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

КОРПОРАТИВНАЯ СИСТЕМА УПРАВЛЕНИЯ И ОБМЕНА ДАННЫМИ. ПЛАТФОРМА – КОНСТРУКТОРСКАЯ ПОДГОТОВКА ПРОИЗВОДСТВА

(КСУОД.ПЛАТФОРМА-КПП)

РУКОВОДСТВО АДМИНИСТРАТОРА

Листов 74

2025

Аннотация

В настоящем документе приведено описание системы «Корпоративная система управления и обмена данными. ПЛАТФОРМА – Конструкторская подготовка производства» (далее – Система).

Документ приводит сведения о требованиях к обслуживающему персоналу, технических требованиях к Системе, установке ПО, удаления ПО, сведения о технической поддержки пользователей.

Оглавление

1 Нормативные ссылки

В настоящем документе приведены ссылки на следующие нормативные документы:

1. Корпоративная система управления и обмена данными. ПЛАТФОРМА – Конструкторская подготовка производства (КСУОД.ПЛАТФОРМА-КПП). Руководство пользователя

2 Термины, определения и сокращения

В настоящем документе применяются следующие сокращения, термины и определения:

БД	–	База данных
КПП	–	Конструкторская подготовка производства
КТПП	–	Конструкторско-технологическая подготовка производства
ОС	–	Операционная система
ПО	–	Программное обеспечение
СУБД	–	Система управления базами данных
ТО	–	Техническое обслуживание
ТПП	–	Технологическая подготовка производства
TC	–	Teamcenter

3 Общие сведения

3.1 Система предназначена для управления процессами конструкторской подготовки производства как на отечественных промышленных предприятиях, так и в конструкторских организациях (бюро).

3.2 Система обеспечивает:

- эффективное управление основными процессами КПП в части формирования конструкторских данных и документов;
- взаимодействие сотрудников на всех этапах жизненного цикла изделий и предоставление им единого источника знаний об этих изделиях и связанных с ними процессах.

3.3 Система имеет возможность гибко подключать новые функциональные компоненты, а также модернизировать существующие.

3.4 Система применяет инструменты, существенно упрощающие и снижающие трудоёмкость по вводу и управлению конструкторскими данными;

3.5 Наличие механизмов управления версиями (ревизиями), опциями и альтернативами позволяет вести в одном электронном составе изделия все его варианты и, при необходимости, конфигурировать состав конкретного экземпляра изделия.

3.6 Система обеспечивает существенное снижение трудоёмкости проведения изменений в конструкции изделия и разработки новых вариантов (модификаций) изделия.

3.7 В Системе предусмотрена возможность непрерывного обмена данными с системами CAD, поэтому существует возможность работы с различными составами (или вариантами состава) изделия непосредственно в системе CAD.

3.8 Особенности Системы заключаются в следующем:

- единая объектная модель обработки всей информации;
- единая база данных, которая исключает повторный ввод информации и позволяет контролировать ее целостность;
- поддержка комплекса стандартов ГОСТ серии ЕСКД;
- устойчивость к сетевым сбоям и возможность настройки на различную пропускную способность сети;
- решения по хранению и обработке данных позволяют работать с объектами, содержащими информацию на различных языках и содержащую различные спецсимволы.

3.9 Система разработана как самостоятельный программный продукт.

3.10 Функции, реализуемые Системой, объединяются в две группы:

– группа функций «Обеспечение доступа к функциональности Системы».

– группа функций «Конструкторская подготовка производства».

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

– управление пользователями Системы;

– контроль доступа к Системе;

– защита от несанкционированного доступа.

3.12 Данные функции решают следующие задачи:

– предоставление пользователям веб-интерфейса для доступа к функциональности Системы;

– формирование и управление системой прав доступа;

– формирование и управление пользователями;

– формирование и управление процессами электронного согласования;

– фиксация и протоколирование событий, происходящих в Системе;

– предоставление доступа к функциональности Системы в соответствии с правами доступа.

3.13 Указанные функции реализованы специализированными сервисами:

– авторизации;

– управления пользователями;

– управления правами доступа;

– логирования.

3.14 Группа функций «Конструкторская подготовка производства» включает функции, которые обеспечивают автоматизацию основных процессов КПП и работу с конструкторскими данными и документами.

3.15 Условная функционально-принципиальная схема Системы приведена на рисунке 1, где:

- функциональный блок «Сервисы» – ПО, являющееся частью Системы и реализующее группу функций обеспечения доступа к функциональности Системы;
- функциональный блок «КПП» – ПО, являющееся частью Системы и реализующее задачи конструкторской подготовки производства и управления конструкторскими данными и документами.

