цифра

Инструкция по развертыванию

Руководство системного администратора Zyfra Quality Lab

Версия ZQL – 5.3.0

Содержание

1. Аннотация

Данный документ разработан для системных инженеров и инженеров внедрения лабораторной информационной менеджмент системы (ЛИМС) **ZQL (Zyfra Quality Lab)** и содержит подробные сведения о пошаговой установке, настройке и обновлению системы.

2. Термины и определения

Helm-чарт (chart) — пакет Helm. Содержит описания ресурсов, необходимых для запуска приложения, инструмента или службы внутри кластера Kubernetes.

Репозиторий (Repository) — хранилище файлов, находящихся под контролем версий, вместе с историей их изменения и другой служебной информацией.

Пайплайн (Pipeline) — автоматизированная последовательность действий, которая позволяет интегрировать, тестировать и доставлять обновления программного обеспечения с максимальной эффективностью.

Задание (Job) — yaml-манифест, который создаёт под (pod) для выполнения разовой задачи.

Под (Pod) — это абстрактный объект Kubernetes, представляющий собой группу из одного или нескольких контейнеров приложения (например, Docker) вместе с ресурсами, которые они совместно используют.

Пространство имен (namespace) — механизм для логического разделения физического кластера Kubernetes на виртуальные, каждый из которых изолирован от других.

Вложенный чарт (subchart) — файлы вложенных чартов, которые не могут быть унифицированы, либо их унификация нецелесообразна.

Шаблон — шаблоны для формирования параметров для всех вложенных чартов.

Экземпляр (instance) — экземпляр продукта, установленный в определенной среде.

Мультитенантность (multi-tenancy) — возможность продукта, обеспечивающая обслуживание нескольких клиентов и предоставляющая для каждого клиента изолированную среду.

Тенант (tenant) — экземпляр изолированной среды, предоставляемый клиенту.

Persistent volume claim (PVC) – раздел для хранения данных в пространстве Kubernetes. Может быть реализован, например, папкой на сетевой файловой системе (NFS) или виртуальным диском в облачном хранилище.

3. Предварительные требования

Прежде чем приступить к изучению данного руководства, рекомендуется ознакомиться со следующими документами и ресурсами:

- Документация по Kubernetes;
- Документация по OKD;
- Документация по Helm.

4. Требования к программному обеспечению

Необходимы следующие предусловия для инициализации процесса установки (развертывания) и работы с сервисами ZQL в кластере Kubernetes:

1. Наличие пространства имен в кластере Kubernetes, предназначенного для развертывания ZQL.
2. Наличие в этом пространстве имен сервис-аккаунта (учётной записи) с достаточными правами для создания и редактирования ресурсов Kubernetes.
3. Установленная в этом же кластере платформа ZIIoT, которая совместима с версией ZQL, предназначенной для установки.
4. Наличие доступов для корректной инициализаций сервисов ZQL:
    a. Административный доступ к realm Keycloak платформы, к которой будет подключен ZQL.
    b. Административный доступ к СУБД PostgreSQL (наличие права CREATE_DB).
    c. Административный доступ к хранилищу S3 MinIO, входящему в состав платформы.
5. Наличие инструмента Helm версии 3 и выше для установки Helm-чарта ZQL.

Для успешной установки Helm-чарта ZQL необходимы следующие предусловия:

1. Наличие доступа к репозиторию, в котором хранится Helm-чарт ZQL. Возможные способы осуществления доступа:
    a. Прямой доступ к репозиторию Helm-чарта.
    b. Зеркалирование репозитория внутри корпоративной сети.
    c. Перенос Helm-чарта в виде архива с последующим использованием.
2. Наличие репозитория с параметрами стенда для возможности настройки под конкретное целевое окружение установки.

5. Этапы выполнения развертывания

Основные этапы установки ZQL

1. Подготовка к установке.
    a. Структура репозитория Helm-чарта ZQL.
    b. Настройка Helm-чарта.
    c. Настройка Платформы ZIIoT.
2. Установка Helm-чарта.
    a. Выполнение раздела Предусловия о подготовке файлов **global_values.yaml**, **versions.yaml**, **global_secrets.yaml**, **service_values.yaml**, **resouces.yaml**, **ConfigMap** с цепочкой доверия, указанного в global_values.yaml.
    b. Запуск Helm, завершение процессов инициализации, запуск и стабилизация всех микросервисов из перечня раздела Шаги по установке.

Дополнительные этапы установки ZQL

1. Установка Helm-чарта zif-proxy.
2. Работа с EntitiesConfiguration.
3. Реализация мультитенантности в виде разных пространств имен ZQL для каждого тенанта.
4. Изменение текста в интерфейсе приложений.

Этапы обновления ZQL

1. Выполнение раздела Предусловия о сборе сведений об именах **Helm-релиза ZQL** и **пространства имен, в котором установлен ZQL**.
2. Шаги по обновлению:
    a. Обновление файлов **global_values.yaml**, **versions.yaml**, **global_secrets.yaml**, **service_values.yaml** под целевой релиз обновления.
    b. Адаптация файла **resouces.yaml** под целевой релиз обновления.
    c. Запуск Helm.
    d. Завершение процессов инициализации, запуск и стабилизация всех микросервисов.

6. Подготовка к установке

6.1. Структура репозитория Helm-чарта ZQL

6.1.1. Репозиторий Helm-чарта

Каталоги репозитория:

1. **helm** - каталог чарта ZQL.
2. **pipelines** - пайплайн сборки Helm-пакета и обслуживающие задания (Jobs). Данный каталог не требуется использовать при установке.
3. **scripts** - вспомогательные скрипты.

Файлы репозитория:

1. **gitlab-ci.yml** - файл с описанием пайплайна проекта.
2. **CHANGELOG.md** - файл с описанием изменений, накопительный.
3. **Dockerfile** - файл для создания образа, предназначенного для сборки Helm-пакета из Helm-чарта.

6.1.2. Каталог helm

Подкаталоги:

1. **app_settings** - каталог с файлами JSON, в которых содержится конфигурация для фронтенд-приложений ZQL.
2. **charts** - каталог с вложенными чартами (subcharts) микросервисов и дополнительными сущностями, такими как конфигурационные карты (configmaps) и секреты (secrets), необходимыми для развертывания и функционирования микросервисов.
3. **scripts** - каталог со скриптами инициализации микросервисов.
4. **secpols** - каталог с JSON-файлами политик безопасности, которые импортируются в zif-security.
5. **templates** - шаблоны манифестов Helm-чарта.
6. **values** - файлы с параметрами микросервисов.

