Общество с ограниченной ответственностью «ГНР ГРУПП»
(ООО «ГНР ГРУПП»)

КОРПОРАТИВНАЯ СИСТЕМА УПРАВЛЕНИЯ И ОБМЕНА ДАННЫМИ. ПЛАТФОРМА – ИНТЕГРАЦИОННАЯ ШИНА (КСУОД.ПЛАТФОРМА-ИШ)
УСТАНОВКА И НАСТРОЙКА

РУКОВОДСТВО АДМИНИСТРАТОРА

Листов 68

2025

Аннотация

В настоящем документе приведено описание процессов установки и настройки системы «Корпоративная система управления и обмена данными. ПЛАТФОРМА – Интеграционная шина» (далее – Система).

Документ приводит сведения о требованиях к инфраструктуре, на которой устанавливается Система, к обслуживающему персоналу. Приведены сведения о процессах установки и настройки программного обеспечения, сведения о технической поддержке пользователей.

Оглавление

1 Нормативные ссылки

В настоящем документе приведены ссылки на следующие нормативные документы:

1 Корпоративная система управления и обмена данными. ПЛАТФОРМА – Интеграционная шина (КСУОД.ПЛАТФОРМА-ИШ). Описание программного обеспечения

2 **Ошибка! Источник ссылки не найден.**

2 Термины, определения и сокращения

В настоящем документе применяются следующие сокращения, термины и определения:

PLM	–	Управление жизненным циклом продукции (сокр. от Product Lifecycle Management)
TC	–	Teamcenter – PLM-система разработки Siemens PLM Software
БД	–	База данных
КБД	–	Консолидированная база данных
НСИ	–	Нормативно-справочная информация
ПО	–	Программное обеспечение
Система	–	Корпоративная система управления и обмена данными. ПЛАТФОРМА – Интеграционная шина
СУБД	–	Система управления базами данных

3 Общие сведения

3.1 Система предназначена для интеграции в единую среду предприятия систем автоматизации производства, а, также, миграции исторических конструкторско-технологических данных (далее – КТД) из замещаемых систем.

3.2 Система обеспечивает:

- обмен данными между различными информационными системами в единой среде предприятия;
- возможность бесшовной интеграции новых компонентов промышленных систем, обеспечивающих автоматизацию и роботизацию предприятия (систем классов WMS, LMS, MES, ERP);
- миграцию КТД от зарубежных систем автоматизации производства (замещаемых систем) в целевые системы.

3.3 Система имеет возможность при необходимости гибко подключать новые функции, а, также, модернизировать существующие.

3.4 Система позволяет интегрировать сформированные во внешних системах конструкторско-технологические данные и в дальнейшем полнофункционально работать с ними.

3.5 Функции, реализуемые Системой, объединяются в две группы:

- Группа функций «Обеспечение доступа к функциональности Системы».
- Группа функций «Интеграция и миграция».

3.6 Группа функций «Обеспечение доступа к функциональности Системы» включает следующие функции:

- управление пользователями Системы;
- контроль доступа к Системе;
- защита от несанкционированного доступа.

3.7 Указанные функции реализованы следующими программными модулями (Сервисами):

- авторизации;
- управления пользователями;
- управления правами доступа;
- логирования;
- обеспечивающими пользователю доступ к функциональности Системы.

3.8 Группа функций «Интеграция и миграция» включает следующие функции:

– экспорт КТД из внешней системы;
– хранение КТД и управление ими;
– обработка поступивших КТД к формату, поддерживаемому целевой системой;
– импорт обработанных КТД в целевую систему
– протоколирование процессов интеграции и/или миграции КТД.

3.9 Указанные функции реализованы следующими программными модулями:

– экспорт;
– интеграционная шина;
– импорт.

3.10 Система включает:

– «Сервисы» – реализуют группу функций «Обеспечение доступа к функциональности Системы».
– «Экспорт» – предназначен для обеспечения взаимодействия «Интеграционной шины» с внешней замещаемой системой. Решает задачу экспорта КТД из внешней системы.
– «Интеграционная шина» – предназначена для обеспечения в рамках единой системы интеграции с внешними системами и миграции данных из замещаемой системы в целевую и в собственный информационный ресурс, который ведется в КБД. Решает задачи обработки, хранения и управления КТД, обеспечивает задачу протоколирования процессов интеграции и/или миграции;
– «Импорт» – предназначен для обеспечения взаимодействия между «Интеграционной шиной» и целевой системой. Решает задачу импорта КТД во внешнюю систему.

3.11 Структурная схема Системы приведена на рисунке **Ошибка! Источник ссылки не найден.**.

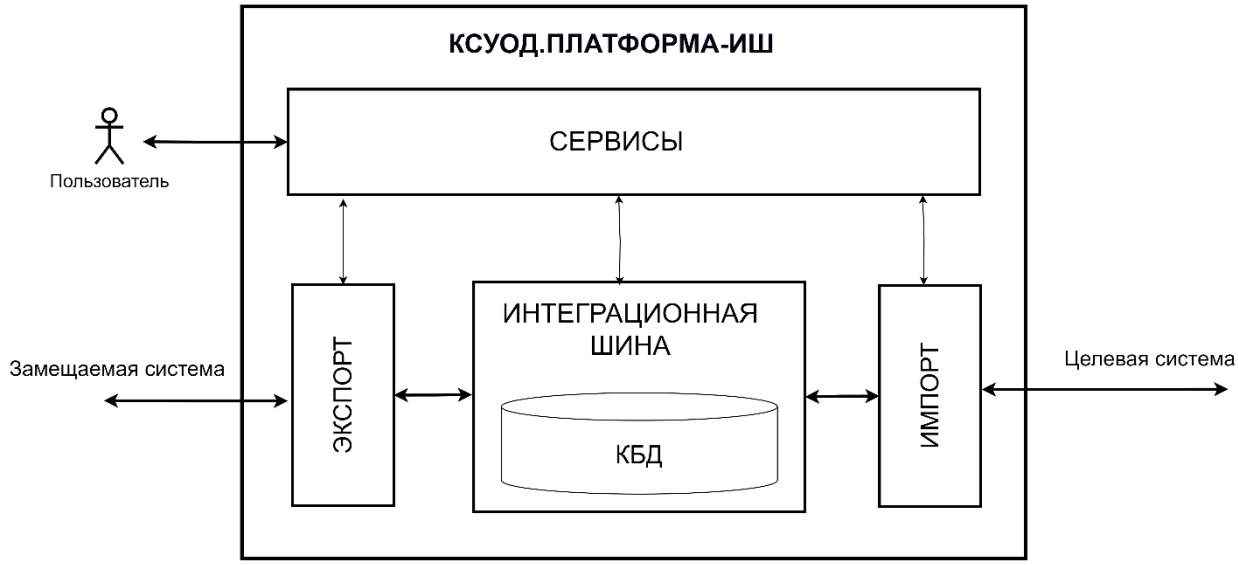


Рисунок 1. – Структурная схема Системы

3.12 Описание Системы приведено в документе «Корпоративная система управления и обмена данными. ПЛАТФОРМА – Интеграционная шина (КСУОД.ПЛАТФОРМА-ИШ). Описание программного обеспечения».

3.13 Система обладает возможностями для расширения своей функциональности путем подключения дополнительных модулей. Модули могут быть разработаны на любых.NET-совместимых языках программирования: C#, C++, J#, JScript, Delphi, Visual Basic и пр.

3.14 Система обеспечивает информационное взаимодействие с другими информационными системами предприятия, используя стандартные механизмы и протоколы взаимодействия приложений (.NET Remoting, COM, SOAP, XML и пр.).

3.15 Процесс развертывания Системы представляет собой последовательность действий, которая обеспечивает инсталляцию программного обеспечения, его настройку и делает Систему доступной для последующего использования.

4 Требования к обслуживающему персоналу

4.1 К установке, настройке и обслуживанию Системы допускаются лица, ознакомившиеся с эксплуатационной документацией на Систему, документацией на поставляемое программное и аппаратное обеспечение и имеющие практические навыки работы с соответствующим программным и аппаратным обеспечением.

4.2 Обслуживающий персонал, допущенный к установке, настройке и обслуживанию Системы, должен иметь права доступа «администратор».

4.3 Перед началом работ администратору необходимо ознакомиться с данным документом, а также с документом «Корпоративная система управления и обмена данными. ПЛАТФОРМА – Интеграционная шина (КСУОД.ПЛАТФОРМА-ИШ). Описание программного обеспечения».

4.4 Основные требования, предъявляемые к администратору:

– высшее специальное (допускается среднее специальное) образование;
– навыки работы с операционными системами (далее – ОС) семейства Windows и Linux;
– знание и понимание принципов функционирования ПО, реализованного в микросервисной архитектуре;
– навыки работы с контейнеризированными приложениями.

4.5 Функции и обязанности администратора Системы:

– развертывание системного и прикладного ПО на предоставленной инфраструктуре;
– администрирование системного обеспечения:
  – администрирование ОС;
  – администрирование систем управления базами данных (далее – СУБД);
  – администрирование системы развертывания и управления контейнеризированными приложениями ***Kubernetes***.
– устранение неполадок в работе ПО;
– обслуживание ПО.

4.6 Также в обязанности администратора входит:

– обеспечение бесперебойной работы ПО;

– оперативное выявление неисправностей технических средств и их устранение;
– техническое обслуживание комплекса технических средств в соответствии с соответствующими инструкциями;
– выполнение правил технической эксплуатации, правил по технике безопасности и требований должностных инструкций.

5 Требования к инфраструктуре

5.1 Требования к аппаратной инфраструктуре

5.1.1 Система размещается в инфраструктуре, предоставляемой Заказчиком.

5.1.2 Требуемая схема аппаратной инфраструктуры, на которой разворачивается Система, приведена на рисунке 2.

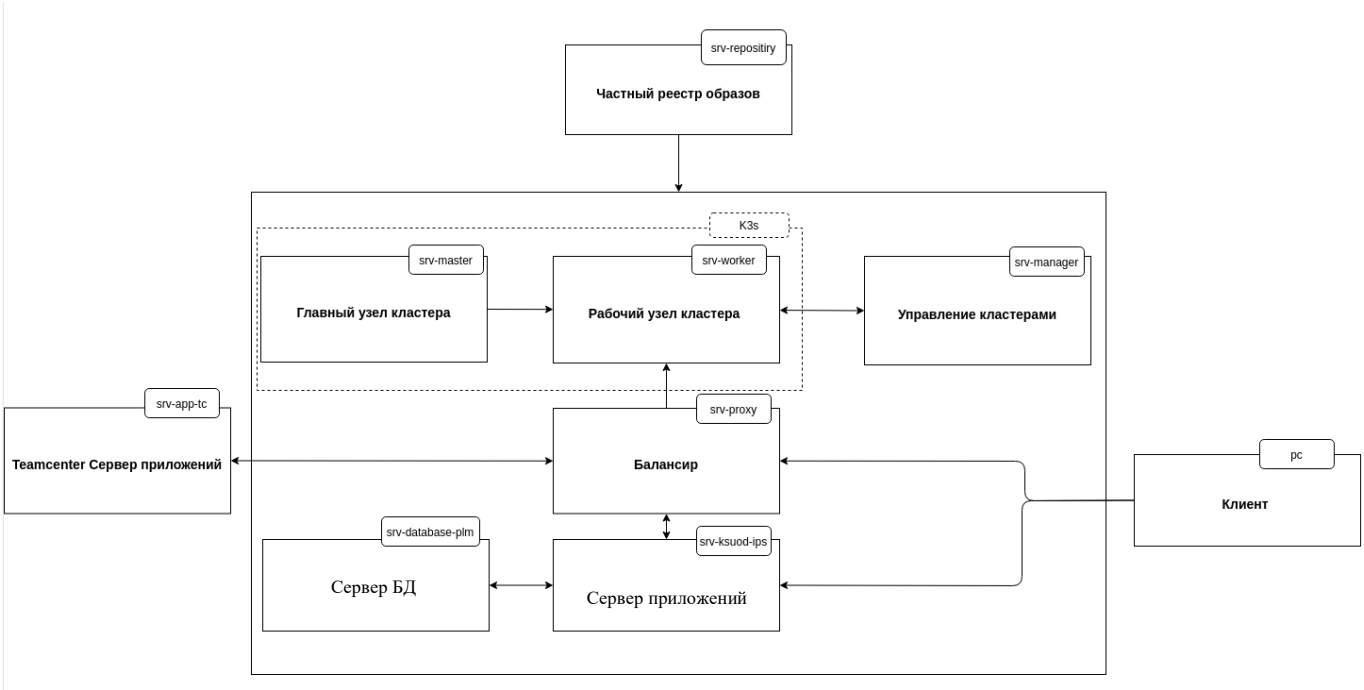


Рисунок 2. – Схема аппаратной инфраструктуры
для размещения Системы

5.1.3 Серверы СУБД и приложений должны находиться в одной сети, между ними должен быть разрешён весь сетевой трафик.

5.1.4 На всех серверах должны быть либо сконфигурированы статические IP-адреса, либо, в случае динамической выдачи IP-адресов DHCP-сервером, эти IP-адреса должны быть зарезервированы.

5.1.5 Для развертывания реестра образов ***Docker*** необходимо дисковое пространство не менее 250 ГБ, к которому должен быть организован доступ по протоколу https.

5.1.6 Должно быть развернуто минимум два рабочих узла ***Kubernetes*** (Worker). Требования к каждому узлу: 8 ядер CPU, 32 ГБ RAM, диск 100 ГБ (для узла приложений), 200 ГБ (для узла баз данных).

Примечания: 1. Рабочий узел (нод) ***Kubernetes*** *– физический или виртуальный компьютер, на котором разворачивается* ***Kubernetes****.*

2. Желательно выделить отдельный рабочий узел для размещения серверов БД с объёмом дисков, рассчитанным на объём баз данных.

5.1.7 Необходимо также развернуть узлы ***Kubernetes*** для ETCD и Control Plane. Требования к каждому узлу: 4 ядра CPU, 8 ГБ RAM, диск 50 ГБ 1000 IOPS (SSD).