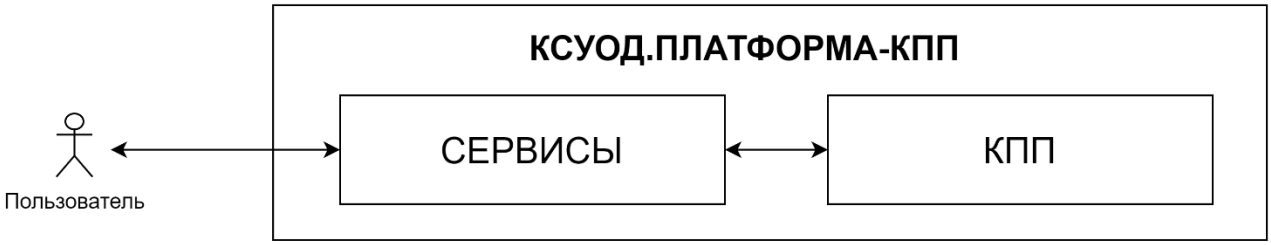


Рисунок 1 – Структурная схема Системы

3.16 Система обеспечивает интерактивное взаимодействие сотрудников, участвующих на всех этапах конструкторской подготовки производства; формирование и предоставление пользователям единого источника знаний об изделиях и связанных с ними процессах.

3.17 Порядок работы пользователей с Системой описывается в документе «**Ошибка! Источник ссылки не найден.**».

4 Требования к обслуживающему персоналу

4.1 К администрированию и обслуживанию Системы, допускаются лица, ознакомившиеся с проектной и эксплуатационной документацией, документацией на поставляемое ПО и аппаратное обеспечение, имеющие практические навыки работы с соответствующим программным и аппаратным обеспечением.

4.2 Администраторы, обеспечивающие работу Системы, должны обладать навыками работы на компьютерах и в локальных сетях, а также иметь представление о том, что такое СУБД.

4.3 В основные обязанности администраторов входит:

- установка ПО на компьютеры;
- создание гибкой модели архивов предприятия;
- ведение списка ролей, пользователей и групп пользователей;
- настройка типов объектов, атрибутов и взаимосвязей между ними;
- назначение прав доступа пользователей к различным объектам системы;
- ведение списка должностей предприятия для поддержки электронных подписей документов;
- определение типовых маршрутов движения документов на предприятии;
- управление журналом регистрации действий пользователей в системе;
- настройка инструментов для работы с хранящимися в системе документами;
- администрирование СУБД и страховое копирование баз данных решения.

5 Технические требования

5.1 Требования к аппаратной инфраструктуре

5.1.1 Схема аппаратной инфраструктуры, на которой разворачивается Система, приведена на рисунке 2.

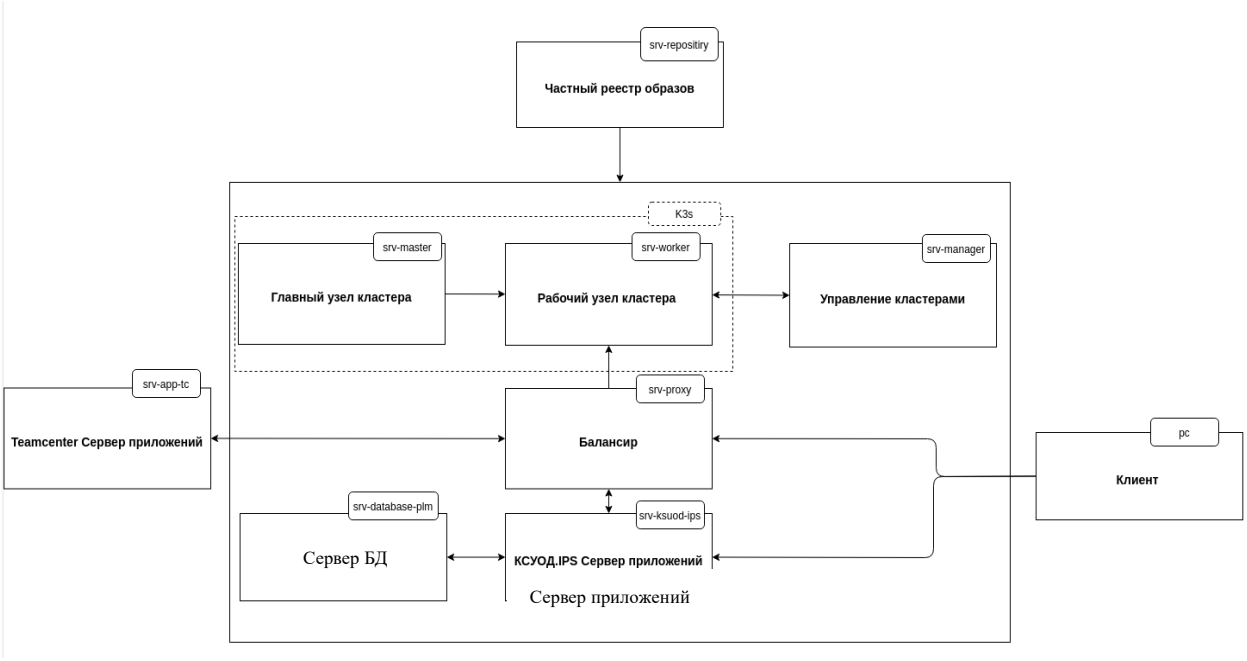


Рисунок 2. – Схема аппаратной инфраструктуры
для размещения Системы

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

5.1.3 Минимальные аппаратные требования к серверам СУБД:

– 1 ядро процессора типа Xeon класса Silver или выше на 10-20 пользователей системы.
– 1 Gb оперативной памяти на 10-20 пользователей системы (+объем памяти, рекомендуемый для используемой ОС).
– Для системной базы данных рекомендуется использовать SSD-диски корпоративного класса суммарным объемом 1 Tb и выше. Для баз данных с файловыми шкафами рекомендуется использовать диски SAS или SATA 10000 rpm. суммарным объемом не менее 2 Tb.