Файлы:

1. **Chart.yaml** - общее описание Helm-чарта и зависимостей. В нем указывается версия Helm-чарта для формирования Helm-пакета.
2. **global_secrets.yaml** - шаблон для формирования файла секретов для установки.
3. **global_values.yaml** - шаблон для формирования файла глобальных параметров чарта.
4. **versions_*.yaml** - файл с версиями микросервисов, который можно использовать в текущем виде или создать кастомный файл версий для стенда.
5. **resources.yaml** - файл с лимитами и реквестами для микросервисов. Не является обязательным для использования. Если файл не указан, то будут использованы лимиты/реквесты по умолчанию, указанные в values.yaml файлах вложенных чартов сервисов.

6.2. Настройка Helm-чарта

Настройка Helm-чарта позволяет адаптировать базовый пакет установки ЛИМС ZQL к конкретному окружению и набору функциональных возможностей, востребованных на стенде.

6.2.1. Настройка файла global_values.yaml

Настройка файла **global_values.yaml** выполняется в репозитории под параметры целевого стенда.

Примечание. В шаблоне файла **global_values.yaml** присутствуют комментарии, помогающие при заполнении его содержания.

1. Скопировать файл шаблона **global_values.yaml** из репозитория Helm-чарта.
2. Вставить скопированный файл в репозиторий или в каталог с параметрами стенда. Наименование файла может быть изменено, но при последующем запуске установки важно учесть новое наименование файла.
3. Установить в разделе **image** все адреса Docker registry.
4. Установить в подразделе **pullSecrets.name** имя секрета к Docker registry. Если требуется создание секрета для загрузки образов во время процесса установки, следует установить значение **pullSecrets.createSecret** = true. В случае использования **pullSecrets.createSecret** = true будет создан секрет с указанным именем из параметра pullSecrets.name, а логин/пароль будут взяты из параметров **registrySecret** файла секретов.
5. Указать в разделе **pv.sc** имя класса хранилища (storage class) для создания PVC.
6. Установить значения параметров **infraUrl**, **platformUrl**, **appsUrl**, **platformNamespace**, **infraNamespace**, **realm, platformTenant**.
7. Установить в разделе **kafka** адреса серверов Kafka.
8. Установить в разделе **postgres** параметры хоста PostgreSQL.

   Внимание! Начиная с версии ZQL 4.7.0 требуется явно указывать хост без использования шаблонизации.
9. Установить в разделе **minio** параметры хоста MinIO.
10. При необходимости установить в разделе **jaeger** параметры подключения к системе трассировки Jaeger.
11. При необходимости установить в разделе **logging** параметры ведения журнала (log) для микросервисов ZQL.

    Примечание. Как правило, значения по умолчанию считаются достаточными.
12. При необходимости установить в разделе **metrics** параметры отбора метрик микросервисами ZQL.
13. Установить в параметре **defaultZiiotServiceRole** роль по умолчанию для создаваемых сервис-аккаунтов клиентов Keycloak.
14. При необходимости включить инициализацию Keycloak, PostgreSQL и MinIO в подразделах **keycloakInit**, **postgresInit** и **minioInit** раздела **preInstallJobs**. Инициализацию необходимо включать при первой установке или изменении набора сервисов в случае обновления. После завершения инициализации и в случае, если состав сервисов не подлежит изменениям, можно отключить процесс инициализации.
15. Настроить в разделе **postInstallJobs** импорт политик и настроек фронтенд приложений в разделах **policiesInit** и **appSettingsInit** соответственно.
16. Установить в разделе **tests** необходимые параметры запуска автотестов.

17. Установить в разделе **site.type** нужный тип манифеста для **ingress: ingress** или **route**. При разворачивании в среде Kubernetes необходимо подготовить сертификаты и ключ для Ingress. Данное действие можно выполнить двумя методами:
    a. Метод 1:
        i. Создать вручную секрет с цепочкой сертификатов (сертификат для домена .Values.global.appsUrl и промежуточные и/или корневой сертификаты) и с ключом от сертификата домена при помощи команды

```
  kubectl create secret tls zif-tls-ingress-secret --
cert=/путь/к/цепочке_сертификатов.crt --key=/путь/к/ключу.key
```

    b. Метод 2:
        i. Удалить комментарий и установить параметры **tlsSecretCreate: true**, **tlsSecretName: "zif-tls-ingress-secret", tlsCertFrom: "certs/ingress.crt"** и **tlsKeyFrom: "certs/ingress.key"** для создания секрета для Ingress при установке Helm-чарта.
        ii. В CI\CD пайплайне добавить шаги для добавления файлов с цепочкой сертификатов в **certs/ingress.crt** и с ключом в **certs/ingress.key**.

            Внимание! Не рекомендуется держать файл с цепочкой сертификатов для домена и ключом от него в проекте окружения стенда. Прежде чем использовать данный метод, стоит учесть политику предприятия по работе с сертификатами и ключами.

            Пример файла с цепочкой сертификатов:

```
  -----BEGIN CERTIFICATE-----
  тело сертификата с CN для домена в параметре {{ .Values.global.appsUrl }} в
global_values.yaml
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  тело промежуточного "сертификата_1", которым подписан вышеуказанный
сертификат для домена
  -----END CERTIFICATE-----
  -----BEGIN CERTIFICATE-----
  тело корневого "сертификата_2", которым подписан промежуточный вышеуказанный
сертификат "сертификата_1"
  -----END CERTIFICATE-----
```

            Пример файла с ключом:

```
  -----BEGIN PRIVATE KEY-----
  тело ключа для сертификата домена (первый сертификат в
./ingress.example.crt).
  ВНИМАНИЕ:
  данный файл рекомендуется шифровать для хранения в проекте окружения или
держать в защищенном хранилище
  и дешифровать из хранилища в процессе установки Helm-чарта непосредственно
до запуска команды `helm upgrade --install`.
  -----END PRIVATE KEY-----
```

18. Подготовить корневые сертификаты (цепочку доверия). Данное действие возможно выполнить двумя методами:
    a. Метод 1:
        i. Скопировать из пространства имен платформы в пространство имен установки ZQL подготовленный ConfigMap **pem-ca-bundle**.
        ii. В разделе **extraCerts.configmap.name** необходимо установить имя ConfigMap из пункта выше.

