаженное решение

https://gitverse.ru/chernov-af/file-watch

Работает и в k8s, и на VM

SSL bundles пока так не умеют в k8s

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
            - 3.2.4.1. Reload серверных SSL-сертификатов
            - 3.2.4.2. Reload клиентских SSL-сертификатов
            - 3.2.4.3. Reload SSL-сертификатов для соединений с БД
            - 3.2.4.4. Reload кредов БД
        - ...

Отделяем слежение за секретами от их hot reload

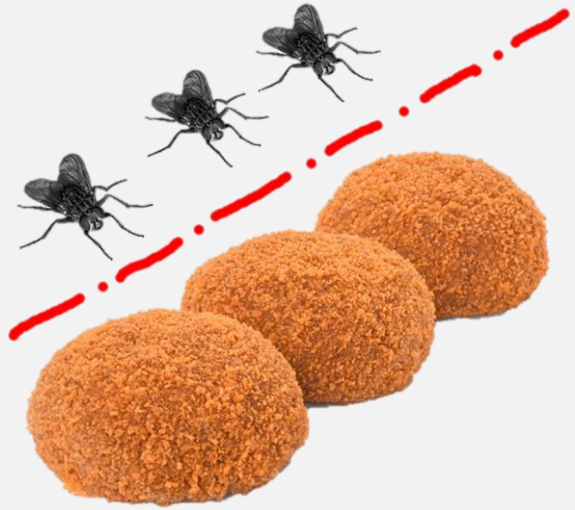
1. У слежения за изменениями файлов с секретами много нюансов 

Отделяем слежение за секретами от их hot reload

1. У слежения за изменениями файлов с секретами много нюансов 
2. У hot reload секретов нюансов не меньше 

Отделяем слежение за секретами от их hot reload

1. У слежения за изменениями файлов с секретами много нюансов 
2. У hot reload секретов нюансов не меньше 
3. Разделим эти слои, используя паттерн Publisher-Subscriber



Отделяем слежение за секретами от их hot reload

Hot reload секретов

```
public interface ChangeSubscriber {

  List<Path> subscriptions();

  /**
   * Hot reload здесь
   */
  void onChange(List<Path> changedFiles);
}
```

Отделяем слежение за секретами от их hot reload

Hot reload секретов

```
public interface ChangeSubscriber {

  List<Path> subscriptions();

  /**
   * Hot reload здесь
   */
  void onChange(List<Path> changedFiles);
}
```

Слежение за изменениями

```
public class ChangePublisher {

  public ChangePublisher(
    List<ChangeSubscriber> subscribers,
    WatchService watchService,
    EnumSet<EventKind> watchEventKinds)
  {
    ...
  }

  public void start() { ... }

  public void stop() { ... }
}
```

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
            - 3.2.4.1. Reload серверных SSL-сертификатов
            - 3.2.4.2. Reload клиентских SSL-сертификатов
            - 3.2.4.3. Reload SSL-сертификатов для соединений с БД
            - 3.2.4.4. Reload кредов БД
        - ...

Следим за keyStore, trustStore и паролями от них

```
public List<Path> subscriptions() {
  return List.of( secretProps.getServerKeyStorePath(),
                  secretProps.getServerKeyStorePasswordPath(),
                  secretProps.getServerKeyPasswordPath(),
                  secretProps.getServerTrustStorePath(),
                  secretProps.getServerTrustStorePasswordPath() );
}

// Tomcat при старте все это применил — теперь нужно переприменить
```

Достаем из Tomcat нужный объект для hot reload

Готовимся к «волшебному слову» для Tomcat

```
@Bean
public TomcatServletWebServerFactory tomcatFactory() {
  var tomcatFactory = new TomcatServletWebServerFactory();
  tomcatFactory.addConnectorCustomizers( new SslProtocolAccessor() );
  return tomcatFactory;
}
```

Достаем из Tomcat нужный объект для hot reload

Готовимся к «волшебному слову» для Tomcat

```
public class SslProtocolAccessor implements TomcatConnectorCustomizer {

  private Http11NioProtocol protocol;

  @Override
  public void customize( Connector connector ) {
    var protocol = ( Http11NioProtocol )connector.getProtocolHandler();
    if ( connector.getSecure() )
        this.protocol = protocol;
  }

  public Http11NioProtocol getProtocol() {
      return protocol;
  }
}
```

Hot reload серверных SSL-сертификатов Tomcat

```
public void onChange( List<Path> changedFiles ) {
  var customizers = tomcatFactory.getTomcatConnectorCustomizers();
  SslProtocolAccessor customizer = <находим наш в коллекции customizers>;
  Http11NioProtocol protocol = customizer.getProtocol();

  // Готовы к «волшебному слову»?
  protocol.reloadSslHostConfig( SSLHostConfig.DEFAULT_SSL_HOST_NAME );
}
```