5.1.4 Минимальные аппаратные требования к серверам приложений:

– 1 ядро процессора типа Xeon класса Silver или выше на 5-15 пользователей системы.
– 1 Gb оперативной памяти на 5-15 пользователей системы (+объем памяти, рекомендуемый для используемой ОС).

– Диски SATA 7200rpm общим объёмом от 1 Tb.

Примечание: Методика расчета характеристик серверов приведена в приложении 1.

5.1.5 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.

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

5.1.7 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.

5.1.8 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).

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

5.1.10 Должен быть развернут балансировщик нагрузки.

5.1.11 Должен быть развернут DNS-сервер.

5.1.12 Должна быть настроена синхронизация времени.

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

5.1.14 Минимальные аппаратные требования к рабочему месту, на котором устанавливается клиент:

– Intel i3 (AMD Ryzen 3), 1 Gb RAM, HDD 500 Gb. Рекомендуется Intel i5 (AMD Ryzen 5), 4 Gb RAM, 1 Tb HDD/SSD.
– При выборе аппаратных средств также следует учитывать требования операционной системы и программ для просмотра и редактирования документов.

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).

5.3 Подготовка базы данных функционального блока «КПП»

Для правильной установки и конфигурирования СУБД PostgreSQL необходимо руководствоваться комплектом технической документации. Установочный пакет и необходимую документацию можно скачать на официальном сайте PostgreSQL (https://www.postgresql.org/download/).

Внимание! При развёртывании КСУОД.ПЛАТФОРМА-КПП после установки системы КСУОД.ПЛАТФОРМА-ТПП отдельно устанавливать СУБД не надо, достаточно настроиться на базу данных ранее установленной системы.

6 Установка функционального блока «Сервисы»

6.1 Функциональный блок 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***.

6.2 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.

6.3 Перед установкой необходимо выполнить настройку:

– рабочих узлов (описана в приложении 2);
– портов (описана в приложении 3);
– репозитория манифестов (описана в приложении 4).

6.4 Далее необходимо развернуть и настроить:

– реестр образов ***Docker*** (описано в приложении 5);
– платформу ***Kubernetes (K3s)*** (описано в приложении 6);
– платформу ***Rancher*** (описано в приложении 7);
– кластер ***Kubernetes*** (описано в приложении 8);
– контроллер ***Sealed Secrets*** (описано в приложении 9;***;***
– утилиту ***Kubeseal*** (описано в приложении 10);
– сервер системы управления версиями ***Git***, доступный по сети для узлов ***Rancher*** (описано в приложении 11).

6.5 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***.

7 Установка функционального блока «КПП»

7.1 Установка .NET Framework

Для работы Системы в системе Windows требуется пакет .Net Framework. Обычно в составе ОС этот пакет уже установлен (предустановлен по умолчанию).

Если .NET Framework, по каким-то причинам не был установлен, можно воспользоваться следующим способом:

- Откройте панель управления.
- Найдите и откройте «Программы и компоненты» (Рисунок 3).

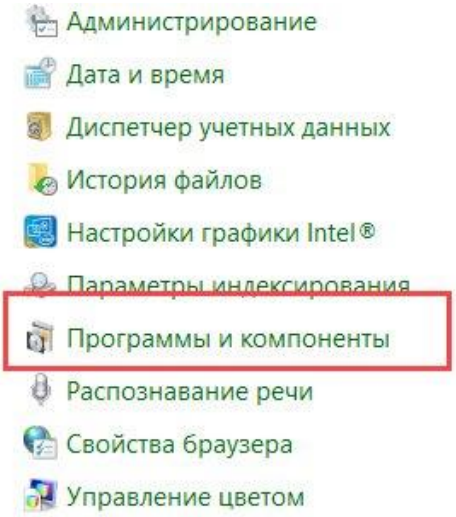


Рисунок 3

- Нажмите «Включение или отключение компонентов Windows» (Рисунок 4).