b. Метод 2:

i. Установить параметр **create: true** и **createFrom: "certs/ca-bundle.crt"** для создания ConfigMap с цепочкой доверия из файла в параметре **createFrom**.

ii. Подготовить файл с цепочкой доверия **certs/ca-bundle.crt** в проекте стенда.
Пример расположения файла с цепочкой доверия в проекте окружения стенда и самого файла с цепочкой доверия:

```
-----BEGIN CERTIFICATE-----
тело промежуточного "сертификата_1"
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
тело промежуточного "сертификата_2"
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
тело корневого "сертификата_1"
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
тело корневого "сертификата_2"
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
тело корневого "сертификата_3"
-----END CERTIFICATE-----
```

6.2.2. Настройка версий микросервисов ZQL

1. Скопировать шаблон файла **versions_*.yaml** для требуемой версии релиза ZQL из Helm-чарта требуемой версии. К примеру, файл **versions_4.1.2.yaml** содержит версии релиза ZQL 4.1.2 и соответствующий ему состав микросервисов ZIAK.
2. Разместить файл шаблона в репозитории или каталоге настроек стенда (понадобится для корректировки версий при необходимости) с именем **versions.yaml**. Наименование файла может быть изменено, но при запуске установки важно учесть новое наименование файла.
3. При необходимости для каждого микросервиса в файле **versions.yaml** можно указать нужный тег образа микросервиса в разделе **%имя_микросервиса%.image.tag**.

6.2.3. Настройка секретов микросервисов ZQL

1. Скопировать шаблон файла секретов **global_secrets.yaml** для требуемой версии релиза ZQL из Helm-чарта требуемой версии.
2. Разместить скопированный файл в репозитории или каталоге настроек стенда.
3. Внести правки в скопированном файле:
   a. Установить значение для секрета registrySecret.registryUser и registrySecret.registryUser. Имя и пароль доступа к Docker registry.
   b. Установить значения ключей zqa-module-config-configmap.secretsData согласно комментариев в файле шаблона:
      i. KEYCLOAK_TEST_CLIENT - идентификатор клиента Keycloak, сервис-аккаунт которого имеет права для настройки портала приложений платформы и политик безопасности.
      ii. KEYCLOAK_TEST_CLIENT_SECRET - секрет клиента Keycloak, сервис-аккаунт которого имеет права для настройки портала приложений платформы и политик безопасности.

iii. KEYCLOAK_ZIF_REALM_ADMIN_USER - имя пользователя с правами администратора целевой конфигурации безопасности (realm) Keycloak.

iv. KEYCLOAK_ZIF_REALM_ADMIN_PASSWORD - пароль пользователя с правами администратора целевой конфигурации безопасности (realm) Keycloak.

v. POSTGRES_ADMIN_USER - имя пользователя Postgres, имеющего привилегированные права (CREATE DB, SUPERUSER).

vi. POSTGRES_ADMIN_PASSWORD - пароль пользователя Postgres, имеющего привилегированные права (CREATE DB, SUPERUSER).

vii. S3STORAGE_DEFAULT_ROOT_ACCESS_KEY - имя аккаунта с правами администратора в MinIO (S3 хранилище платформы).

viii. S3STORAGE_DEFAULT_ROOT_SECRET_KEY - секрет аккаунта с правами администратора в MinIO (S3 хранилище платформы).

4. Выполнить генерацию секретов. Генерацию секретов требуется выполнять для всех ключей в файле yaml секретов, кроме **registrySecret** и **zqa-module-config-configmap**. Альтернативный метод представлен в пункте 6.2.4

5. Установить значения секретов в виде обычного текста (plain text) вручную или с использованием соответствующего генератора (например, scripts/pw_gen.py из репозитория Helm-чарта ZQL). При формировании секретов важно соблюдать политику паролей, установленную для стенда.
Пример генерации случайных значений секретов из заготовки global_secrets.yaml (пункты 1-2):

```
  python3 pw_gen.py -i global_secrets.yaml -o global_secrets.non-encrypted.yaml
```

6. Разместить файл с секретами в безопасном хранилище системы непрерывной интеграции/непрерывной поставки (CI/CD), такой как GitLab или Jenkins, в соответствии с установленными политиками и правилами безопасности, принятыми для конкретного стенда или предприятия. Рекомендуется также провести шифрование данного файла, например, с использованием ключа Age, после чего поместить зашифрованную версию в репозиторий или каталог параметров стенда. В качестве примера шифрования можно использовать утилиты sops и age-keygen, как показано в следующем примере с шифрованием файла **global_secrets.non-encrypted.yaml**:

```
  age-keygen -y -o /path/to/age.pub /path/to/your/private/age.key
  sops -e --age $(cat /path/to/age.pub) global_secrets.non-enc.yaml > global_secrets.yaml
```

7. В дальнейшем этот файл требуется дешифровать во временный каталог пайплайна для его последующего использования в качестве values-файла для Helm. Пример дешифровки **global_secrets.yaml** с помощью утилиты sops:

```
  export SOPS_AGE_KEY=YOUR-PRIVATE-AGE-KEY
  sops -d global_secrets.yaml > global_secrets.non-encrypted.yaml
```

6.2.4. Автогенерация секретов при установке Helm-чарта

Внимание! Перед включением данной функциональности стоит учесть политику секретов/паролей стенда/предприятия.

Данная функциональность позволяет использовать механизм автогенерации случайных секретов/паролей при установке Helm-чарта с помощью встроенной функции Helm randAlphaNum. Автоматическая генерация заменяет ручную настройку, описанную в п.6.2.3.

По умолчанию длина секретов/паролей - 20 символов, но число символов может быть изменено параметром **randomPasswordLength** в **global_values.yaml**. После включения данной

функциональности Helm-чарт будет игнорировать секреты/пароли для микросервисов, указанные в **global_secrets.yaml**.

Внимание. Данная функциональность работает только при условии, что в **global_values.yaml** следующие параметры имеют значение **true**:

```
  alwaysGenerateRandomPassword: true
  keycloakInit:
    enabled: true
  postgresInit:
    enabled: true
  minioInit:
    enabled: true
```

Иначе Helm-чарт будет использовать секреты/пароли для микросервисов, указанные в **global_secrets.yaml**.