5.1.8 Должен быть развернут сервер ***Rancher***. Требования: 4 ядра CPU, 8 ГБ RAM, диск 40 ГБ 1000 IOPS (SSD).

5.1.9 Должен быть развернут балансировщик нагрузки.

5.1.10 Должен быть развернут DNS-сервер.

5.1.11 Должна быть настроена синхронизация времени (например, NTP).

5.1.12 Должен быть развернут сервер Git с доступом по протоколу https.

5.2 Требования к программному обеспечению

5.2.1 Для функционирования Системы необходимо установить и настроить ряд программных продуктов, которые обеспечивают надежную, масштабируемую и безопасную работу Системы.

5.2.2 Перед выполнением установки и настройки Системы рекомендуется предварительно ознакомиться со следующими технологиями, инструментами и спецификациями:

Kubernetes (https://*Kubernetes*.io/docs/home/)
Rancher (https://*Rancher*manager.docs.*Rancher*.com/v2.10?_gl=1)
Fleet (https://fleet.*Rancher*.io/0.11)
Kustomize (https://kubectl.docs.*Kubernetes*.io/references/kustomize/)
Git (https://git-scm.com/do)
Sealed Secrets (https://github.com/bitnami-labs/sealed-secrets)
JSONPath (https://www.ietf.org/archive/id/draft-goessner-dispatch-jsonpath-00.html)

5.2.3 Требования, предъявляемые к инфраструктурным программным средствам:

- Операционная система Astra Linux (ядро 6.0 и выше, версия 1.7.5).
- Платформа для оркестрации контейнеров: ***Kubernetes*** (v1.31.7) – для управления приложениями в распределенной среде, масштабируемости и автоматизации.
- Система управления аутентификацией и авторизацией: Keycloak (v22.0.4) – обеспечивает безопасность и управление доступом пользователей через единую точку входа.
- Хранилища данных:

  - СУБД MongoDB (v8.0.3) – документно-ориентированная база данных для хранения неструктурированных данных.
  - СУБД PostgreSQL (v15) – реляционная база данных.
  - Объектная система хранения MinIO.
- Платформа для управления ***Kubernetes***-кластерами: ***Rancher*** Server (v2.10.3).
- Контейнеризатор приложений: ***Docker*** (v24.0.2) – программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации.
- Программная платформа Java JRE (Temurin-11.0.22) Eclips.
- Технические средства компиляции – JDK (Temurin-11.0.22).

6 Установка и настройка Системы

6.1 Общие сведения

6.1.1 Система развёртывается в виде ресурсов ***Kubernetes*** в кластере под управлением платформы ***Rancher*** с помощью системы ***Fleet***, работающей по методике ***GitOps***. При этом обеспечивается автоматическое развёртывание ресурсов на основе их описаний (манифестов), находящихся в репозитории манифестов на сервере ***Git***.

6.1.2 Платформа ***GitOps*** отслеживает изменения в указанном репозитории, синхронизирует их с кластером и автоматически применяет новые или изменённые манифесты. При этом образы контейнеров ***Docker*** находятся в частом реестре образов и устанавливаются управляющей системой кластера автоматически.

6.1.3 Перед установкой Системы необходимо выполнить настройку:

– рабочих узлов;
– портов;
– репозитория манифестов.

6.1.4 Далее необходимо развернуть и настроить:

– реестр образов ***Docker***;
– платформу ***Kubernetes (K3s)***;
– платформу ***Rancher***;
– кластер ***Kubernetes***;
– контроллер ***Sealed Secrets;***
– утилиту ***Kubeseal***;
– сервер системы управления версиями ***Git***, доступный по сети для узлов ***Rancher***.

6.1.5 В комплект установочных материалов входят:

– файлы приложений (в каталоге *k8s-manifests*);
– файлы образов Системы (в каталоге *images*);
– скрипт развертывания объектов базы данных PostgreSQL (*initdb/postgresql-init.sql*).
– архив, с которого производится загрузка образов в частный реестр ***Docker***.

6.2 Настройка рабочих узлов

6.2.1 На всех узлах необходимо:

– Установить уровень безопасности «Орел»;
– Настроить статический IP;

– Настроить разрешение DNS;
– Отключить Firewall;
– Настроить маршрут по умолчанию.

6.2.2 Для доступа к реестру ***Docker*** следует установить сертификаты SSL (TLS) следующими командами:

```
sh
sudo cp ~/<сертификат>.pem /etc/ssl/certs/
sudo update-ca-certificates --fresh
```

6.2.3 На узлах ***Kubernetes*** (ETCD, Control Plane, Worker, ***Rancher***) необходимо выполнить команды:

```
sh
openssl s_client -showcerts -connect Docker-plm.ru:443 < /dev/null
| awk '/-----BEGIN CERTIFICATE-----/{n++} n==3{print; if (/-----
END CERTIFICATE-----/) exit}' > ~/globalsign-ca.crt.bak

sh
sudo cp ./globalsign-ca.crt /usr/local/share/ca-certificates/Docker-
plm-ca.crt

sh
sudo update-ca-certificates --fresh
```

6.2.4 Установить ***Docker*** и ***Docker Compose*** командами:

```
sh
sudo apt install Docker.io Docker-compose
sudo usermod -aG Docker ${USER}
```

6.2.5 Перелогиниться в текущей сессии, (чтобы *Docker*-команды работали без *sudo*) следующим образом:

```
sh
logout
```

6.2.6 Выполнить вход в частный реестр для каждого узла кластера (ETCD, Control Plane, Worker):

```
sh
Docker login Docker-plm.ru
```

6.2.7 Ввести логин и пароль для подключения к реестру.

6.3 Настройка портов

6.3.1 Порт 6443 узла сервера ***Rancher*** должен быть доступен всем узлам.

6.3.2 Все узлы должны быть доступны друг другу по протоколу UDP и порту 8472. Данный порт используется модулем CNI (Container Network Interface) Flannel и Canal.

6.3.3 Если используется сервер метрик, нужно на всех узлах открыть порт 10250.

6.3.4 Проверить firewall:

```
sh
sudo ufw status
```

Если команда выводит: *Status: active,* требуется выполнить команду

```
sh
sudo ufw disable
```

6.3.5 Для того чтобы открыть порты нужно отредактировать файл настроек *iptables*:

```
sh
sudo nano /var/lib/iptables/rules-save
```

6.3.6 В редакторе прописать правило для открытия нужного порта. Пример файла `*rules-save*` для узла калстера ***Kubernetes:***

```
*filter
:INPUT DROP [24:1332]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
:ACL-SSH - [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j
ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACL-SSH
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10050 -m comment --comment
"Accept zabbix connections." -j ACCEPT
-A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 1337 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 2376 -j ACCEPT
```

-A INPUT -p tcp -m tcp --dport 2379 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 2380 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 6443 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 6444 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 8443 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 9099 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 9443 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 9796 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10248 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10250 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10254 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10256 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10257 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10259 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10010 -j ACCEPT
-A INPUT -p udp -m udp --dport 4789 -j ACCEPT
-A INPUT -p udp -m udp --dport 8472 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 6443 -j ACCEPT
-A OUTPUT -p tcp -m tcp --dport 6444 -j ACCEPT
-A OUTPUT -j ACCEPT
-A ACL-SSH -s 10.0.0.0/8 -j ACCEPT
-A ACL-SSH -j DROP
COMMIT

Пример файла `*rules-save*` для узла сервера ***Rancher***:

**filter*
:INPUT DROP [24:1332]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
:ACL-SSH - [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACL-SSH
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10050 -m comment --comment "Accept zabbix connections." -j ACCEPT

-A INPUT -p tcp --dport 443 -j ACCEPT
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 6443 -j ACCEPT
-A INPUT -p udp --dport 8472 -j ACCEPT
-A INPUT -p tcp --dport 2379 -j ACCEPT
-A INPUT -p tcp --dport 2380 -j ACCEPT
-A INPUT -p tcp --dport 6444 -j ACCEPT
-A INPUT -p tcp --dport 10010 -j ACCEPT
-A INPUT -p tcp --dport 8443 -j ACCEPT
-A INPUT -p tcp --dport 9099 -j ACCEPT
-A INPUT -p tcp --dport 10250 -j ACCEPT
-A INPUT -p tcp --dport 22 -j ACCEPT
-A OUTPUT -j ACCEPT
-A ACL-SSH -s 10.0.0.0/8 -j ACCEPT
-A ACL-SSH -j DROP
COMMIT

6.3.7 Сохранить изменения. Затем выполнить команду:

sh
sudo iptables-restore < /var/lib/iptables/rules-save

6.3.8 Входящие правила для узлов ***Rancher Server*** приведены в **Ошибка! Источник ссылки не найден.**, исходящие – в Таблица **2**.

Таблица 1 – Входящие правила для узлов ***Rancher*** *Server*

Протокол	***Порт***	***Источник***	***Описание***
TCP	80	Балансировщик/прокси, выполняющий терминацию SSL	Пользовательский интерфейс и API, если используется внешняя терминация SSL
TCP	443	Узлы сервера, агентов, кластеров. Любой источник, который должен иметь доступ к интерфейсу или API ***Rancher***	Агент *Rancher*, UI/API *Rancher*, `kubectl`
TCP	6443	Узлы сервера K3s	*Kubernetes API*
UDP	8472	Узлы сервера и агентов ***Rancher***	Если используется Flannel или Canal CNI
TCP	10250	Узлы сервера и агентов ***Rancher***	`kubelet`

Таблица 2 – Исходящие правила для узлов ***Rancher Server***

Протокол	***Порт***	***Назначение***	***Описание***
TCP	443	git.*Rancher*.io	*Rancher* catalog
TCP	6443	*Kubernetes* API	Сервер *Kubernetes* API

6.3.9 Порты узлов сервера ***Rancher Server***

Для трафика между узлами ***Rancher Server*** используются порты в соответствии с **Ошибка! Источник ссылки не найден.** 3.

Таблица 3. – Порты ***Rancher Server***

Протокол	***Порт***	***Описание***
TCP	443	Агенты ***Rancher***
TCP	10010	Container Runtime Interface
TCP	6444	Порт для работы внутреннего API
TCP	2379	Запросы клиентов ETCD
TCP	2380	Одноранговая коммуникация ETCD
TCP	6443	Сервер ***Kubernetes*** API
TCP	8443	Nginx Ingress's Validating Webhook
UDP	8472	Сетевой слой Canal/Flannel VXLAN
TCP	9099	Зондирование готовности/ функционирования Canal/Flannel
TCP	10250	Коммуникация сервера метрик со всеми узлами

6.3.10 Узлы кластера ***Kubernetes***

TCP 443: от узлов кластера к узлам ***Rancher***

TCP 22, 2376: от узлов ***Rancher*** к узлам кластера ***Kubernetes***

6.3.11 Порты, открываемые на узлах ***Kubernetes***, приведены в таблице 4.

Таблица 4. – Порты, открываемые на узлах ***Kubernetes***

Протокол	***Порт***	***Описание***
TCP	179	Calico BGP (если используется Calico CNI)
TCP	2376	Node driver Docker daemon TLS
TCP	2379	etcd client requests
TCP	2380	etcd peer communication
UDP	8472	Canal/Flannel VXLAN overlay networking
UDP	4789	Flannel VXLAN overlay networking on Windows cluster
TCP	8443	Rancher webhook

TCP	9099	Зондирование готовности/функционирования Canal/Flannel
TCP	9443	Rancher webhook
TCP	9796	Стандартный порт мониторинга для node-exporters
TCP	6783	Weave (если используется Weave CNI)
UDP	6783-6784	Weave UDP (если используется Weave CNI)
TCP	10250	Коммуникация сервера метрик со всеми узлами
TCP	10254	Зондирование готовности/функционирования контоллера Ingress
TCP	10257	kube-controller-manager - self
TCP	10259	kube-scheduler - self
TCP	10256	kube-proxy
TCP	10248	kubelet healthz endpoint
TCP/UDP	30000-32767	Порты сервисов типа NodePort (открывать по мере использования)

6.4 Настройка репозитория манифестов

6.4.1 Автоматическое развёртывание ресурсов инфраструктуры и приложений обеспечивается репозиторием манифестов. Манифест – специальный файл, содержащий описание ресурсов.

6.4.2 Репозиторий манифестов ***ksuod-ci*** используется системой ***Fleet*** для автоматического развёртывания в кластере ресурсов инфраструктуры и приложений. Он содержит:

- манифесты ***Kubernetes***;
- манифесты преобразования ***Kustomize***;
- пакеты ***Helm***;
- конфигурационные файлы ***Fleet***;
- конфигурационные файлы Системы.

6.4.3 Репозиторий ***ksuod-ci*** должен быть опубликован на сервере ***Git***, к которому имеется прямой сетевой доступ у сервера платформы ***Rancher***, управляющего кластером ***Kubernetes*** используемым для развёртывания ресурсов Системы.

6.4.4 Манифесты в репозитории разбиты на подкаталоги, каждый из которых представляет собой пакет ***Fleet*** (***Fleet Bundle***) – пакет ресурсов, развёртываемых одновременно. Каждый пакет может иметь свой приоритет, что позволяет развёртывать группы ресурсов в определённом порядке. Приоритеты развертывания пакетов приведены в таблице 5.

Таблица 5. – Пакеты *Fleet* в порядке приоритета развёртывания

№	Наименование	Приоритет	Описание
1	namespaces	1	Определения пространств имён и прочие глобальные настройки кластера.
2	volumes	1	Определения постоянных томов (Persistence Volume)
3	secrets	2	Определения ресурсов Sealed Secret для логинов, паролей и других чувствительных данных
4	operators	3	Операторы *Kubernetes* для установки СУБД, брокеров сообщений и прочих компонентов инфраструктуры
5	infrastructure	4	Манифесты CRD (Custom Resource Definition) для создания ресурсов, управляемых операторами, установленными пакетом operators – баз данных, очередей сообщений, учётных записей пользователей и прочих компонентов инфраструктуры.
6	init-db	5	Определения задач (Job) для выполнения скриптов, которые должны выполняться после развёртывания пакета infrastructure
7	apps	6	Развёртывание и конфигурация Системы

6.4.5 Репозиторий может описывать ресурсы для нескольких управляемых кластеров. Манифесты ресурсов разбиты на базовые шаблоны, общие для всех кластеров (*base*), и наборы переопределений (*overlay*), которые изменяют базовые шаблоны индивидуально для каждого кластера или пространства имён в кластере для конкретной среды выполнения.