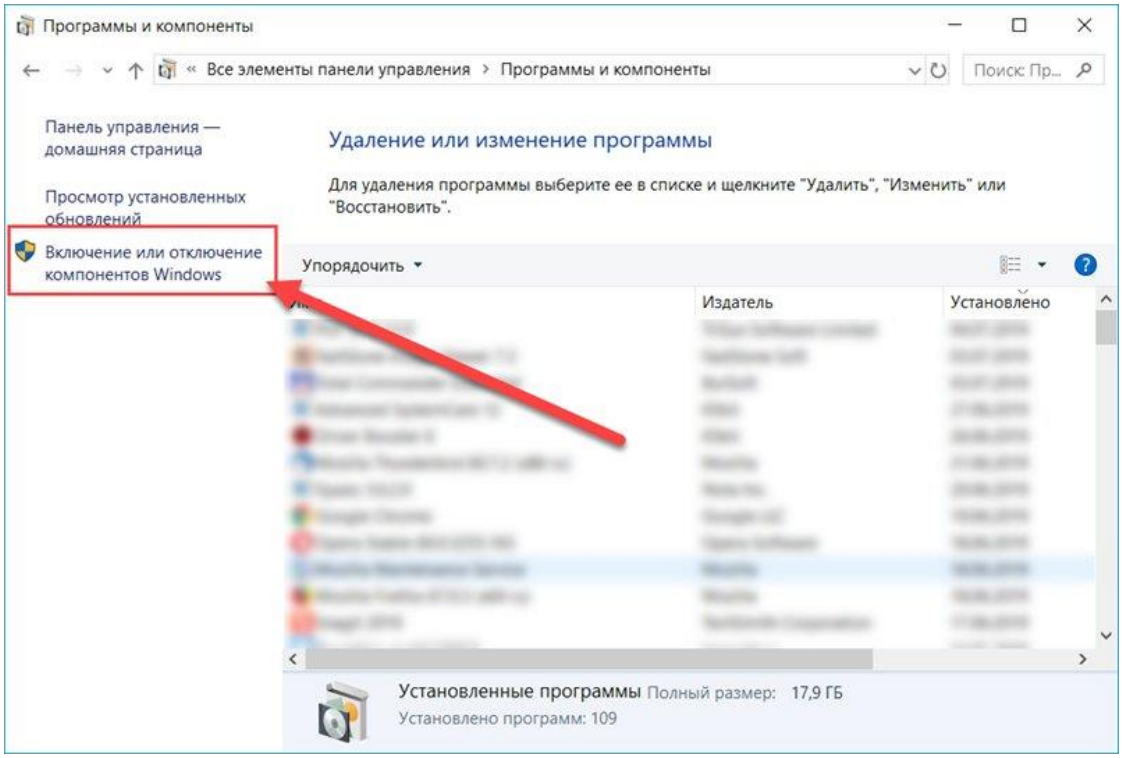


Рисунок 4

- Выберите .NET Framework и нажмите кнопку «ОК» (Рисунок 5).

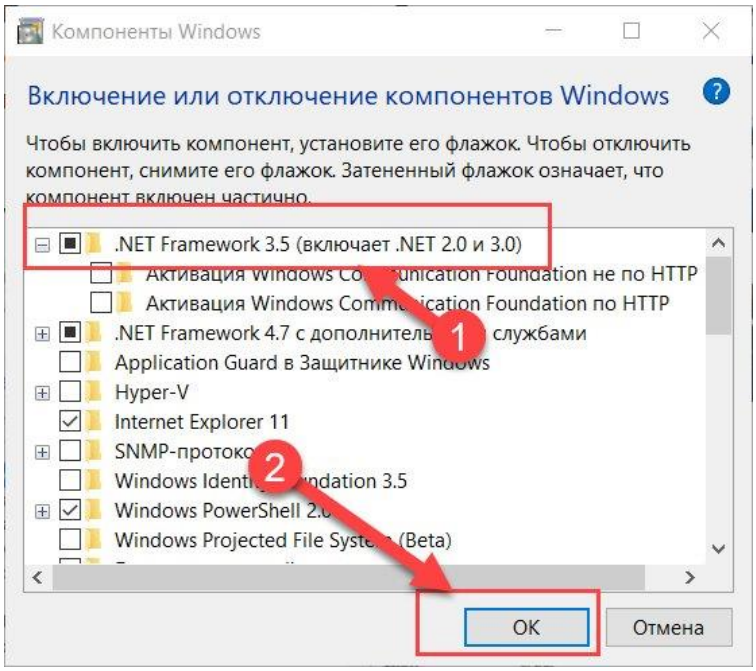


Рисунок 5

После этого начнётся загрузка .NET Framework на компьютер.

7.2 Установка Системы.

Установка Системы, начинается с установки серверной части.

Однопользовательская установка представляет собой одновременную установку серверной и клиентской частей на одной машине.

Для запуска инсталлятора следует запустить установочный файл из состава инсталляционного пакета.

Инсталлятор отобразит мастер установки, для перемещения по окнам необходимо использовать кнопки «Далее» и «Назад».

В процессе установки (Рисунок 6) производится указание имени пользователя и организации, указывается тип установки (только сервер или сервер и клиент), осуществляется настройка доступа к web-службам решения.

Для настройки портала\узла следует указать параметры настройки серверной части (в качестве портала/узла портала) – пользователю предоставляется возможность установить портал, либо настроить серверную часть, как узел портала.

Следует определить следующие параметры:

– Наименование соединения – поле для текстового описания настраиваемого соединения узла с порталом Системы.

– Хост портала – имя компьютера, на котором развернут портал.

