Руководство по установке

Продукт «Platform V DataSpace»

Код продукта: APT

Руководство по установке компонента DataSpace Core (DSPC)

О документе

Настоящий документ содержит описание процесса установки и настройки компонента DataSpace Core продукта Platform V DataSpace (далее — DataSpace или компонент).

DataSpace Core предоставляет возможность описывать модель данных предметной области и формировать набор сервисов для работы с этими данными: создание, изменение, удаление, объединение в транзакционные группы, гибкие возможности поиска и получения данных.

DataSpace не имеет собственной модели данных предметной области, а ожидает ее от потребителя. Именно сочетание DataSpace и модели потребителя образует полноценный сервис платформы.

Основные понятия

В таблице приведены основные аббревиатуры и сокращения:

Аббревиатура, сокращение	**Расшифровка**
БД	База данных
СУБД	Система управления базами данных
КТС	Комплекс технических средств
ОС	Операционная система
ТУЗ	Технологическая учетная запись
CD	Continuous Delivery. Непрерывная поставка
SDK	Software development kit. Набор средств разработки
CI	Continuous Integration. Непрерывная интеграция
CLI	Command Line Interface. Интерфейс командной строки
CPU	Central processing unit. Центральное обрабатывающее устройство
DDL	Data Definition Language — команды языка SQL для создания и изменения объектов БД
LDM	Logical Data Model. Логическая модель данных
PG	СУБД PostgreSQL
SQL	Structured Query Language
XML	eXtensible Markup Language. Расширяемый язык разметки
ПЖ	Прикладной журнал

В таблице ниже приведены основные термины и определения:

Термин	**Определение**

DevOps	Development и Operations. Методология активного взаимодействия специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимная интеграция их рабочих процессов друг в друга
Платформа	Набор продуктов Platform V, правообладателем которых является АО “СберТех”. Перечень таких продуктов обозначен в документации на конкретный продукт
Прикладной журнал	Компонент продукта Platform V Data Tools
Компонент	Компонент DataSpace Core продукта Platform V DataSpace
Helm	Средство упаковки с открытым исходным кодом, которое помогает установить приложения Kubernetes и управлять их жизненным циклом
StandIn	Режим дублирования ключевых систем (англ. StandIn — дублирующий)
PostgreSQL	СУБД PostgreSQL (рекомендован Platform V Pangolin SE)

Системные требования

Системные требования, приведенные в данном разделе, являются минимальными.

Аппаратные требования

Серверы приложений, использующих DataSpace, должны отвечать следующим минимальным требованиям:

- Процессор: 2 x CPU.
- Оперативная память: 2 ГБ или больше.

В дополнение к этому серверы приложений, использующих DataSpace, должны отвечать требованиям к остальным компонентам реализуемых приложений.

Минимальные требования к КТС для БД:

- CPU — 4шт;
- RAM — 8GB;
- HDD — 20GB;
- IOPS > 50.

Типовые требования к dev КТС для БД PostgreSQL:

- CPU — 8шт;
- RAM — 64GB;
- HDD — 150GB;
- IOPS — 2000.

При использовании опции функционального StandIn необходимо иметь две БД на разных КТС.

Программные требования

На серверы приложений, использующих DataSpace, должна быть установлена технологическая платформа Kubernetesили Red Hat OpenShift 4+ или аналогичная среда контейнеризации приложений.

Требования к ядру Linux:

- Версия ядра — 3.10.x (протестировано, на других версиях ядра стабильная работа не гарантируется).
- Поддержка только cgroupv1 (cgroupv2 не поддерживается).

Требования к инфраструктуре и конфигурациям

Имеются следующие требования:

- Для создания проекта с помощью архетипа-необходим доступ к репозиторию Maven Central или его зеркалу.
- Наличие основной БД (PostgreSQL, H2 или Oracle). В зависимости от используемых возможностейDataSpace может потребоваться наличие StandIn БД.
- Создан и настроен namespace Kubernetes (см. раздел “Чек-лист по настройке namespace Kubernetes”).
- Наличие Kubectl на компьютере, с которого производится установка.
- Наличие пакетного менеджера Helm на ПК, с которого производится установка.
- При построении кластера серверов приложений сервиса DataSpace в случае использования алгоритма SNOWFLAKE для обеспечения уникальности генерации прикладных идентификаторов необходимо соблюсти уникальность IPv4-адресов узлов кластера. Детали см. в разделе “Генерация идентификатора алгоритмом SNOWFLAKE” в документе «Руководство по эксплуатации» в разделе «Руководство по ведению модели».

Требования к ПО:

Внешний компонент (по отношению к DataSpace Core)	**Версия**	**Цель применения**	**Применение**
Операционная система	Linux (рекомендован ОС Альт 8 СП)	Возможность функционирования продукта	Обязательно
JVM	Open JDK 11	Возможность функционирования продукта	Обязательно
Основная БД (PostgreSQL, H2 или Oracle)	Platform V Pangolin 4.6.0+ (рекомендован);	Основная БД для хранения данных компонента DataSpace Core	Обязательно

	PostgreSQL 12.x; Oracle 12.2+		
StandIn БД (PostgreSQL, H2 или Oracle)	Platform V Pangolin 4.6.0+ (рекомендован); PostgreSQL 12.x; Oracle 12.2+	БД для организации функционального StandIn для хранения данных компонента DataSpace Core	Опционально
Среда контейнеризации	Kubernetes 1.23.x(рекомендован); Red Hat OpenShift 4+	Область среды контейнеризации, выделенная под компоненты DataSpace Core и/или приложения потребителя	Обязательно
Kubectl	Версия, соответствующая версии Kubernetes	Интерфейс командной строки для управления Kubernetes, являющийся его составной частью, устанавливаемый на рабочей станции	Обязательно
Helm	3.4.x	Пакетный менеджер для Kubernetes, используется для установки компонента DataSpace Core в Kubernetes	Обязательно
Компонент Прикладной журнал (APLJ) продукта Platform V Data Tools	03.035+	Организация функционального StandIn	Опционально
Компонент Сервис межкластерной индексации продукта (CCIX) Platform V Application Sharding	D-04.002.12+	Публикация информации о размещении данных в шардах для поддержки прикладного шардирования	Опционально
Prometheus	2.22.2+	Сбор и хранение метрик мониторинга работы приложения	Опционально
Influx	1.8+	Хранение метрик мониторинга работы приложения	Опционально
Grafana	7.3+	Мониторинг работы приложения	Опционально

Fluentbit	1.4.5	Подготовка и публикация журналов приложения	Опционально
Istio (рекомендован Platform V Synapse Service Mesh)	1.12+	Маршрутизация и защита межсервисных взаимодействий	Опционально

Чек-лист по настройке namespace Kubernetes

Пространство Kubernetes будет считаться готовым к развертыванию сервисов DataSpace после выполнения всех следующих требований:

- В namespace Kubernetes создан secret типа Dockersecret с логином и паролем ТУЗ.
- Созданный secret прописан в service accounts: `default` и `dataspace-core`.
- Произведена настройка Ingress Gateway (опционально).
- Произведена настройка Egress Gateway (опционально).

Подготовка базы данных

Для работы DataSpace требуется развернуть БД. Поддерживаются СУБД PostgreSQL 12.x и выше, а также Oracle 12.2 и выше (далее Oracle). Для dev-тестов может использоваться h2 версии не ниже 1.4.200.

Рекомендуется использовать продукт Platform V Pangolin SE.

> **Примечание**
>
> Для именования объектов БД (учетных записей, ролей, схем, табличных пространств и т.д.) следует руководствоваться требованиями используемой СУБД для имен без использования кавычек. Обычно в допустимый набор символов входят латинские буквы (a-z, A-Z), цифры (0-9) и символ подчеркивания (_), имена должны начинаться с буквы или символа подчеркивания.

Создание учетных записей

Для работы модулей DataSpace и для работы скриптов развертывания в БД необходимо создать технологические учетные записи (ТУЗ):

Имя ТУЗ PG	Имя ТУЗ Oracle	Назначение	Условия использования
as_admin	DSPC	Владелец схемы БД. Для работы скриптов развертывания DataSpace — создание объектов схемы БД средствами Liquibase	**ТУЗ обязательна**. При использовании Platform V Pangolin в роль as_admin включается заранее созданная доменная ТУЗ владельца схемы БД. Доменная ТУЗ с ролью as_admin, as_admin должна являться владельцем объектов БД, для этого перед выполнением DDL-команд

			должна выполняться команда `set role as_admin`
dsc_appl	DSC_APPL	Подключение к БД модуля dataspace-core, чтение и запись	**ТУЗ обязательна**. Исключение — локальная БД разработчика для целей тестирования DataSpace
dsa_appl	DSA_APPL	Подключение к БД модуля dataspace-applier, чтение и запись	ТУЗ опциональна — определяется потребителем. Нужна только при использовании функционального StandIn для возможности раздельного мониторинга основной и вспомогательной нагрузки и управления ресурсами на уровне БД
dss_appl	DSS_APPL	Подключение к БД модуля dataspace-search, чтение	ТУЗ опциональна — определяется потребителем. Нужна только при использовании отдельной конфигурации DataSpace в режиме search для возможности раздельного мониторинга основной и поисковой нагрузки и управления ресурсами на уровне БД

Создание схемы

Необходимо создать:

- Кластер PostgreSQL.
- Роли as_admin, as_TUZ (valid until указать в соответствии с требованиями). Роль as_admin будет являться владельцем схемы, как и в случае использования БД Platform V Pangolin SE.

```
-- Скрипт выполняет DB admin с привилегией на создание ролей и пользователей
CREATE USER as_admin WITH PASSWORD <пароль> LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER NOINHERIT NOREPLICATION NOBYPASSRLS VALID UNTIL '2099.12.31';
CREATE USER "as_TUZ" WITH PASSWORD <пароль> LOGIN NOCREATEROLE NOCREATEDB NOSUPERUSER NOINHERIT NOREPLICATION NOBYPASSRLS VALID UNTIL '2099.12.31';
```

- Каталоги Linux для табличных пространств. Пользователь Linux Postgres должен иметь для них все права (mkdir /pgdata/11/ts/dspc_t; chown postgres /pgdata/11/ts/dspc_t; chmod u+rwx /pgdata/11/ts/dspc_t).

- Табличные пространства необходимого размера, БД и схему. Необходимо выдать права владельца.

```
-- Названия объектов БД и пути даны в качестве примера (обычно они подходят для типовой установки)
-- Их необходимо проверить и заменить на актуальные
-- 1) Создание табличного пространства. Выполняет DB admin.
CREATE TABLESPACE dspc_t OWNER as_admin LOCATION '/pgdata/11/ts/dspc_t';
GRANT CREATE ON TABLESPACE dspc_t TO as_admin;
```

```
-- 1.1) Следующие 2 команды в случае, если для индексов используется отдельн
ое табличное пространство.
CREATE TABLESPACE dspc_i OWNER as_admin LOCATION '/pgdata/11/ts/dspc_i';
GRANT CREATE ON TABLESPACE dspc_i TO as_admin;
-- 2) Создаем БД. Выполняет DB admin
CREATE DATABASE dspc_db OWNER as_admin ENCODING 'UTF-8' LC_COLLATE 'en_US.UT
F-8' LC_CTYPE 'en_US.UTF-8' TABLESPACE dspc_t;
GRANT ALL PRIVILEGES ON DATABASE dspc_db TO as_admin;
-- 3) Создаем схему. Выполняет DB admin или as_admin
CREATE SCHEMA dspc AUTHORIZATION as_admin;
REVOKE ALL ON SCHEMA public FROM public;
GRANT ALL PRIVILEGES ON SCHEMA dspc TO as_admin;
ALTER DEFAULT PRIVILEGES IN SCHEMA dspc GRANT ALL PRIVILEGES ON TABLES TO as
_admin;
GRANT USAGE ON SCHEMA dspc TO "as_TUZ";
GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TABLES IN SCHEMA dspc TO "as_TUZ
";
ALTER DEFAULT PRIVILEGES IN SCHEMA dspc GRANT SELECT, INSERT, UPDATE, DELETE
ON TABLES TO "as_TUZ";
ALTER ROLE as_admin SET search_path to 'dspc';
-- 4) Назначаем as_admin владельцем схемы (опционально, если схему создавала
УЗ с ролью as_admin)
ALTER SCHEMA dspc OWNER TO as_admin;
```

- ТУЗ владельца схемы — dspc. ТУЗ приложения — dsc_appl, dsa_appl, dss_appl. Выдать ТУЗ права DML на таблицы схемы DML.

```
-- Выполняет DB admin с привилегиями на создание ролей
-- ТУЗ для Liquibase
CREATE USER dspc WITH ENCRYPTED PASSWORD <пароль> NOINHERIT;
ALTER ROLE dspc SET search_path to 'dspc';
GRANT as_admin TO dspc;
ALTER ROLE dspc SET ROLE as_admin; -- as_admin является владельцем схемы
-- ТУЗ для модуля dataspace-core
CREATE USER dsc_appl WITH ENCRYPTED PASSWORD <пароль> INHERIT;
GRANT "as_TUZ" TO dsc_appl;
ALTER ROLE dsc_appl SET search_path to 'dspc';
-- ТУЗ для модуля dataspace-applier  – если необходимо
CREATE USER dsa_appl WITH ENCRYPTED PASSWORD <пароль> INHERIT;
GRANT "as_TUZ" TO dsa_appl;
ALTER ROLE dsa_appl SET search_path to 'dspc';
-- ТУЗ для модуля dataspace-search  – если необходимо
CREATE USER dss_appl WITH ENCRYPTED PASSWORD <пароль> INHERIT;
GRANT USAGE ON SCHEMA dspc TO dss_appl;
ALTER ROLE dss_appl SET search_path to 'dspc';
GRANT SELECT ON ALL TABLES IN SCHEMA dspc TO dss_appl;
ALTER DEFAULT PRIVILEGES IN SCHEMA dspc GRANT SELECT ON TABLES TO dss_appl;
```

Создание схемы с помощью Liquibase

Параметры для запуска Liquibase:

Параметр	Пример	Комментарий
tableSpaceT	dspc_t	
tableSpaceI	dspc_i (или dspc_t, если создается только одно табличное пространство)	
tableSpaceL	dspc_l (только для Oracle, если отдельное табличное пространство для LOB-объектов не используется — указать dspc_t)	
defaultSchemaName	dspc	
index_parallel_count	Степень параллелизма при создании индексов (количество параллельных процессов Oracle или максимальное число параллельных процессов PostgreSQL). Для Oracle значение по умолчанию — 1, для PostgreSQL — в соответствии с настройками сервера БД	
dbUser	dspc (для Oracle), ТУЗ с ролью as_admin (для PostgreSQL)	ТУЗ владельца схемы БД для выполнения Liquibase
dbPassword	Пароль dbUser	
dbCredId	Соответственно credentials для dbUser (задается dbCredId или dbUser/dbPassword)	ТУЗ владельца схемы БД для выполнения Liquibase
rollType	roll	
urlToChangelogZip	Ссылка на дистрибутив, содержащий chanelog	
dbDriver	org.postgreSQL.Driver	
dbUrl	jdbc:postgreSQL:	
liquibase.oracle.online	ONLINE	для использования опции ONLINE при создании индексов при использовании СУБД Oracle установить параметр в ONLINE
liquibase.pg.online	CONCURRENTLY	для использования опции CONCURRENTLY при создании индексов при использовании СУБД

		PostgreSQL установить параметр в CONCURRENTLY

Параметры для передачи в secret

В secrets MainDbsecret и StandInDbsecret необходимо прописать реквизиты ТУЗ приложений:

- Для dataspace-core необходимо прописать следующие параметры:

```
spring.datasource.username=dsc_appl
spring.datasource.password=<пароль dsc_appl>
```

- Для dataspace-applier (gigabas) необходимо прописать следующие параметры:

```
spring.datasource.username=dsa_appl
spring.datasource.password=<пароль dsa_appl>
```

- Для dataspace-search необходимо прописать следующие параметры:

```
spring.datasource.username=dss_appl
spring.datasource.password=<пароль dss_appl>
```

Настройка соединения с БД в сервисах DataSpace

В данном разделе описаны шаги настройки сервисов DataSpace и пространства Kubernetes (или OpenShift) для подключения БД (PostgreSQL или Oracle).