6.4.6 Каталог пакета ***Fleet***

6.4.6.1 Общая структура каталога пакета ***Fleet*** (***Fleet Bundle***) следующая:

- base\: манифесты в формате утилиты ***Kustomize***, которые описывают общие конфигурации ресурсов для всех кластеров.

- ***overlays\cluster-N***: манифесты переопределений для кластера ***cluster-N***.

- ***fleet.yml***: файл конфигурации системы ***Fleet***.

6.4.6.2 Файл ***fleet.yml*** содержит настройки всех кластеров, в которые должны устанавливаться ресурсы, описанные в манифестах ***Kustomize***, находящихся в том же каталоге и его подкаталогах.

6.4.6.3 Каждый каталог, в котором находится файл ***fleet.yml***, является пакетом ***Fleet*** (***Fleet Bundle***).

6.4.6.4 Файл ***fleet.yml*** для ***Kustomize*** имеет следующий формат:

```
labels:
        priority: "priority-2"
dependsOn:
            - selector:
                matchLabels:
                        priority: "priority-1"
targetCustomizations:
        - name: cluster-n-bundle
                clusterSelector:
                        matchLabels:
                                management.cattle.io/cluster-name: c-r8rkq
                kustomize:
                        dir:./overlays/cluster-N
```

где:

labels.priority: приоритет развёртывания (сортируется по алфавиту).

dependsOn[0].selector.matchLabels.priority: приоритеты пакетов, после развёртывания которых развёртывается данный пакет.

targetCustomizations: массив описаний управляемых кластеров.

name: наименование управляемого кластера – должно быть уникальным в пределах массива *targetCustomizations*.

clusterSelector.matchLabels.management.cattle.io/cluster-name: идентификатор кластера в платформе ***Rancher*** – значение метки *management.cattle.io/cluster-name* управляемого кластера (можно посмотреть в конфигурации кластеров в разделе ***Continuous Delivery***/***Clusters*** интерфейса ***Rancher***).

kustomize.dir: каталог, в котором находится корневой манифест ***kustomization.yaml*** утилиты ***Kustomize***. Задается относительно файла ***fleet.yaml***.

6.4.6.5 Некоторые пакеты ***Fleet*** развёртываются через менеджер пакетов ***Helm***. В этом случае каждый каталог пакета ***Helm*** (***Helm Chart***) является отдельным пакетом (bundle) ***Fleet***.

6.4.6.6 Формат файла ***fleet.yml*** для ***Helm*** следующий:

```
labels:
    priority: "priority-3"
dependsOn:
    - selector:
        matchLabels:
            priority: "priority-2"
targetCustomizations:
    - name: cluster-n-bundle-helm-n
        clusterSelector:
            matchLabels:
             management.cattle.io/cluster-name: c-r8rkq
        defaultNamespace: kafka
        helm:
            releaseName: strimzi-cluster-operator
            chart: ""
            values:
                defaultImageRegistry: registry.suod.local
                defaultImageRepository: quay.io/strimzi
                image:
                imagePullSecrets:
        - name: regcred
```

где:

labels и *dependsOn* - аналогичны тем, которые используются для типа ***Kustomize***.

targetCustomizations: Настройка пакета индивидуально для каждого кластера:

name: наименование пакета ***Fleet***, уникальное в пределах массива *targetCustomizations*.

clusterSelector.matchLabels.management.cattle.io/cluster-name: идентификатор кластера в платформе ***Rancher*** (см. выше).

defaultNamespace: пространство имён ***Kubernetes***, в которое будет устанавливаться пакет ***Helm***.

helm:

releaseName: наименование пакета ***Helm***

chart: "" (константа)

values: параметры конфигурации пакета ***Helm***.

6.4.7 Настройка кластера в репозитории ***ksuod-ci***

6.4.7.1 Общая процедура настройки кластера

Для развёртывания ресурсов на кластере, нужно добавить манифесты и настройки в репозиторий ***ksuod-ci*** следующим образом:

– В каждый из каталогов пакета ***Fleet*** в подкаталог *overlays*/ создать подкаталог с условным наименованием кластера, например, чтобы добавить кластер *cluster-n* в пакет *namespaces*:

```
namespaces
├── base
└── overlays
    ├── ksuodkk
    └── cluster-n
```

– Добавить в новый каталог подкаталоги и файлы ресурсов: скопировать эти настройки из подкаталога *overlays/_template/* этого же пакета (или подкаталога *overlays*/ другого кластера, если он содержит подходящие переопределения базовых ресурсов, чтобы использовать его в качестве шаблона). Отредактировать настройки (в соответствии с п.5.4.9).

– В файл ***fleet.yml*** пакета ***Fleet*** добавить секцию *targetCustomizations*. Например, для добавления кластера *cluster*-n в пакете *namespaces*:

```
- name: cluster-n-namespaces
    clusterSelector:
        matchLabels:
            management.cattle.io/cluster-name: c-a2u17
    kustomize:
        dir:./overlays/cluster-n
```

где:

cluster-n-*namespaces* – наименование пакета

c-a2u17 – идентификатор кластера в ***Rancher***.

*Примечание: Чтобы узнать идентификатор кластера в **Rancher**, нужно в интерфейсе **Rancher** открыть раздел Continuous Delivery > Clusters > [cluster-n] > Configuration, – идентификатор кластера находится поле Name.*

./overlays/cluster-n – путь к подкаталогу конфигурации *Kustomize* нового кластера.

– Выполнить команды *commit* и *push* системы ***Git*** – после этого изменения будут обнаружены системой ***Fleet*** и применены к соответствующему кластеру ***Kubernetes***.

6.4.8 Редактирование настроек кластера (применительно к пакетам Fleet):

6.4.8.1 **namespaces**

1. Добавить в *namespaces/fleet.yml#/targetCustomizations* (здесь и далее по тексту для обозначения полей в файле YAML используется синтаксис JSONPath)):

- name: <имя_кластера>-namespaces # Уникальное имя пакета
clusterSelector:
matchLabels:
management.cattle.io/cluster-name: <id-кластера> # Имя кластера в ***Rancher***
kustomize:
dir:./overlays/<имя_кластера>

где:

name – уникальное имя пакета в пределах секции *clusterSelector.matchLabels.management.cattle.io/cluster-name:* имя кластера в разделе *Continuous Delivery > Git Repos* веб-интерфейса ***Rancher***.

kustomize.dir – относительный путь к каталогу *namespaces/overlays/<имя_кластера>*

2. Создать каталог *namespaces/overlays/<имя_кластера>* (можно использовать любое правильное имя каталога, однозначно обозначающее кластер или среду развёртывания ***Kubernetes***).

3. Скопировать в каталог *namespaces/overlays/<имя_кластера>* содержимое каталога *namespaces/overlays/_template* (либо каталога другого кластера, если он подходит в качестве шаблона).

6.4.8.2 **volumes**

1. Добавить в *volumes/fleet.yml#/targetCustomizations*:

- name: <имя_кластера>-volumes # Уникальное имя пакета
clusterSelector:
matchLabels:
management.cattle.io/cluster-name: id-кластера # Имя кластера в Rancher

kustomize:

dir:./overlays/<имя_кластера>

Значения полей см. в п. 5.4.9.1.

2. Создать каталог *volumes/overlays/<имя_кластера>*.

3. Скопировать в каталог *volumes/overlays/<имя_кластера>* содержимое каталога *volumes/overlays/_template*.

4. Отредактировать файлы *volumes/overlays/<имя_кластера>/worker-*.yaml*, – указать реальные имена узлов кластера в поле [0].*value*:

- op: replace

path:

/spec/nodeAffinity/required/nodeSelectorTerms/0/matchExpressions/0/values/0

value: worker-1 # Заменить на реальное имя узла

где *worker-1* – сетевое имя узла (значение метки *Kubernetes.io/hostname*) соответствующего узла *worker*.

При необходимости, добавить аналогичные файлы *worker-*.yaml* для каждого из узлов либо удалить лишние.

5. Настроить соответствие томов (*Persistent Volume*) узлам *Worker* кластера в *kustomization.yaml#/patches* в зависимости от расположения томов на узлах:

Тома располагаются на разных узлах: для каждого *patches*[*].*target.name* указать файл *worker-*.yaml*, соответствующий нужному узлу *worker*:

patches:

- target:

name: dev-mongo-logs-pv-0 # Имя тома

path: worker-1.yaml # Файл для узла Worker

Все тома располагаются на одном узле: оставить только фрагмент, применяющий ко всем ресурсам *PersistentVolume* файл *worker-*.yaml*, соответствующий нужному узлу:

patches:

- target:

kind: PersistentVolume # Применить ко всем ресурсам PersistentVolume

path: worker-1.yaml # Файл-патч для узла Worker

6. Создать каталоги томов, указанные в поле *spec.local*.path файлов *volumes/base/*.yaml* на тех узлах, на которых планируется разместить соответствующие тома:

/mnt/volumes/minio
/mnt/volumes/dev-mongo/data
/mnt/volumes/dev-mongo/log
/mnt/volumes/dev-pg-0
/mnt/volumes/dev-camunda
/mnt/volumes/dev-debezium
/mnt/volumes/dev-tcfms
/mnt/volumes/dev-tcfms-direct
/mnt/volumes/kafka-pv-volume-0

7. Назначить права (*chmod* 755) на все каталоги томов и назначить владельцев для томов с множественным доступом:

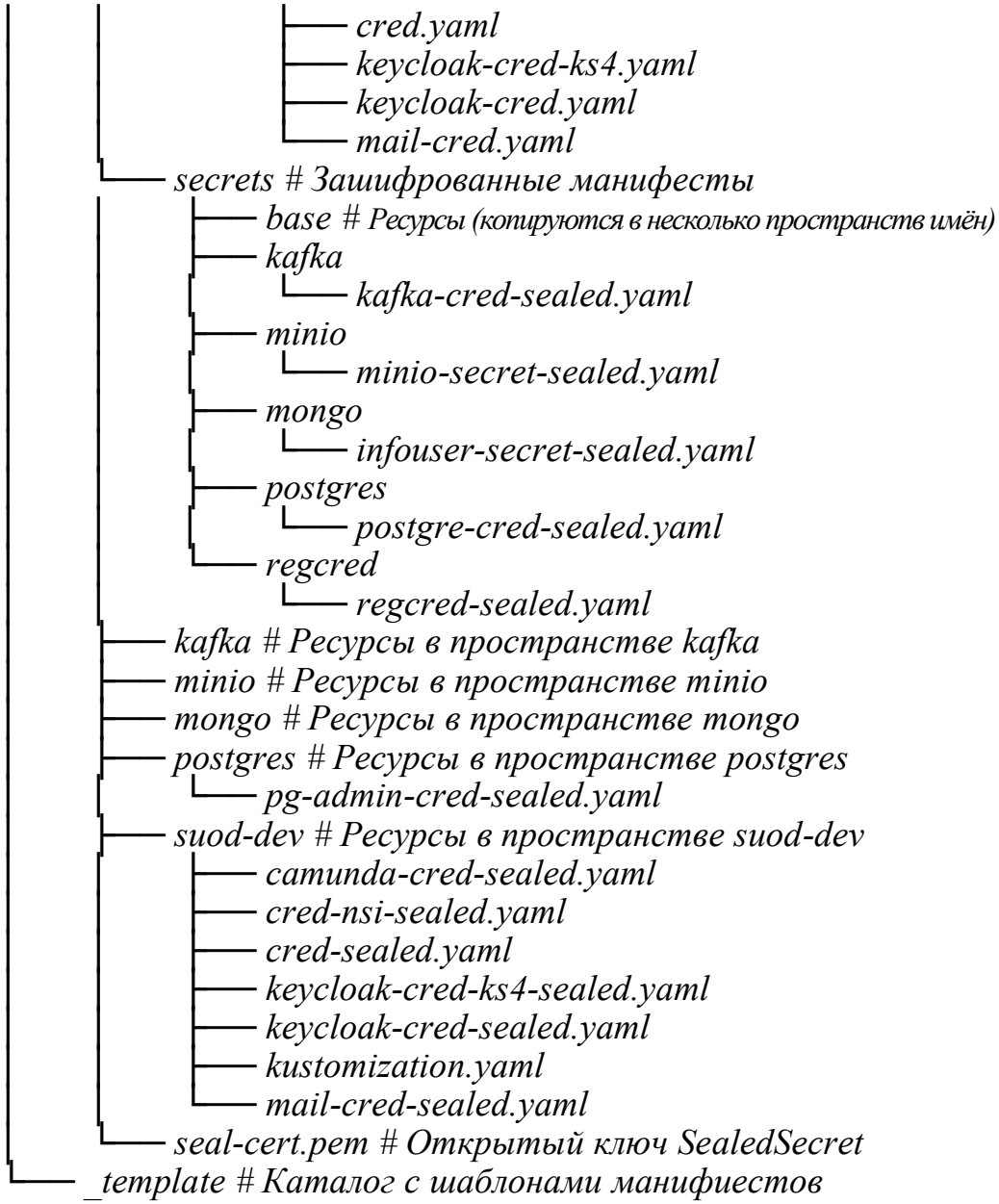
chown:2000 -R /mnt/volumes/dev-tcfms
chown:2000 -R /mnt/volumes/dev-tcfms-direct/
chown:101 -R /mnt/volumes/dev-camunda/

Внимание! *Создать каталоги томов на узлах роли Worker и назначить права доступа следует перед началом развёртывания.*

6.4.8.3 **secrets**

Структура каталогов пакета *secrets*:

```
secrets/
├── fleet.yml # Файл настройки пакета Fleet
├── seal.sh (.ps1) # Скрипт шифрования манифестов
└── overlays
    ├── cluster-n # Каталог манифестов кластера cluster-n
    │   ├── _unsealed # Незашифрованные (исходные) манифесты
    │   │   ├── base
    │   │   │   ├── kafka
    │   │   │   │   └── kafka-cred.yaml
    │   │   │   ├── minio
    │   │   │   │   └── minio-secret.yaml
    │   │   │   ├── mongo
    │   │   │   │   └── infouser-secret.yaml
    │   │   │   ├── postgres
    │   │   │   │   └── postgre-cred.yaml
    │   │   │   ├── regcred
    │   │   │   │   └── regcred.yaml
    │   │   │   │   ├── postgres
    │   │   │   └── pg-admin-cred.yaml
    │   │   └── suod-dev
    │   │       ├── camunda-cred.yaml
    │   │       └── cred-nsi.yaml
```



1. Добавить в *secrets/fleet.yml#/targetCustomizations*:

- name: <имя_кластера>-secrets # Уникальное имя пакета
clusterSelector:
matchLabels:
management.cattle.io/cluster-name: <id-кластера> # Имя кластера в ***Rancher***
kustomize:
dir:./overlays/<имя_кластера>

2. Создать каталог *secrets/overlays/<имя_кластера>*.