Hot reload серверных SSL-сертификатов Tomcat

```
public void onChange( List<Path> changedFiles ) {
  var customizers = tomcatFactory.getTomcatConnectorCustomizers();
  SslProtocolAccessor customizer = <находим наш в коллекции customizers>;
  Http11NioProtocol protocol = customizer.getProtocol();

  // Готовы к «волшебному слову»?
  protocol.reloadSslHostConfig( SSLHostConfig.DEFAULT_SSL_HOST_NAME );
}
```

Сначала нужно задать новые пароли из файлов

Hot reload серверных SSL-сертификатов Tomcat

```
public void onChange( List<Path> changedFiles ) {
  var customizers = tomcatFactory.getTomcatConnectorCustomizers();
  SslProtocolAccessor customizer = <находим наш в коллекции customizers>;
  Http11NioProtocol protocol = customizer.getProtocol();
  setNewPasswordsInto( protocol );
  // Готовы к «волшебному слову»?
  protocol.reloadSslHostConfig( SSLHostConfig.DEFAULT_SSL_HOST_NAME );
}

private void setNewPasswordsInto( Http11NioProtocol protocol ) {
  SSLHostConfig sslHostConfig = <дефолтный из protocol.findSslHostConfigs()>;
  SSLHostConfigCertificate cert = <первый из sslHostConfig.getCertificates()>;
  cert.setCertificateKeystorePassword(
          stringFromFile( secretProps.getServerKeyStorePasswordPath() ) );
  cert.setCertificateKeyPassword(
          stringFromFile( secretProps.getServerKeyPasswordPath() );
  sslHostConfig.setTruststorePassword(
          stringFromFile( secretProps.getServerTrustStorePasswordPath() );
}
```

Hot reload серверных SSL-сертификатов Tomcat

```
public void onChange( List<Path> changedFiles ) {
  var customizers = tomcatFactory.getTomcatConnectorCustomizers();
  SslProtocolAccessor customizer = <находим наш в коллекции customizers>;
  Http11NioProtocol protocol = customizer.getProtocol();
  setNewPasswordsInto( protocol );
  // Готовы к «волшебному слову»!
  protocol.reloadSslHostConfig( SSLHostConfig.DEFAULT_SSL_HOST_NAME );
}

private void setNewPasswordsInto( Http11NioProtocol protocol ) {
  ...
}
```

Hot reload серверных SSL-сертификатов Tomcat

```
public void onChange( List<Path> changedFiles ) {
  var customizers = tomcatFactory.getTomcatConnectorCustomizers();
  SslProtocolAccessor customizer = <находим наш в коллекции customizers>;
  Http11NioProtocol protocol = customizer.getProtocol();
  setNewPasswordsInto( protocol );
  // Готовы к «волшебному слову»!
  protocol.reloadSslHostConfig( SSLHostConfig.DEFAULT_SSL_HOST_NAME );
}

private void setNewPasswordsInto( Http11NioProtocol protocol ) {
  ...
}
```

Пауза 100 – 500 мс
Клиенты ждут

Hot reload серверных SSL-сертификатов Tomcat

```
public void onChange( List<Path> changedFiles ) {
  var customizers = tomcatFactory.getTomcatConnectorCustomizers();
  SslProtocolAccessor customizer = <находим наш в коллекции customizers>;
  Http11NioProtocol protocol = customizer.getProtocol();
  setNewPasswordsInto( protocol );
  // Готовы к «волшебному слову»!
  protocol.reloadSslHostConfig( SSLHostConfig.DEFAULT_SSL_HOST_NAME );
}

private void setNewPasswordsInto( Http11NioProtocol protocol ) {
  ...
}
```

Пауза 100 – 500 мс
Клиенты ждут

Работает на
Spring Boot 2 и 3

Hot reload серверных SSL-сертификатов Tomcat

```
public void onChange( List<Path> changedFiles ) {
  var customizers = tomcatFactory.getTomcatConnectorCustomizers();
  SslProtocolAccessor customizer = <находим наш в коллекции customizers>;
  Http11NioProtocol protocol = customizer.getProtocol();
  setNewPasswordsInto( protocol );
  // Готовы к «волшебному слову»!
  protocol.reloadSslHostConfig( SSLHostConfig.DEFAULT_SSL_HOST_NAME );
}

private void setNewPasswordsInto( Http11NioProtocol protocol ) {
  ...
}
```

Пауза 100 – 500 мс
Клиенты ждут

Работает на
Spring Boot 2 и 3

В SSL bundles так же

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
            - 3.2.4.1. Reload серверных SSL-сертификатов
            - 3.2.4.2. Reload клиентских SSL-сертификатов
            - 3.2.4.3. Reload SSL-сертификатов для соединений с БД
            - 3.2.4.4. Reload кредов БД
        - ...