Шаги настройки на Kubernetes (или OpenShift)

Необходимо выполнить следующие действия:

1. Получить и настроить базу(ы) данных (см. раздел "Подготовка базы данных").
2. Создать secret(s) с параметрами подключения к базе(ам) данных в Kubernetes (или OpenShift). Смотрите раздел "Создание secret в Kubernetes (или OpenShift)".
3. Обеспечить доступ из пространства Kubernetes (или OpenShift) к базе(ам) данных. Смотрите раздел "Настройка пространства Kubernetes (или OpenShift)".
4. Передать в параметры названия созданных secrets в сервисы DataSpace при развертывании в Kubernetes (или OpenShift). Полный список параметров можно найти в манифестах OpenShift в дистрибутиве. Названия параметров:
   - MainDataBasesecretId;
   - StandInDataBasesecretId — при наличии StandIn базы данных.

Конфигурирование сервисов DataSpace

В сервисах DataSpace конфигурация настроек подключения к БД происходит средствами spring properties.

Обратите внимание, что параметры для Main и StandIn баз данных отличаются (spring <-> StandIn).

Необходимо указать следующие параметры:

- Основная (Main) база данных:

```
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.url=jdbc:postgreSQL://dataspace-core-deposit-16923100-pgdb:543
2/dataspace
dataspace.datasource.primary.dbschema=schema_name
spring.datasource.driver-class-name=org.postgreSQL.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

- StandIn база данных:

```
standin.datasource.username=dataspace
standin.datasource.password=dataspace
standin.datasource.url=jdbc:postgreSQL://dataspace-core-deals-17731375-StandIn-p
gdb:5432/dataspace
dataspace.datasource.standin.dbschema=schema_name_si
standin.datasource.driver-class-name=org.postgreSQL.Driver
standin.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Для основной и резервной (StandIn) базы данных необходимо указывать диалект SQL:

- Пример параметров в случае использования PostgreSQL:

```
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

standin.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
standin.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

- Пример параметров в случае использования Oracle:

```
spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
spring.jpa.hibernate.dialect=org.hibernate.dialect.Oracle12cDialect

standin.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
standin.jpa.hibernate.dialect=org.hibernate.dialect.Oracle12cDialect
```

Значения параметров в зависимости от БД

Параметры:

- PostgreSQL:

```
#Main db
spring.datasource.driver-class-name=org.postgreSQL.Driver
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialec
#standin db
standin.datasource.driver-class-name=org.postgreSQL.Driver
standin.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

- Oracle:

```
#Main db
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
#standin db
```

```
standin.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
standin.jpa.hibernate.dialect=org.hibernate.dialect.Oracle12cDialect
```

Создание secret в Kubernetes (или OpenShift)

Для конфигурирования на стендах Kubernetes (или OpenShift) используются объекты secret, содержащие stringData с параметрами подключения. Создать secret можно через cli-приложение (OpenShift cli или kube_ctl).

> **Внимание!**
>
> Secrets для Main и StandIn базы данных отличаются по своему содержимому.

Пример secret для основной (Main) базы данных PostgreSQL

```
apiVersion: v1
kind: secret
metadata:
  name: Main-db-secret
data:
stringData:
  secret.properties: |-
    spring.datasource.username=dsc_appl
    spring.datasource.password=<пароль>
    spring.datasource.url=jdbc:postgresql://dataspace-core-deposit-16923100-pgdb
:5432/dataspace
    dataspace.datasource.primary.dbschema=dspc
    spring.datasource.driver-class-name=org.postgresql.Driver
    spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
    spring.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Пример secret для StandIn базы данных PostgreSQL

```
apiVersion: v1
kind: secret
metadata:
  name: StandIn-db-secret
data:
stringData:
  secret.properties: |-
    standin.datasource.username=dsc_appl
    standin.datasource.password=<пароль>
    standin.datasource.url=jdbc:postgresql://dataspace-core-deals-17731375-stand
in-pgdb:5432/dataspace
    dataspace.datasource.standin.dbschema=dspc
    standin.datasource.driver-class-name=org.postgresql.Driver
    standin.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
    standin.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Пример secret для основной (main) базы данных Oracle

```
apiVersion: v1
kind: Secret
metadata:
  name: main-db-secret
data:
```

```
stringData:
  secret.properties: |-
    spring.datasource.username=dsc_appl
    spring.datasource.password=dsc_appl_password
    spring.datasource.url=jdbc:oracle:thin:@<ip-адрес>:1521:pprb
    dataspace.datasource.primary.dbschema=dspc
    spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
    spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
    spring.jpa.hibernate.dialect=org.hibernate.dialect.Oracle12cDialect
```

Пример secret для StandIn базы данных Oracle

```
apiVersion: v1
kind: Secret
metadata:
  name: standin-db-secret
data:
stringData:
  secret.properties: |-
    standin.datasource.username=dsc_appl
    standin.datasource.password=dsc_appl_password
    standin.datasource.url=jdbc:oracle:thin:@<ip-адрес>:1521:pprbsi
    dataspace.datasource.standin.dbschema=dspc
    standin.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
    standin.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect
    standin.jpa.hibernate.dialect=org.hibernate.dialect.Oracle12cDialect
```

Описание параметров, используемых в secrets

Используются следующие параметры:

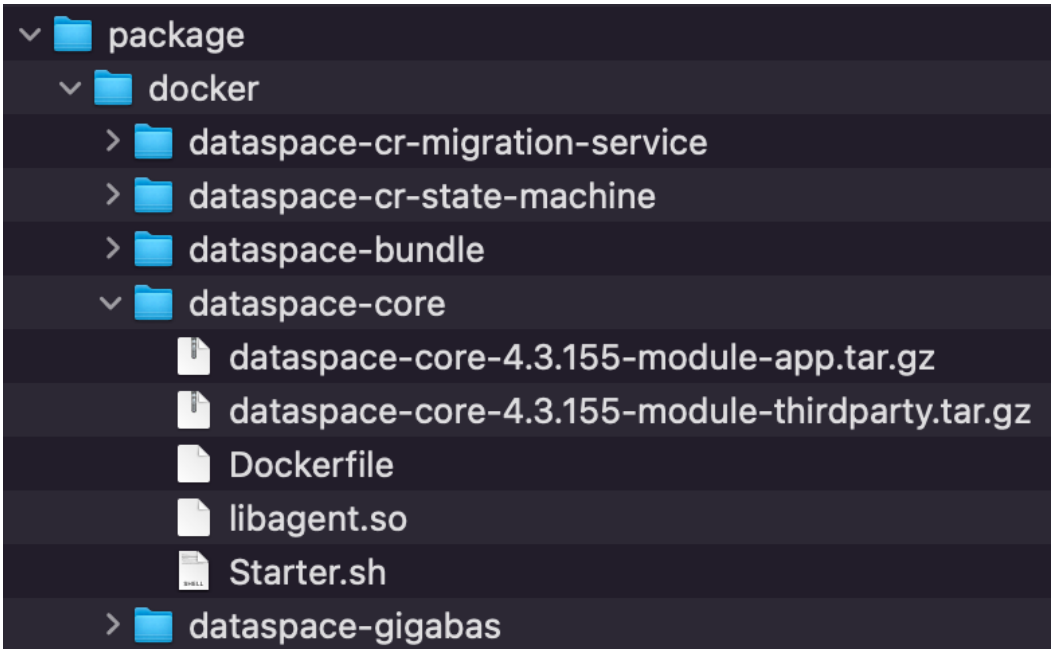
- `datasource.username` — имя пользователя;
- `datasource.password` — пароль пользователя;
- `datasource.url` — jdbc-строка соединения с базой данных;
- `dbschema` — имя схемы;
- `datasource.driver-class-name` — имя драйвера для соединения с БД;
- `database-platform` — имя диалекта БД;
- `hibernate.dialect` — имя диалекта БД, аналогично значению `database-platform`.

Установка

Ручная сборка дистрибутива и образов сервисов DataSpace

Сборка базового дистрибутива

В составе дистрибутива в папке docker лежит все необходимое для сборки базового дистрибутива компонентов DataSpace.



Основные передаваемые при сборке аргументы:

- **ARG registryUrl** – URL образа операционной системы, на основе которой будет собираться базовый дистрибутив DataSpace
- **ARG OpenJdkUrl** – URL OpenJDK 11 (Опционально, применяется в случае, если в выбранной ОС отсутствует требуемая версия OpenJDK)

Сборка производится следующим образом

```
export registryUrl=<url образа>
export OpenJdkUrl=<url OpenJDK>
docker build -t <имя:тэг> ./ --build-arg registryUrl --build-arg OpenJdkUrl
```

Ручная сборка образов сервисов DataSpace с моделью потребителя

В данной инструкции описаны шаги сборки образов сервисов DataSpace с целью их дальнейшей инсталляции в Kubernetes (или OpenShift).

Для сборки образов сервисов DataSpace необходимо выполнить следующие шаги:

1. Выбрать версию базового дистрибутива сервиса DataSpace.
2. Скачать базовый дистрибутив.
3. Разработать модель согласно инструкциям, описанным в документе «Руководство по эксплуатации» в разделе «Руководство по ведению модели».
4. Осуществить генерацию артефактов согласно инструкциям, описанным в документе «Руководство по эксплуатации» в разделе «Быстрый старт».
5. Создать следующую структуру папок и файлов:
   - *docker/*
     - *ext/* — папка с JPA-классами, сгенерированными в шаге 4.

- *lib/* — папка, в которую можно добавить дополнительные библиотеки. Эти библиотеки будут добавлены в classpath сервиса DataSpace (например, `oracleJdbcDriver`).

6. Скопировать сгенерированные на шаге 4 JPA-классы в созданную папку *./docker/ext*.

7. В случае необходимости работы с Oracle, добавить в созданную папку *./docker/lib* артефакт OJDBC. Пример maven-зависимости OJDBC:

```
<artifactItem>
    <groupId>com.oracle.jdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>18.3.0.0</version>
</artifactItem>
```

8. Создать в папке *./docker* файл *dockerfile* со следующим ниже содержимым:

   – *urlToDataspaceService* — ссылка на базовый образ интересующего сервиса DataSpace.
   – *serviceVersion* — версия данного сервиса DataSpace.

```
FROM urlToDataspaceService:serviceVersion
```

9. Должна получиться следующая структура файлов:

   – *docker* (папка)
     - *ext*
       – *model-jpa.jar*
     - *lib*
       – *ojdbc8.jar* ((опционально, в дистрибутиве не поставляется))
     - *dockerfile*

10. В корне папки *docker* выполнить команду сборки docker образа, где `yourServiceName:tag` — имя и тег результирующего образа. Обычно присваивают наименование по следующему шаблону: `{имяСервиса}-{имяМодели}` (например, `dataspace-core-deposit:1.0.0`).

```
docker build -t yourServiceName:tag ./
```

11. Произвести загрузку собранного образа в хранилище образов:

```
docker push imageTag
```

После осуществления всех шагов ручной сборки необходимо выполнить следующие шаги:

1. Произвести загрузку Liquibase-скриптов на базу данных, к которой будет подключен сервис DataSpace. Параметры, которые необходимо передать в Liquibase:
   – *–username=scott* — имя пользователя с правами на создание таблиц;
   – *–password=tiger* — пароль от пользователя;
   – *–defaultSchemaName=schema* — имя схемы в которую проливается Liquibase;

- –*index_parallel_count=1* — Степень параллелизма при создании индексов (количество параллельных процессов Oracle или максимальное число параллельных процессов PostgreSQL);
- –*liquibase.oracle.online=ONLINE* – для использования опции `ONLINE` при создании индексов при использовании СУБД Oracle$
- –*liquibase.pg.online=CONCURRENTLY* – для использования опции `CONCURRENTLY` при создании индексов при использовании СУБД PostgreSQL;
- –*changeLogFile="./db/changelog.xml*;
- –*tablespace_t=tablespaceT* — имя табличного пространства для Создания таблиц. Обязательный;
- –*tablespace_i=tablespaceI* — имя табличного пространства для индексов, может быть равен tablespace_t;
- –*tablespace_l=tablespaceL* — имя табличного пространства для LOB поля для Oracle, для PG неприменимо (может быть равен tablespace_t);
- –*role=as_admin* — имя роли для сессии под которой произойдет создание таблиц в change.log (применимо к PostgreSQL);
- –*setRole=as_admin* — дублирование параметра, для обратной совместимости в некоторых версиях DataSpace;
- –*enable_index_ddl=true*;
- –*grant_session=true*.

```
java -jar Liquibase.jar
```

2. Произвести установку сервиса при помощи HelmChart или OpenShift Template , хранящихся в базовом дистрибутиве.

   Внимание!

   Базовый дистрибутив не содержит ссылки на образ. Ссылку на образ необходимо передать при установке шаблона HelmChart или OpenShift, как параметр шаблона, или указать в секцию `image: ***` в сущности `Deployment`.

   Дополнительные сведения можно найти в разделе “Инструкция ручной установки DataSpace приложений в Kubernetes (или OpenShift)”.

Возможность поставки объединенного модуля

DataSpace включает несколько компонентов: dataspace-core, dataspace-applier, dataspace-stateMachine, dataspace-migration, dataspace-referenceUpdater. Пользователи, для которых развертывание всех модулей является затратной операцией, а также избыточной в части ознакомления с функциональностью, могут воспользоваться развертыванием объединенного модуля.