6.3. Настройка Платформы ZIIoT

Внимание. Блок **modules** в файле **services/service-list-patch.yaml** не может повторяться.

Внимание. После добавления файлов с новыми параметрами следует применить изменения, запустив установщик платформы.

6.3.1. Изменение версий сервисов в платформе ZIIoT, необходимых для группы приложений ZQL

Список версий сервисов, которые отличаются от версий в поставке платформы ZIIoT и требуются для релизов ZQL:

- Релиз 5.0.0:
  - в конфигурации платформы версии 2.16.x и выше изменение версии сервиса zif-sm-testspecification не требуется
- Релиз 4.8.0:
  - zif-sm-testspecification 1.9.2
- Релиз 4.7.0:
  - zif-sm-testspecification 1.9.1
- Патч-версия 4.6.7:
  - zif-sm-testspecification 1.9.1
- Патч-версия 4.6.6:
  - zif-sm-testspecification 1.9.1
- Патч-версия 4.6.5:
  - zif-sm-testspecification 1.9.1
- Патч-версия 4.6.3:
  - zif-sm-testspecification 1.8.5-mod-zql
- Патч-версия 4.6.x:
  - zif-sm-testspecification 1.8.4-mod-zql.

Для включения кастомных версий микросервисов в платформе ZIIoT следует:

1. Создать файл **services/service-list-patch.yaml** в каталоге параметров платформы.
2. Указать в файле следующее содержимое:

```
  modules:
  - name: <имя модуля сервиса>
    enabled: true
    services:
      - name: <имя сервиса>
        tag: <версия сервиса>
        <прочие конфигурационные параметры сервиса>
```

<имя модуля сервиса> и **<прочие конфигурационные параметры сервиса>** следует взять из файла **service-list.yaml**, который расположен в репозитории zifctl в каталоге **services**.

Альтернативным способом получения файла **service-list.yaml** является запуск docker-образа zifctl с нужным тегом (тег образа = версия платформы). К примеру, для платформы ZIIoT 2.13.6 список модулей с их параметрами и списком сервисов, включая версий и параметры сервисов, можно получить с помощью команды:

```
  docker run -it --rm --entrypoint 'cat' docker-group.idp.yc.ziiot.ru/infra/zifctl:2.13.6 /platform/services/service-list.yaml
```

6.3.2. Включение модулей (сервисов) ZIIoT, необходимых для группы приложений ZQL

Для обеспечения правильного функционирования ZQL требуется включить в его компоненты сервисы bp-workers. Для этого следует:

1. Создать файл **services/service-list-patch.yaml** в каталоге параметров платформы.
2. Добавить в файл следующее содержимое:

```
  modules:
  - name: bp-workers
    enabled: true
```

Для политик доступа **zif-file-storage** в пользовательском интерфейсе MinIO необходимо внести изменения:

1. Перейти в раздел **Access**.
2. Открыть политику с именем **ziiot--zif-file-storage**.
3. Перейти в раздел **Raw Policy**.
4. Изменить значение "arn:aws:s3:::ziiot--zif-file-storage/" на "arn:aws:s3:::*".

6.3.3. Добавление переменных сервисов платформы ZIIoT, необходимых для функционирования с приложениями ZQL

zif-reporting

1. Добавить файл в репозиторий с параметрами платформы **values/process/zif-reporting.yaml.gotmpl**.
2. Вставить в файл следующее содержимое:

```
  environmentVars:
  # {{ .Values.externalBaseHostname }} соответствует базовому URL инстанса/тенанта платформы
  EXPORTED_DOCUMENT_EXTERNAL_URL_PREFIX: "https://{{ .Values.externalBaseHostname }}/zif-reporting"
```

3. Внести правки. В переменной **EXPORTED_DOCUMENT_EXTERNAL_URL_PREFIX** указать значение **REST_ZIF_REPORTING_URL** из ресурса ConfigMap платформы zif-service-external-links. Пример:

```
  EXPORTED_DOCUMENT_EXTERNAL_URL_PREFIX: https://%platformUrl%/zif-reporting,
где %platformUrl% - это базовый URL экземпляра платформы.
```

zif-workflow

1. Добавить файл в репозиторий с параметрами платформы **values/process/zif-workflow.yaml.gotmpl**.
2. Вставить в файл следующее содержимое:

```
  environmentVars:
    REST_ZIF_OM_EXTENSIONS_CORE_URL:  'https://`%appsUrl%`/zif-om-extensions-
core'
    REST_ZQA_ANALYSISREQUESTS_SIMPLE_URL:           'https://`%appsUrl%`/zqa-
analysisrequests-simple'
    REST_ZQA_MATERIALS_URL: 'https://`%appsUrl%`/zqa-materials'
    REST_ZQA_PERSONNEL_URL: 'https://`%appsUrl%`/zqa-personnel'
    REST_ZQA_SAMPLETEMPLATES_URL: 'https://`%appsUrl%`/zqa-sampletemplates'
    REST_ZQA_SAMPLINGPOINTS_URL: 'https://`%appsUrl%`/zqa-samplingpoints'
    REST_ZQA_ADMIN_URL: 'https://`%appsUrl%`/zqa-admin'
    REST_ZQA_SAMPLES_URL: 'https://`%appsUrl%`/zqa-samples'
    REST_ZQA_IQC_URL: 'https://`%appsUrl%`/zqa-iqc'
    REST_ZQA_CONTENTSTORAGE_URL: 'https://`%appsUrl%`/zqa-contentstorage'
    REST_UDL_DFAWEBAPI_URL: "https://{{ .Values.externalBaseHostname }}/zif-
udl-dfawebapi"
    GRAPHQL_ZIF_OM_GRAPHQL_URL:   'https://{{   .Values.externalBaseHostname
}}/zif-om-graphql/graphql'
    REST_ZIF_EVENTS_URL:  'https://{{  .Values.externalBaseHostname  }}/zif-
events'
    REST_ZIF_MATERIAL_LOT_URL:    'https://{{    .Values.externalBaseHostname
}}/zif-material-lot'
    REST_ZIF_OM_OBJECT_URL: 'https://{{ .Values.externalBaseHostname }}/zif-
om-object'
    REST_ZIF_OM_PROPERTIES_URL:   'https://{{   .Values.externalBaseHostname
}}/zif-om-object'
    REST_ZIF_OM_PROPERTIES_VIEW_URL: 'https://{{ .Values.externalBaseHostname
}}/zif-om-object/propertyview'
    REST_ZIF_RDM_COMMON_URL: 'https://{{ .Values.externalBaseHostname }}/zif-
rdm-common'
    REST_ZIF_REPORTING_URL: 'https://{{ .Values.externalBaseHostname }}/zif-
reporting/api/v0.1'
    REST_ZIF_SM_DIRECTORIES_URL:   'https://{{   .Values.externalBaseHostname
}}/zif-sm-directories'
    REST_ZIF_SM_OPERATIONPERFORMANCE_URL:                              'https://{{
.Values.externalBaseHostname }}/zif-sm-operationperformance'
    REST_ZIF_SM_WORKSCHEDULE_URL:  'https://{{  .Values.externalBaseHostname
}}/zif-sm-workschedule'
    # GUID модели GlobalModel в zif-sm-directories для процесса
"Пробоподготовка" QL.AnalysisRequest.Workflow.SamplePreparation
    DEFAULT_GLOBAL_MODEL_ID: 00000000-0000-0000-0000-000000000000
```