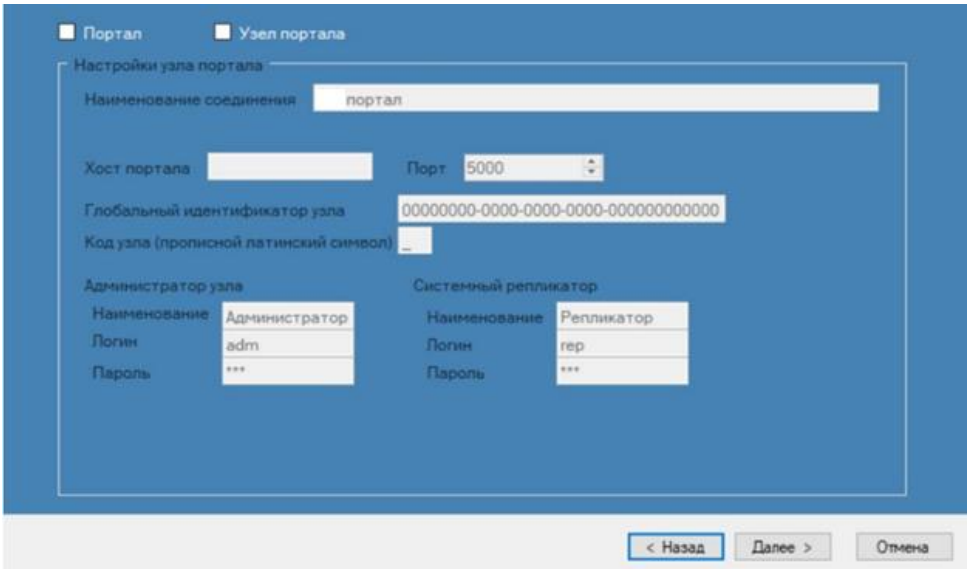


Рисунок 6

– Порт – порт для связи с порталом.

– Глобальный идентификатор узла – это уникальный идентификатор, который выдается на стороне портала для нового узла портала.

– Код узла – символ, идентифицирующий узел, уникальный в информационном пространстве.
– Администратор узла – это пользователь для администрирования узла.
– Системный репликатор – это системный пользователь, от имени которого запускаются фоновые задачи публикации и импорта.

В процессе установки системы могут быть установлены дополнительные компоненты – файлы документации и примеров (Рисунок 7).

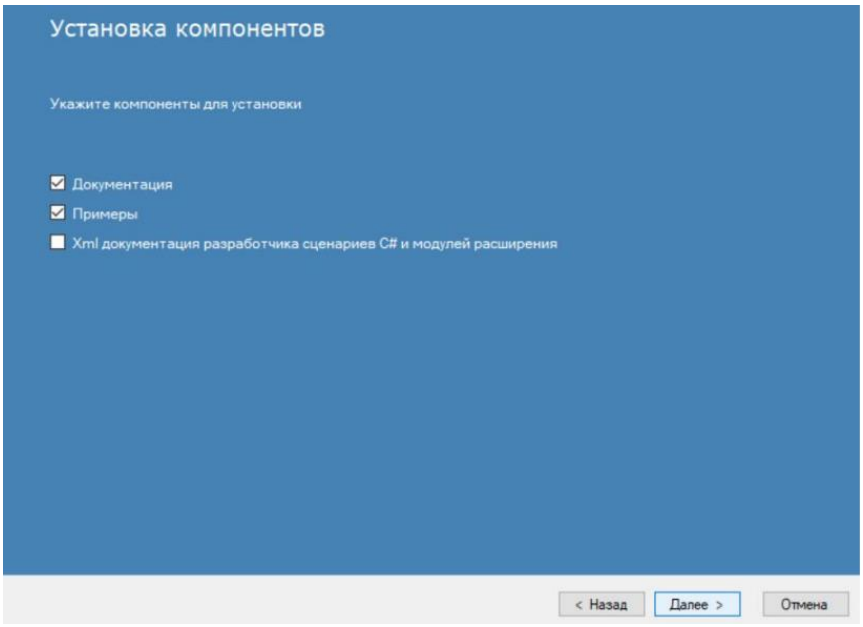


Рисунок 7

По нажатию кнопки «Далее» инсталлятор предложит выбрать вариант установки сервера решения – в виде консольного приложения или системной службы (Рисунок 8).