3. Скопировать в каталог *secrets/overlays/<имя_кластера>* содержимое каталога *secrets/overlays/_template*.

Внимание! Следует убедиться:

– Что каталог secrets/overlays/<имя_кластера>/_unsealed включён в.gitignore или.git/info/exclude для предотвращения отправки на сервер Git.

– Что доступ к этому каталогу закрыт для посторонних.

4. В каталог *secrets/overlays/<имя_кластера>/secrets/* поместить файл открытого ключа *seal-cert.pem,* выгруженный из контроллера инструмента ***Kubeseal*** командой (либо через веб-интерфейс ***Rancher***):

kubeseal --controller-name=sealed-secrets --controller-namespace=kubesystem --fetch-cert > seal-cert.pem

5. В файлах *secrets/overlays/<имя_кластера>/_unsealed/base//*.yaml* заменить слова *<change>* на реальные пароли:

1. *secrets/overlays/<имя_кластера>/_unsealed/base/regcred/regcred.yaml:*

stringData:

.Dockerconfigjson: >-

{

"auths": {

"<registry1>": {

"username": "<change>",

"password": "<change>"},

"<registry2>": {

"username": "<change>",

"password": "<change>"}}

}

Здесь *<registry1>*, *<registry2>* – адреса реестров ***Docker***, из которых извлекаются образы для ресурсов (если их несколько).

2. *secrets/overlays/<имя_кластера>/_unsealed/suod-dev/keycloak-cred.yaml:*

admin-password: пароль администратора домена *master* – суперпользователя *Keycloak*.

3. *secrets/overlays/<имя_кластера>/_unsealed/suod-dev/keycloak-cred-*.yaml*: Заполняется для каждого нового домена (*realm*) *Keycloak*.

Поля *admin-password*, *admin-user*, *client-secret, rsa-public-key*, – заполняются значениями из параметров нового домена Keycloak после развёртывания ресурсов кластера и создания нового домена в административной консоли приложения Keycloak.

В случае добавления дополнительного домена Keycloak, нужно добавить файл *keycloak-cred-*.yaml* в каталог *_unsealed/* и соответствующую строку с суффиксом - *sealed* в файл *kustomization.yaml* в каталоге *secrets/suod-dev*. Например, для *keycloak-cred-nsi.yaml*:

secrets/overlays/<имя_кластера>/_unsealed/suod-dev/keycloak-cred-nsi.yaml:

metadata:

name: keycloak-cred-nsi

secrets/overlays/<имя_кластера>/secrets/suod-dev/kustomization.yaml:

resources:

- keycloak-cred-nsi-sealed.yaml

4. *secrets/overlays/<имя_кластера>/_unsealed/suod-dev/cred*.yaml*: отдельный манифест для каждого подключения к серверу. Аналогично *keycloak-cred-*.yaml*, каждый новый файл добавляется с уникальным значением поля *metadata.name*, и в файл *kustomization.yaml* добавляется соответствующая строка с суффиксом *-sealed*.

5. *secrets/overlays/<имя_кластера>/_unsealed/base/kafka/kafka-cred.yaml.*

*Примечание: Заполняется после развёртывания ресурсов оператора Kafka Strimzi – то есть, нужно редактировать его уже после развёртывания всех ресурсов (статус можно проверить в разделе **Rancher** > Continuous Delivery > Git Repos > Bundles):*

1. Дождаться развёртывания пакета *ksuod-ci-operators*.

2. Заполнить *secrets/overlays/<имя_кластера>/_unsealed/base/kafka/kafkacred.yam*l данными из ресурсов в пространстве имён *kafka* (см. таблицу 6).

Таблица 6. – Данные из ресурсов в пространстве имен *kafka*

kafka-cred.yaml#/stringData	*Наименование Secret*	*Поле Secret*
admin-password	admin	password
user-password	user	password
truststore-password	kafka-cluster-cluster-ca-cert	ca.password

3. Скопировать файл сертификата.*data.ca.p12* из ресурса *kafka/kafka-clustercluster-ca-cert* в ресурс *suod-dev/kafka-truststore через kubectl*:

kubectl get secret kafka-cluster-cluster-ca-cert -n kafka -o jsonpath='{.data.ca\.p12}' | base64 -d > kafka-cluster-cluster-cacert.p12

kubectl create secret generic kafka-truststore -n suod-dev --fromfile=truststore.p12=kafka-cluster-cluster-ca-cert.p12

4. Перезапустить ресурсы *deployment*, использующие *Kafka*:
access-management-deployment
freecadapi-deployment
integration-logging-deployment
integration-logging-gateway-deployment
json-schema-deployment
object-script-service-deployment
product-data-import-deployment
product-data-import-deployment-tflex
product-data-integration-deployment
product-data-integration-deployment-tflex
users-integration-deployment
users-integration-deployment-ksuod
6. Выполнить шифрование открытым ключом:
1. Перейти в каталог *secrets/overlays/<имя_кластера>/*
2. Выполнить команду (на рабочей станции должна быть установлена клиентская утилита *kubeseal*):
для Linux:
../../seal.sh
для Windows (Powershell):
..\..\seal.ps1
При этом все файлы из подкаталога *_unsealed/* будут зашифрованы открытым ключом *secrets/seal-cert.pem* и зашифрованные копии будут перемещены в подкаталог secrets/ с сохранением структуры каталогов и суффиксом *-sealed* в имени файла.

*Примечание. Контроллер SealedSecret для каждого из зашифрованных манифестов типа секрет (файлы *-sealed.yaml) создаёт CRD типа SealedSecret с тем же именем и в том же пространстве, что и зашифрованный ресурс Secret. Затем он расшифровывает данные и создаёт (или обновляет) соответствующий ресурс Secret.*

В случае, если в созданный с помощью SealedSecret ресурс типа Secret вручную внесены изменения, они остаются там до тех пор, пока не будет обновлён (автоматически через Fleet либо вручную) одноимённый CRD SealedSecret.

6.4.8.4 **operators**

1. Добавить в operators/fleet.yml#/targetCustomizations:

- name: <имя_кластера>-operators # Уникальное имя пакета
clusterSelector:
matchLabels:
management.cattle.io/cluster-name: <id-кластера> # Имя кластера в ***Rancher***
kustomize:
dir:./overlays/<имя_кластера>

2. Создать каталог operators/overlays/<имя_кластера>

3. Скопировать в него содержимое operators/overlays/_template

4. Отредактировать файл operators/overlays/<имя_кластера> /kustomization.yaml:

configMapGenerator:
- name: env-config-operators
literals:
- IMAGEREPO=<change>

configMapGenerator.literals[0].IMAGEREPO: имя частного реестра образов и путь к образам инфраструктуры, например registry.suod.local/***Rancher***.

5. Отредактировать файл operators/helm/strimzi-kafka-operator/fleet.yml:

В секцию targetCustomizations добавить:

targetCustomizations:
- name: <имя_кластера>-operators-helm
clusterSelector:
matchLabels:
management.cattle.io/cluster-name: <id-кластера>
defaultNamespace: kafka
helm:
releaseName: strimzi-cluster-operator
chart: ""
values:
defaultImageRegistry: <частный_реестр_образов/путь>
defaultImageRepository: Rancher/quay.io/strimzi
image:
imagePullSecrets:
- name: regcred

Заменить:

<имя_кластера>-operators-helm: уникальное имя пакета

<id-кластера>: значение поля Name в разделе Continuous Delivery/Clusters интерфейса ***Rancher***.

<частный_реестр_образов/путь>: имя частного реестра образов и путь к образам инфраструктуры, например registry.suod.local/***Rancher***.

6.4.8.5 infrastructure

1. Добавить в infrastructure/fleet.yml#/targetCustomizations:

- name: <имя_кластера>-infrastructure # Уникальное имя пакета
clusterSelector:
matchLabels:
*management.cattle.io/cluster-name: <id-кластера> # Имя кластера в **Rancher***
kustomize:
dir:./overlays/<имя_кластера>

2. Создать каталог infrastructure/overlays/<имя_кластера>
3. Скопировать в него содержимое infrastructure/overlays/_template
4. Отредактировать файл infrastructure/overlays/<имя_кластера>/env:

IMAGEREPO=<change>
INGRESS_HOST=<change>

IMAGEREPO: имя частного реестра образов и путь к образам инфраструктуры.

INGRESS_HOST: доменное имя сайта, через который будет осуществляться доступ к вебконсолям инфраструктуры (Minio, pgAdmin), например, worker-1.suod.local.

6.4.8.6 init-db

1. Добавить в init-db/fleet.yml#/targetCustomizations:

- name: <имя_кластера>-init-db # Уникальное имя пакета
clusterSelector:
matchLabels:
management.cattle.io/cluster-name: <id-кластера> # Имя кластера в Rancher
kustomize:
dir:./overlays/<имя_кластера>

2. Создать каталог init-db/overlays/<имя_кластера>
3. Скопировать в него содержимое init-db/overlays/_template

6.4.8.7 **apps**

1. Добавить в apps/fleet.yml#/targetCustomizations:

- name: <имя_кластера>-apps # Уникальное имя пакета
clusterSelector:
matchLabels:
management.cattle.io/cluster-name: <id-кластера> # Имя кластера в Rancher
kustomize:
dir:./overlays/<имя_кластера>

2. Создать каталог *apps/overlays/<имя_кластера>*.
3. Скопировать в него содержимое *apps/overlays/_template*.
4. Отредактировать файл *apps/overlays/<имя_кластера>/env*:

IMAGEREPO=<change>
IMAGEREPO_INFRA=<change>
FRONT_HOST=<change>
BACK_HOST=<change>
API_URL=<change>
MAIL_HOST=<change>

IMAGEREPO: адрес реестра образов ***Docker*** для приложений КСУОД, например, registry.suod.local/ksuod.

IMAGEREPO_INFRA: адрес реестра образов для инфраструктуры, например, registry.suod.local/*Rancher*.

FRONT_HOST: доменное имя хоста основного веб-интерфейса (панели администрирования и прочих веб-приложений КСУОД), например, www.suod.local.

BACK_HOST: доменное имя хоста для запросов к API от веб-клиентов и сторонних приложений, например api.suod.local.

API_URL: Полный адрес (с указанием схемы) служб API до базового пути, например, https://api.suod.local – адрес сайта должен совпадать со значением BACK_HOST.

MAIL_HOST: доменное имя почтового сервера для отправки уведомлений пользователям, например, mail.suod.local.

5. *apps/overlays/<имя_кластера>/authentication-deployment-patch.yaml*: заполнить значения полей value в секции /spec.template.spec.containers[0].env для соответствующих значений полей name:

KEYCLOAK_REALMS_0_REALM: наименование первого домена аутентификации (кроме master) в Keycloak, например, suod.

KEYCLOAK_REALMS_0_CLIENT_ID: идентификатор клиента первого домена в Keycloak, например, suod-client.

KEYCLOAK_REALMS_0_CLIENT_SECRET:

valueFrom.secretKeyRef.name: имя ресурса Secret, созданного для первого домена Keycloak (см п. 5.8.2.3 пп. 3).

valueFrom.secretKeyRef.key: client-secret

6. *apps/overlays/<имя_кластера>/admin-panel/deploymentpatch.yaml#/spec.template.spec.containers[0].env* – настройки панели администрирования:

ENV_VITE_APP_REALM: имя домена Keycloak для аутентификации пользователей.

ENV_VITE_APP_CLIENT_ID: идентификатор клиента домена Keycloak для аутентификации пользователей.

ENV_VITE_APP_CLIENT_SECRET:

valueFrom.secretKeyRef.name: имя ресурса Secret с учётными данными домена Keycloak для аутентификации пользователей.

valueFrom.secretKeyRef.key: client-secret

7. *apps/overlays/<имя_кластера>/api-gateway/deploymentpatch.yaml#/spec.template.spec.containers[0].env/:*

KEYCLOAK_SECURITY_JWT_RSA_PUBLIC_KEYS_0:

valueFrom.secretKeyRef.name: имя ресурса *Secret* с учётными данными домена *Keycloak* для аутентификации пользователей.

valueFrom.secretKeyRef.key: rsa-public-key

8. *apps/overlays/<имя_кластера>/product-data-export/main/deployment-patch.yaml* – настройки экспорта данных в основной сервер:

SERVER_URL: адрес сервера

USER:

valueFrom.secretKeyRef.name: имя ресурса *Secret* с учётными данными основного сервера.

valueFrom.secretKeyRef.key: ключ ресурса *Secret* с учётными данными основного сервера, в значении которого хранится имя пользователя, от имени которого осуществляется экспорт данных, например, *admin-user*.