Следим за keyStore, trustStore и паролями от них

```
public List<Path> subscriptions() {
  return List.of( secretProps.getClientKeyStorePath(),
                  secretProps.getClientKeyStorePasswordPath(),
                  secretProps.getClientTrustStorePath(),
                  secretProps.getClientTrustStorePasswordPath() );
}

// Из этого при старте был собран SSLContext для Jersey client
```

Hot reload SSL-сертификатов HTTP-клиента

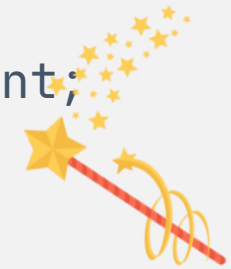
```
public void onChange( List<Path> changedFiles ) {
  ClientBuilder clientBuilder = ClientBuilder.newBuilder().withConfig(...);
  clientBuilder.sslContext( createSslContext( secretProps ) );
  Client newJerseyClient = clientBuilder.build();

  ...
}
```

Hot reload SSL-сертификатов HTTP-клиента

```
public void onChange( List<Path> changedFiles ) {
  ClientBuilder clientBuilder = ClientBuilder.newBuilder().withConfig(...);
  clientBuilder.sslContext( createSslContext( secretProps ) );
  Client newJerseyClient = clientBuilder.build();

  Client oldJerseyClient = Application.volatileHighloadClient;
  Application.volatileHighloadClient = newJerseyClient;
}
```



Hot reload SSL-сертификатов HTTP-клиента

```
public void onChange( List<Path> changedFiles ) {
  ClientBuilder clientBuilder = ClientBuilder.newBuilder().withConfig(...);
  clientBuilder.sslContext( createSslContext( secretProps ) );
  Client newJerseyClient = clientBuilder.build();

  Client oldJerseyClient = Application.volatileHighloadClient;
  Application.volatileHighloadClient = newJerseyClient;

  scheduleCloseOf( oldJerseyClient );  // abort connections, отложенный
                                       // на время > read timeout
}
```

Hot reload SSL-сертификатов HTTP-клиента

```
public void onChange( List<Path> changedFiles ) {
  ClientBuilder clientBuilder = ClientBuilder.newBuilder().withConfig(...);
  clientBuilder.sslContext( createSslContext( secretProps ) );
  Client newJerseyClient = clientBuilder.build();

  Client oldJerseyClient = Application.volatileHighloadClient;
  Application.volatileHighloadClient = newJerseyClient;

  scheduleCloseOf( oldJerseyClient );  // abort connections, отложенный
                                       // на время > read timeout
}
```

HikariDataSource с @RefreshScope
рвет соединения без ожидания

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
            - 3.2.4.1. Reload серверных SSL-сертификатов
            - 3.2.4.2. Reload клиентских SSL-сертификатов
            - 3.2.4.3. Reload SSL-сертификатов для соединений с БД
            - 3.2.4.4. Reload кредов БД
        - ...

Следим за pem-файлами с SSL-сертификатами

```
public List<Path> subscriptions() {
  return List.of( secretProps.getDbSslCertPath(),
                  secretProps.getDbSslKeyPath(),
                  secretProps.getDbSslRootCertPath() );
}

// При старте это было «зашито» в JDBC URL
```

Hot reload SSL-сертификатов для БД

```
public void onChange( List<Path> changedFiles ) {
  hikariDataSource.getHikariPoolMXBean().softEvictConnections();
}

// Даже файлы с секретами самим читать не нужно!
```

Hot reload SSL-сертификатов для БД

```
public void onChange( List<Path> changedFiles ) {
  hikariDataSource.getHikariPoolMXBean().softEvictConnections();
}

// Даже файлы с секретами самим читать не нужно!
```

Без недоступности работы с БД

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
            - 3.2.4.1. Reload серверных SSL-сертификатов
            - 3.2.4.2. Reload клиентских SSL-сертификатов
            - 3.2.4.3. Reload SSL-сертификатов для соединений с БД
            - 3.2.4.4. Reload кредов БД
    - ...