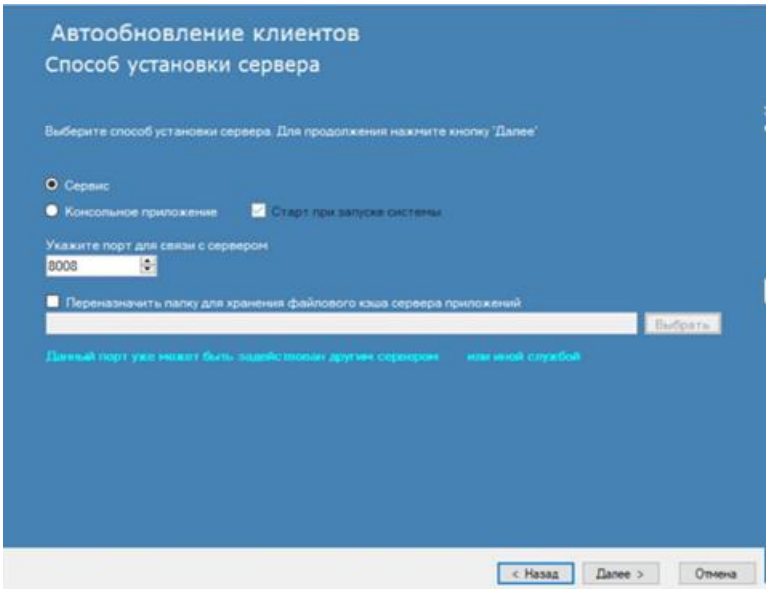


Рисунок 8

Вариант установки сервера решения в виде системной службы является рекомендуемым вариантом, так как не требует входа пользователя в операционную систему. Поле Порт для связи с сервером позволяет настроить номер порта, по которому клиенты будут устанавливать соединение с сервером.

Установка флажка «Переназначить папку для хранения файлового кэша сервера приложений» позволяет переназначить папку для хранения файлового кэша сервера приложений с папки по умолчанию на любую другую локальную папку дисковой системы сервера (Рисунок 9).

Затем инсталлятор предложит указать необходимость установки службы автоматического обновления клиентов на машинах, где планируется установить клиентскую часть (Рисунок 10).

Далее инсталлятор предложит указать тип СУБД, который будет использоваться сервером для доступа к базе данных (Рисунок 11).

При установке серверной части Системы можно указать некоторые настройки, которые будут применены к клиентам при их установке (Рисунок 12).