PASSWORD:

valueFrom.secretKeyRef.name: имя ресурса *Secret* с учётными данными основного сервера.

valueFrom.secretKeyRef.key: ключ ресурса *Secret* с учётными данными основного сервера, в значении которого хранится пароль пользователя, от имени которого осуществляется экспорт данных, например admin-password.

REST_API_ACCESS_LEVEL_ID: уровень доступа – строковое значение, например, '0'.

SPRING_DATA_MONGODB_DATABASE: имя базы данных сервиса экспорта. Уникально в пределах одного экземпляра сервиса Mongodb, предварительно должно быть задано в базовом манифесте *infrastructure/base/mongo/dev-mongo.yaml* либо в его переопределении для текущего кластера

*infrastructure/overlays/<имя_кластера>/mongo/dev-mongo-patch.yam*l в массиве *spec.users[0].roles*.

KSUOD_PARTY_CONFIG_ID: идентификатор сервиса экспорта данных – уникален для каждого экземпляра сервиса.

KSUOD_SCHEDULER_EXPORT_ITEM_REVISIONS: строка в формате chrontab, задающая периодичность запуска процедуры экспорта, например "*/15 * * * * *" (каждые 15 секунд).

KSUOD_EMAIL_ENABLE: признак включения отправки почтовых уведомлений, 'true' – включить, 'false' – выключить.

KSUOD_EMAIL_FROM: e-mail адрес отправителя сообщения электронной почты для уведомлений.

SERVER_ID: обозначение сервера, используемое в скриптах экспорта и логах.

KSUOD_EMAIL_SERVER_FROM: обозначение сервера в сообщениях электронной почты.

9. *apps/overlays/<имя_кластера>/product-data-export/nsi/deployment-patch.yaml*:

настройки экспорта данных в дополнительный сервер – заполняются аналогично предыдущему пункту, обеспечивая уникальность значений.

По мере надобности можно добавлять новые экземпляры сервиса экспорта, для чего:

1. Создать переопределения в новом подкаталоге в *apps/overlays/<имя_кластера>/product-data-export/<суффикс>* с использованием другого переопределения в качестве шаблона.

2. Заполнить значения в *apps/overlays/<имя_кластера>/product-dataexport/<суффикс>/deployment-patch.yaml*, соблюдая уникальность значений.

3. Включить относительный путь нового каталога переопределения в файл *apps/overlays/<имя_кластера>/product-data-export/kustomization.yaml* в поле *resources.*

10. *apps/overlays/<имя_кластера>/product-data-import/deploymentpatch.yaml# /spec.template.spec.containers[0].env*: настройки сервиса импорта данных:

KSUOD_PRODUCT_DATA_IMPORT_CLEAR_MESSAGE_DATA: включить ('true') или выключить ('false') удаление промежуточных данных импорта из базы данных (рекомендуется выключать только для отладки).

KSUOD_EMAIL_ENABLE: включить ('true') или выключить ('false') отправку уведомлений по электронной почте.

KSUOD_EMAIL_FROM: обратный адрес электронной почты.

11. *apps/overlays/<имя_кластера>/users-integration/deploymentpatch.yaml# /spec.template.spec.containers[0].env* – переопределение настроек сервиса экспорта данных о пользователях:

XML_IMPORT_CONFIGURATION_ID: идентификатор конфигурации импорта XML для данных о пользователях.

KEYCLOAK_REALM: домен ***Keycloak***, из которого экспортировать данные пользователей.

KEYCLOAK_CLIENT_ID: идентификатор клиента домена ***Keycloak*** для аутентификации сервиса.

KEYCLOAK_CLIENT_SECRET – данные ресурса *Secret* домена ***Keycloak*** для аутентификации сервиса:

valueFrom.secretKeyRef.name: имя ресурса.

valueFrom.secretKeyRef.key: client-secret.

KEYCLOAK_LOGIN_USERNAME: данные ресурса *Secret* домена ***Keycloak*** для аутентификации сервиса:

valueFrom.secretKeyRef.name: имя ресурса.

valueFrom.secretKeyRef.key: admin-user

KEYCLOAK_LOGIN_PASSWORD: данные ресурса *Secret* домена ***Keycloak*** для аутентификации сервиса:

valueFrom.secretKeyRef.name: имя ресурса

valueFrom.secretKeyRef.key: admin-password.

SERVER_URL: адрес сервера.

USER – данные ресурса *Secret* с учётными данными сервера:

valueFrom.secretKeyRef.name: имя ресурса

valueFrom.secretKeyRef.key: admin-user

PASSWORD – данные ресурса *Secret* с учётными данными сервера:

valueFrom.secretKeyRef.name: имя ресурса

valueFrom.secretKeyRef.key: admin-password

12. *apps/base/product-data-import/scripts*: базовый каталог скриптов импорта данных (для всех кластеров).

13. *apps/base/product-data-export/config-scripts/scripts*: базовый каталог скриптов экспорта данных (для всех кластеров).

В случае необходимости использования индивидуальных версий скриптов для конкретного кластера:

1. Создать каталог *apps/overlays/<имя_кластера>/product-data-export /config-scripts* с набором скриптов.

2. Скопировать в него содержимое *apps/base/product-data-export/config-scripts*.

3. Поместить версию скриптов для кластера *<имя_кластера>* в каталог *apps/overlays/<имя_кластера>/product-data-export/config-scripts/scripts*.

4. Отредактировать, если необходимо, *apps/overlays/<имя_кластера>/ product-dataexport/config-scripts/kustomization.yaml#configMapGenerator[*].files* в соответствии с именами файлов из подкаталога *scripts/*.

5. Если необходимо, в *apps/overlays/<имя_кластера>/product-data-export/configscripts/patch.yaml* отредактировать */spec.template.spec.volumes* и */spec.template.spec.containers[0].volumeMounts*, чтобы они монтировали файлы в нужные пути файловой системы контейнера.

6. Заменить содержимое файла *apps/overlays/<имя_кластера>/product-dataexport/main/kustomization.yaml* на:

apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
resources:
-../../../../base/product-data-export/base # Взять базовые определения манифестов
-../../../../base/product-data-export/config # Взять базовую конфигурацию приложения
namePrefix: product-data-export- # Добавить префикс к именам ресурсов из базовой конфигурации
labels: # Добавить метки к ресурсам базвой конфигурации
- pairs:
app: product-data-export
includeSelectors: true
components:
-../config-scripts # Применить переопределённые скрипты

-../../../../components/config-volume # Применить базовую настройку томов конфигурации приложения

6.4.9 Добавление репозитория GitOps в систему Fleet сервера ***Rancher***

Настроить Fleet в интерфейсе ***Rancher*** на использование репозитория ksuod-ci, если это не было сделано раньше для других кластеров под управление этого же сервера ***Rancher***:

1. Открыть раздел Continuous Delivery > Git Repos.
2. Нажать кнопку [Add Repository].
3. Заполнить поля:

Name: ksuod-ci

Repository URL: Ссылка на репозиторий ksuod-ci на сервере Git, например:

https://gitea.suod.local/SUOD/ksuod-ci.git.

Watch: выбрать Branch

Branch Name: ввести имя ветки репозитория (обычно main или master для основного развёртывания).

Git Authentication:

Create a HTTP Basic Auth Secret, если нужно ввести новые логин и пароль к серверу Git для доступа по http(s),

Create a SSH Key Secret – если нужно ввести новый публичный и секретный ключи для доступа по протоколу ssh.

Если учетные данные были введены ранее, можно выбрать их из списка уже существующих учётных данных (все учётные данные сохраняются в ресурсе типа Secret кластера, на котором развёрнута платформа ***Rancher***).

TLS Certificate Verification:

Require a valid certificate: если сайт сервера Git работает с сертификатом, подписанным доверенным центром сертификации операционной системы, на которой работает Rancher.

Specify additional certificates to be accepted: если сертификат сервера Git не заверен доверенным центром, то его можно ввести в формате PEM в поле Certificates.

Accept any certificate (insecure): если есть уверенность, что соединение с сервером Git не может быть подвержено атаке типа MITM.

Enable Self-Healing: Если включено, то изменённые в кластере ресурсы автоматически восстанавливаются из репозитория.

Внимание!

На период установки и отладки рекомендуется отключить. В случае необходимости восстановления ресурсов использовать команду Force Update в контекстном меню репозитория или на панели инструментов раздела Git Repos.

Always Keep Resources: Если включено, то удалённые из репозитория ресурсы остаются в кластере.

Внимание!

В период промышленной эксплуатации рекомендуется включать, чтобы избежать непреднамеренного удаления ресурсов. Включать только перед планируемым свёртыванием ресурсов посредством удаления их из репозитория.

Target: Выбрать All Clusters in the Workspace – целевой кластер будет определяться значением поля targetCustomizations[0].clusterSelector файла fleet.yml каждого пакета Fleet.

4. Нажать [Create].

6.5 Настройка реестра образов *Docker*

6.5.1 Развернутый реестр образов ***Docker*** должен быть доступен для всех узлов кластера ***Kubernetes***.

6.5.2 Сбор и загрузка образов в реестр ***Docker*** выполняется из установочного архива.

6.5.3 Установочный архив имеет структуру:

- helm:

- *- cert-manager-crd.yaml: манифест ресурса cert-manager*
- *- cert-manager-v1.11.0.tgz: архив репозитория Helm модуля cert-manager*
- *- helm-v3.17.3-linux-amd64.tar.gz: архив утилиты Helm*
- *- helm-v3.17.3-linux-amd64.tar.gz.sha512: контрольная сумма*
- *- Rancher-2.10.3.tgz: архив репозитория Helm Rancher*
- *- sealed-secrets-2.17.2.tgz: архив репозитория Helm контроллера шифрования ресурсов Kubernetes Sealed Secrets*

- *- images: архивы образов для установки в частный реестр*
- *- cert-manager: модуль cert-manager*
- *- kafka: оператор Kubernetes Strimzi для развёртывания брокера сообщений Kafka*
- *- keycloak: менеджер учётных записей и аутентификации Apache Keycloak*
- *- minio: система хранения файловых данных Minio*
- *- mongo: оператор Kubernetes для развёртывания СУБД Mongo*
- *- postgres: СУБД Postgresql*
- *- Rancher: система Rancher*
- *- sealed-secrets: модуль шифрования ресурсов Kubernetes Sealed Secrets*
- *- Rancher-load-images.sh: скрипт загрузки образов из архива в частный реестр*
- *- Rancher-save-images.sh: скрипт загрузки образов из реестров в файл архива*
- *- k3s: файлы для установки реализации Kubernetes K3s*
- *- install.sh: скрипт установки K3s*
- *- k3s: исполнимый файл системы K3s*
- *- k3s.sha512: контрольная сумма для проверки целостности файла k3s*
- *- k3s-airgap-images-amd64.tar: архив образов системы K3s*
- *- k3s-airgap-images-amd64.tar.sha512 : контрольная сумма*
- *- sealed-secrets/cli: файлы клиентской утилиты контроллера Sealed Secrets*
- *- install.sh: скрипт установки в системе Linux*
- *- kubeseal-0.29.0-linux-amd64.tar.gz: архив с версией утилиты для ОС Linux*
- *- kubeseal-0.29.0-linux-amd64.tar.gz.sha512: контрольная сумма*
- *- kubeseal-0.29.0-windows-amd64.tar.gz: архив с версией утилиты для ОС Windows*
- *- kubeseal-0.29.0-windows-amd64.tar.gz.sha512: контрольная сумма*

6.5.4 В реестр образов ***Docker*** из архива необходимо установить:

– образы инфраструктуры:
bitnami/sealed-secrets-controller:0.29.0
busybox:latest

dpage/pgadmin4:8.14
groundnuty/k8s-wait-for:v1.6
minio/minio:RELEASE.2025-04-22T22-12-26Z
postgres:14.2
quay.io/jetstack/cert-manager-cainjector:v1.11.0
quay.io/jetstack/cert-manager-controller:v1.11.0
quay.io/jetstack/cert-manager-ctl:v1.11.0
quay.io/jetstack/cert-manager-webhook:v1.11.0
quay.io/keycloak/keycloak:22.0.4
quay.io/mongodb/mongodb-agent-ubi:108.0.6.8796-1
quay.io/mongodb/mongodb-community-server:7.0.19
quay.io/mongodb/mongodb-Kubernetes-operator-version-upgrade-post-start-hook:1.0.10
quay.io/mongodb/mongodb-Kubernetes-operator:0.13.0
quay.io/mongodb/mongodb-Kubernetes-readinessprobe:1.0.23
quay.io/spotahome/redis-operator:v1.2.4
quay.io/strimzi/kafka-bridge:0.28.0
quay.io/strimzi/kafka:0.41.0-kafka-3.6.2
quay.io/strimzi/kaniko-executor:0.41.0
quay.io/strimzi/maven-builder:0.41.0
quay.io/strimzi/operator:0.41.0
redis:6.2.6-alpine

– образы приложений:
access-management-service:latest
admin-panel:latest
api-gateway:latest
audit-service:latest
authentication-service:latest
camunda-service:latest
debezium-service:latest
freecadapi:latest
integration-logging:latest
integration-logging-gateway:latest
json-schema:latest
kafka-test:latest
quay.io/keycloak/keycloak:22.0.4

object-script-service:latest
product-data-export:latest
product-data-import:latest
product-data-integration:latest
product-direct-import:latest
request-logging:latest
users-integration:latest

6.5.5 Для загрузки образов в реестр ***Docker*** необходимо:

– Выполнить команду:

sh
sudo Docker login Docker-plm.ru

– Ввести логин и пароль

– На рабочей станции, имеющей доступ к частному реестру образов ***Docker***, выполнить команды для каждого подкаталога `images/`:

sh
cd /images/Rancher
../Rancher-load-images.sh -l Rancher-images.txt -i Rancher-images.tar.gz --registry Docker-plm.ru > out 2>error

где:

`*/images/Rancher*`: каталог с файлом архива образов (здесь – системы *Rancher*)

`*Rancher-images.txt*`: текстовый файл со списком тегов образов, которые нужно загрузить

`*Rancher-images.tar.gz*`: файл архива образов

`*registry.private.local*`: имя сервера частного реестра, в который загружать образы из архива

`*out*`: имя файла, в который запишется стандартный вывод скрипта

`*error*`: имя файла, в который запишется вывод ошибок скрипта

Пример для *Cert-manager*:

sh
cd /images/cert-manager
../Rancher-load-images.sh -l cert-manager.txt -i cert-manager.tar.gz --registry Docker-plm.ru

Внимание: Проверьте вывод скрипта на наличие ошибок. Убедитесь, что все образы импортировались успешно.