Следим за секретными файлами с кредами БД

```
public List<Path> subscriptions() {
  return List.of( secretProps.getDbUsernamePath(),
                  secretProps.getDbPasswordPath() );
}

// При старте это было передано в HikariDataSource
```

Hot reload кредов БД

```
public void onChange( List<Path> changedFiles ) {
  hikariDataSource.getHikariConfigMXBean().setUsername(
          stringFromFile( secretProps.getDbUsernamePath() ) );
  hikariDataSource.getHikariConfigMXBean().setPassword(
          stringFromFile( secretProps.getDbPasswordPath() ) );
  hikariDataSource.getHikariPoolMXBean().softEvictConnections();
}
```

Hot reload кредов БД

```
public void onChange( List<Path> changedFiles ) {
  hikariDataSource.getHikariConfigMXBean().setUsername(
          stringFromFile( secretProps.getDbUsernamePath() ) );
  hikariDataSource.getHikariConfigMXBean().setPassword(
          stringFromFile( secretProps.getDbPasswordPath() ) );
  hikariDataSource.getHikariPoolMXBean().softEvictConnections();
}
```

Без недоступности работы с БД

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
    - 3.1. Применение секретов при старте сервиса
    - 3.2. Hot reload секретов при их изменении
        - 3.2.1. Почему не рестарт, а hot reload?
        - 3.2.2. Почему не стандартные средства Spring для hot reload?
        - 3.2.3. Слежение за изменениями файлов с помощью WatchService
        - 3.2.4. Собственно, hot reload секретов под нагрузкой!
        - 3.2.5. Неодновременный hot reload секретов в разных pods/узлах 
4. Подводим итоги

Проблема одновременного hot reload

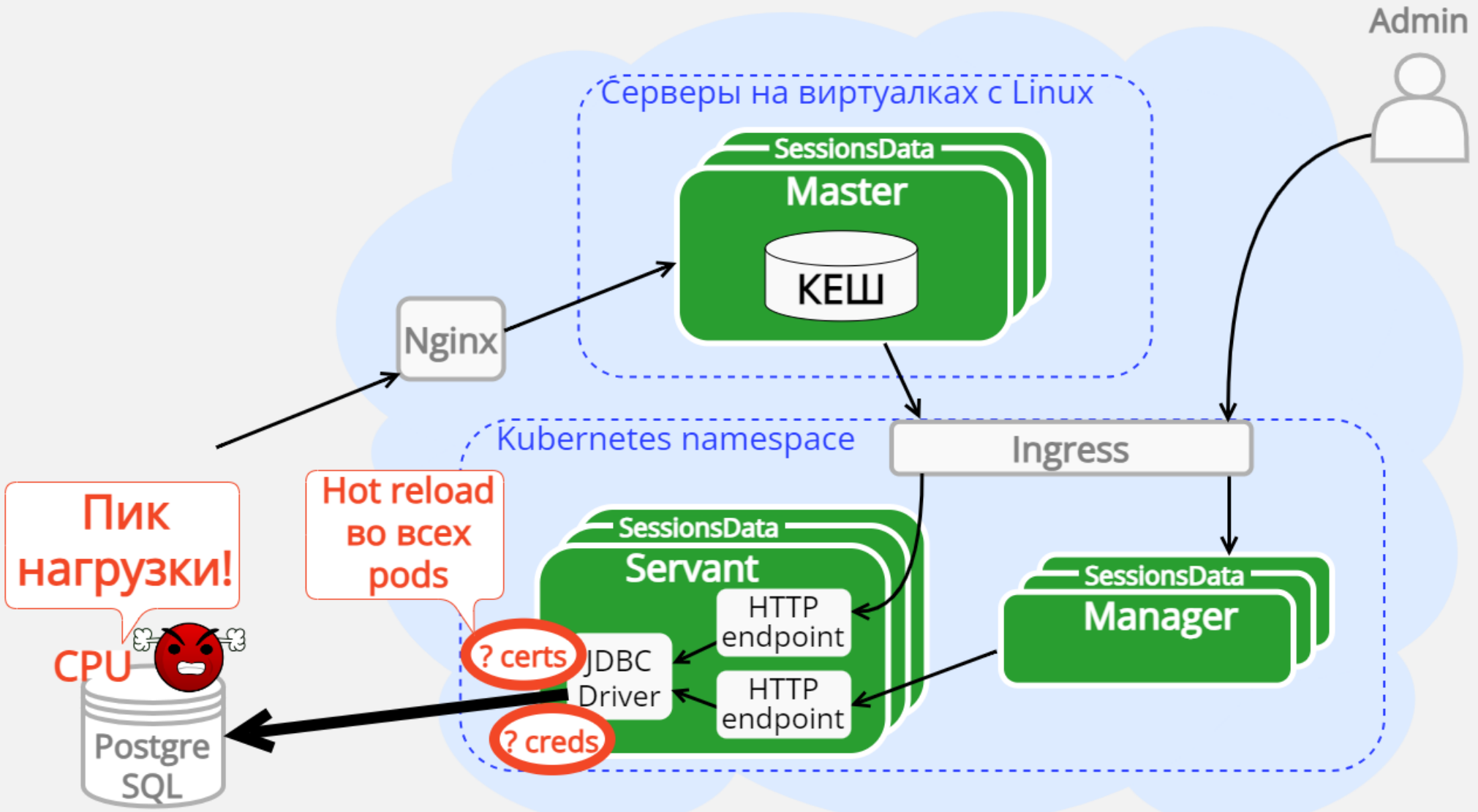
Одновременно обновляешь клиентские SSL-сертификаты — **создаешь лавину** пересозданий SSL-коннектов на сервере.