Конфигурирование требуемых к включению модулей осуществляется через шаблон:

```
Installer:
  parameters:
    dataspace-bundle-template:
      MODULE_NAME: dataspace-bundle
      core_module_enable: true
      applier_module_enable: true
```

```
state_machine_module_enable: true
migration_module_enable: true
reference_updater_module_enable: true
```

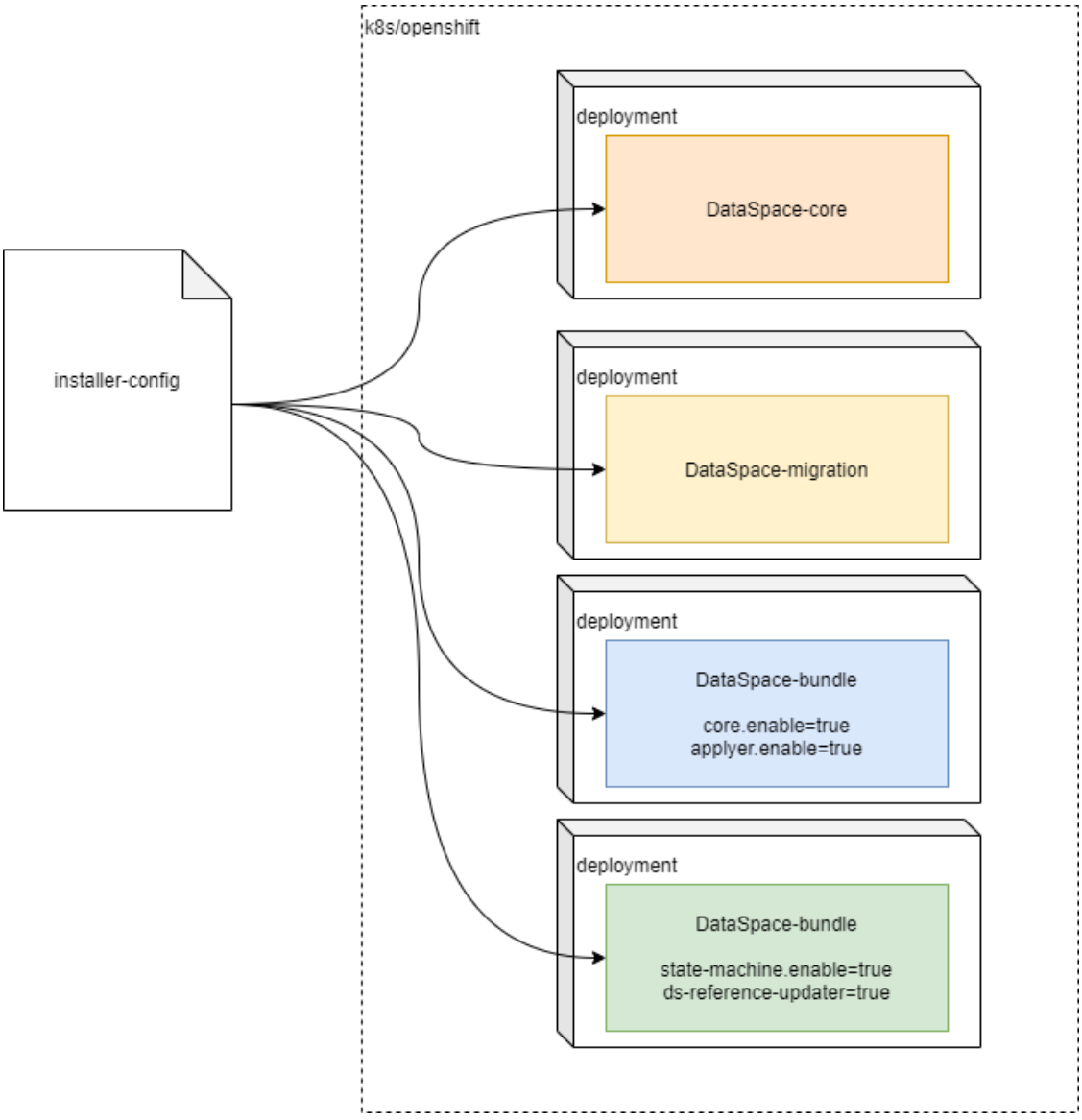
В зависимости от потребности можно включить один или несколько функциональных модулей DataSpace в единый deployment.

Примечание

Разные модули могут требовать доступность разных систем, соответственно при включении компонента с зависимостью на работу другой системы накладывается требование на весь бандл. Например, при включении applier_module_enable: true требуется настройка StandIn (Platform V Data Tools).

Для более гибкого контроля потребления ресурсов рекомендуется использование раздельных модулей.

Пример развертывания deployment:



Установка DataSpace с помощью Helm в Kubernetes (или OpenShift)

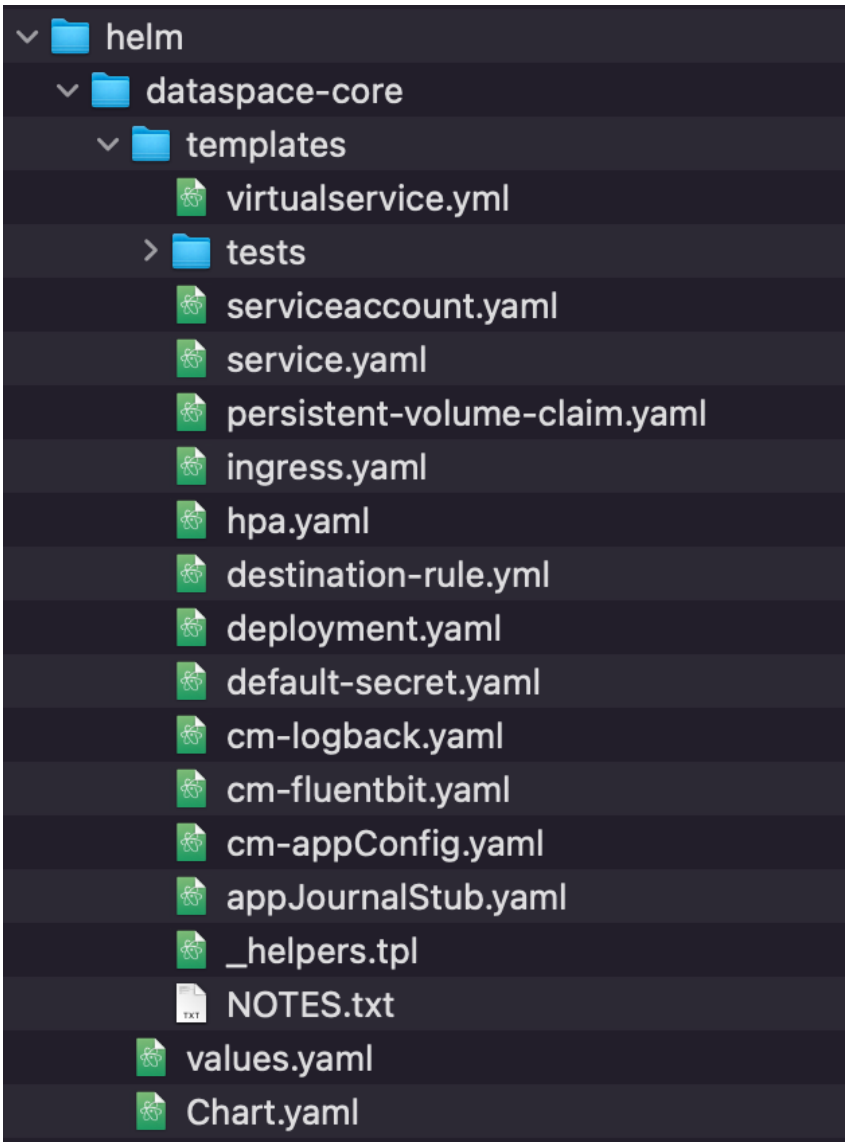
Примечание

Необходимо наличие Helm на компьютере, с которого производится установка.

Реализована возможность установки DataSpace с помощью Helm – пакетного менеджера Kubernetes. Для данного вида установки нужен специально подготовленный установочный пакет *helm chart*.

Пакет helm chart

Структура пакета *helm chart* DataSpace:



Внутри дистрибутива находится папка *helm*, содержащая:

- *Chart.yaml* – файл со служебной информацией о chart и приложении, который он разворачивает.
- *values.yaml* — содержит свойства chart с их дефолтными значениями.

- *templates/*.yaml* — файлы с шаблонами объектов Kuberenetes.
- *templates/tests/*.yaml* — файлы с шаблонами для тестов Helm.

Для базовой установки дистрибутива DataSpace необходимо заполнить следующие значения в values.yaml:

```
appConfig:
  packagesToScan: ru.sbt.platformv.dev.duser1 – GroupID Maven артефакта

fluentBit:
  enabled: true – включить/выключить использование Fluentbit
  image: '' – URL образа Fluentbit

defaultSecret:
  create: true – создавать/не создавать (false) дефолтный secret, задается в def
ault-secret.yaml

image:
  registryUrl: '' – URL Nexus
  registryProject: '' – Project ID
  name: "dataspace-core-m7107684626437177346:0.0.1" – имя образа

istio:
  enabled: true – использовать/не использовать
```

Проверка пакета *helm chart*:

```
helm install --dry-run --debug <имя> <путь к папке с chart на локальной машине>
```

Процесс установки

Для установки необходимо выполнить команд

у вида:

```
helm install <имя> <путь к папке с chart на локальной машине>
```

В случае успешной установки, в консоль выводится сообщение вида:

```
NAME: <имя деплоймента>
LAST DEPLOYED: <дата последнего успешного деплоя>
NAMESPACE: <неймспейс с установленным деплойментом>
STATUS: deployed
REVISION: 1
NOTES:
You have installed dataspace-core app with model
1.Get the application URL in namespace
http://svc-dataspace-core:8080
2. Get the application URL by running these commands:
```

Откат деплоймента при установке с помощью Helm

Если по какой-то причине установка не прошла или есть необходимость удаления существующего деплоймента, следует выполнить команду:

```
helm uninstall <имя>
```

Это делается для того, чтобы избежать ошибок следующего вида:

```
helm install dataspace-core dataspace-core-123
Error: INSTALLATION FAILED: cannot re-use a name that is still in use
```

Опциональные настройки

Опционально настраиваются следующие файлы:

- appJournalStub.yaml – пример настройки подключения к Прикладному журналу (по умолчанию ведет на “заглушку”).

```
apiVersion: v1
kind: Secret
metadata:
  name: {{ include "dataspace-core.appJournalSetting" . }}
data:
stringData:
  appJournal.properties: |-
    standin.cloud.client.stub=true
    standin.cloud.client.zone-id=test
    standin.cloud.client.kafka.bootstrapServers=localhost
    standin.cloud.client.mmtProxyUrl=localhost
  appJournalForCore.properties: |-
    dataspace.replication.enabled=false
```

- cm-appConfig.yaml – основной config map модуля DataSpace, в котором задаются override.properties, переопределяющие стандартные значения application.properties, например:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: {{ include "dataspace-core.appConfigName" . }}
data:
  override.properties: |-
    #-------------------   Static application properties start Don't change
this properties after build ds with model----------------------
    dataspace-core.model.packagesToScan={{ .Values.appConfig.packagesToScan
}}
    #-------------------   Application Mode end ----------------------
    # Активация бинов для работы с пакетом

    dataspace.packet.enable={{ .Values.appConfig.dataspace.packet.enabled }}

    # Активация бинов для работы с поиском

    dataspace.search.enable={{ .Values.appConfig.dataspace.packet.enabled }}

    dataspace.endpoint.jsonrpc.enabled=true
```

В этом же файле имеется секция standOverride.yml:

```
  standOverride.yml: |-
    {{- with .Values.appConfig.overrideProperties }}
```

В standOverride.yml дописываются параметры, определенные в overrideProperties из values.yaml. Это полезно в том случае, когда нужно переопределить стандартные значения override.properties, добавить свои параметры приложений DataSpace, а так же добавить или изменить параметры, общие для SpringBoot приложений. Кроме того в cm-appConfig.yaml прописываются JVM-аргументы, задаваемые values.yaml (представлен вариант по-умолчанию):

```
 jvmArguments: "-XX:+AlwaysPreTouch -server -XX:InitialRAMPercentage=50.0 -X
X:MaxRAMPercentage=70.0 -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:HeapDumpPat
h=./ -XX:+UnlockDiagnosticVMOptions -XX:+HeapDumpOnOutOfMemoryError -XX:+Dis
ableExplicitGC -XX:+ScavengeBeforeFullGC -XX:PrintSafepointStatisticsCount=1
-XX:+ParallelRefProcEnabled -XX:+PrintClassHistogram -XX:+PrintFlagsFinal -X
X:+LogVMOutput -Xlog:gc*=debug:file=/tmp/gc_log_memory.log:time,uptime,level
,tags -Xlog:safepoint=debug:/tmp/gc_log_memory.log:time,uptime,level,tags -D
com.sun.management.jmxremote.port=5000 -Dcom.sun.management.jmxremote.authen
ticate=false -Dcom.sun.management.jmxremote.ssl=false -Djdk.attach.allowAtta
chSelf=true --add-exports java.base/jdk.internal.perf=ALL-UNNAMED --add-expo
rts java.management/com.sun.jmx=ALL-UNNAMED"
```

- cm-fluentbit.yaml – дефолтный config map Fluentbit, включается в deployment.yaml через значения из values.yaml:

```
fluentBit:
  enabled: true – включить/выключить
  image: '' – путь к образу Fluentbit
  secret: fluentBitSecret
  config: '' – если тут пусто, создается конфигмап по умолчанию. Если тут ук
азан собственный конфигмап Fluentbit, то дефолтный не создается и использует
ся указанный в данном поле.
```

- cm-logback.yaml – дефолтный config map для логбэка, его использование задается значением *customLogback: ”* в values.yaml.

- default-secret.yaml – дефолтный secret для подключения main-db. Пример:

```
kind: Secret
apiVersion: v1
metadata:
  name: {{ include "dataspace-core.mainDataBaseSecret" . }}
stringData:
  secret.properties: |-
    dataspace.replication.enabled=false
    spring.datasource.url=jdbc:h2:mem:test
    spring.datasource.username=sa
    spring.datasource.password=
    spring.datasource.driver-class-name=org.h2.Driver
    spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.hibernate.ddl-auto=none
spring.liquibase.enabled=true
spring.liquibase.change-log=classpath:db/changelog.xml
spring.liquibase.parameters.role=sa
spring.liquibase.parameters.tablespace_t=sc_t
spring.liquibase.parameters.tablespace_i=sc_i
spring.liquibase.parameters.tablespace_l=sc_l
dataspace.warmup.enable=false
```

По умолчанию создается при развертывании с локальной базой данных H2, если в values.yaml не указан иной secret для подключения к базе данных.

```
db:
...
  main:
    secret: ''
```

Аналогичным образом можно создать свои secrets для Kubernetes (или OpenShift).

- destination-rule.yml – сетевые правила для Istio. Задействуется в том случае, если в values.yaml “включены” значения.

```
istio:
  enabled: true
```

и

```
destinationRule:
  enabled: true – по умолчанию установлено значение false
```

- hpa.yaml – правила горизонтального масштабирования, берет из values.yaml следующие параметры:

```
autoscaling:
  enabled: true – включено/выключено
  minReplicas: 1 – минимальное количество одновременно запущенных реплик
  maxReplicas: 3 – максимальне количество одновременно запущенных реплик
  targetCPUUtilizationPercentage: 80 – процент утилизации CPU
```

Необходимо для того, чтобы при увеличении нагрузки на приложение, автоматически создавались новые pods (реплики) с приложением и распределяли нагрузку между собой. Когда нагрузка уменьшается, “лишние” реплики удаляются. В базовом варианте указано минимальное значение – “1”, максимальное – “3” и утилизация по CPU. Это значит, что в случае повышения нагрузки на pod более 80% процессорного времени, создается вторая реплика приложения и нагрузка распределяется между ними. Когда нагрузка уменьшается, “лишняя” реплика удаляется.

- ingress.yaml – настройки Ingress, заполняется параметрами из values.yaml.