6.6 Установка и настройка *Kubernetes*

6.6.1 Подготовка каталога с образами

6.6.1.1 На узле сервера ***Rancher*** следует создать каталог образов (*images*) и скопировать в него установочные файлы:

```
sh
sudo mkdir -p /var/lib/Rancher/k3s/agent/images/
sudo cp ./k3s/k3s-airgap-images-amd64.tar/var/lib/Rancher/k3s/agent/images/
```

где

`./k3s/k3s-airgap-images-amd64.tar` – файл архива образов ***K3s***

6.6.1.2 Следует создать специальный файл реестра, для чего:

– Создать директорию: `Rancher/k3s`:

```
sh
  sudo mkdir -p /etc/Rancher/k3s
```

– Создать файл `/etc/Rancher/k3s/registries.yaml`:

```
sh
  sudo nano /etc/Rancher/k3s/registries.yaml
```

– Файл *registries.yaml* имеет следующее содержание:

```
yaml
---
mirrors:
  customreg:
endpoint:
  - "https://Docker-plm.ru/Rancher"
configs:
  customreg:
auth:
  username: kk-plm-ci
  password: 2AZDalz#mW9Iafmt1!o2pejbx47yvjtYsL@
#tls:
#  ca_file: /etc/ssl/certs/self-signed-CA.pem
```

где:

`mirrors.customreg.endpoint`: URL сервера частного реестра образов.

`configs.customreg.auth`: логин и пароль к частному реестру образов (`anonymous/anonymous` – если авторизация не включена)

`tls.ca_file`: путь к файлу с сертификатом CA сервера реестра

Примечание: Если файл сертификата отсутствует, следует закомментировать секцию tls с помощью символа `#`.

6.6.2 Установка K3s

– Перейти в директорию `*k3s/*`

sh

cd k3s

– Скопировать файлы из установочного каталога `*k3s/*` на узел сервера ***Rancher***:

– `k3s` – в `/usr/local/bin`

– `install.sh` – в любой доступный каталог (например, `~/`)

sh

sudo cp k3s /usr/local/bin/

sudo cp install.sh ~/

Примечание: Установить разрешение для исполняемых файлов:

sh

sudo chmod +x /usr/local/bin/k3s

sudo chmod +x ~/install.sh

– Перейти в директорию с файлом `*install.sh*`:

sh

cd ~/ # Если install.sh был скопирован в домашнюю директорию

– Запустить установочный скрипт ***K3s***:

sh

sudo INSTALL_K3S_SKIP_DOWNLOAD=true INSTALL_K3S_VERSION="v1.31.7+k3s1" ./install.sh

где:

`INSTALL_K3S_VERSION="v1.31.7+k3s1"`: указывает версию K3s

`INSTALL_K3S_SKIP_DOWNLOAD=true`: включает режим установки в изолированой среде

– После завершения выполнения скрипта выполнить проверку запуска сервиса:

sh

sudo systemctl status k3s

Примечание: В случае установки кластера K3s на нескольких узлах, на каждом дополнительном узле выполнить установку агента K3s:

sh

sudo INSTALL_K3S_SKIP_DOWNLOAD=true INSTALL_K3S_VERSION="v1.31.7+k3s1" K3S_URL=https://<SERVER>:6443 K3S_TOKEN=<TOKEN> ./install.sh

где:

`<SERVER>`: IP-адрес или DNS-имя управляющего сервера K3s

`<TOKEN>`: файл по пути `/var/lib/*Rancher*/k3s/server/node-token` на узле сервера K3s.

6.6.3 Настройка рабочей станции для работы с кластером ***K3s***

Для организации работы рабочей станции с кластером ***K3s*** сервера ***Rancher*** следует выполнить следующие действия:

– Установить на рабочую станцию, имеющую подключение к серверу ***Rancher*** утилиту `*kubectl*`.

– Скопировать на рабочую станцию в каталог `*~/.kube/config*` файл `*/etc/**Rancher**/k3s/k3s.yaml*`.

– Отредактировать в нём свойство `*clusters.cluster.server*`, чтобы оно указывало на адрес узла сервера ***Rancher***.

Примечание: Настройки утилиты фиксируются в файле kubeconfig. Для указания конкретного файла kubeconfig используется параметр --kubeconfig утилиты kubectl.

– Выполнить проверку установки кластера ***K3s*** командами:

sh

kubectl --kubeconfig ~/.kube/config/k3s.yaml get pods --all-namespaces

*Примечание: Утилита `kubectl` устанавливается также на сам сервер **K3s**, поэтому для управление кластером можно использовать узел сервера **Rancher**. В этом случае на сервере **Rancher** следует использовать команду `sudo`.*

6.7 Установка и настройка *Rancher*

6.7.1 Предварительно следует выполнить установку *Helm*. Для этого на узле сервера ***Rancher*** (и/или рабочей станции):

– Скопировать из установочного каталога файл `*helm/helm-v3.17.3-linux-amd64.tar.gz*`.

– Распаковать его:

sh

tar -zxvf helm-v3.17.3-linux-amd64.tar.gz

– Установить Helm:

sh

sudo cp linux-amd64/helm /usr/local/bin/helm

– Проверить работоспособность:

sh

helm help

6.7.2 Установка менеджера сертификатов *cert-manager*

Внимание! На сервере ***Rancher*** предварительно следует переключиться на суперпользователя:

sh

sudo su

Для установки менеджера следует:

– Задать путь к контексту K3S:

sh

export KUBECONFIG=/etc/Rancher/k3s/k3s.yaml

– Скопировать на рабочую станцию (сервер) файлы из установочного каталога:

- `helm/cert-manager-crd.yaml`

- `helm/cert-manager-v1.11.0.tgz`

– Создать пространство имён в кластере ***K3s***:

sh

kubectl create namespace cert-manager

– Создать CRD (CustomResourceDefinitions):

sh

kubectl apply -f cert-manager-crd.yaml

– Установить *cert-manager*:

sh

registry_host="Docker-plm.ru"

*helm install cert-manager ./cert-manager-v1.11.0.tgz *

*--namespace cert-manager *

--set image.repository=$registry_host/quay.io/jetstack/cert-manager-controller

*--set webhook.image.repository=$registry_host/quay.io/jetstack/cert-manager-webhook *

*--set cainjector.image.repository=$registry_host/quay.io/jetstack/cert-manager-cainjector *

--set startupapicheck.image.repository=$registry_host/ quay.io/jetstack/cert-manager-ctl

где:

`registry_host`: DNS-имя сервера частного реестра ***Docker*** с ранее установленными образами из установочного каталога.

6.7.3 Установка ***Rancher***

Для установки необходимо:

– Скопировать на рабочую станцию (сервер) файлы из установочного каталога:

- `helm/Rancher-2.10.3.tgz`

– Создать пространство имён `*cattle-system*`:

sh

kubectl create namespace cattle-system

– Выполнить команды:

sh

registry_host="Docker-plm.ru"

*helm install Rancher ./Rancher-2.10.3.tgz *

*--namespace cattle-system *

*--set hostname=kk-srv-ks4-t.npo.izhmash *

*--set RancherImageTag=v2.10.3 *

*--set certmanager.version=1.11.0 *

*--set RancherImage=$registry_host/Rancher/Rancher *

*--set systemDefaultRegistry=$registry_host *

--set useBundledSystemChart=true

где:

registry_host: DNS-имя сервера частного реестра ***Docker*** с ранее установленными образами из установочного каталога

hostname: имя машины в сети узла сервера ***Rancher***

– Для проверки имени машины следует использовать:

`sudo cat /etc/hosts`

Внимание! Заменить стандартное значение переменной hostname на имя машины, с которой происходит инсталляция.

– Дождаться выполнения инсталляции. После завершения выполнить

sh

kubectl get pods -n cattle-system

Внимание! Все контейнеры должны быть в состоянии Running или Completed.

– Получить пароль:

sh

kubectl get secret --namespace cattle-system bootstrap-secret -o go-template='{data.bootstrapPassword|base64decode}{"\n"}'

Следующие действия выполняются через веб-интерфейс сервера ***Rancher***.

– Открыть интерфейс сервера ***Rancher***, по URL `*https://**Rancher**.host*` (где `***Rancher**.host*` – доменное имя узла сервера ***Rancher***). В поле *Password* вставить пароль, полученный на предыдущем шаге.

– Создать новый пароль для входа в кластер или использовать предложенный.

– Отключить телеметрию, для чего выбрать опцию: ☰ > Global Settings > Settings > telemetry-opt: `Opt-out of Telemetry`

6.8 Развертывание кластера *Kubernetes*

6.8.1 Необходимо развернуть кластер ***Kubernetes*** под управлением платформы ***Rancher.*** Для развертывания кластера ***Kubernetes*** следует:

1. Открыть веб-интерфейс сервера ***Rancher***.
2. Нажать ☰ > Cluster Management.
3. На странице Clusters, нажать Create.
4. Выбрать RKE1.
5. Выбрать Custom.
6. Ввести имя кластера.
7. В секции Cluster options:

- *Kubernetes* Version: `1.31.x`
- Network provider: `Canal`
- Private Registry: `Enabled`:
- URL: доменное имя частного реестра с образами *Rancher* (например, `*Docker*-plm.ru`).
- User: имя пользователя реестра (или `*anonymous*`, если авторизация не включена).
- Password: пароль пользователя реестра или `anonymous`, если авторизация не включена.
- Остальные параметры – по умолчанию.

6.8.2 На каждом узле кластера ***Kubernetes*** (master/worker) выполнить создание директории:

sudo mkdir -p /mnt/volumes/volume

6.8.3 Следует отредактировать конфигурационный файл. Для этого в конфигурационном файле нужно найти соответствующую секцию и вставить следующие команды:

yaml
Rancher_Kubernetes_engine_config:
...
services:
kubelet:
extra_binds
private_registries:
user: kk-plm-ci
password: '2AZDalz#mW9Iafmt1!o2pejbx47yvjtYsL@'
ca_cert:
-----BEGIN CERTIFICATE-----
(второй, промежуточный сертификат)
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
(третий, корневой сертификат)
-----END CERTIFICATE-----

Примечание: Сертификаты можно получить командами:

sh
openssl s_client -showcerts -connect Docker-plm.ru:443 < /dev/null

Отобразится 3 сертификата, берем под номером 2 и 3.

6.8.4 В случае неудачных попыток нужно остановить и удалить контейнеры с master/worker нод

sudo Docker stop $(sudo Docker ps -q)
sudo Docker rm -f $(sudo Docker ps -a -q)

Затем удалить зависший кластер в графическом интерфейсе ***Rancher*** и создать кластер заново.

8. Нажать Next
9. В Node Role выбрать:

- `etcd`, `control plane` – скопировать сгенерированную команду ***Docker*** и выполнить её на узле кластера, назначенном на роль ***Kubernetes*** Control Plane.

- `worker` – выполнить сгенерированную команду на узлах, назначенных на роль ***Kubernetes*** Worker.

10. Дождаться завершения выполнения команд на узлах и нажать Done.
11. Дождаться перехода нового кластера в состояние Active.

6.9 Установка контроллера *Sealed Secrets*

6.9.1 Контроллер ***Sealed Secrets*** предназначен для шифрования открытым ключом на клиентской стороне манифестов ресурсов ***Kubernetes*** типа *Secret* и расшифровки их на стороне контроллера ***Kuberntetes*** закрытым ключом.

6.9.2 Для установки контроллера необходимо

– Скопировать из установочного каталога файл `*helm/sealed-secrets-2.17.2.tgz*` на рабочую станцию с установленными `*kubectl*` и `*helm*`.

– Выполнить команду установки:

```
sh
registry_host="Docker-plm.ru"
sudo KUBECONFIG=/etc/Rancher/k3s/k3s.yaml helm install sealed-secrets ./sealed-secrets-2.17.2.tgz \
  --namespace kube-system \
  --set image.registry=$registry_host
```

где:

`registry_host`: DNS-имя сервера частного реестра ***Docker*** с ранее установленными образами из установочного каталога.

6.10 Установка утилиты *Kubeseal*

6.10.1 Утилита ***Kubeseal*** предназначена для шифрования секретов в ***Kubernetes***

6.10.2 Установка утилиты выполняется на рабочей станции оператора GitOps из установочного каталога `*sealed-secrets/cli/*`:

– Если используется ОС Linux:

```
sh
sudo tar -xvzf kubeseal-0.29.0-linux-amd64.tar.gz kubeseal
sudo install -m 755 kubeseal /usr/local/bin/kubeseal
```

– Если используется ОС Windows:

Распаковать архив `*kubeseal-0.29.0-windows-amd64.tar.gz*` и скопировать утилиту в любой доступный каталог.

6.10.3 Для шифрования файлов манифестов без доступа к кластеру можно экспортировать открытый ключ шифрования из кластера. Для этого на рабочей станции с установленным *kubectl* и настроенным доступом к кластеру ***Kubernetes*** выполнить команду:

sh

sudo KUBECONFIG=/etc/Rancher/k3s/k3s.yaml kubeseal --controller-name=sealed-secrets --controller-namespace=kube-system --fetch-cert > seal-cert.pem

6.10.4 Шифрование манифеста *Secret*

Для шифрования манифестов *Secret* использовать команду:

```
sudo KUBECONFIG=/etc/Rancher/k3s/k3s.yaml kubeseal \
  --controller-name=sealed-secrets \
  --controller-namespace=kube-system \
  --format=yaml \
  --cert=seal-cert.pem \
  < secret.yaml > secret-sealed.yaml
```

где:

`seal-cert.pem`: файл открытого ключа контроллера sealed-secret

`secret.yaml`: оригинальный манифест ресурса ***Kubernetes*** Secret

`secret-sealed.yaml`: итоговый зашифрованный ресурс SealedSecret.