Проблема одновременного hot reload

Одновременно обновляешь клиентские SSL-сертификаты — **создаешь лавину** пересозданий SSL-коннектов на сервере.

Установка SSL-коннектов **тратит много CPU**.

Проблема одновременного hot reload



Неодновременный hot reload — нет проблемы

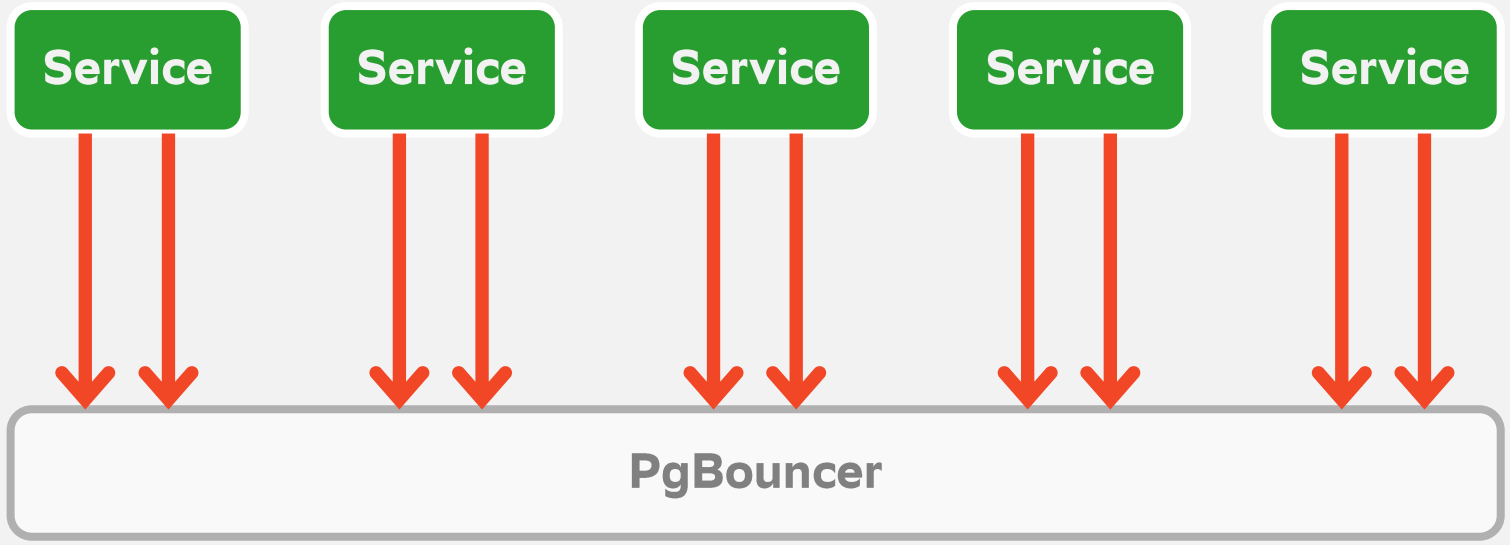
Избегаем одновременного hot reload во всех инстансах.
Random-ожидание (**jitter**) перед применением новых секретов.

Неодновременный hot reload — нет проблемы

```
public void onChange( List<Path> changedFiles ) {
   sleep( <jitter from 5 to 150 seconds> );
   // Применение изменений в changedFiles
}
```