3. Внести правки. Установить значение **%appsUrl%** - базовый URL приложения ZQL, который указывается в глобальных параметрах в ключе appsUrl. Где:
    - **%appsUrl%** —базовый URL приложения ZQL, который указывается в глобальных параметрах в ключе appsUrl (необходимо поменять);
    - **{{ .Values.externalBaseHostname }}** — значение, в которое подставляется базовый URL экземпляра/тенанта при развертывании платформы (подставляется автоматически при развертывании).

zif-sm-operationperformance

1. Добавить файл **values/sm/zif-sm-operationperformance.yaml.gotmpl** в репозиторий с параметрами платформы.
2. Внести следующее содержимое.

```
environmentVars:
DATETIME_KIND_MODE: "REPLACE_TO_UTC"
```

zui-app-shell

1. Добавить файл **values/portal/zui-app-shell.yaml.gotmpl** в репозиторий с параметрами платформы.
2. Внести следующее содержимое.

```
environmentVars:
REDIRECT_ TO_FAVORITE_APPLICATION: false
```

zif-propertyset

1. Добавить файл **values/om/zif-propertyset.yaml.gotmpl** в репозиторий с параметрами платформы.
2. Внести следующее содержимое.

```
environmentVars:
OBJECT_MODEL_PERSONNEL_PROTOTYPE_NAME: имя прототипа модели Персонал
```

3. Указать значение переменной **OBJECT_MODEL_PERSONNEL_PROTOTYPE_NAME**. Имя прототипа модели Персонал следует взять из файла values/zqa-personnel-data-configmap.yaml, находящегося в репозитории Helm-чарта ZQL. Если были внесены изменения в файлы values и создана кастомная конфигурация для Helm-чарта ZQL в репозитории стенда, то имя модели также можно извлечь из соответствующего файла в этой кастомной конфигурации. Альтернативным решением является подстановка в значение вышеуказанной переменной имени прототипа модели Персонал после установки Helm-чарта ZQL из приложения Объектная модель. Значения по умолчанию см. в файле zif-propertyset.yaml.gotmpl.

zif-notifications

1. Добавить файл values/notifications/zif-notifications.yaml.gotmpl в репозиторий с параметрами платформы.
2. Внести следующее содержимое.

```
environmentVars:
  ZIF_OM_OBJECT_MODEL_PROTOTYPE_CODE: имя модели Персонал
```

3. Указать значение переменной **ZIF_OM_OBJECT_MODEL_PROTOTYPE_CODE**. Имя модели Персонал следует взять из файла values/zqa-personnel-data-configmap.yaml, находящегося в репозитории Helm-чарта ZQL. Если были внесены изменения в файлы values и создана кастомная конфигурация для Helm-чарта ZQL в репозитории стенда, то имя модели также можно извлечь из соответствующего файла в этой кастомной конфигурации. Альтернативным решением является подстановка в значение вышеуказанной переменной из приложения Объектная модель после установки Helm-чарта ZQL. Значения по умолчанию см. в файле zif-notifications.yaml.gotmpl.

zui-app-om

1. Добавить файл **values/om/zui-app-om.yaml.gotmpl** в репозиторий с параметрами платформы.
2. Внести следующее содержимое.

```
  environmentVars:
  ZIF_OM_EQUIPMENT_MODEL_PROTOTYPE_ID: ID прототипа модели Оборудования
  ZIF_OM_PERSONNEL_MODEL_PROTOTYPE_ID: ID прототипа модели Персонал
  ZIF_OM_MATERIAL_MODEL_PROTOTYPE_ID: ID прототипа модели Материалы
  ZIF_OM_PERSONNEL_MODEL_ID: ID модели Персонал
  ZIF_OM_EQUIPMENT_MODEL_ID: ID модели Оборудование
  ZIF_OM_WORKDEFINITION_MODEL_ID: 00000000-0000-0000-0000-000000000000
```

В значения вышеуказанных переменных после установки Helm-чарта следует подставить соответствующие ID из приложения Объектная модель. Значения по умолчанию см. в файле zui-app-om.yaml.gotmpl.

7. Установка Helm-чарта

7.1. Предусловия

Для установки или обновления Helm-чарта ZQL должны быть выполнены условия:

1. Сформирован файл **global_values.yaml** и настроен согласно вышеуказанным разделам и комментариев в нем.
2. Сформирован файл **versions.yaml** и, при необходимости, настроены версии согласно вышеуказанным разделам.
3. Сформирован файл с секретами **global_secrets.yaml** согласно правил и политик стенда/предприятия.
4. Сформирован файл **resouces.yaml**. Данный пункт не является обязательным.
5. Из пространства имен платформы (экземпляра или тенанта) в пространство имен для установки приложений ZQL скопированы следующие объекты Kubernetes: ConfigMap pem-ca-bundle или иной ConfigMap с цепочкой доверия, указанный в global_values.yaml, согласно вышеуказанным разделам.