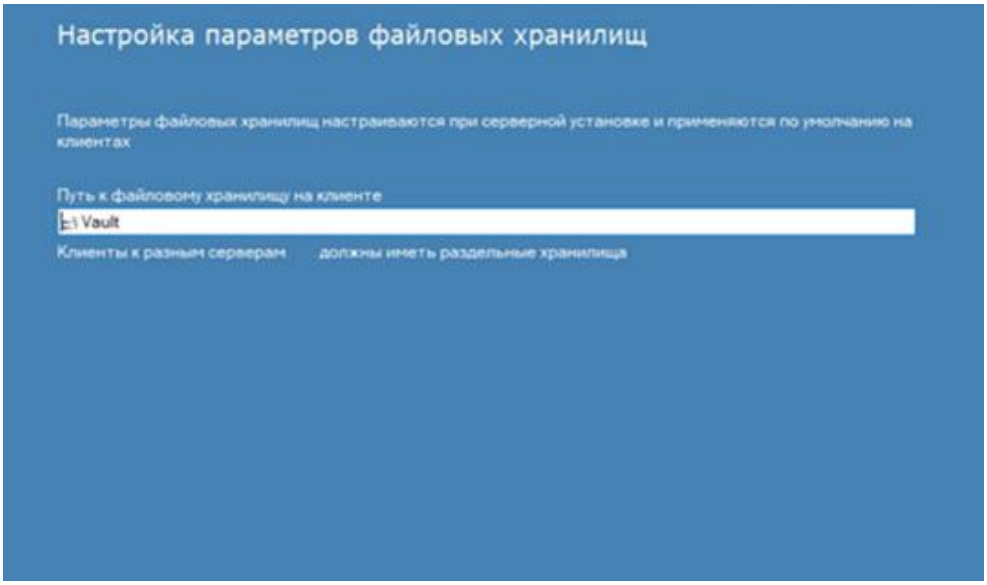


Рисунок 9

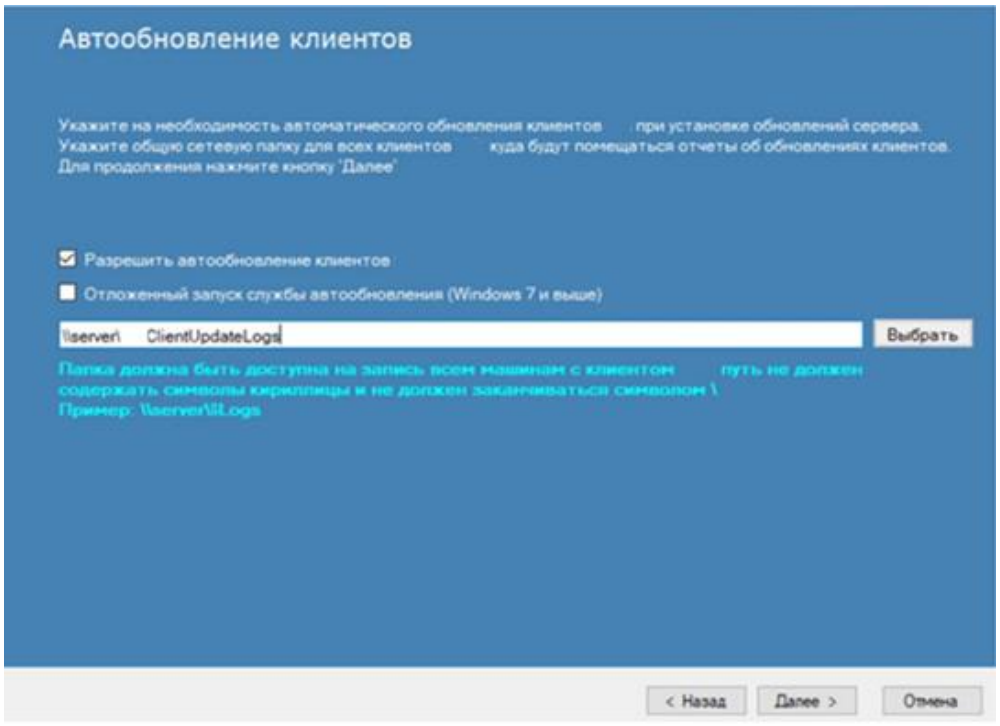


Рисунок 10

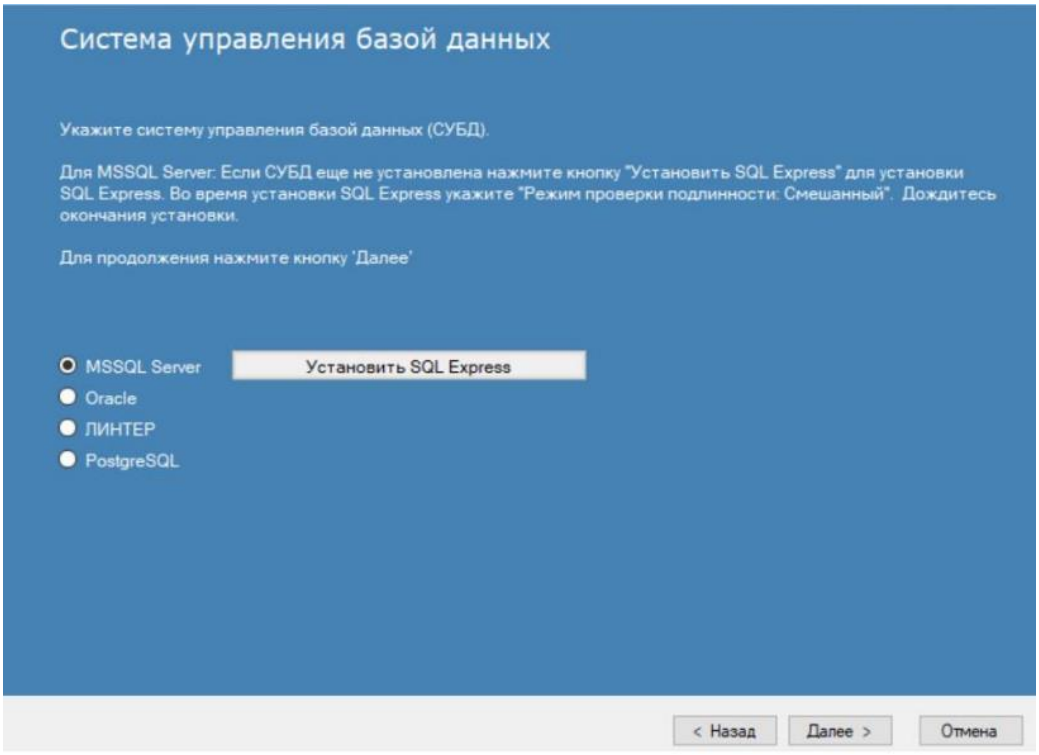


Рисунок 11

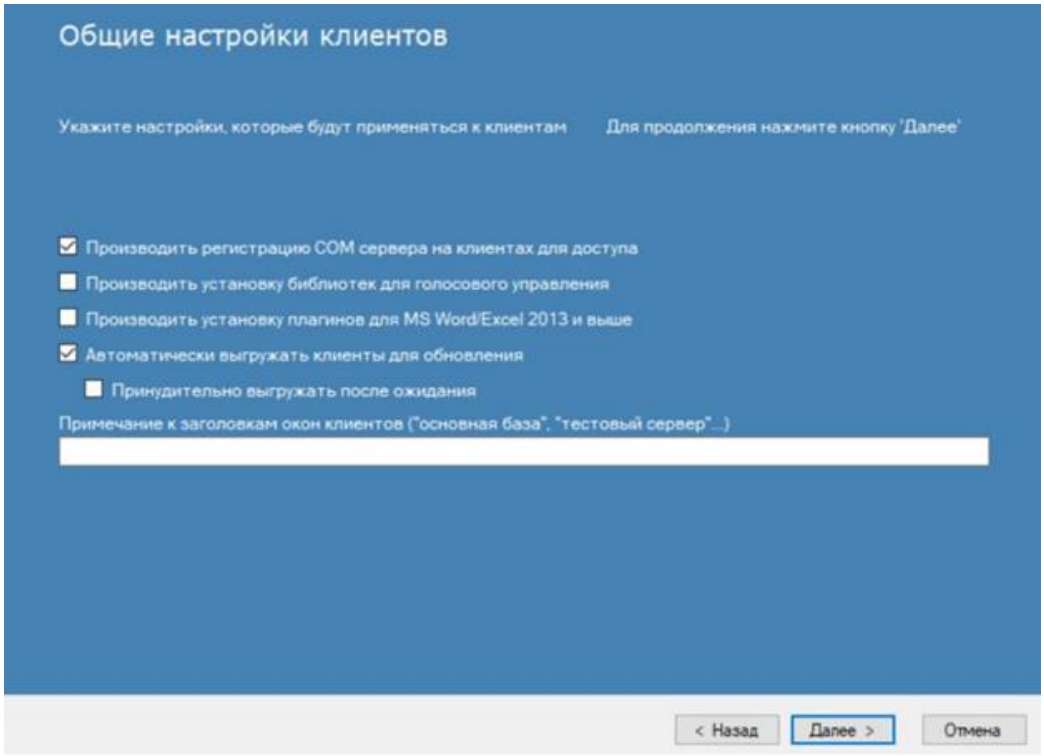


Рисунок 12

На рисунке 12 показаны следующие доступные настройки:

– Производить регистрацию COM сервера на клиентах для доступа – позволяет разрешить доступ к API на клиентских машинах. По умолчанию доступ разрешен.

– Производить установку библиотек для голосового управления – указывает на необходимость установки на клиентских машинах библиотек голосового управления Microsoft Speech Platform.

– Производить установку плагинов для MS Word/Excel 2013 и выше – параметр позволяет интегрировать в приложения пакета Microsoft Office команды взаимодействия с клиентами Системы.

– Автоматически выгружать клиенты для автообновления – параметр указывает службе автоматического обновления клиентов выполнять действия по периодической проверке необходимости автообновления, а также давать сигнал к закрытию клиентов для последующего проведения обновлений.