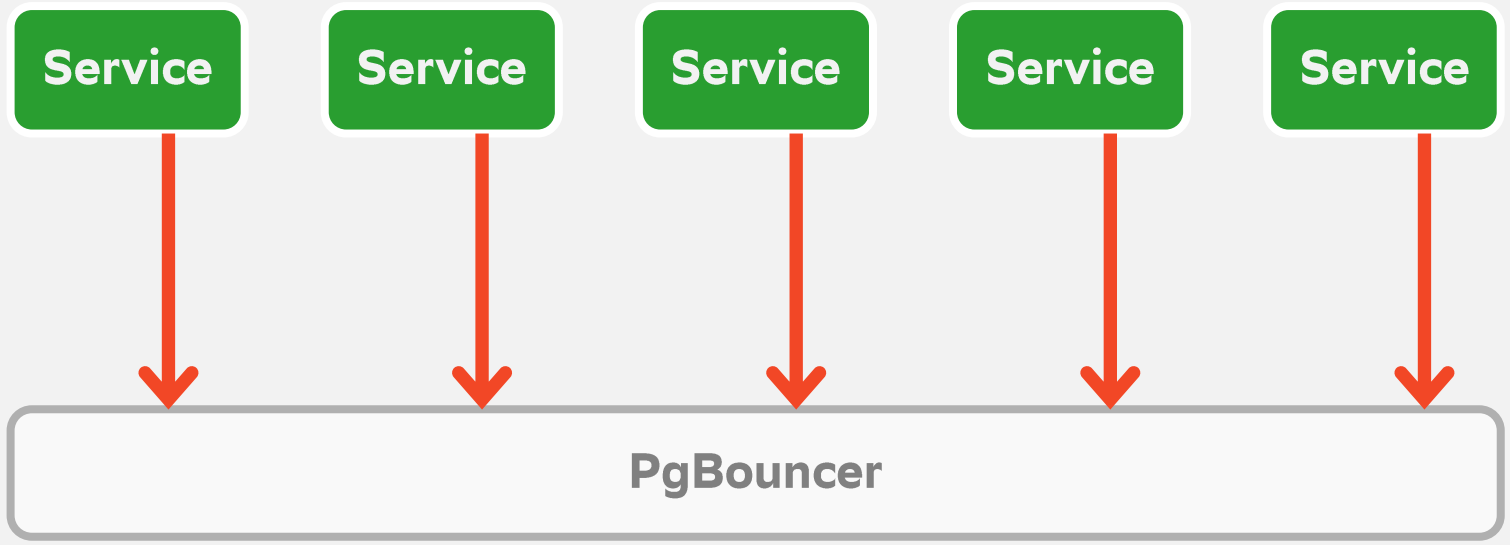
Неодновременный hot reload — нет проблемы

```
public void onChange( List<Path> changedFiles ) {
    sleep( <jitter from 5 to 150 seconds> );
    // Применение изменений в changedFiles
}
```



Неодновременный hot reload — нет проблемы

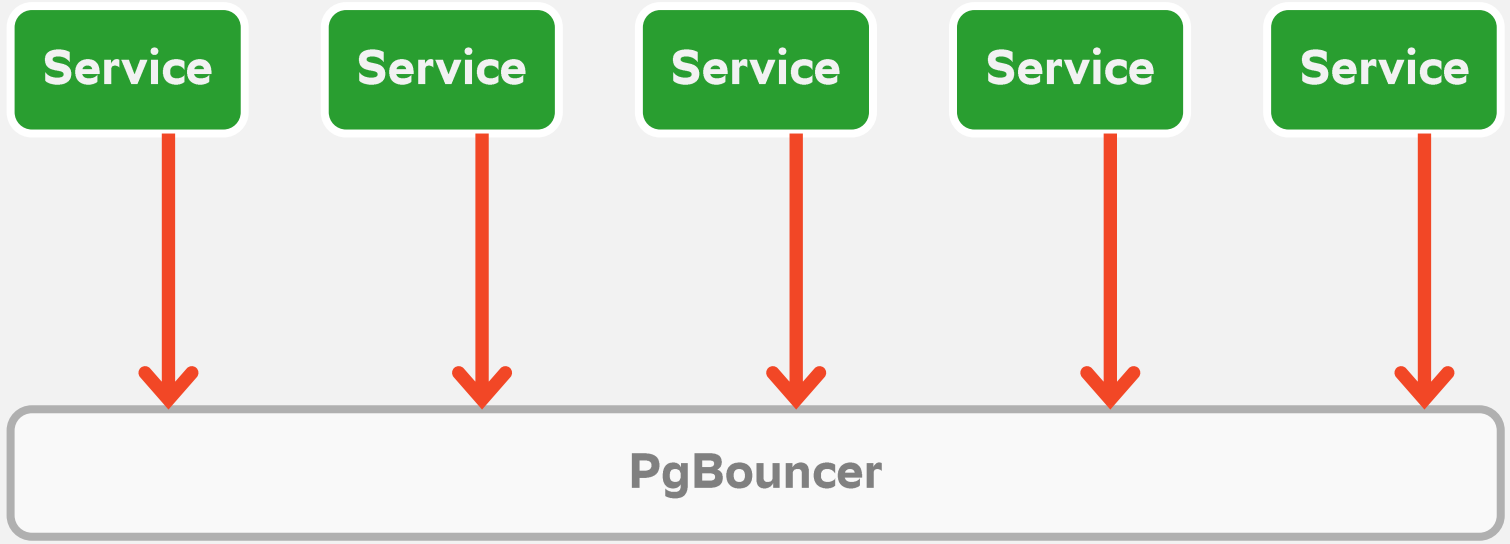
```
public void onChange( List<Path> changedFiles ) {
    sleep( <jitter from 5 to 150 seconds> );
    // Применение изменений в changedFiles
}
```