Внимание! *После шифрования оригинального манифеста Secret необходимо предотвратить его публикацию в репозитории Git или удалить.*

6.11 Настройка GitOps Fleet

6.11.1 Настройка репозитория ***Git***

– На доступном из сервера ***Rancher*** сервере ***Git*** необходимо создать репозиторий.

– Создать пользователя с правами на чтение-запись (`*push*`) в ветку `*main*`.

– В корень ветки `*main*` поместить файл *fleet.yml* следующего содержания:

```
yaml
targetCustomizations:
  - name: suod-local # Имя для отображения
    clusterSelector:
      matchLabels:
        management.cattle.io/cluster-name: c-2pff9 # Идентификатор кластера в Rancher
    kustomize:
      dir: clusters/suod-local # Каталог репозитория
```

где:

`management.cattle.io/cluster-name`: значение из поля ☰ > Continous Delivery > Clusters > выбрать кластер > Config > поле Name в интерфейсе ***Rancher***.

6.11.2 Выполнить установку ***Fleet***, для чего в интерфейсе ***Rancher***:

- ☰ > Continuous Delivery

- Namespace: `*fleet-default*` (включает все нижестоящие кластеры, зарегистрированные в *Rancher*)

– Git Repos:

- Name: Уникальное имя (например, `*suod-fleet*`)

- Repository URL: скопировать из сервера Git (Gitea, Gitlab), например: `*https://gitea.private.local/KSUOD/ksuod-fleet.git*`

- Watch: имя ветки или ревизии Git (`A Barnch`\> `main`)

- Git Authentication: `Create HTTP Basic Auth Secret`

- TLS Certificate Verification: Экспортировать сертификат Git-сервера в браузере (или curl) и вставить в поле Certificates

- Resource Handling: включить

- Deploy To: выбрать кластер (например, `ksuod-cluster`)

– Остальное оставить как есть.

6.11.3 Выполнить установку ***HA proxy*** в изолированной среде

– На всех требуемых серверах:

- Настроить статический IP

- Настроить разрешение DNS

- Отключить firewall

- Настроить маршрут по умолчанию

- Установить ***HA proxy***

– Проверить *firewall*:

sh

sudo ufw status

– Если команда выводит: *Status: active, в* этом случае требуется выполнить команду:

sh

sudo ufw disable

– Для того чтобы открыть порты нужно отредактировать файл настроек *iptables*:

sh

sudo nano /var/lib/iptables/rules-save

– В редакторе прописать правило для открытия нужного порта. Пример файла `*rules-save*`

*filter
:INPUT DROP [31:1752]
:FORWARD DROP [0:0]
:OUTPUT DROP [0:0]
:ACL-SSH - [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p tcp -m tcp --dport 22 -j ACL-SSH
-A INPUT -p icmp -m icmp --icmp-type 8 -j ACCEPT
-A INPUT -p tcp -m tcp --dport 10050 -m comment --comment "Accept zabbix connections." -j ACCEPT
-A INPUT -p tcp --dport 8080 -j ACCEPT
-A INPUT -p tcp --dport 80 -j ACCEPT
-A INPUT -p tcp --dport 443 -j ACCEPT
-A OUTPUT -j ACCEPT
-A ACL-SSH -s 10.0.0.0/8 -j ACCEPT
-A ACL-SSH -j DROP
COMMIT

– Сохранить изменения. После сохранения требуется выполнить команду

sh
sudo iptables-restore < /var/lib/iptables/rules-save

– Выполнить установку ***haproxy***:

sh
sudo apt install haproxy

– Начать редактирование конфигурационного файла

sh
sudo nano /etc/haproxy/haproxy.cfg

– В секции listen stats указать корректный путь до сертификата и изменить stats auth suod:<change_me> на ваш пароль.

listen stats
*bind *:8080 ssl crt /etc/ssl/private/suod.local.pem*
http-request redirect scheme https unless { ssl_fc }
stats enable
stats uri /

stats refresh 5s
stats realm Haproxy\ Statistics
stats auth suod:<change_me>
http-request redirect scheme https unless { ssl_fc }

– В секции *frontend suod-front* указать путь до сертификата, в *bind* указать адрес текущего сервера. порты 80 и 443 предварительно должны быть открыты!

frontend suod-front
bind 10.10.10.2:80
bind 10.10.10.2:443 ssl crt /etc/ssl/private/suod.local.pem
http-request redirect scheme https unless { ssl_fc }
default_backend suod-ingress

– В секции *backend suod-ingress* изменять имя *worker1* и адрес сервера на реальное имя *worker* узла ***Kubernetes***. В случае, если узлов *worker* несколько, следует указать каждый. На целевых узлах должен быть открыт порт 80!

backend suod-ingress
balance roundrobin
server worker1 10.10.10.5:80

– Сохранить изменения и перезапустить сервис ***HA proxy***:
sudo systemctl restart haproxy

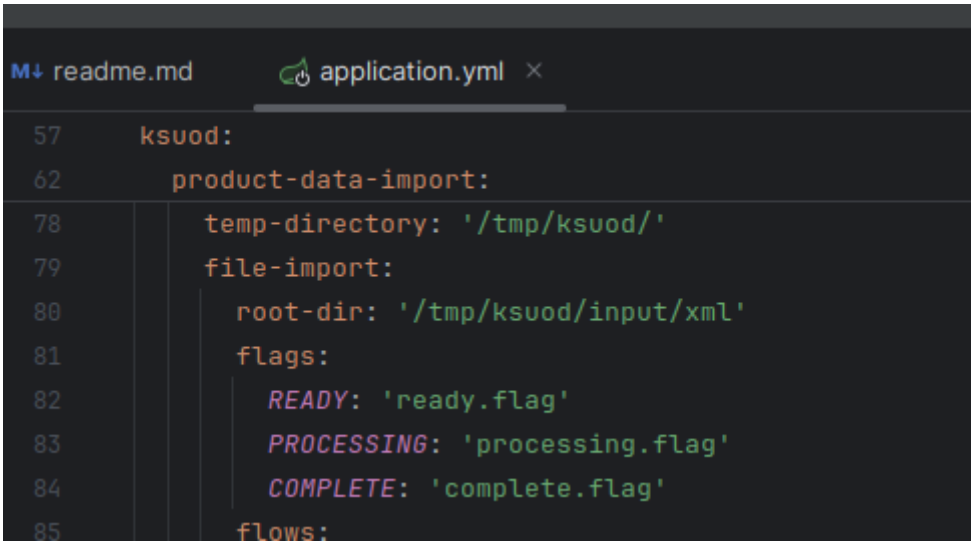
– Проверить статус сервиса ***HA proxy***
sudo systemctl status haproxy

6.12 Настройка Системы

6.12.1 Для того, чтобы Система приступила к обработке (например, миграции) входных данных, эти данные, предварительно подготовленные, должны быть размещены в определенной папке.

6.12.2 Для доступа к источнику входных данных, необходимо предоставить доступ с серверов входящих в кластер, к ресурсу на сервере на котором в папке находятся данные для миграции.

6.12.3 В конфигурационном файле микросервиса *product-data-import (*далее также – *import-data)* необходимо указать полный путь к источнику данных. Например:



6.12.4 Для обеспечения процесса интеграции также следует выполнить настройку маппинга. Маппинг (от англ. mapping) — процесс, который позволяет установить соответствие между несколькими группами данных, атрибутов или структурами. Маппинг, применяемый при интеграции КТД, настраивается и ведется в специальных конфигурационных файлах YML-формата, содержащих правила соответствия отдельных объектов, атрибутов, связей, специальных символов, моделей данных между собой. Порядок настройки маппинга приведен в приложении 1.

7 Техническая поддержка

7.1 Техническая поддержка Системы включает:

- гарантийную поддержку;
- дополнительную техническую поддержку.

7.2 В случае выявления неисправностей или возникших вопросов в ходе установки и настройки Системы, Заказчик может обратиться в службу технической поддержки Разработчика путем направления запроса по электронной почте на адрес ssa@ksuod.ru.

7.3 Регламенты технической поддержки описаны в документе «**Ошибка! Источник ссылки не найден.**».

Приложение 1. Настройка маппинга

Маппинг (от англ. mapping) — процесс, который позволяет установить соответствие между несколькими группами данных, атрибутов или структурами. Маппинг, применяемый при миграции данных, настраиваемый и ведется в специальных конфигурационных файлах YML-формата, содержащих правила соответствия отдельных объектов, атрибутов, связей, специальных символов, моделей данных между собой.

Выделяют следующие группы правил, в соответствии с которыми выполняется маппинг:

- Правила сопоставления атрибутов;
- Правила сопоставления объектов;
- Правила сопоставления связей.

П1.1 Правила сопоставления атрибутов:

- один к одному — необходимо заполнить атрибуты:
  «Атрибут системы-источника» – имя атрибута из исходного XML;
  «Атрибут системы-приемника» – GUID атрибута в которой необходимо конвертировать.
- атрибут объекта в атрибут связи — необходимо заполнить атрибуты:
  «Атрибут системы-источника» – имя атрибута из исходного XML;
  «Атрибут системы-приемника» – GUID атрибута в которой необходимо конвертировать;
  «Признак атрибута связи» – заполнить любым значением.
- атрибут в атрибут с ед. измерения — необходимо заполнить атрибуты:
  «Атрибут системы-источника» – имя атрибута из исходного XML;
  «Атрибут системы-приемника» – GUID атрибута в которой необходимо конвертировать;
  «Атрибут со значением ЕИ» – наименование атрибута содержащего единицу измерения в исходном XML.
- атрибут значение и атрибут единицы измерения в атрибут с ед. измерения — необходимо заполнить атрибуты:
  «Атрибут системы-источника» – имя атрибута из исходного XML;
  «Атрибут системы-приемника» – GUID атрибута в которой необходимо конвертировать;
  «Значение ЕИ» – код ОКЕИ для единицы измерения.
- атрибут значение в атрибут для конкретного типа объекта — необходимо заполнить атрибуты:
  «Атрибут системы-источника» – имя атрибута из исходного XML;

«Атрибут системы-приемника» – GUID атрибута в которой необходимо конвертировать;

«Тип объекта в системе-источнике» – имя типа объекта из исходного XML;

П1.2 Правила сопоставления объектов:

- один к одному— необходимо заполнить атрибуты:

«Тип объекта в системе-источнике» – имя объекта из исходного XML;

«Тип объекта в системе-приемнике» – GUID объекта, в который необходимо конвертировать.

- один к одному с формирование первоначального заголовка— необходимо заполнить атрибуты:

«Тип объекта в системе-источнике» – имя объекта из исходного XML;

«Тип объекта в системе-приемнике» – GUID объекта, в который необходимо конвертировать;

«Правило формирования заголовка объекта» – имена атрибутов в исходном XML.

- тип объекта после конвертации зависит от вхождения— необходимо заполнить атрибуты:

«Тип объекта в системе-источнике» – имя объекта из исходного XML;

«Тип объекта в системе-приемнике» – GUID объекта, в который необходимо конвертировать;

«Тип родительского объекта-источника» – имя родительского объекта из исходного XML.

Приоритетное правило, проверяется первым.

П1.3 Правила сопоставления связей:

- связь зависит от типа родительского и дочернего объекта— необходимо заполнить атрибуты:

«Тип объекта-родителя связи» – имя объекта из исходного XML;

«Тип дочернего объекта связи» – имя объекта из исходного XML;

«Тип связи системы-приемника» – GUID типа связи, в который необходимо конвертировать.

Если тип связи должен создаваться одинаковым при неизменном родительским или дочернем типе объекта, то вместо имени можно вставить знак «» который означает любой тип объекта.*

- связь зависит от типа связи, указанной в исходном XML— необходимо заполнить атрибуты:

«Тип связи системы-источника» – имя типа связи из исходного XML;
«Тип связи системы-приемника» – GUID типа связи, в который необходимо конвертировать.

Указанные правила задются в специальных конфигурационных файлах:

- application-metadata.yml
- application-meta-objects.yml
- application-meta-attributes.yml
- application-meta-relations.yml

Данные файлы хранят конфигурационную информацию об объектах, атрибутах и связях, которые связаны между собой следующим образом (рисунок П1.1). Описание конфигурационных файлов приведено в разделе 6.

Процедура валидации вносимых изменений в решении не предусмотрена.