7.2. Шаги по установке

1. Перейти в каталог с Helm-чартом.
2. Выполнить слияние всех файлов с параметрами микросервисов. Пример команды в среде bash:

```
  cat ./values/*.yaml > ./service_values.yaml
```

3. Разместить файл **service_values.yaml** в репозитории или каталоге, из которого будет производиться установка.
4. Запустить Helm с параметрами:

```
  helm upgrade --install <lims> <helm> -f ./global_values.yaml -f
./service_values.yaml-f ./<secrets_values_file> -f ./versions.yaml -f
./resources.yaml -n lims_namespace --timeout 40m --debug
```

Где:

- **<lims>** - необходимое имя релиза Helm;

- **<helm>** - каталог с чартом (в зависимости от требований может именоваться иначе);
- **./global_values.yaml** - путь к файлу с глобальными параметрами;
- **./service_values.yaml** - путь к файлу из пункта 3 с параметрами микросервисов;
- **./<secrets_values_file>** - путь к файлу с секретами, созданному согласно вышеуказанным разделам;
- **./versions.yaml** - путь к файлу с версиями микросервисов;
- **./resources.yaml** - путь к файлу с лимитами и реквестами микросервисов. Опциональный параметр, если файл не указан, то будут использованы лимиты/реквесты по умолчанию, указанные в charts/<имя микросервиса>/values.yaml;
- **lims_namespace** - пространство имен установки ZQL.

5. Дождаться отработки процессов инициализации, которые запускаются в виде объектов Jobs в Kubernetes в ходе установки Helm-чарта.
6. Дождаться запуска и стабилизации всех микросервисов.

8. Установка Helm-чарта zif-proxy

Установка Helm-чарта zif-proxy представляет собой альтернативное решение для фронтенд-приложений при размещении ZQL в пространстве имен, отличном от платформенного.

После развертывания Helm-чарта ZQL в отдельном пространстве имен необходимо установить Helm-чарт zif-proxy.

8.1. Подготовка к установке Helm-чарта zif-proxy

8.1.1. Настройка файла values.yaml для zif-proxy для Kubernetes

1. В разделе **common.registry** указать реестр с образом, описанном в разделах **zif_proxy.image** и **zif_proxy.imageTag**.
2. В разделе **common.host** указать доменное имя Платформы.
3. В разделе **zif_proxy.resources** установить соответствующее значение лимита CPU, исходя из предполагаемой нагрузки на приложения ZQL. Рекомендуемое значение - cpu: 1, чтобы избежать ограничения производительности по CPU.
4. В разделе **ingress.tls** при использовании самоподписанных сертификатов включить флаг enabled: true, а также указать название объекта Secret в Kubernetes с цепочкой сертификатов и ключом в разделе secretName: <имя объекта Secret в Kubernetes>. В случае, если такого объекта в Kubernetes нет, то его необходимо создать вручную.
5. В разделе **zif_proxy.zui_app_list** подготовить список для настройки переадресации:

```
  zui_app_list:
       - name: <zui-app-name>.<app-namespace>
         path: /<hostAppUrl>
```

Где:

- **<zui-app-name>** - имя сервиса фронтенд-приложения в пространстве имен, в котором установлены приложения ZQL.
- **<app-namespace>** - имя пространства имен, в котором установлены приложения ZQL.
- **<hostAppUrl>** - параметр path из объекта Ingress в Kubernetes для сервиса <zui-app-name>.

Для получения списка переадресации можно воспользоваться следующим bash-скриптом.

Внимание:

- данный скрипт актуален только для Kubernetes;
- при использовании параметра **limsTenantName** в **global_values.yaml** скрипту требуется передать его значение, как 3-й аргумент.

```
       #!/bin/bash

       LIMS_KUBECONF=$1
       LIMS_NS="$2"
       LIMS_TENANT="$3"

       if [[ "${LIMS_KUBECONF}x" == "x" ]] || [[ "${LIMS_NS}x" == "x" ]] || [[
"$@" =~ "*help*" ]]; then
         echo -e "This script helps to get 'zui_app_list' (basically, a list of
service names and their paths from ingresses) that should be set into values
of Helm chart zif-proxy."
```

```
        echo -e "Execute script example: \n\tbash $0 <path/to/lims/kubeconfig
for Kubernetes> <lims-ns-name> <value of limsTenantName parameter if it's set
in global_values.yaml>"
        exit 1
      fi

      echo "  zui_app_list:"

      kubectl --kubeconfig ${LIMS_KUBECONF} -n $LIMS_NS -ojson get ingress | \
      jq -r --arg n "$LIMS_NS" --arg t "-$LIMS_TENANT" --args '
        .items[]
        | select(.metadata.name | startswith("zui-app"))
        | .spec.rules[] | .http.paths[]
        | "    - name:     " + .backend.service.name + "." + $n + "\n" +
            if ($t!="-") then
          "      path:     " + (.path | sub("\\(.*";$t)) + "\n" +
          "      sub_from: " + (.path | sub("\\(.*";"")) + "\n" +
          "      sub_to:   " + (.path | sub("\\(.*";$t)) + "\n"
            else
          "      path:     " + (.path | sub("\\(.*";"")) + "\n"
            end
        ' | \
      grep -v "^$"
```

8.1.2. Настройка файла openshift.values.yaml для zif-proxy для OKD\OpenShift

1. В разделе **common.registry** указать реестр с образом, описанном в разделах **zif_proxy.image** и **zif_proxy.imageTag**.
2. В разделе **common.host** указать доменное имя Платформы.
3. В разделе **zif_proxy.resources** установить соответствующее значение лимита CPU, исходя из предполагаемой нагрузки на приложения ZQL. Рекомендуемое значение: cpu: 1, чтобы избежать ограничения производительности по CPU.
4. В разделе **ingress** установить флаг **enabled: false.**
5. В разделе **zif_proxy.zui_app_list** подготовить список для настройки переадресации:

```
  zui_app_list:
       - name: <zui-app-name>.<app-namespace>
         path: /<hostAppUrl>
```

Где:

- <zui-app-name> - и имя сервиса фронтенд-приложения в пространстве имен, в котором установлены приложения ZQL.
- <app-namespace> - имя пространства имен, в котором установлены приложения ZQL.
- <hostAppUrl> - параметр path из объекта Ingress в k8s для сервиса <zui-app-name>
- параметр path из объекта Ingress в Kubernetes для сервиса <zui-app-name>.