Экономим «железо»

Неодновременный hot reload — нет проблемы

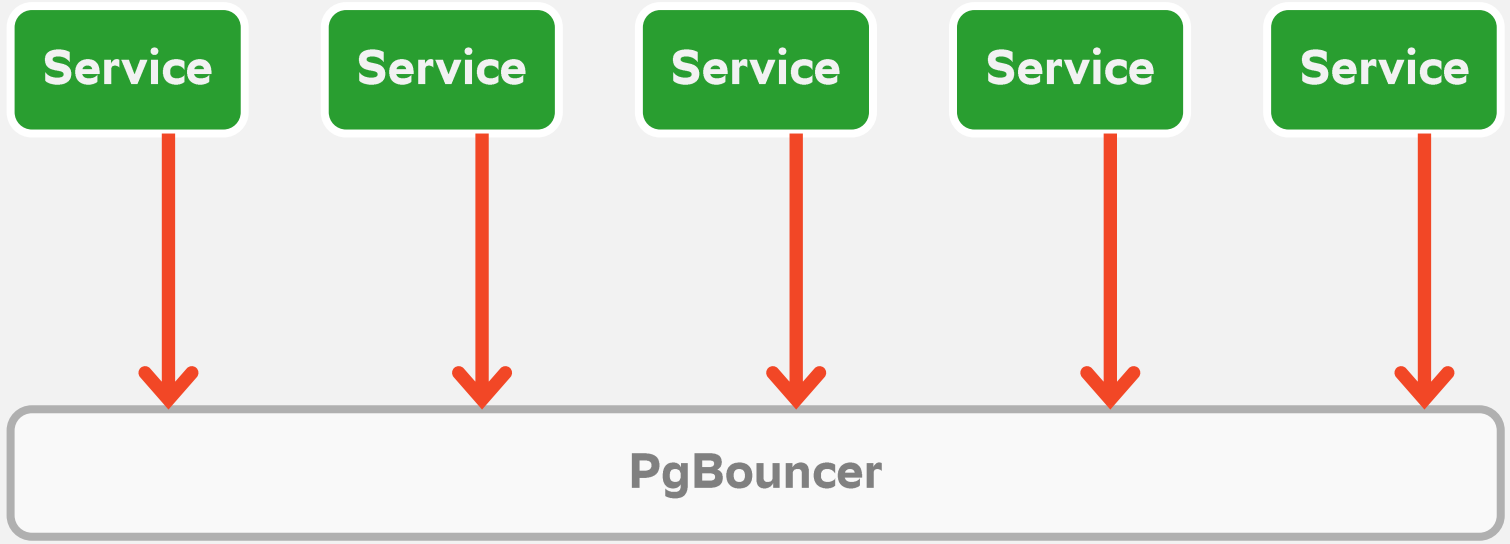
```
public void onChange( List<Path> changedFiles ) {
    sleep( <jitter from 5 to 150 seconds> );
    // Применение изменений в changedFiles
}
```