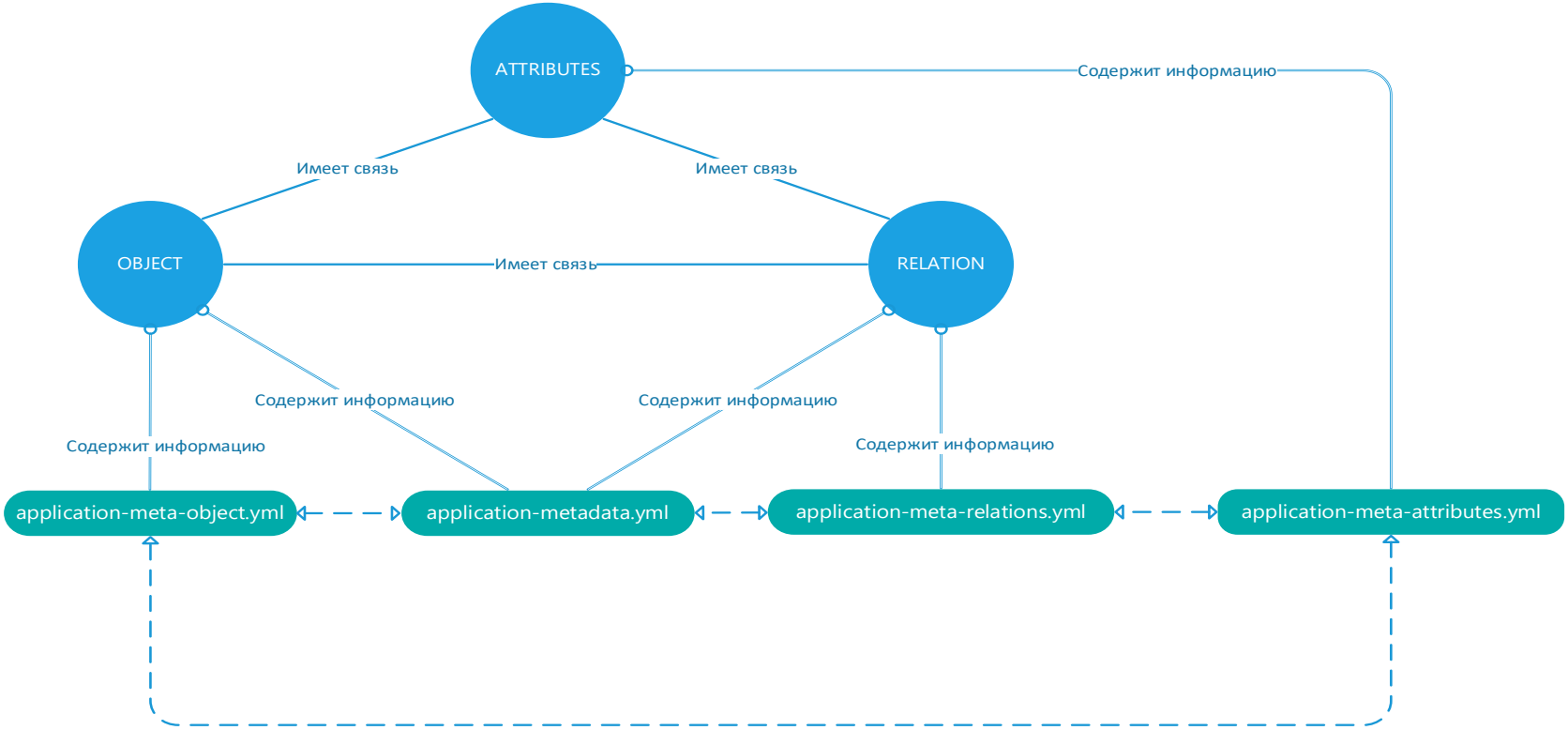


Рисунок П1.1. – Связь конфигурационных файлов

```
'[DirectModel]':
  type-name: JT-представление документа
  guid: cadd94e9-306c-11d8-b4e9-00304f19f545
  id: 1715
  attributes: [ revision, designation, name, description, 'Файл', 'Владелец (строка)',
```

Рисунок П1.2. – Связь параметров конфигурационных файлов

П1.4 Пример настройки маппинга

Порядок действий при настройке маппинга рассмотрим на следующих примерах.

П1.4.1 Добавление объекта нового типа

Допустим, в процессе работы возникла необходимость добавить в маппинг объект нового типа (object_type=«**Сборочные единицы**»).

В исходном xml-файле это выглядит так:

```
<PRODUCT item_id="176453" uom="796" owning_user="s.y.rozov" object_type="Сборочные единицы"
  <VERSION owning_user="s.y.rozov" uid="jjldK4494Tv0cD" version="000">
    <ATTRLIST>
      <ATTR name="Наименование" value="Сборка 2" />
      <ATTR name="Головное изделие" value="0" />
      <ATTR name="Литера" value="" />
      <ATTR name="Масса" value="0.000" />
```

Последовательность действий следующая:

1. Находим соответствующий тип объекта в базе. Если соответствующего объекта нет, его необходимо предварительно создать.

2. Добавляем в конфигурационный файл ***application-meta-object.yml*** информацию о найденном объекте следующим образом:

```
14        Сборочные единицы:
15          type-name: Сборочные единицы
16          guid: cad00132-306c-11d8-b4e9-00304f19f545
17          id: 1074
18          attributes: [ ]
19          parent: Products
```

3. Сравниваем название найденного объекта и название объекта во входном xml файле.

4. Если имена совпадают («**Сборочные единицы**» и «**Сборочные единицы**»), изменение в маппинг завершено.

5. Если имена не совпадают («**Сборочные единицы**» и «**Сборочная единица**»), редактируем (добавляем строку) в ***application-metadata.yml***, в которой указываем название из xml и найденного объекта:

```
object-type-mapping:
  SPB5_PKI: Part
  SPB5_Assy: Assembly
  '[Деталь]': Part
  '[Комплектующее]': Part
  '[Сборочная единица]': 'Сборочные единицы'
```

На этом внесение изменений в маппинг завершено.

П1.4.2. Добавление новых атрибутов к объекту

Допустим, в процессе работы возникла необходимость добавить в процесс маппинга обработку нового атрибута «***Масса***».

В исходном xml-файле этот атрибут выглядит так:

```
<PRODUCT item_id="176453" uom="796" owning_user="s.y.rozov" object_type="Сборочная единица"
  <VERSION owning_user="s.y.rozov" uid="jjldK4494Tv0cD" version="000">
    <ATTRLIST>
      <ATTR name="Наименование" value="Сборка 2" />
      <ATTR name="Головное изделие" value="0" />
      <ATTR name="Литера" value="" />
      <ATTR name="Масса" value="0.000" />
```

Последовательность действий следующая:

- Находим соответствующий атрибут в базе. Допустим, его Обозначение = Масса, guid= cad00275-306c-11d8-b4e9-00304f19f545
- Найденный в базе атрибут записываем в конфигурационный файл ***application-meta-attributes.yml***, заполняя соответствующие строки:

```
'[Масса]':
  name: Масса
  guid: cad00275-306c-11d8-b4e9-00304f19f545
  id: 1000
  type: ftMeasured
```

- В исходном XML-файле смотрим, к какому объекту принадлежит атрибут

```
<PRODUCT item_id="176453" uom="796" owning_user="s.y.rozov" object_type="Сборочная единица"
  <VERSION owning_user="s.y.rozov" uid="jjldK4494Tv0cD" version="000">
    <ATTRLIST>
      <ATTR name="Наименование" value="Сборка 2" />
      <ATTR name="Головное изделие" value="0" />
      <ATTR name="Литера" value="" />
      <ATTR name="Масса" value="0.000" />
```

- Находим соответствующий объект в базе
- Найденный в базе объект записываем в конфигурационный файл ***application-meta-objects.yml***:

```
14          Сборочные единицы:
15            type-name: Сборочные единицы
16            guid: cad00132-306c-11d8-b4e9-00304f19f545
17            id: 1074
18            attributes: [ 'Масса' ]
19            parent: Products
```

- Если атрибут во входном XML записан вне секции <VERSION>,

```
<PRODUCT item_id="СИ01.6458.С61" uom="796" owning_user="s.y.rozov" object_type="Сборочная единица" semanti
    <ATTRLIST>
        <ATTR name="Масса" value="7"/>
    </ATTRLIST>
    <VERSION owning_user="s.y.rozov" uid="E3kZt3f34Tv0cD" version="000">
```

то записываем его обработку в ***ItemRevisionConverter.groovy*** так:

```
attribute 'Код организации', { versionOrgCode != null ? versionOrgCod
attribute 'Масса', { obj.properties?.'product'?.'Масса' as String }
attribute 'Головное изделие (признак)', {
```

- Если атрибут во входном XML записан в <VERSION>,

```
    <VERSION owning_user="s.y.rozov" uid="E3kZt3f34Tv0cD" version="000"
        <ATTRLIST>
            <ATTR name="Наименование" value="Корпус в сборе"/>
            <ATTR name="Головное изделие" value="0"/>
            <ATTR name="Литера" value="01"/>
            <ATTR name="Масса" value="204"/>
```

то записываем его обработку в *ItemRevisionConverter.groovy* так

```
}
attribute 'Масса', { obj.properties?.'version'?.'Масса' as String }
attribute 'Формат (изделие)', { obj.p
```

- Если атрибут записан в секции <BOMLINE>

```
        <BOMLINE item_id="Круг10 ГОСТ 7417 50 ГОСТ 1050"
                 semantic_uid=""
                 occ_type="Основной материал"
                 sequence_no="10"
                 quantity="1.0"
                 quantity_uom="166"
                 object_type="Полуфабрикат">
            <ATTRLIST>
                <ATTR name="Масса" value="7"/>
```

- Смотрим в *xml*, к какой связи относится атрибут (в данном случае, «Полуфабрикат» относится к «Заготовке», связь «Заготовка^Полуфабрикат»)

```
<PRODUCT item_id="060896" uom="796" owning_user="al.vl.popov"
         object_type="Заготовка" semantic_uid="" uid="ZLb9Ragr4Tv0cD">
   <VERSION owning_user="al.vl.popov" uid="ZPR9Ragr4Tv0cD" version="00
      <ATTRLIST>
         <ATTR name="Идентификатор" value="060896"/>
         <ATTR name="Ревизия" value="000"/>
         <ATTR name="Наименование" value="Заготовка"/>
      </ATTRLIST>
      <STATUS status_id="-1073663586" name="ОКР" from="2020-09-03T09:1
      <BOM name="060896-000-Состав" uid="ZXf9Ragr4Tv0cD">
         <BOMLINE item_id="Круг10 ГОСТ 7417 50 ГОСТ 1050" semantic_uid
                  object_type="Полуфабрикат">
```

- Ищем связь в файле *application-meta-relations.yml.*

```
'[Заготовка^Полуфабрикат]': ProcessStructure
```

Если там связь не найдена или связь составная (например, «Заготовка^Полуфабрикат»), то сначала выполняем поиск в *application-metadata.yml,* затем найденное наименование типа ищем в *application-meta-relations.yml*

```
ProcessStructure:
  type-name: Технологический состав
  guid: cad0019f-306c-11d8-b4e9-00304f19f545
  id: 1002
  attributes: [ position, 'Тип вхождения в ТП', 'Сортировка', 'Номер объект
                'Н.расх (Сред)', 'Н.расх (Факт)', 'Единица величины', 'Номе
                'Ширина реза от инструмента (В)', 'Масса заготовки', 'Отход
                'Чистовые размеры', 'КДЗ', 'Геом. параметры', 'КИМ']
```

- Если атрибута в найденной связи нету, добавляем его

```
ProcessStructure:
  type-name: Технологический состав
  guid: cad0019f-306c-11d8-b4e9-00304f19f545
  id: 1002
  attributes: [ position, 'Тип вхождения в ТП
                'Н.расх (Сред)', 'Н.расх (Фак
                'Ширина реза от инструмента
                'Чистовые размеры', 'КДЗ', 'Г
                'Масса']
```

- Добавляем обработку атрибута в *StructureRelationConverter.groovy*

```
attributes.add(attribute(typeKey, key: "Масса",
        occurrence?.properties?.'Масса' as String))
```

- Если атрибут записан в <ICO>,

```
<ICO class_id="PLM0606">
    <ATTRLIST>
        <ATTR name="Macca" value="7"/>
```

пишем обработку в *ItemRevisionConverter*.groovy

```
attribute 'Macca',
        { obj.properties?.'icoEntryAttrs'?.'Macca' as String }
```

- Если атрибут имеет тип «*ftDateTime*»

```
'[Дата внедрения ТП]':
  name: Дата внедрения ТП
  guid: 16b70f6e-317a-43ed-8e97-b92a9fb1d19a
  id: 18790
  type: ftDateTime
```

пишем обработку в *ItemRevisionConverter*.groovy

```
dateTimeAttribute 'Дата внедрения ТП',
        { obj.properties?.'version'?.'Дата внедрения' as String }
```

- Если атрибут имеет тип *ftObjectLink,* при его записи возможно добавление поля *link-guid.*

```
'[Ед. измерения]':
  name: Ед. измерения
  guid: 34d11d34-1904-400a-b91d-0a1021d7703f
  id: 18766
  type: ftObjectLink
  link-guid: cad0000b-306c-11d8-b4e9-00304f19f545
```

Обработка атрибута с таким полем

```
linkAttribute | key: 'Ед. измерения',
        supplierValue: { obj.properties?.'version'?.'uom' as String }
```

- На этом добавление нового атрибута завершается

П1.4.3. Добавление новой связи

Допустим, в процессе работы возникла необходимость добавить в процесс маппинга обработку новой связи «***Ссылка на ЦЗ***»

В исходном xml-файле связь «***Ссылка на ЦЗ***» выглядит так:

```
<PRODUCT item_id="PreRSC_Example_1_111" uom="796" owning_user="u.m.mihaylova" object_type="ФЦ3">
    <VERSION owning_user="u.m.mihaylova" uid="X5gZeQcb4Tv0cF" version="1">
        <ATTRLIST>
            <ATTR name="Цех-участник основной (признак)" value="1"/>
            <ATTR name="shop" value="110.04"/>
            <ATTR name="Назначенное бюро" value="БМН"/>
            <ATTR name="Начальник тех.бюро" value="i.i.ivanov"/>
            <ATTR name="Начальник цеха" value="p.p.petrov"/>
            <ATTR name="Вид работ" value="21"/>
        </ATTRLIST>
        <RELATION rel_type="Ссылка на ЦЗ" item_id="Цехозаход_1" semantic_uid=""
                  object_type="Передельный техпроцесс"/>
    </VERSION>
</PRODUCT>
```

Последовательность действий следующая:

- В базе находим связь, которой соответствует связь «***Ссылка на ЦЗ***» (допустим она называется «***Технологическая связь с расцеховкой***»).
- Записываем следующий блок в ***application-meta-relations.yml***:

```
'[Технологическая связь с расцеховкой]':
  type-name: Технологическая связь с расцеховкой
  guid: cad005b0-306c-11d8-b4e9-00304f19f545
  id: 1009
  attributes: [ ]
```

- Если наименование связи в исходном xml-файле не совпадает с наименованием соответствующей связи в базе, прописываем в ***application-metadata.yml*** следующую строку в разделе *relation-type-mapping*:

```
relation-type-mapping:
  '[Ссылка на ЦЗ]': 'Технологическая связь с расцеховкой'
  '[Документы]': ProductDocumentation
```

- На этом добавление новой связи завершается.