```
ingress:
    enabled: false   # По умолчанию отключено, для использования поменять на
true
    servicePort: ''  # Порт сервиса, с которого ожидается входящий трафик. Е
сли одного порта недостаточно, то для других портов прописываются правила ма
ршрутизации
    annotations: {}
    rules: {}        # Здесь указываются правила маршрутизации для ингресса
    hosts:           # Хосты, с которых ожидается входящий трафик
        - host: chart-example.local
          paths: []
    tls: []         (Путь к файлам сертификатов, при использовании ssl
```

- NOTES.txt – параметры сообщения, выдаваемого в консоли после успешного запуска DataSpace. Если включен и настроен Ingress, также выводит в консоль ссылку для внешнего доступа к развернутому приложению.
- service.yaml – настройка сетевого сервиса приложения.
- serviceaccount.yaml – параметры для service account Kubernetes (или OpenShift), создаваемого если это было задано:

```
serviceAccount:
  create: false
  annotations: {}
  name: ''
```

- virtualservice.yml – параметры виртуального сервиса для Istio, заполняется следующими значениями из values.yaml:

```
virtualService:
  enabled: false    # По умолчанию отключено, включить поставив в значение t
rue
  gateways: []      # Имена шлюзов и сайдкаров для маршрутизации
  hosts: []         # Список хостов, на которые направляется трафик
  http:             # Правила маршрутизации HTTP-трафика
    - name: reviews-v1-route
      route:
        - destination:
          host: reviews.prod.svc.cluster.local
          subset: v1
```

Альтернативный вариант установки DataSpace в OpenShift

Обязательные параметры для передачи в OpenShift template при установке:

- `MODULE_NAME` — имя модуля.
- `MainDataBasesecretId` — имя secret с конфигурацией подключения к Primary БД (см. раздел “Настройка соединения с БД в сервисах DataSpace”).
- `FLUENTBIT_IMAGE` — ссылка на образ FluentBit.

С дополнительными возможностями конфигурации (ресурсами, readiness/liveness probe и прочим) можно ознакомиться в OpenShift template конкретного сервиса в секции **parameters**.

> **Примечание**
>
> Blue/green развертывание — это метод развертывания, который снижает время простоя и риски за счет запуска двух идентичных производственных сред, называемых `blue` и `green`. В любой момент времени только одна из сред является активной (рабочая среда). Активная среда обслуживает весь производственный трафик.

Данное руководство содержит информацию о загрузке Liquibase-скриптов на базу данных, а также проведения blue/green развертывания средствами Istio в ручном режиме.

Необходимо выполнить следующие действия:

1. Сконфигурировать подключения к базам данных согласно разделу "Настройка соединения с БД в сервисах DataSpace".

2. Подключиться к OpenShift через OpenShift cli (oc):

   ```
   oc login
   ```

3. Скачать дистрибутив, который необходимо установить, из Nexus.
   Архив дистрибутива содержит:

   - Манифесты (template) OpenShift с сервисами Data (параметры перечислены для Liquibase cli).
   - Скрипты Liquibase для создания физики в Базе данных (папка db)

4. Выполнить скрипты Liquibase при помощи любой формы Liquibase: https://docs.Liquibase.com/tools-integrations/cli/home.html.

   Параметры, которые необходимо передать в Liquibase:

   - *–username=scott* — имя пользователя с правами на создание таблиц;
   - *–password=tiger* — пароль от пользователя;
   - *–defaultSchemaName=schema* — имя схемы в которую проливается Liquibase;
   - *–index_parallel_count=1* — Степень параллелизма при создании индексов (количество параллельных процессов Oracle или максимальное число параллельных процессов PostgreSQL);
   - *–liquibase.oracle.online=ONLINE* – для использования опции `ONLINE` при создании индексов при использовании СУБД Oracle;
   - *–liquibase.pg.online=CONCURRENTLY* – для использования опции `CONCURRENTLY` при создании индексов при использовании СУБД PostgreSQL;
   - *–changeLogFile="./db/changelog.xml*;
   - *–tablespace_t=tablespaceT* — имя табличного пространства для Создания таблиц. Обязательный;

- *–tablespace_i=tablespaceI* — имя табличного пространства для индексов, может быть равен tablespace_t;
- *–tablespace_l=tablespaceL* — имя табличного пространства для LOB поля для Oracle, для PG неприменимо (может быть равен tablespace_t);
- *–role=as_admin* — имя роли для сессии под которой произойдет создание таблиц в change.log (применимо к PostgreSQL);
- *–setRole=as_admin* — дублирование параметра, для обратной совместимости в некоторых версиях DataSpace;
- *–enable_index_ddl=true*;
- *–grant_session=true*.

```
java -jar Liquibase.jar
```

5. Установку сервисов DataSpace необходимо производить в соответствии с приоритетом. Чтобы узнать приоритеты сервисов, необходимо открыть манифесты (template.yml) в дистрибутиве и найти следующие поля:

```
apiVersion: template.OpenShift.io/v1
kind: Template
labels:
   app: ${MODULE_NAME}
   version: dataspace-release-snapshot
   type: application
   installPriority: "4"
   template: dataspace-core-template
   template_version: v1
   deleteLabel: ${MODULE_NAME}
```

   Информация о приоритете хранится в labels.installPriority.
   Порядок установки: первым устанавливается приложение с меньшим значением installPriority.

6. Необходимо узнать, какая версия установлена на стенд в текущий момент, чтобы правильно переключать нагрузку средствами istio. Текущая версия хранится в yaml deployments в labels.version.
   Выполнить команду, где MODULE_NAME: имя устанавливаемого модуля.

```
oc describe deployments --selector app=MODULE_NAME | grep version
```

```
 % oc describe deployments --selector app=dataspace-core-deals-17724333 | gr
ep version
                     template_version=v1
                     version=develop-17724333-snapshot
   Selector:               app=dataspace-core-deals-17724333,version=develop
-17724333-snapshot
   version=develop-17724333-snapshot
```

7. Создать правило маршрутизации DestinationRule, где:

   - MODULE_NAME — имя устанавливаемого модуля;

- blueVersion — версия в текущий момент установленного сервиса (если нет, то создать равным текущей устанавливаемой версии);
- greenVersion — версия устанавливаемого сервиса.

```
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: dr-${MODULE_NAME}
spec:
  host: svc-${MODULE_NAME}
  subsets:
    – name1: oldVersion
      labels:
        version: ${blueVersion}
    – name2: canaryVersion
      labels:
        version: ${greenVersion}
```

Данное правило создает два subset, которыми мы сможем в будущем воспользоваться для маршрутизации трафика.

8. Создать правило маршрутизации VirtualService, где MODULE_NAME — имя устанавливаемого модуля.

```
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
   name: vs-${MODULE_NAME}
spec:
   exportTo:
      – .
   hosts:
      – svc-${MODULE_NAME}
   http:
      – route:
           – destination1:
                host: svc-${MODULE_NAME}
                subset: oldVersion
             weight: 100
           – destination2:
                host: svc-${MODULE_NAME}
                subset: canaryVersion
             weight: 0
```

Данным правилом мы гарантируем, что весь трафик в текущий момент будет идти на blue (рабочую) версию (если она имеется).

9. Произвести установку сервиса из манифеста (template).

```
oc process -f имяФайла.yaml -p MODULE_NAME=имяМодуля -p ...(перечисляем все параметры, которые необходимо передать при установке) | oc apply -f -
```

10. Дождаться успешного развертывания сервиса.
    Deployment ready:

    ```
    oc get deployments --selector app=dataspace-core-deals
    ```

    Вывод команды:

    ```
    NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
    dataspace-core-deals-4.0.0    1/1     1            1           3h16m
    ```

    READY 1/1 означает, что Pods успешно поднялись.

11. В случае, если Pods долго не переходят в состояние READY необходимо зайти в логи Pod, deployment или event, найти причину и устранить проблемы.
    Если проблема не была устранена, произвести удаление установленных объектов deployment, configmap.

12. В случае успешного поднятия всех Pods произвести переключение трафика на green-версию. Применить DestinationRule, где

    - MODULE_NAME — имя устанавливаемого модуля;
    - greenVersion — версия устанавливаемого сервиса.

    ```
    apiVersion: networking.istio.io/v1alpha3
    kind: DestinationRule
    metadata:
      name: dr-${MODULE_NAME}
    spec:
      host: svc-${MODULE_NAME}
      subsets:
        – name: oldVersion
          labels:
            version: ${greenVersion}
        – name: canaryVersion
          labels:
            version: ${greenVersion}
    ```

    Данная операция исключит из маршрутизации старую blue-версию, сделает рабочей новую установленную версию.

13. Повторить пункты 5-11 для всех template в поставке.

Обновление

Обновление модели данных

Для обновления модели данных необходимо выполнить следующие шаги:

1. Внести обратно совместимые правки в модель данных (файл *model.xml*).
2. Пройти CI-процесс генерации новой релизной версии сервисов DataSpace с обновленной моделью данных.
3. Пройти CD-процесс с новой версией сервисов DataSpace (см. раздел “Установка DataSpace”):

   - Обновить схему базы данных скриптами Liquibase из дистрибутива сервисов DataSpace.
   - Установить новую версию сервисов в пространство Kubernetes (или OpenShift).
4. Проверить работоспособность.

Обновление версии базовых сервисов DataSpace

Для обновления версии сервисов DataSpace необходимо выполнить следующие шаги:

1. Поднять версию зависимости dataspace-bom в проекте с моделью.
2. Пройти CI-процесс генерации новой релизной версии сервисов DataSpace.
3. Пройти CD-процесс с новой версией сервисов DataSpace (см. раздел "Установка DataSpace").
   - Обновить схему базы данных скриптами Liquibase из дистрибутива сервисов DataSpace.
   - Установить новую версию сервисов в пространство Kubernetes (или OpenShift).
4. Проверить работоспособность DataSpace.

Обновление существующей установки с помощью Helm

Для обновления необходимо выполнить команду вида:

```
helm upgrade --install <имя> <путь к папке с chart на локальной машине>
```

Проверка работоспособности

Для проверки работоспособности необходимо выполнить следующие шаги:

1. Проверить выполнение пунктов из раздела "Чек-лист валидации установки".
2. Произвести вызов функционального endpoint сервиса DataSpace при помощи SDK или GraphQL. Требуемый код ответа: `200`.

Откат

В разделе описаны шаги, которые необходимо выполнить для отката изменений в сервисах DataSpace и используемых базах данных.

Откат сервисов DataSpace необходимо производить в следующей последовательности:

1. Откат сервисов DataSpace.
2. Откат изменений БД (если необходимо).

Откат сервисов DataSpace

В пространство Kubernetes (или OpenShift) необходимо установить старую версию сервисов. Для этого достаточно пройти CD-процесс со старой версией сервисов DataSpace.

Откат изменений БД

Для отката изменений БД применяется механизм Liquibase rollback. Для выполнения отката требуется передать тег, до которого необходим откат (см. раздел "Формирование тега rollback").

Формирование тега rollback

После выпуска новой релизной версии модели в changelog добавляются все изменения БД относительно предыдущего выпуска. Все изменения размещаются между тегами:

- `<modelName>-<version>-before` — тег, обозначающий начало новых изменений;
- `<modelName>-<version>-applied` — тег, обозначающий окончание изменений.

Формирование тегов позволяет осуществить быстрый откат до необходимой версии.

Ручной откат с помощью Liquibase

Для ручного отката изменений БД необходимо наличие последнего дистрибутива и утилиты Liquibase. С принципами отката с помощью Liquibase можно ознакомиться в официальной документации утилиты.

Необходимо распаковать дистрибутив, перейти в `./configs/db` и запустить утилиту Liquibase:

```
shell script Liquibase --changeLogFile=changelog.xml rollback <tag>
```

Пример тега: `deposit-4.1.0-applied`.

Чек-лист валидации установки

В данном разделе описаны методы проверки валидности установки сервисов DataSpace.

Для того, чтобы удостовериться в валидности установки, необходимо проверить следующие пункты:

1. Успешно пролиты скрипты Liquibase на базу данных.
2. Успешная установка в Kubernetes (или OpenShift).
3. Endpoint сервиса DataSpace доступен из других установленных в Kubernetes (или OpenShift) сервисов.

Успешно пролиты скрипты Liquibase на базу данных

Для валидации успешного выполнения скриптов Liquibase необходимо проверить выполнение следующего условия: при подключении к базе данных видны все таблицы для сущностей, описанных в модели (в файле *model.xml*).

Успешная установка в Kubernetes (или OpenShift)

Для валидации установки в Kubernetes (или OpenShift) необходимо проверить выполнение следующего условия: deployment перешел в статус "Ready". Увидеть его можно при помощи cli. Для этого необходимо:

- Авторизоваться в Kubernetes с использованием kubectl.
- Выполнить следующую команду, заменив переменные своими значениями:

  ```
  kubectl get deployment имяДеплойментВашегоСервиса -n имя Вашего namespace
  ```

- Проверить, что все Pods находятся в статусе “ready”. Столбец должен иметь следующий вид: `Ready 1/1`, `Ready 2/2` и т.д. (в зависимости от количества поднятых Pods).

Endpoint сервиса DataSpace доступен из других установленных в Kubernetes (или OpenShift) сервисов

Для проверки доступности сервиса необходимо выполнить следующие действия:

1. Из терминала любого Pod другого сервиса выполнить команду:

   ```
   curl -v svc-имяСервисаDataSpace:8080/actuator/info
   ```

   Например:

   ```
   curl -v svc-dataspace-core-deposit:8080/actuator/info
   ```

   При этом должен вернуться json с информацией.

   Имя сервиса можно посмотреть следующей командой:

   ```
   kubectl get service -n ИмяВашегоNamespace
   ```

2. Из терминала любого Pod другого сервиса выполнить команду:

```
curl -v -X GET svc-имяСервисаDataSpace:8080/packet
```

Должна вернуться ошибка: `HTTP ERROR 400`.

Дополнительные операции

Настройка SSL-соединения с БД в сервисах DataSpace

В данном разделе описаны шаги конфигурирования базы данных, сервисов DataSpace для активации SSL режима соединения. Для выполнения шагов инструкции понадобится ПК (КТС) с установленными утилитами openssl и keytool.

Должны быть выполнены следующие требования:

1. Сертификаты client и root ca должны подключаться через secret.
2. Необходимо указать признак работы *с tls / без tls* через jdbc-строку соединения.
3. Для Main и SI необходимо использовать одинаковые сертификаты, т.к. это один и тот же объект доступа. Для основного приложения и для DataSpace Applier (gigabas) необходимо использовать одинаковые сертификаты, т.к. это один субъект доступа в двух частях.

Настройка SSL-соединения Kubernetes-DataBase

Подключение к БД по SSL в сервисах DataSpace должно производиться с помощью KeyStore, как хранилища сертификатов.

Чтобы настроить SSL-соединение с базой из установленного в Kubernetes сервиса DataSpace необходимо выполнить следующие действия:

1. Сконфигурировать базу. Инструкции по конфигурации баз (PostgreSQL или Oracle) можно найти ниже по тексту текущего раздела.
2. Создать keystore с клиентскими сертификатами.
3. Создать kind secret с keystore(.jks) в Kubernetes.
4. Сконфигурировать kind secret с настройками подключения к БД. Выставить SSL-подключение. Обратите внимание, что для каждого типа базы данных имеется свой способ указания типа подключения.
5. Передать имя secret с keystore в параметры dbSslKeyStoresecret и secrets с данными подключения в параметры (mainDataBaseSecretId/standinDataBaseSecretId).

Инструкция для PostgreSQL

Для подключения к базе данных PostgreSQL необходимо выполнить следующие шаги:

1. Выпустить сертификаты.
2. Для версии PostgreSQL ниже 4.2.1 сконфигурировать базу данных.
3. Сконфигурировать secrets kind secret для SSL-подключения к базам данных:
   - основной БД;
   - резервной (StandIn) БД.
4. Проверить правильность настройки.

Выпуск сертификатов

Производить выпуск сертификатов необходимо согласно инструкции соответствующей СУБД.

Необходимо получить следующие файлы:

- `root.crt` (сертификат CA);
- `root.key`;
- `server.crt` (сертификат для БД);
- `server.key`, `client.crt` (сертификат для пользователя);
- `client.key`.

Конфигурирование базы данных

Производить конфигурирование базы данных необходимо согласно инструкции соответствующей СУБД.

Для **подключения к основной БД** необходимо в `Main-db-secret` добавить параметры `keyStorePassword` и `trustStorePassword`, как показано в примере ниже:

```
apiVersion: v1
kind: Secret
metadata:
  name: main-db-secret
data:
stringData:
  secret.properties: |-
    spring.datasource.username=username
    spring.datasource.password=password
    spring.datasource.url=jdbc:postgreSQL://dataspace-core-deposit-16923100-pgdb
:5432/dataspace?sslmode=verify-ca&sslfactory=org.postgreSQL.ssl.DefaultJavaSSLFa
ctory
    spring.datasource.driver-class-name=org.postgreSQL.Driver
    spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
    dataspace.db.ssl.keyStorePassword=changeit
    dataspace.db.ssl.trustStorePassword=changeit
```

Для **подключения к резервной БД** параметры `keyStorePassword` и `trustStorePassword` не нужны, как показано в примере `StandIn-db-secret' ниже:

```
apiVersion: v1
kind: Secret
metadata:
  name: standin-db-secret
data:
stringData:
  secret.properties: |-
    standin.datasource.username=username
    standin.datasource.password=password
    standin.datasource.url=jdbc:postgreSQL://dataspace-core-deals-17731375-Stand
In-pgdb:5432/dataspace?sslmode=verify-ca&sslfactory=org.postgreSQL.ssl.DefaultJa
vaSSLFactory
    standin.datasource.driver-class-name=org.postgreSQL.Driver
    standin.jpa.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
```

Проверка правильности настройки

Для проверки SSL-соединения должны быть выполнены следующие требования:

1. Модуль успешно устанавливается в Kubernetes.
2. При выполнении SQL-запроса к БД отображается свойство TLS *True* для соединения, указанного в данных подключения пользователя.

Настройка репликации в StandIn при использовании DataSpace

Возможности StandIn при использовании DataSpace

При использовании DataSpace для работы с данными консистентная репликация изменений в БД StandIn обеспечивается автоматически при выполнении следующих условий:

- агрегатоцентричная модель данных;

- наличие на стенде корректно настроенного и функционирующего компонента Прикладной журнал (ПЖ), в том числе плагинов в ПЖ (для репликации в StandIn используются плагины Kafka **EXPORT_FUNC_SI** и **EXPORT_FUNC_SI_LCK**);
- корректные настройки приложения dataspace-core подключения к ПЖ и к БД StandIn;
- разворачивание модуля применения векторов изменений в OS с корректными настройками.

Этапы настройки репликации в StandIn для новых пользователей DataSpace

Этап 1 — проверка модели данных

Необходимо убедиться, что модель данных удовлетворяет критерию агрегатоцентричности или разработать такую модель данных (сведения о реализации предметной модели можно найти в документе «Руководство по эксплуатации» в разделе “Руководство по ведению модели”).

Этап 2 — проверка наличия ПЖ

Убедиться, что на стенде имеется развернутый компонент Прикладной журнал (ПЖ), или заказать разворачивание этого сервиса. За подробностями можно обратиться к документации компонента Прикладной журнал продукта Platform V Data Tools.

Режим ожидания подтверждения коммита

> **Внимание!**
>
> Данный раздел взят по материалам документации ПЖ. Рекомендуется дополнительно согласовать все действия по данному разделу с ПЖ.

Концепция:

- В этом режиме ПЖ буферизирует вектора изменений до получения флага успешного коммита соответствующей транзакции с БД источника.
- Если вместо флага подтверждения пришел флаг отката, то транзакция не будет передана в модуль применения векторов.
- Если истекло время ожидания, и не пришло ни подтверждение, ни отмена транзакции - ПЖ отправит запрос на переподтверждение в модуль применения векторов.

Действия для потребителя

Для функционирования данного режима вводятся новые топики, по которым нужно получить права на чтение/запись:

- journal_request_topic_confirmed_{zoneId}_{datatype};
- journal_confirmation_{zoneId}_{datatype}.

Действия для сопровождения ПЖ

Сопровождение ПЖ должно включить “надежные транзакции”, для этого необходимо:

1. Выполнить sql-запрос вида:

```
UPDATE KAFKA_EXPORT_MAPPING
set WAIT_FOR_CONFIRM=1
where ZONE_ID='<укажите имя своей зоны – MY_ZONE>'
  and DATA_TYPE='DATASPACE' and PLUGIN_CODE='EXPORT_FUNC_SI';
```

2. Перезагрузить модули Writer.

Поддержка режима ожидания подтверждения коммита во всех модулях DataSpace

Рекомендуется прописать в secret ПЖ следующие параметры:

- dataspace.replication.confirmation-mode=confirmed;
- dataspace.standin.applier.lock-control-logic-mode=VERSION_WITH_DELETE.

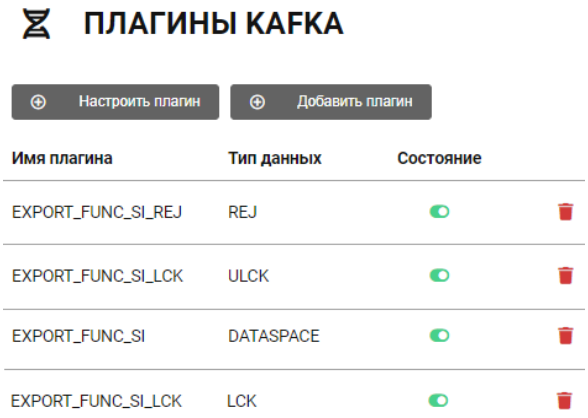
Этап 3 — создание новой зоны ПЖ

Необходимо выбрать идентификатор StandIn-зоны (зоны ПЖ) и заказать создание новой зоны ПЖ с данным идентификатором на стенде. За подробностями можно обратиться к документации компонента Прикладной журнал продукта Platform V Data Tools.

Этап 4 — настройка плагинов ПЖ

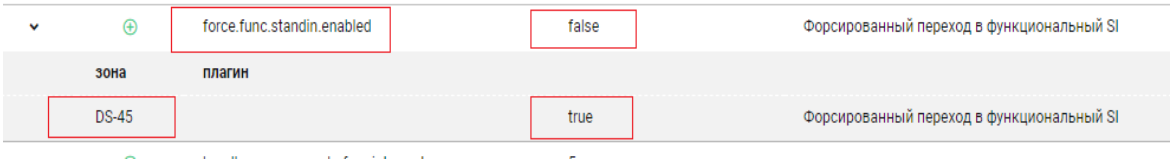
Необходимо заказать настройку плагинов ПЖ для DataSpace или убедиться, что такая настройка произведена. Настройка осуществляется согласно таблице:

Имя плагина	Тип данных	Комментарий
EXPORT_FUNC_SI	DATASPACE	Вектора изменений DataSpace, формат JSON
EXPORT_FUNC_SI_LCK	LCK	StandIn блокировки агрегатов, формат JSON
EXPORT_FUNC_SI_LCK	ULCK	StandIn разблокировки агрегатов, формат JSON
EXPORT_FUNC_SI_REJ	REJ	StandIn флаг отката, формат JSON



Дополнительно необходимо заказать настройку использования блокировок для указанной зоны: параметр *force.func.standin.enabled* должен быть установлен в “true”.

Обычно ПЖ по умолчанию устанавливает в false, и явно для тех кто запросил, устанавливает в true



Этап 5 — настройка secret ПЖ

После настройки ПЖ необходимо запросить параметры подключения к ПЖ на данном стенде и создать secret с определением ресурса `appJournal.properties`.

Пример шаблона secret для dev-стенда с ПЖ:

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-appjournalsettings
data:
stringData:
  appJournal.properties: |-
    standin.cloud.client.stub=false

    standin.cloud.client.zoneId=<укажите свою зону>

    standin.cloud.client.kafka.bootstrapServers=<адрес:порт>
```

Пример шаблона secret для конфигурации подключения к ПЖ на стендах с указанием параметров SSL подключения:

```
apiVersion: v1
kind: Secret
metadata:
  name: secret-appjournalsettings
data:
stringData:
  appJournal.properties: |-
    standin.cloud.client.stub=false

    standin.cloud.client.zoneId=${ZONE_ID}

    standin.cloud.client.kafka.bootstrapServers=${KAFKA_BOOTSTRAP_SERVERS}

    standin.cloud.client.kafka.producerConfig."[security.protocol]"=SSL
    standin.cloud.client.kafka.producerConfig."[ssl.key.password]"=${SSL_KEY_PAS
SWORD}
    standin.cloud.client.kafka.producerConfig."[ssl.keystore.location]"=/opt/key
store/kafka/server.keystore.jks
    standin.cloud.client.kafka.producerConfig."[ssl.keystore.password]"=${SSL_KE
YSTORE_PASSWORD}
    standin.cloud.client.kafka.producerConfig."[ssl.truststore.location]"=/opt/k
eystore/kafka/trust.jks
    standin.cloud.client.kafka.producerConfig."[ssl.truststore.password]"=${SSL_
TRUSTSTORE_PASSWORD}
```

```
    standin.cloud.client.kafka.producerConfig."[ssl.keystore.type]"=JKS
    standin.cloud.client.kafka.producerConfig."[ssl.truststore.type]"=JKS
    standin.cloud.client.kafka.producerConfig."[ssl.protocol]"=TLS
    standin.cloud.client.kafka.producerConfig."[ssl.enabled.protocols]"=TLSv1.2
    standin.cloud.client.kafka.producerConfig."[ssl.endpoint.identification.algo
rithm]"=

    standin.cloud.client.kafka.consumerConfig."[security.protocol]"=SSL
    standin.cloud.client.kafka.consumerConfig."[ssl.key.password]"=${SSL_KEY_PAS
SWORD}
    standin.cloud.client.kafka.consumerConfig."[ssl.keystore.location]"=/opt/key
store/kafka/server.keystore.jks
    standin.cloud.client.kafka.consumerConfig."[ssl.keystore.password]"=${SSL_KE
YSTORE_PASSWORD}
    standin.cloud.client.kafka.consumerConfig."[ssl.truststore.location]"=/opt/k
eystore/kafka/trust.jks
    standin.cloud.client.kafka.consumerConfig."[ssl.truststore.password]"=${SSL_
TRUSTSTORE_PASSWORD}
    standin.cloud.client.kafka.consumerConfig."[ssl.keystore.type]"=JKS
    standin.cloud.client.kafka.consumerConfig."[ssl.truststore.type]"=JKS
    standin.cloud.client.kafka.consumerConfig."[ssl.protocol]"=TLS
    standin.cloud.client.kafka.consumerConfig."[ssl.enabled.protocols]"=TLSv1.2
    standin.cloud.client.kafka.consumerConfig."[ssl.endpoint.identification.algo
rithm]"=
```

Все свойства с префиксом `standin.cloud.client.kafka` отвечают за соединение с Kafka ПЖ.

Параметру `standin.cloud.client.zoneId` необходимо прописать значение идентификатора зоны, выбранное на Этапе 3.

Параметру `standin.cloud.client.stub` необходимо прописать значение “false”.

По значениям остальных параметров можно обратиться к документации компонента Прикладной журнал продукта Platform V Data Tools.

Этап 6 — обеспечение доступа к Kafka ПЖ из кластера

Необходимо обеспечить доступ из пространства Kubernetes (или OpenShift) к Kafka ПЖ (см. раздел “Настройка пространства Kubernetes (OpenShift)”.

При интеграции на dev-средах, можно создать ServiceEntry по образцу:

```yaml
kind: ServiceEntry
spec:
  addresses:
    - 1.2.3.4
  endpoints:
    - address: 1.2.3.4
  exportTo:
    - .
  hosts:
    - dataspace.client.kafka.poseidon
```

```
  location: MESH_EXTERNAL
  ports:
    - name: tcp
      number: 9092
      protocol: TCP
  resolution: DNS
```

Этап 7 — конфигурирование параметров приложений DataSpace

Для обеспечения репликации в SI помимо остальных приложений DataSpace требуется разворачивать также приложение для применения векторов изменений DataSpace Applier (или Gigabas). Необходимо указывать параметры SpringBoot-приложений явно:

- Обязательно должно быть сконфигурировано соединение с Kafka ПЖ по аналогии с secret-appJournalSettings из этапа 5 (добавьте актуальные значения всех параметров из appJournal.properties в конфигурационные файлы приложений DataSpace Core и DataSpace Gigabas, например в application.properties).
- Должны быть сконфигурированы оба datasource:
  - для main БД – это свойства `spring.datasource.*` (`spring.datasource.url`, `spring.datasource.username`, `spring.datasource.password` и при необходимости другие)
  - для standin БД – это те же свойства, только с префиксом *standin* вместо *spring*: `standin.datasource.url`, `standin.datasource.username`, `standin.datasource.password` и при необходимости другие.

Настройка параметров применения (DataSpace Applier)

Существует возможность переопределения параметров применения векторов изменений. Для переопределения следует указать актуальные значения в шаблоне `dataspace-gigabas-template`.

```
Installer:
  parameters:
    dataspace-gigabas-template:
      ...
      applierKafkaConcurrency: 10
      applierRetryCount: 5
      applierRetryTimeout: 10000
```

Параметр	**Значение**	**Комментарий**
`subscriptionKafkaConcurrency`	Количество потоков, читающих топик с журналами на одно приложение	Настройка `standin.cloud.client.subscriptionKafkaConcurrency` клиентской библиотеки ПЖ. Рекомендуемое значение – 10. Это означает, что один модуль applier(gigabas) будет использовать 10 потоков для обработки партиций в топиках кафки, и рекомендуемое значение количества партиций – тоже 10, потому что: а) если количество партиций будет меньше 10, тогда applier(gigabas) будет

		иметь неиспользуемые зарезервированные ресурсы, и это не хорошо; б) если количество партиций будет больше 10, в этом случае 10 потоков applier(gigabas) будут переключаться для обработки излишних партиций, и это не хорошо. Если один под applier(gigabas) не справляется с нагрузкой, рекомендуется для каждого нового пода applier(gigabas) использующего 10 потоков, добавлять 10 новых партиций в топики. Например, при использовании трех подов applier(gigabas) на одном плече и трех подов applier(gigabas) на втором плече рекомендуется создать 60 партиций для топиков.
`applierRetryCount`	Количество попыток применения вектора изменений	Настройка `standin.cloud.client.kafka-retry` клиентской библиотеки ПЖ
`applierRetryTimeout`	Задержка между попытками обработать вектор изменений с ошибками репликации (в мс)	Настройка `standin.cloud.client.retry-timeout` клиентской библиотеки ПЖ

Проверка интеграции DataSpace с ПЖ на стенде разработки/тестирования

Проверка интеграции DataSpace с ПЖ на стенде разработки /тестирования осуществляется: по логам; через UI ПЖ; через тестовый запрос.