☝ *Похоже на RollingUpdate для обновления секретов*

Неодновременный hot reload — нет проблемы

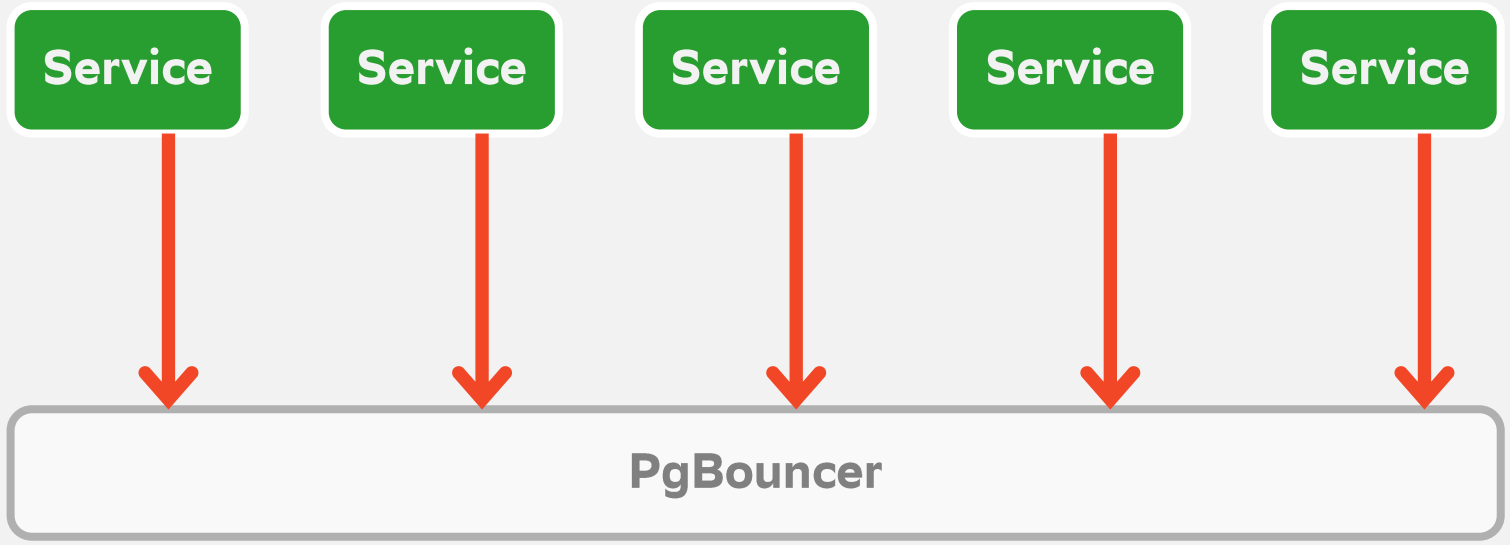
```
public void onChange( List<Path> changedFiles ) {
    sleep( <jitter from 5 to 150 seconds> );
    // Применение изменений в changedFiles
}
```



А в SSL bundles одновременный hot reload

Неодновременный hot reload — нет проблемы

```
public void onChange( List<Path> changedFiles ) {
    sleep( <jitter from 5 to 150 seconds> );
    // Применение изменений в changedFiles
}
```



Готово к production

План

1. Каждому сервису нужны секреты
2. Получение секретов для сервисов
3. Применение секретов в сервисах на Java
4. Подводим итоги

Выводы

1. Используйте внешний провайдер секретов

Централизованное, гибкое и безопасное управление секретами

Не важно, на чем написаны ваши сервисы и как они деплоятся

1.1. Разделите слои принесения секретов и их потребления

Выводы

1. Используйте внешний провайдер секретов
   Централизованное, гибкое и безопасное управление секретами
   Не важно, на чем написаны ваши сервисы и как они деплоятся

   1.1. Разделите слои принесения секретов и их потребления
   Секреты в виде файлов — легко поменять провайдера секретов

Выводы

1. Используйте внешний провайдер секретов
2. **Не используйте rolling update для применения новых секретов**
   Временное уменьшение числа pods — ухудшение качества сервиса
   Временное увеличение числа pods — нужно больше «железа»,
   а значит, денег

Выводы

1. Используйте внешний провайдер секретов
2. Не используйте rolling update для применения новых секретов
3. **Не бойтесь выполнять hot reload секретов под нагрузкой**
   В Java на Spring Boot обновить секреты «на горячую» дешевле, чем rolling update

   3.1. Разделите слои обнаружения изменений секретов и их hot reload

Выводы

1. Используйте внешний провайдер секретов
2. Не используйте rolling update для применения новых секретов
3. Не бойтесь выполнять hot reload секретов под нагрузкой

   В Java на Spring Boot обновить секреты «на горячую» дешевле, чем rolling update

   3.1. Разделите слои обнаружения изменений секретов и их hot reload

   Легко поменять слой слежения за изменениями, не трогая hot reload

Выводы

1. Используйте внешний провайдер секретов
2. Не используйте rolling update для применения новых секретов
3. Не бойтесь выполнять hot reload секретов под нагрузкой
4. **Выполняйте hot reload секретов в разных pods в разное время**
   Избегаем пиковых нагрузок на CPU сервера — экономим на «железе»

Выводы

1. Используйте внешний провайдер секретов
2. Не используйте rolling update для применения новых секретов
3. Не бойтесь выполнять hot reload секретов под нагрузкой
4. Выполняйте hot reload секретов в разных pods в разное время

Надеюсь, было полезно

SBER TECH

СПАСИБО!

Андрей Чернов
Java архитектор в СберТехе

@ chernovaf@mail.ru
chernovaf

SBER TECH

ДОПОЛНИТЕЛЬНЫЕ СЛАЙДЫ