Для получения списка переадресации можно воспользоваться следующим bash-скриптом:

```
      #!/bin/bash

      LIMS_KUBECONF=$1
      LIMS_NS="$2"
      LIMS_TENANT="$3"
```

```
      if [[ "${LIMS_KUBECONF}x" == "x" ]] || [[ "${LIMS_NS}x" == "x" ]] || [[
"$@" =~ "*help*" ]]; then
        echo -e "This script helps to get 'zui_app_list' (basically, a list of
service names and their paths from ingresses) that should be set into values
of Helm chart zif-proxy."
        echo -e "Execute script example: \n\tbash $0 <path/to/lims/kubeconfig
for OKD\OpenShift> <lims-ns-name> <value of limsTenantName parameter if it's
set in global_values.yaml>"
        exit 1
      fi

      echo "  zui_app_list:"

      kubectl --kubeconfig ${LIMS_KUBECONF} -n $LIMS_NS -ojson get routes | \
      jq -r --arg n "$LIMS_NS" --arg t "-$LIMS_TENANT" --args '
        .items[]
        | select(.metadata.name | startswith("zui-app"))
        | "    - name:     " + .metadata.name + "." + $n + "\n" +
            if ($t!="-") then
          "      path:     " + (.spec.path | sub("/?$";$t)) + "\n" +
          "      sub_from: " + (.spec.path | sub("/$";"")) + "\n" +
          "      sub_to:   " + (.spec.path | sub("/?$";$t)) + "\n"
            else
          "      path:     " + (.spec.path | sub("/?$";"")) + "\n"
            end
        ' | \
      grep -v "^$"
```

Внимание! Данный скрипт актуален только для OKD/OpenShift. При использовании параметра **limsTenantName** в **global_values.yaml** скрипту требуется передать его значение как 3-й аргумент.

8.2. Установка Helm-чарта zif-proxy

Выполняется стандартная процедура установки, аналогичная установке любого Helm-чарта. Для этого указывается файл **values** с необходимыми параметрами и выбирается пространство имен (namespace) для платформы. Пример команды:

```
  helm upgrade --install zif-proxy <zif-proxy-helm-dir> -f ./<values-file> -n
<platform-ns> --debug
```

Где:

- **<zif-proxy-helm-dir>** - каталог с Helm-чартом zif-proxy;
- **<platform-ns>** - имя пространства имен, в котором установлены приложения ZQL;
- **<values-file>** - путь к файлу values, сконфигурированному соответственно одному из вышеуказанных разделов.

9. Работа с EntitiesConfiguration

Конфигурационные карты (ConfigMaps) с предоставляемыми значениями для файлов EntitiesConfiguration могут использоваться по умолчанию на проекте или настраиваться индивидуально для каждого проекта. Если используются значения по умолчанию из EntitiesConfiguration, дополнительные шаги не требуются. Для использования собственной конфигурации EntitiesConfiguration необходимо выполнить следующее:

1. Создать файл **entities.yaml** (можно использовать собственное именование, но необходимо учитывать его при развертывании).
2. Поместить в этот файл все конфигурации из файлов "/helm/values/*-data-configmap.yaml". Для выполнения вышеуказанных шагов можно воспользоваться командой: "cat /helm/values/*-data-configmap.yaml > entities.yaml".
3. Настроить развертывание Helm таким образом, чтобы последним аргументом был "-f entities.yaml". При необходимости указать полный путь.
4. Переопределить нужные параметры в соответствующих EntitiesConfiguration.

После такой перестройки развертывания требуется самостоятельно поддерживать актуальность entities.yaml на основе изменения исходных файлов EntitiesConfiguration.

10. Реализация мультитенантности в виде разных пространств имен ZQL для каждого тенанта

Реализация мультитенантности включает в себя систему кастомизации настроек развертывания для отдельных экземпляров ZQL для каждого тенанта.

Шаги по настройке мультитенантности:

1. Настройка параметров инициализации для клиентов Keycloak, базы данных Postgres, а также связанных с ними ролей, контейнеров и сервис-аккаунтов MinIO (S3).
2. Адаптация имен топиков в Kafka и других параметров, которые индивидуально задаются для каждого тенанта.
3. Кастомизация идентификаторов политик безопасности.
4. Персонализация процесса размещения приложений в портале платформы и их настроек.

Управление функциональностью мультитенантности осуществляется путем использования параметра **limsTenantName** в файле **global_values.yaml**.

Внимание! Параметр принимает только строковое значение, нельзя использовать имя неймспейса или другое зарезервированное значение, а также специальные символы.

Если этот параметр установлен (не закомментирован), то происходит активация механизма настройки под разные тенанты. В противном случае выполняется установка с использованием значений по умолчанию.

Внимание! В текущей реализации известна проблема жестко закодированных путей передачи статического контента, что приводит к тому, что параметры CONTEXT_PATH не могут быть шаблонизированы. Для решения данной проблемы предлагается использовать zif-proxy с возможностью переопределения пути в ответе сервиса.

10.1. Создание клиентов в Keycloak

Включает в себя следующие параметры:

- пример значения по умолчанию - ziiot__zqa-samples;
- пример значения для мультитенанта - ziiot__tenant1__zqa-samples.

Где:

- **ziiot** - имя тенанта платформы;
- **tenant1** - имя тенанта ZQL;
- **zqa-samples** - имя микросервиса.

10.2. Создание БД Postgres и ролей

Включает в себя следующие параметры:

- пример значения по умолчанию - ziiot__zqa-samples;
- пример значения мультитенанта - ziiot__tenant1__zqa-samples.

Где:

- **ziiot** - имя тенанта платформы;
- **tenant1** - имя тенанта ZQL;

- **zqa-samples** - имя БД и роли.

10.3. Создание/именование в параметрах контейнеров MinIO и сервисных аккаунтов

Включает в себя следующие параметры:

- пример значения по умолчанию - zql-files;
- пример значения мультитенанта - tenant1--zql-files.

Где:

- zql-files - имя бакета по умолчанию;
- tenant1 - имя тенанта ZQL.

10.4. Публикация фронтенд-приложений в портале платформы

Включает в себя следующие параметры:

- пример значения по умолчанию - /hostSampleLogs, /sample-logs, samplelogs-root;
- пример значения мультитенанта - /hostSampleLogs-tenant1, /sample-logs-tenant1, samplelogs-root-tenant1.

Где:

- **/hostSampleLogs** - базовый путь (baseHref) по умолчанию для приложения;
- **/sample-logs** - маршрут по умолчанию (match route) для приложения;
- **samplelogs-root** - селектор по умолчанию для приложения;
- **tenant1** - имя тенанта ZQL.