> **Внимание!**
>
> Рекомендуется использовать проверку через тестовый запрос.

Проверка интеграции DataSpace с ПЖ на стенде по логам

Выполните создание сущности (в данном примере создается сущность Product) и убедитесь, что она появилась в БД основного контура.

В логах DataSpace найдите следующие текстовые термы: Sent LCK+DATA journal. Данная строка лога означает, что в ПЖ отправлен вектор с данными. В этой же строке уникальный идентификатор в терминах ПЖ — id сервиса. Примерный вид данной информации такой: serviceId 909ee533-36d2-4be9-b214-4ca55c676c7f_69046525374425989l3#Product, где:

- 909ee533-36d2-4be9-b214-4ca55c676c7f — идентификатор транзакции;
- 6904652537442598913 — идентификатор агрегата;
- Product — тип агрегата.

Пример строки лога:

```
2020-12-10 18:38:14,884 [general-pool-thread-3] [INFO] (sbp.com.sbt.dataspace.re
plication.PostTransactionConfirmatorImpl) [sbp.com.sbt.dataspace.replication.Pos
tTransactionConfirmatorImpl::sendAck:23] mdc:()| tx 909ee533-36d2-4be9-b214-4ca5
5c676c7f: Received ACK
```

В логах Applier найти следующие текстовые термы: Applied journal createMode = 0, dataType = DATASPACE (означает, что вектор с данными успешно применен). В этой же строке находится id сервиса: serviceId = 909ee533-36d2-4be9-b214-4ca55c676c7f_69046525374425989l3#Product (тот же, что и в логах DataSpace).

Пример строки лога:

```
2020-12-10 18:38:15,123 [consumer-2-C-1] [INFO] (sbp.dataspace.standin.journal.S
tandInJournalConsumer) [sbp.dataspace.standin.journal.StandInJournalConsumer::ha
ndle:60] mdc:()| Applied journal createMode = 0, dataType = DATASPACE, serviceId
= 909ee533-36d2-4be9-b214-4ca55c676c7f_69046525374425989l3#Product
```

Проверка интеграции DataSpace с ПЖ на стенде через UI ПЖ

Для того, чтобы осуществить проверку через UI ПЖ, необходимо:

- В режиме журнала установить фильтр по полю **ID Сервиса**.
- В качестве значения указать значение из лога DataSpace: 909ee533-36d2-4be9-b214-4ca55c676c7f_69046525374425989l3#Product.

В итоге должно быть найдено две строки:

- первая – с типами данных LCK и DATASPACE;
- вторая – с типом данных ULCK.

Проверка интеграции DataSpace с ПЖ на стенде через тестовый запрос

Для того, чтобы осуществить проверку через тестовый запрос, необходимо:

1. Убедиться, что активным является Основной контур (Normal).

2. Выполнить создание сущности в контуре Main (например, Product), получить в ответе идентификатор (например, 6904652537442522222).

3. Выполнить SQL-запрос в БД основного контура и убедиться, что данная сущность присутствует. Для контроля убедиться, что дата создания сущности соответствует действительности.

4. Выполнить SQL-запрос в БД контура SI и убедиться, что данная сущность присутствует. Для контроля убедиться, что дата создания сущности соответствует действительности и совпадает с датой на основном контуре.

5. Убедиться, что все поля в таблице по данной сущности (Product с id = 6904652537442522222) полностью совпадают на обоих контурах.

6. Выполнить переход в функциональный StandIn.

7. Выполнить создание сущности в контуре StandIn (например, Product), получить в ответе идентификатор (например, 6904652537442577777).

8. Выполнить SQL-запрос в БД контура SI и убедиться, что данная сущность присутствует. Для контроля убедиться, что дата создания сущности соответствует действительности.

9. Выполнить SQL-запрос в БД основного контура и убедиться, что данная сущность отсутствует, поскольку при переходе в SI автоматически отключаются плагины репликации.

10. Выполнить переход в Основной контур (Normal). При этом плагины автоматически включатся, и репликация в основной должна быть выполнена.

11. Выполнить SQL-запрос в БД основного контура и убедиться, что данная сущность присутствует. Для контроля убедиться, что дата создания сущности соответствует действительности и совпадает с датой в контуре StandIn.

12. Убедиться, что все поля в таблице по данной сущности (Product с id = 6904652537442577777) полностью совпадают на обоих контурах.

Отключение репликации и отправки векторов в ПЖ

Возможна ситуация, когда все вышеперечисленное на этапе настроено и работает, но при этом необходимо отключить репликацию и отправку векторов в ПЖ (как правило, временно). Для этого нужно выполнить следующие действия:

1. В secret ПЖ установить параметр `standin.cloud.client.stub=true`.
2. В параметрах шаблона Core установить флаг `standinEnabled: false` и переустановить Core.

Чтобы включить репликацию, необходимо удалить параметры из secret ПЖ, для этого:

1. В secret ПЖ установить параметр `standin.cloud.client.stub=false`.
2. В параметрах шаблона Core установить флаг `standinEnabled: true` и переустановить Core.

Часто встречающиеся проблемы и пути их устранения

Проблема 1: приложение не поднимается

Приложение не поднимается, например:

```
Failed to configure a DataSource: 'url' attribute is not specified and no embedd
ed datasource could be configured
```

Решение:

- необходимо проверить доступность к БД Main и StandIn **изнутри Kubernetes (или OpenShift)**, которые были указаны в secret.
- Обратить внимание на распространенную ошибку при настройке secrets основного контура и StandIn:
  - для secret основного контура используется префикс *spring*;
  - для secret контура StandIn используется префикс *standin*.

Проблема 2: Любая неясная ситуация, связанная с отсутствием репликации

Если возникла любая не понятная ситуация, связанная с отсутствием репликации, необходимо проверить, что включены основные настройки для Core (не для applier (он же gigabas)):

- *standin.configuration.enabled = true*;
- *standin.multitenant-entity-manager.enable = true*.

Один из вариантов ошибки в логах по этой причине следующий:

```
Caused by: ru.sbrf.journal.client.JournalClientException: Невозможно определить
режим создания журнала из контекста и он не передан отправителем.
```

Решение: проверьте с помощью actuator/env (лучше всего в терминале pod Core запросить: `curl http://localhost:8080/actuator/env`), что имеются параметры со значениями dataspace.replication.enabled=true.

Проблема 3: запись должна выполняться в БД SI, но выполняется в БД Main

Вектора изменений применяются в БД Main вместо того, чтобы применяться в БД SI.

При модификации созданных сущностей выдаются ошибки вида:

```
"message":"Object <ID сущности>#<тип сущности> is locked by StandIn","data":"sbp
.sbt.sdk.exception.detailedexception.SystemLockException"
```

Чтобы проверить наличие проблемы, необходимо убедиться, что:

1. Активный контур в ПЖ — Main.
2. При создании сущности запись в БД производится в БД Main.
3. Переходим в ПЖ в StandIn. При создании сущности запись в БД производится снова в БД Main, а не в БД SI.

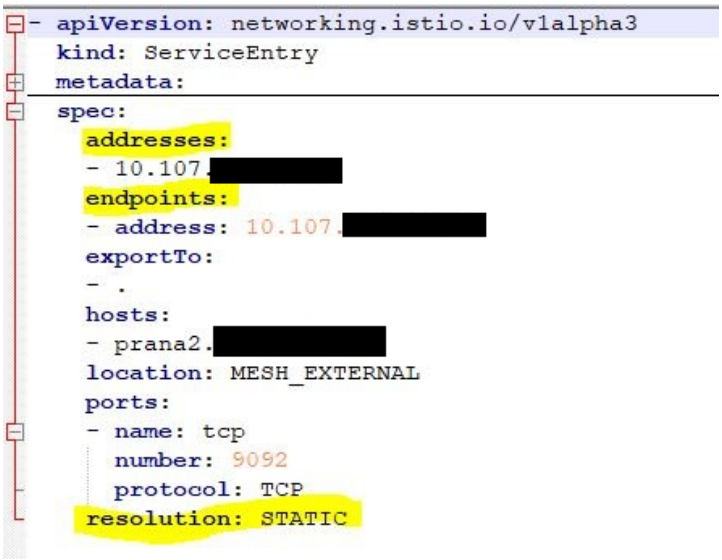
Решение:

- С помощью команды curl http://localhost:8080/actuator/env в терминале pod модуля dataspace-core получить информацию о значениях параметров подключения к БД:
  - *spring.datasource.url*;
  - *standin.datasource.url*.
- Убедиться, что значения заданы корректно и направлены на разные и правильные БД.
- Если настройки заданы верно, значит ошибку необходимо искать в настройках Kubernetes (или OpenShift).

Описание первого метода решения:

Ранее через 2 VirtualService приходили запросы на хосты host1.dev.local:6544 и host2.dev.local:6544, которые маршрутизировались на Egress, где был открыт порт 2798 для TCP-трафика. Здесь имелось пересечение трафика. Далее трафик при помощи 2 VirtualService маршрутизировался с порта 2798 на данные указанные хосты. Было предположено, что селектор VirtualService не отрабатывает, и созданы новые порты 3000 и 3001, чтобы трафик для контура Main шел через порт 3001, а VirtualService – через 3000.

Пример второго метода решения:



Проблема 4: приложение не поднимается из-за проблем с соединением с БД SI

Приложение не поднимается, в логах выводится информация вида:

```
[com.sbt.pprb.standin.em.dialect.DialectAutoDetector::getDialectResolutionInfo:53]dataSource=proxyDataSourceDecorator [net.ttddyy.dsproxy.support.ProxyDataSource] -> StandInDataSource [com.zaxxer.hikari.HikariDataSource]
org.postgreSQL.util.PSQLException: The connection attempt failed.
```

Решение: указать параметры БД Main для secret SI. Если ошибка исчезает, то проблема с БД SI или внутри OS не верно настроена маршрутизация на внешнюю БД SI. Метод решения проблемы см. в разделе “Проблема 3: запись должна выполняться в БД SI, но выполняется в БД Main”.

Проблема 5: приложение не поднимается из-за проблем с соединением с Kafka ПЖ

Приложение не поднимается, в логах выводится информация вида:

```
2022-01-18 11:28:33,579 [consumer-0-C-1] [INFO] (org.apache.kafka.clients.consum
er.internals.AbstractCoordinator)
[org.apache.kafka.clients.consumer.internals.AbstractCoordinator::markCoordinato
rUnknown:849] mdc:()|
[Consumer clientId=consumer-event-dataspace-core-def-02.020.00-5df84f64dc-h6hvh_
-2,
groupId=event-dataspace-core-tam-02.020.00-5df84f64dc-h6hvh_]
Group coordinator 127.0.0.1:9092 (id: 2147483643 rack: null) is unavailable or i
nvalid, will attempt rediscovery
```

или

```
2022-01-18 11:28:33,683 [consumer-0-C-1] [INFO] (org.apache.kafka.clients.consum
er.internals.AbstractCoordinator)
[org.apache.kafka.clients.consumer.internals.AbstractCoordinator::joinGroupIfNee
ded:455] mdc:()|
[Consumer clientId=consumer-event-dataspace-core-def-02.020.00-5df84f64dc-h6hvh_
-2,
groupId=event-dataspace-core-tam-02.020.00-5df84f64dc-h6hvh_]
Join group failed with org.apache.kafka.common.errors.DisconnectException
```

Решение:

1. Проверить параметры подключения к Kafka ПЖ (в secret-appjournalsettings).
2. Проверить доступ из namespace в Kafka (например, с pod выполнить `curl –v kafkaBrokerHost:9093`):
   - если доступа нет, то проверить наличие и корректность настроек примитивов Kubernetes (или OpenShift), обеспечивающих такой доступ;
   - если доступ есть — обратиться к команде ПЖ для диагностики проблем с Kafka.

Руководство по установке компонента DS Lab (DSLB)

О документе

Настоящий документ содержит описание процесса установки и настройки компонента DS Lab продукта Platform V DataSpace (далее — DS Lab или компонент).

DS Lab — компонент, предоставляющий возможность описывать модель данных предметной области через UI и разворачивать модель в виде сервиса Platform V Dataspace для работы с этими данными: создание, изменение, удаление, объединение в транзакционные группы, гибкие возможности поиска и получения данных.

Основные понятия

В таблице приведены основные аббревиатуры и сокращения:

Аббревиатура, сокращение	Расшифровка
БД	База данных
КТС	Комплекс технических средств
ОС	Операционная система
СУБД	Система управления базами данных
ТУЗ	Технологическая учетная запись
SDK	Клиентская часть программного продукта DataSpace
CPU	Central processing unit. Центральное обрабатывающее устройство
DDL	Data Definition Language. Команды языка SQL для создания и изменения объектов БД
HDD	Hard Disk Drive
IOPS	Input/Output Operations Per Second. Количество операций ввода/вывода в секунду
k8s	Среда контейнеризации Kubernetes
Node	Узел Kubernetes
PG	PostgreSQL
RAM	Random Access Memory. Оперативная память
SQL	Structured Query Language
UI	User Interface. Графический пользовательский интерфейс
VM	Virtual machine
XML	eXtensible Markup Language. Расширяемый язык разметки

В таблице ниже приведены основные термины и определения:

Термин	Определение
Платформа	Набор продуктов Platform V, правообладателем которых является АО “СберТех”. Перечень таких продуктов обозначен в документации на конкретный продукт
Продукт	Platform V DataSpace
DS Lab	Компонент DS Lab продукта Platform V DataSpace
Helm	Средство упаковки с открытым исходным кодом, которое помогает установить приложения Kubernetes и управлять их жизненным циклом
Liquibase	Система управления версиями базы данных
PostgreSQL	СУБД PostgreSQL (рекомендован Platform V Pangolin SE)

Системные требования

Системные требования, приведенные в данном разделе, являются минимальными.

Аппаратные требования

Минимальные требования для работы модулей DS Lab:

- Для модуля dataspace-core-meta:
  - Процессор: 2 x CPU.
  - Оперативная память: 2 ГБ или больше.
- Для модуля dataspace-meta:
  - Процессор: 1 x CPU.
  - Оперативная память: 1 ГБ или больше.

Дополнительные минимальные требования для каждого разворачиваемого клиентского DataSpace:

- Процессор: 0.25 x CPU.
- Оперативная память: 512 Мб.
- Для модуля dataspace-meta:
  - Процессор: 2 x CPU.
  - Оперативная память: 2 ГБ или больше.

Дополнительные минимальные требования для каждого разворачиваемого клиентского Dataspace:

- Процессор: 0.25 x CPU.
- Оперативная память: 512 Мб.

Типовые требования к серверу БД для модуля dataspace-core-meta:

- СУБД на базе PostgreSQL 12+;
- CPU — 2шт;
- RAM — 4GB;
- HDD — 40GB;
- IOPS — 1000.

Типовые требования к серверу БД для клиентского модуля dataspace-core:

- СУБД на базе PostgreSQL 12+;
- CPU — 4шт;
- RAM — 8GB;
- HDD — 120GB;
- IOPS — 2000.

Требования к кластеру среды контейнеризации приложений: должно быть не менее двух Node.

Программные требования

На серверы приложений, использующих DataSpace, должна быть установлена среда контейнеризации Kubernetes, Red Hat OpenShift 4+ или аналогичная среда контейнеризации приложений.

Требования к ядру Linux:

- Версия ядра — 3.10.x (протестировано, на других версиях ядра стабильная работа не гарантируется).
- Поддержка только cgroupv1 (cgroupv2 не поддерживается).

Рекомендуется использовать ОС “Альт 8 СП”.

Требования к инфраструктуре и конфигурациям

Имеются следующие требования:

- Наличие БД для модуля dataspace-core-meta в режиме standby.
- Созданы и настроены два namespace в Kubernetes (или OpenShift):
  - namespace для клиентских dataspace;
  - namespace для остальных сервисов (nginx/dslab/prometheus).
- Наличие ОС Kubectl (CLI) на ПК, с которого производится установка.
- Наличие пакетного менеджера Helm на ПК, с которого производится установка.
- Наличие docker на машине для сборки images dataspace-core-meta/dataspace-meta/dslab-builder.
- Наличие пакета liquibase версии 4.4.2.
- Наличие jdbc-драйвера PostgreSQL версии не ниже 42.2.18.
- Доступ к частному или публичному registry image.
- Доступ к частному или публичному package repository.
- Доступ к следующим общим образам (в скобках приведены ссылки на hub.docker.com):
  - curlimages/curl;
  - maven:3.8.1-adoptopenjdk-11;
  - alpine/helm;
  - liquibase/liquibase.

Необязательные требования (в зависимости от используемых функций):

- Наличие установленного ПО Influx + Grafana.
- Наличие установленного ПО Prometheus + Grafana.

Описание внешнего ПО, используемого для установки, настройки и контроля, с указанием выполняемых функций приведено в разделе “Прочие аспекты” документа “Детальная архитектура”.

Настройка namespace

Namespaces k8s или OpenShift будет считаться готовым к развертыванию DS Lab после выполнения всех следующих требований:

- Установлен ingress nginx.
- Для namespace создан secret типа Dockersecret с логином и паролем ТУЗ для image pull.
- Созданный secret прописан в service accounts: `default`.

- Для namespace, в котором разворачивается ControlPlane, создан secret с сертификатами Kubernetes.
- Для namespace, в котором разворачивается ControlPlane, создан secret с параметрами подключения к БД.
- Для namespace клиентских Dataspace создан секрет с kubeconfig.
- Для namespace клиентских Dataspace создан секрет с settings.xml.

Создание pull image secret:

`shell script kubectl create secret docker-registry <secret-name> --docker-server=<your-registry-server> --docker-username=<your-name> --docker-password=<your-pword> --docker-email=<your-email>` Создание secret с сертификатами Kubernetes:

`shell script kubectl apply -f <имя файла>` где файл:

```
kind: Secret
apiVersion: v1
metadata:
  name: k8s-certifacte
data:
  ca.crt: <ca.crt in base64>
  client.crt: <client.crt in base64>
  client.key: <client.key in base64>
type: Opaque
```

Создание secret для подключения к БД:

`shell script kubectl create secret generic <secret-name> --from-file=<имя файла>` где файл:

```
apiVersion: v1
kind: Secret
metadata:
  name: main-db-secret
stringData:
  secret.properties: |-
    spring.datasource.username=<имя пользователя для DML операций>
    spring.datasource.password=<пароль пользователя>
    spring.datasource.url=<jdbc строка для подключения к бд>
    dataspace.datasource.primary.dbschema=<имя схемы>
    spring.datasource.driver-class-name=<java driver class>
```

Создание secret с kubeconfig:

`shell script kubectl apply -f <имя файла>`

Пример файла:

```
kind: Secret
apiVersion: v1
metadata:
  name: kube-config
data:
```

```
  config: <kuberconfig in base64>
type: Opaque
```

Создание secret с settings.xml:

```
shell script kubectl apply -f <имя файла>
```

Пример файла:

```
apiVersion: v1
kind: Secret
metadata:
  name: maven-settings
stringData:
  settings.xml: |-
    <settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
                      http://maven.apache.org/xsd/settings-1.0.0.xsd">
        <servers>
            <server>
                <id>release.repo</id>
                <username></username>
                <password></password>
            </server>
            <server>
                <id>internal</id>
                <username></username>
                <password></password>
            </server>
        </servers>
        <profiles>
            <profile>
                <id>settings</id>
                <repositories>
                    <repository>
                        <id>release.repo</id>
                        <url></url>
                    </repository>
                </repositories>
                <pluginRepositories>
                    <pluginRepository>
                        <id>release.repo</id>
                        <url></url>
                    </pluginRepository>
                </pluginRepositories>
            </profile>
        </profiles>
        <activeProfiles>
            <activeProfile>settings</activeProfile>
        </activeProfiles>
  </settings>
```

Подготовка базы данных

Для работы DS Lab требуется развернуть БД. Поддерживается СУБД PostgreSQL 12.x и выше (далее PostgreSQL или PG). Рекомендуется использовать продукт Platform V Pangolin SE.

> **Примечание**
>
> Для именования объектов БД (учетных записей, ролей, схем, табличных пространств и т.д.) следует руководствоваться требованиями используемой СУБД в части имен без использования кавычек. Обычно в допустимый набор символов входят латинские буквы (a-z, A-Z), цифры (0-9) и символ подчеркивания (_), имена должны начинаться с буквы или символа подчеркивания.

Создание учетных записей

На БД PostgreSQL выполнить следующие скрипты:

```
CREATE USER as_admin WITH PASSWORD 'Passw0rd' LOGIN NOCREATEROLE NOCREATEDB NOSU
PERUSER NOINHERIT NOREPLICATION NOBYPASSRLS VALID UNTIL '2021.12.31';
CREATE DATABASE dspc_db OWNER as_admin ENCODING 'UTF-8' LC_COLLATE 'en_US.UTF-8'
LC_CTYPE 'en_US.UTF-8';
CREATE USER "as_tuz" WITH PASSWORD 'Passw0rd' LOGIN NOCREATEROLE NOCREATEDB NOSU
PERUSER NOINHERIT NOREPLICATION NOBYPASSRLS VALID UNTIL '2021.12.31';
CREATE USER dsc_appl WITH ENCRYPTED PASSWORD 'Passw0rd' INHERIT;
GRANT "as_tuz" TO dsc_appl;
```

Ручная сборка образов модулей DS Lab

Сборка образа dataspace-core-meta

> **Примечание**
>
> Должен быть собран базовый образ dataspace-core.

В составе дистрибутива в папке *docker* лежит все необходимое для сборки базового дистрибутива компонентов DataSpace.

Произвести сборку образа, пример:

```
shell script docker build -t dataspace-core-meta:4.3.455 . --build-arg
BASE_IMAGE=<image>
```

Сборка образа dataspace-meta

В составе дистрибутива в папке *docker* лежит все необходимое для сборки базового дистрибутива компонентов DataSpace.

Произвести сборку образа, пример:

```
shell script docker build -t dataspace-core-meta:4.3.455 . --build-arg
registryUrl=<url> --build-arg rhelOpenJdkUrl=<image> --build-arg
version=<version>
```

Сборка образа DS builder

Необходимо подложить папку m2 из дистрибутива продукта Platform V Dataspace в каталог с Dockefile, остальное для сборки лежит в составе дистрибутива рядом с Dockerfile

Произвести сборку образа, пример:

```
shell script docker build -t ds-builder:<версия dataspace-bom> . --build-arg
BASE_IMAGE=maven:3.8.1-adoptopenjdk-11
```

Установка

Для установки компонента DS Lab необходимо выполнить следующие действия:

3. Распаковать дистрибутив компонента DS Lab.

4. Создать файл liquibase.properties, содержащий следующую конфигурацию:

   ```
   changelog-file: <путь к changelog.xml из папки db>
   username: <пользователь с правами DDL>
   password: <пароль пользователя>
   url: <jdbc адрес подключения к бд>
   defaultSchemaName: <имя схемы>
   driver: org.postgresql.Driver
   parameter.defaultSchemaName: <имя схемы>
   parameter.tablespace_t: <имя tablespace для таблиц>
   parameter.tablespace_l: <имя tablespace для ЛОБов>
   parameter.tablespace_i: <имя tablespace для индексов>
   ```

5. Выполнить скрипты Liquibase при помощи любой формы Liquibase. Пример для liquibase cli:

   ```
   shell script  liquibase update -defaultsFile=/path/to/liquibase.properties
   ```

6. Установка DS Lab производится с помощью пакетного менеджера Helm для установки.

7. Сформировать конфигурационный файл для модуля dataspace-core-meta для последующей установки с помощью Helm:

   ```
   releaseName: ds-core-meta
   values:
   replicaCount: 2 # количество поднимаемых подов
   appConfig:
     packagesToScan: "com.sbt.meta.model" # groupId модели DS Lab
     fluentBit:
       enabled: false
     db:
       main:
         secret: "ds-db-main-secret" # имя секрета, созданного в пункте "Создан
   ие secret для подключения к бд"
     overrideProperties:
       graphiql:
   ```

```
      endpoint:
        graphql: /ds-core-meta/graphql #Т.к используется nginx необходимо пе
реопределить пути статики. Должно совпадать с путем в ingress
        subscriptions: /ds-core-meta/subscriptions #Т.к используется nginx н
еобходимо переопределить пути статики. Должно совпадать с путем в ingress
      basePath: /ds-core-meta/ #Т.к используется nginx необходимо переопреде
лить пути статики. Должно совпадать с путем в ingress
    dataspace:
      endpoint:
        graphiql:
          enabled: true
        graphql:
          enabled: true
    server:
      tomcat:
        general-thread-analytics: true
        threadstatistics: false
    spring:
      jpa:
        database-platform: org.hibernate.dialect.H2Dialect
        show-sql: false
service:
  port: 8080
  type: NodePort
  unsecPort: 9999
ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$2
  tls:
    - secretName: ds-tls-secret
      hosts:
        - dsmeta.ddns.net
  hosts:
    - host: 'dsmeta.ddns.net'
      paths:
        - path: /ds-core-meta(/|$)(.*)
    - host: ''
      paths:
        - path: /ds-core-meta(/|$)(.*)
resources:
  limits:
    cpu: 2000m
    memory: 2048Mi
  requests:
    cpu: 2000m
    memory: 2048Mi
image:
  registryUrl: "base.sw.sbc.space"
  registryProject: "pprb/ci00682829_cdm"
imagePullSecrets:
```

```
  - name: default-secret
  - name: nexusbasesw # имя секрета созданного в пункте Создание pull image
secret
```

8. Установить модуль dataspace-core-meta с помощью Helm:

```
shell script  helm upgrade dataspace-core-meta ./dataspace-core-meta -f
./<file-name-config>.yaml --wait --install --namespace <namespace-name>
```

9. Сформировать конфигурационный файл для dataspace-meta для последующей установки с помощью Helm:

```
releaseName: ds-meta
values:
  replicaCount: 2
  appConfig:
    k8s:
      secret: "k8s-certifacte" # значение, имя созданного secret
    fluentBit:
      enabled: true
      image: "fluent/fluent-bit:1.4.5"
    overrideProperties:
      zuul:
        routes:
          gql-plgr-stitching:
            url: http://svc-ds-stitching-{{modelId}}.dataspace-tenant.svc.cl
uster.local:8080/graphql
      customer-dataspace-host: http://svc-ds-{{modelId}}.dataspace-tenant.sv
c.cluster.local:8080
      database:
        enabled: false
      sbercloud:
        database:
          type: PG
        grafana:
          url: http://172.16.0.144:3000
          extUrl: https://pv-api-test.sbc.space
          dataSource:
            url: http://ds-prometheus-k8s-1.dataspace.sbercloud.ru:31076
        k8s:
          url: https://172.16.253.233:5443
          caCertFile: file:/deployments/config/secrets/k8s/ca.crt
          clientCertFile: file:/deployments/config/secrets/k8s/client.crt
          clientKeyFile: file:/deployments/config/secrets/k8s/client.key
          builder:
            image-pull-secrets: nexusdzosw
            container-image: dzo.sw.sbc.space/sbt_dev/ci90000026_dspc_dev/ds
-builder:kss9-SNAPSHOT
            docker-config: nexusdzosw
            nexus.type: nexus3
            nexus:
              url: https://dzo.sw.sbc.space/nexus-ci
              repository: sbt-maven
```

```
            docker:
              registry: dzo.sw.sbc.space
              username: tuz_sbt_ci_dataspace
              password: 1Jlyf_Uh#,fyfz_Hjpjdfz_Hjpf
          helm:
            init-container-image: curlimages/curl
            container-image: dzo.sw.sbc.space/sbt/ci90000026_dspc/alpine/hel
m
          liquibase:
            init-container-image: curlimages/curl
            container-image: dzo.sw.sbc.space/sbt/ci90000026_dspc/liquibase/
liquibase:latest
            image-pull-secrets: default,nexusdzosw
        nexus:
          username: <username>
          password: <password>
          restEndpoint: https://dzo.sw.sbc.space/nexus-ci/service/rest/v1
      sber:
        cloud:
          k8s:
            tenant:
              namespace: dataspace-tenant
        controlplane:
          mock:
            endpointprefix: "http://37.18.121.198:80"
          pipelines:
            "[default]":
              deploy: RollScriptToDbDeployTask,SecretRegistrationDeployTask,
BuilderTask,FilePersistingDeployTask,LiquibaseDeployTask,DeployTask,GrafanaD
eployTask,ApiGatewayDeployTask
              rollback: LiquibaseRollbackTask,SecretRegistrationRollbackTask
,RollScriptToDbRollbackTask
          taskProperties:
            "[GrafanaDeployTask]":
              "[url]": http://172.16.0.144:3000
              "[exturl]": https://pv-api-test.sbc.space
              "[apikey]": Bearer <api key in base64>"
              "[username]": <username>
              "[password]": <password>
              "[userprefix]": dash_
              "[dataSourceUrl]": http://ds-prometheus-k8s-1.dataspace.sbercl
oud.ru:31076
              "[dataSourceType]": PROMETHEUS
              "[dataSourcePrometheusScrapeInterval]": 10s
              "[dataSourcePrometheusQueryTimeout]": 10s
              "[dataSourcePrometheusHttpMethod]": POST
            "[ApiGatewayDeployTask]":
              "[execution_time]": 10
              "[externalIp]": http://37.230.196.201/
        build:
          groupIdPrefix: ru.sbt.platformv.dev
          #groupId: sbp.com.sbt.dataspace.smartapp
```

```
        dataspacebom:
          version: 4.3.255
      endpoints:
        env:
          keys-to-sanitize: secret,credentials,password
      server:
        port: 8080
      core:
        unsecure:
          url: "http://svc-ds-core-meta.dataspace.svc.cluster.local:9999"
        url: "http://svc-ds-core-meta.dataspace.svc.cluster.local:8080"
      controlPlaneUrl: "http://localhost:8080/controlPlane"
  service:
    port: 8080
    type: NodePort
  ingress:
    enabled: true
    annotations:
      kubernetes.io/ingress.class: nginx
    hosts:
      - host: 'dsmeta.ddns.net'
        paths:
          - path: /
      - host: ''
        paths:
          - path: /
  resources:
    limits:
      cpu: 600m
      memory: 512Mi
    requests:
      cpu: 300m
      memory: 256Mi
  image:
    registryUrl: "base.sw.sbc.space"
    registryProject: "pprb/ci00682829_cdm"
  imagePullSecrets:
    - name: default-secret
    #- name: nexus3swpublic
    - name: nexusbasesw
  hostAliases: #В связи с проблемами с DNS в инфраструктуре sbercloud. Досту
пно с версии меты 4.1.75
    - ip: 37.18.122.120
      hostnames:
        - apig.ru-moscow-1.hc.sbercloud.ru
    - ip: 10.255.254.17
      hostnames:
        - base.sw.sbc.space
    - ip: 10.10.10.20
      hostnames:
        - dzo.sw.sbc.space
  core:
```

```
    readinessProbe:
      failureThreshold: 10
    livenessProbe:
      failureThreshold: 10
```