10.5. Имена топиков Kafka

Включает в себя следующие параметры:

- пример значения по умолчанию - zqa-object-status-changed;
- пример значения мультитенанта - tenant1--zqa-object-status-changed;

Где:

- **zqa-object-status-changed** - имя топика по умолчанию;
- **tenant1** - имя тенанта ZQL.

11. Изменение текста в интерфейсе приложений

Для установки кастомных текстов в интерфейсе (UI) приложений ZQL требуется:

1. Настроить JSON конфигурацию текста, основываясь на значениях по умолчанию (можно использовать только необходимые параметры).
2. Создать дополнительный файл значений (Values) для этих параметров, например, customUItexts.yaml.
3. Заполнить файл структурой:

```
zui-app-acs:
  customUiTexts:
    ru_RU.json: |
      {
        Настройки текста
      }
    en_US.json: |
      {
        Настройки текста
      }
```

Где:

- **zui-app-acs** — это сервис фронтенд-приложения, для которого устанавливаются кастомные настройки текста.

Внимание. В файле значений (Values) необходимо присутствие обоих ключей - как для файла ru_RU.json, так и для en_US.json. Если, например, для английской локали не требуются изменения, соответствующий JSON-файл должен быть пустым. Путь к файлу значений следует передавать при установке или обновлении ZQL с использованием Helm при помощи ключа "-f", например, "-f ./customUItexts.yaml".

11.1. Изменение текста в интерфейсе приложений для сервиса zui-app-ps-planning

Для сервиса zui-app-ps-planning реализован собственный механизм из-за отличия путей переопределения файла с текстами UI. Для данного сервиса реализован ConfigMap - zui-app-ps-planning-data-configmap-env. Для переопределения его параметров требуется:

1. Создать дополнительный файл значений (Values) для этих параметров, например, customPlanningUItexts.yaml.
2. Заполнить файл структурой:

```
zui-app-ps-planning-data-configmap:
  fullnameOverride: zui-app-ps-planning-data-configmap
  env:
    ru_RU.json: |
      {
        настройки текста
      }
```

Путь к файлу значений следует передавать при установке или обновлении ZQL с использованием Helm при помощи ключа "-f", например, "-f ./customPlanningUItexts.yaml".

12. Обновление установленного продукта

12.1. Предусловия

Процедура обновления продукта ZQL включает в себя обновление уже установленного продукта с использованием Helm-чарта.

Необходимо учесть следующую информацию перед процессом обновления:

1. **Имя Helm-релиза ZQL**: Это имя требуется при выполнении команды обновления Helm. Уточните соответствующее имя Helm-релиза ZQL перед началом процесса обновления.
2. **Пространство имен, в котором установлен ZQL**: Установка ZQL произведена в определенном пространстве имен Kubernetes. Удостоверьтесь, что вы знаете и указываете соответствующее пространство имен при выполнении операций обновления.

12.2. Шаги по обновлению

Для выполнения обновления с использованием Helm-чарта из Git-репозитория (каталога с чартом на файловой системе) необходимо:

1. Обновить файл global_values.yaml в репозитории конфигурации Helm-чарта стенда, используя в качестве основы файл шаблона global_values.yaml из текущего репозитория.
    a. Включить инициализацию Keycloak, PostgreSQL и MinIO, если они были предварительно отключены, подробнее см. пункт 6.2.1 шаг 14.
    b. Включить инициализацию настроек портала.

    Внимание. Это действие может представлять опасность для стенда, так как может привести к изменению текущих настроек приложений. Если необходимо избежать повторной установки настроек приложений, требуется выполнить ручную публикацию фронтенд-приложений, особенно в случае, если они были добавлены в продукт в качестве новых микросервисов.

    c. Создать на основе файла **global_values.yaml** файл для установки, используя предоставленный шаблон и заменив в нем соответствующие значения.
    d. Разместить сформированный файл в репозитории или каталоге, из которого будет осуществляться процесс обновления.
2. Взять шаблон настроек версий для нужного релиза из файла versions_X.Y.Z.yaml, расположенного в корне каталога с Helm-чартом, где X.Y.Z представляет релиз приложений ZQL.
    a. Сформировать файл версий для вашего стенда из этого шаблона. Если не используется кастомный набор сервисов, скопировать содержимое шаблона, заменяя текущий файл версий для вашего стенда.
    b. Разместить обновленный файл версий в репозитории или каталоге, из которого будет производиться обновление.
3. Взять шаблон секретов **global_secrets.yaml** из корневого каталога чарта и внести изменения в существующий файл значений (values-файл) с секретами. Данное действие можно выполнить двумя способами:
    a. Проанализировать новый шаблон секретов и сравнить его с текущим состоянием, добавив недостающие секреты для новых микросервисов.
    b. Пересоздать файл секретов, используя, например, скрипт pw_gen.py или другие средства генерации значений для YAML-файла.

После формирования файла секретов следует разместить его в репозитории или каталоге, из которого будет производиться обновление.

4. Выполнить слияние всех файлов с параметрами микросервисов. Пример команды в среде bash:

```
cat ./values/*.yaml > ./service_values.yaml
```

5. Разместить файл **service_values.yaml** в репозитории или каталоге, из которого будет производиться обновление.
6. При необходимости выполнить кастомизацию ресурсов микросервисов в файле **resources.yaml**.
7. Запустить Helm с параметрами:

```
helm upgrade --install <lims> <helm> -f ./global_values.yaml -f ./service_values.yaml-f ./<secrets_values_file> -f ./versions.yaml -f ./resources.yaml -n lims_namespace --timeout 40m --debug
```

Где:

- **<lims>** - необходимое имя релиза Helm;
- **<helm>** - каталог с чартом (в зависимости от требований может именоваться иначе);
- **./global_values.yaml** - путь к файлу с глобальными параметрами;
- **./service_values.yaml** - путь к файлу из пункта 3 с параметрами микросервисов;
- **<secrets_values_file>** - путь к файлу с секретами, созданному согласно вышеуказанным разделам;
- **./versions.yaml** - путь к файлу с версиями микросервисов;
- **./resources.yaml** - путь к файлу с лимитами и реквестами микросервисов (опционально);
- **lims_namespace** - пространство имен установки ZQL.

8. Дождаться завершения выполнения процессов инициализации, которые запускаются в виде объектов Jobs в Kubernetes в ходе установки Helm-чарта.
9. Дождаться запуска и стабилизации всех микросервисов.

13. Автотесты

С версии ZQL 4.6.2-mod-a автотесты исключены из развертывания в составе ZQL.