10. Установить модуль dataspace-meta с помощью Helm:

```
shell script  helm upgrade dataspace-meta ./dataspace-meta -f ./<file-name-config>.yaml --wait --install --namespace <namespace-name>
```

```
releaseName: ds-core-meta
values:
replicaCount: 2 # количество поднимаемых подов
appConfig:
  packagesToScan: "com.sbt.meta.model" # groupId моделиDS Lab
  fluentBit:
    enabled: false
  db:
    main:
      secret: "ds-db-main-secret" # имя секрета, созданного в пункте "Создание secret для подключения к бд"
  overrideProperties:
    graphiql:
      endpoint:
        graphql: /ds-core-meta/graphql #Т.к используется nginx необходимо переопределить пути статики. Должно совпадать с путем в ingress
        subscriptions: /ds-core-meta/subscriptions #Т.к используется nginx необходимо переопределить пути статики. Должно совпадать с путем в ingress
      basePath: /ds-core-meta/ #Т.к используется nginx необходимо переопределить пути статики. Должно совпадать с путем в ingress
    dataspace:
      security:
        suppressorMode: "aggregate"
        rawHeader:
          port: 8080
          key: owner-id
        aggregateRootComparison:
          fields: "VModel:owner.entityId"
          statusesMode: blacklist
          statuses: VModel:BLOCKED
        unsecure:
          port: 9999
      endpoint:
        graphiql:
          enabled: true
        graphql:
          enabled: true
    server:
      tomcat:
        general-thread-analytics: true
        threadstatistics: false
```

```
    spring:
      jpa:
        database-platform: org.hibernate.dialect.H2Dialect
        show-sql: false
service:
  port: 8080
  type: NodePort
  unsecPort: 9999
ingress:
  enabled: true
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /$2
  tls:
    - secretName: ds-tls-secret
      hosts:
        - dsmeta.ddns.net
  hosts:
    - host: 'dsmeta.ddns.net'
      paths:
        - path: /ds-core-meta(/|$)(.*)
    - host: ''
      paths:
        - path: /ds-core-meta(/|$)(.*)
resources:
  limits:
    cpu: 2000m
    memory: 2048Mi
  requests:
    cpu: 2000m
    memory: 2048Mi
image:
  registryUrl: "base.sw.sbc.space"
  registryProject: "pprb/ci00682829_cdm"
imagePullSecrets:
  - name: default-secret
  - name: nexusbasesw # имя секрета созданного в пункте Создание pull image
secret
```

11. Установить модуль dataspace-core-meta с помощью Helm:

```
shell script  helm upgrade dataspace-core-meta ./dataspace-core-meta -f
./<file-name-config>.yaml --wait --install --namespace <namespace-name>
```

12. Сформировать конфигурационный файл для dataspace-meta для последующей установки с помощью Helm (пример):

```
releaseName: ds-meta
values:
  replicaCount: 2
  appConfig:
    k8s:
      secret: "k8s-certifacte" # значение, имя созданного secret
```

```
    fluentBit:
      enabled: true
      image: "fluent/fluent-bit:1.4.5"
    overrideProperties:
      zuul:
        routes:
          gql-plgr-stitching:
            url: http://svc-ds-stitching-{{modelId}}.dataspace-tenant.svc.cl
uster.local:8080/graphql
      customer-dataspace-host: http://svc-ds-{{modelId}}.dataspace-tenant.sv
c.cluster.local:8080
      database:
        enabled: false
      sbercloud:
        database:
          type: PG
        grafana:
          url: http://172.16.0.144:3000
          extUrl: https://pv-api-test.sbc.space
          dataSource:
            url: <url_grafana>
        k8s:
          url: https://172.16.253.233:5443
          caCertFile: file:/deployments/config/secrets/k8s/ca.crt
          clientCertFile: file:/deployments/config/secrets/k8s/client.crt
          clientKeyFile: file:/deployments/config/secrets/k8s/client.key
          builder:
            image-pull-secrets: nexusdzosw
            container-image: <образ ds builder>
            docker-config: nexusdzosw
            nexus.type: nexus3
            nexus:
              url: https://dzo.sw.sbc.space/nexus-ci
              repository: sbt-maven
            docker:
              registry: dzo.sw.sbc.space
              username: tuz_sbt_ci_dataspace
              password: 1Jlyf_Uh#,fyfz_Hjpjdfz_Hjpf
          helm:
            init-container-image: curlimages/curl
            container-image: dzo.sw.sbc.space/sbt/ci90000026_dspc/alpine/hel
m
          liquibase:
            init-container-image: curlimages/curl
            container-image: dzo.sw.sbc.space/sbt/ci90000026_dspc/liquibase/
liquibase:latest
            image-pull-secrets: default,nexusdzosw
        nexus:
          username: <username>
          password: <password>
          restEndpoint: https://dzo.sw.sbc.space/nexus-ci/service/rest/v1
      sber:
```

```
        cloud:
          k8s:
            tenant:
              namespace: dataspace-tenant
        controlplane:
          mock:
            endpointprefix: "http://37.18.121.198:80"
          pipelines:
            "[default]":
              deploy: RollScriptToDbDeployTask,SecretRegistrationDeployTask,
BuilderTask,FilePersistingDeployTask,LiquibaseDeployTask,DeployTask,GrafanaD
eployTask,ApiGatewayDeployTask
              rollback: LiquibaseRollbackTask,SecretRegistrationRollbackTask
,RollScriptToDbRollbackTask
          taskProperties:
            "[GrafanaDeployTask]":
              "[url]": http://172.16.0.144:3000
              "[exturl]": https://pv-api-test.sbc.space
              "[apikey]": Bearer <api key in base64>"
              "[username]": <username>
              "[password]": <password>
              "[userprefix]": dash_
              "[dataSourceUrl]": http://ds-prometheus-k8s-1.dataspace.sbercl
oud.ru:31076
              "[dataSourceType]": PROMETHEUS
              "[dataSourcePrometheusScrapeInterval]": 10s
              "[dataSourcePrometheusQueryTimeout]": 10s
              "[dataSourcePrometheusHttpMethod]": POST
            "[ApiGatewayDeployTask]":
              "[execution_time]": 10
              "[externalIp]": http://37.230.196.201/
        build:
          groupIdPrefix: ru.sbt.platformv.dev
          #groupId: sbp.com.sbt.dataspace.smartapp
          dataspacebom:
            version: 4.3.255
      endpoints:
        env:
          keys-to-sanitize: secret,credentials,password
      server:
        port: 8080
      core:
        unsecure:
          url: "http://svc-ds-core-meta.dataspace.svc.cluster.local:9999"
        url: "http://svc-ds-core-meta.dataspace.svc.cluster.local:8080"
      controlPlaneUrl: "http://localhost:8080/controlPlane"
  service:
    port: 8080
    type: NodePort
  ingress:
    enabled: true
    annotations:
```

```
      kubernetes.io/ingress.class: nginx
    hosts:
      - host: 'dsmeta.ddns.net'
        paths:
          - path: /
      - host: ''
        paths:
          - path: /
  resources:
    limits:
      cpu: 600m
      memory: 512Mi
    requests:
      cpu: 300m
      memory: 256Mi
  image:
    registryUrl: "base.sw.sbc.space"
    registryProject: "pprb/ci00682829_cdm"
  imagePullSecrets:
    - name: default-secret
    #- name: nexus3swpublic
    - name: nexusbasesw
  hostAliases: #В связи с проблемами с DNS в инфраструктуре sbercloud. Досту
пно с версии меты 4.1.75
    - ip: 37.18.122.120
      hostnames:
        - apig.ru-moscow-1.hc.sbercloud.ru
    - ip: 10.255.254.17
      hostnames:
        - base.sw.sbc.space
    - ip: 10.10.10.20
      hostnames:
        - dzo.sw.sbc.space
  core:
    readinessProbe:
      failureThreshold: 10
    livenessProbe:
      failureThreshold: 10
```

13. Установить модуль dataspace-meta с помощью Helm:

```
shell script helm upgrade dataspace-meta ./dataspace-meta -f ./<file-name-
config>.yaml --wait --install --namespace <namespace-name>
```

Обновление

Для обновления версии модулей DS Lab необходимо повторить шаги, описанные в разделе “Установка”.

Проверка работоспособности

Для проверки работоспособности необходимо выполнить HTTP-запрос get на адрес `<адрес сервиса>/actuator/health`. В случае успеха будет возвращен ответ вида:


```
{
    "status": "UP",
    "components": {
        "activeDataSource": {
            "status": "UP",
            "details": {
                "database": "H2",
                "validationQuery": "isValid()"
            }
        },
        "db": {
            "status": "UP",
            "components": {
                "activeDataSource": {
                    "status": "UP",
                    "details": {
                        "database": "H2",
                        "validationQuery": "isValid()"
                    }
                },
                "dataSource": {
                    "status": "UP",
                    "details": {
                        "database": "H2",
                        "validationQuery": "isValid()"
                    }
                },
                "standinDataSource": {
                    "status": "UP",
                    "details": {
                        "database": "H2",
                        "validationQuery": "isValid()"
                    }
                }
            }
        },
        "discoveryComposite": {
            "description": "Discovery Client not initialized",
            "status": "UNKNOWN",
            "components": {
                "discoveryClient": {
                    "description": "Discovery Client not initialized",
                    "status": "UNKNOWN"
                }
            }
        },
        "diskSpace": {
```

```
            "status": "UP",
            "details": {
                "total": 499963174912,
                "free": 331588116480,
                "threshold": 10485760,
                "exists": true
            }
        },
        "hystrix": {
            "status": "UP"
        },
        "ping": {
            "status": "UP"
        },
        "reactiveDiscoveryClients": {
            "description": "Discovery Client not initialized",
            "status": "UNKNOWN",
            "components": {
                "Simple Reactive Discovery Client": {
                    "description": "Discovery Client not initialized",
                    "status": "UNKNOWN"
                }
            }
        },
        "refreshScope": {
            "status": "UP"
        }
    },
    "groups": [
        "liveness",
        "readiness"
    ]
}
```


Для представленных выше настроек необходимо наличие соответствующих значений работоспособности: UP, isValid().

Откат

В разделе описаны шаги, которые необходимо выполнить для отката изменений в модулях DS Lab и используемых базах данных.

Откат модулей DS Lab необходимо производить в следующей последовательности:

14. Откат модулей DS Lab. В пространство Kubernetes необходимо установить старую версию модулей. Для этого достаточно выполнить шаги, описанные в разделе “Установка”.

15. Откат изменений БД (если необходимо). С принципами отката с помощью Liquibase можно ознакомиться в официальной документации утилиты.

    Необходимо распаковать дистрибутив, перейти в db и запустить утилиту liquibase:

```
shell script  liquibase --changeLogFile=changelog.xml rollback <tag>
```

Для выполнения отката требуется передать тег, до которого необходим откат (см. раздел “Формирование тега rollback”).

Формирование тега rollback

После выпуска новой релизной версии модели в changelog добавляются все изменения БД относительно предыдущего выпуска. Все изменения размещаются между тегами:

- `<modelName>-<version>-before` — тег, обозначающий начало новых изменений;
- `<modelName>-<version>-applied` — тег, обозначающий окончание изменений.

Формирование тегов позволяет осуществить быстрый откат до необходимой версии.

Ручной откат с помощью Liquibase

Для ручного отката изменений БД необходимо наличие последнего дистрибутива и утилиты Liquibase. С принципами отката с помощью Liquibase можно ознакомиться в официальной документации утилиты.

Часто встречающиеся проблемы и пути их устранения

Проблемы модуля dataspace-meta

При разворачивании модуля dataspace-meta при проверке статуса следующая информация:

```
NAME                                    READY   STATUS             RESTARTS
AGE
ds-meta-74cdcdfc4b-9jthb             0/2     ImagePullBackOff   0          21h
```

В этом случае требуется проверить:

- доступность image registry с кластера kubernetes;
- аутентификацию в image registry на основе созданного secret на шаге “Создание pull image secret” (см. раздел Настройка namespace);
- корректность указания в deployment imagePullSecrets.

При разворачивании модуля dataspace-core-meta при проверке статуса следующая информация:

```
NAME                                    READY   STATUS             RESTARTS
AGEds-core-meta-74cdcdfc4b-9jthb            1/2     CrashLoopBackOff   0
21h
```

В этом случае требуется получить логи с пода с помощью команды в консоли:

```
kubectl  logs ds-core-meta-74cdcdfc4b-9jthb -c dataspace-core-meta -n <namespace
>
```

Логи передать линии поддержки.

При разворачивании модуля dataspace-core-meta при проверке статуса следующая информация:

```
NAME                                    READY   STATUS             RESTARTS
AGE
ds-meta-75564d9d6c-h9j49                0/2     ImagePullBackOff   1
21h
```

В этом случае требуется проверить:

- доступность image registry с кластера Kubernetes;
- аутентификацию в image registry на основе созданного secret на шаге “Создание pull image secret” (см. раздел Настройка namespace);
- корректность указания в deployment imagePullSecrets.

Чек-лист валидации установки

Проверка успешной установки модуля dataspace-core-meta

Для проверки успешности установки модуля выполнить следующую команду в консоли:

```
kubectl get pod -n <namespace>
```

Пример вывода в консоли:

```
NAME                                    READY   STATUS             RESTARTS
AGE
deploy-dataspace-nginx-695b8dbbbb-p8wtd   0/1     CrashLoopBackOff   9847
34d
ds-core-meta-74cdcdfc4b-9jthb           2/2     Running            0
21h
ds-core-meta-74cdcdfc4b-flhzq           2/2     Running            0
21h
ds-meta-75564d9d6c-h9j49                2/2     Running            1
21h
ds-meta-75564d9d6c-kkr6x                2/2     Running            0
21h
```

В списке выведенных pods найти ds-core-meta и убедиться, что статус поднятых pods – `STATUS=Running`.

Проверка успешной установки модуля dataspace-meta

Для проверки успешности установки модуля выполнить следующую команду в консоли:

```
kubectl get pod -n <namespace>
```

Пример вывода в консоли:

```
NAME                                    READY   STATUS             RESTARTS
AGE
deploy-dataspace-nginx-695b8dbbbb-p8wtd   0/1     CrashLoopBackOff   9847
```

```
34d
ds-core-meta-74cdcdfc4b-9jthb             2/2     Running            0
21h
ds-core-meta-74cdcdfc4b-flhzq             2/2     Running            0
21h
ds-meta-75564d9d6c-h9j49                  2/2     Running            1
21h
ds-meta-75564d9d6c-kkr6x                  2/2     Running            0
21h
```

В списке выведенных pods найти ds-meta и убедиться, что статус поднятых pods – `STATUS=Running`.