– Принудительно выгружать после ожидания – управляет процессом выгрузки клиентов из системы. Параметр может быть включен только одновременно с параметром «Автоматически выгружать клиенты для автообновления». При включении параметра, если обновляемые процессы клиента не смогли закрыться автоматически, то они будут закрыты принудительно. Следует иметь в виду, что при включении параметра может происходить потеря несохраненных данных в тех случаях, когда процесс не может закрыться автоматически из-за ожидания ответа на запрос сохранения, и при этом в течении нескольких минут не последовала реакция пользователя на этот запрос.

На следующем шаге установки следует указать на необходимость использования брокера подключений к серверам приложений Системы (Рисунок 13).

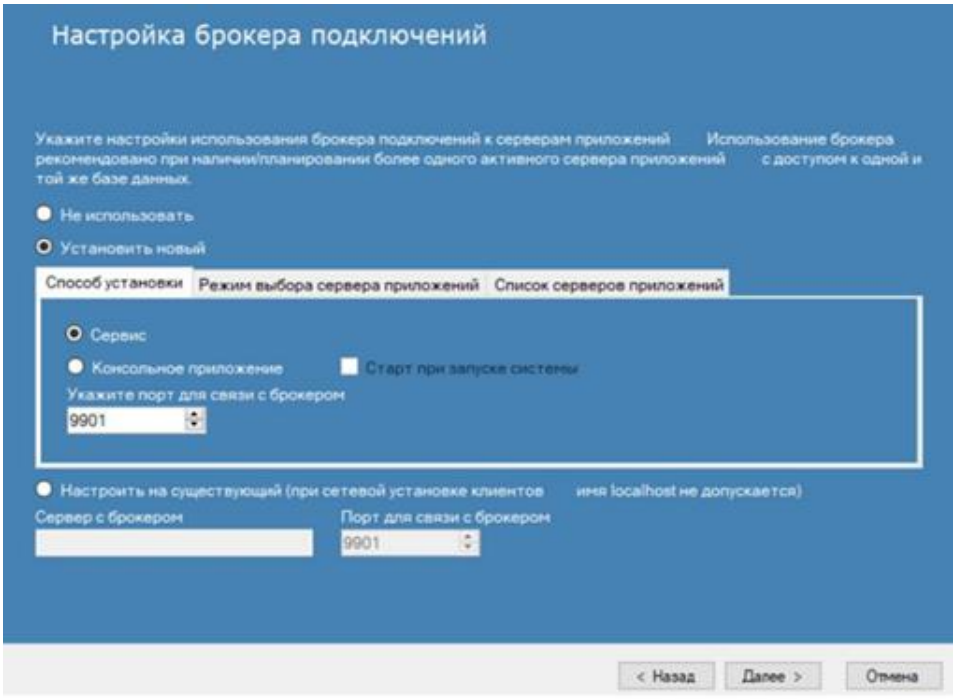


Рисунок 13

На данном этапе можно пропустить установку брокера, установить новый брокер, настроить сервер приложений на существующий брокер.

На следующем шаге установки необходимо указать папку куда будет выполнена установка серверной части Системы (Рисунок 14).

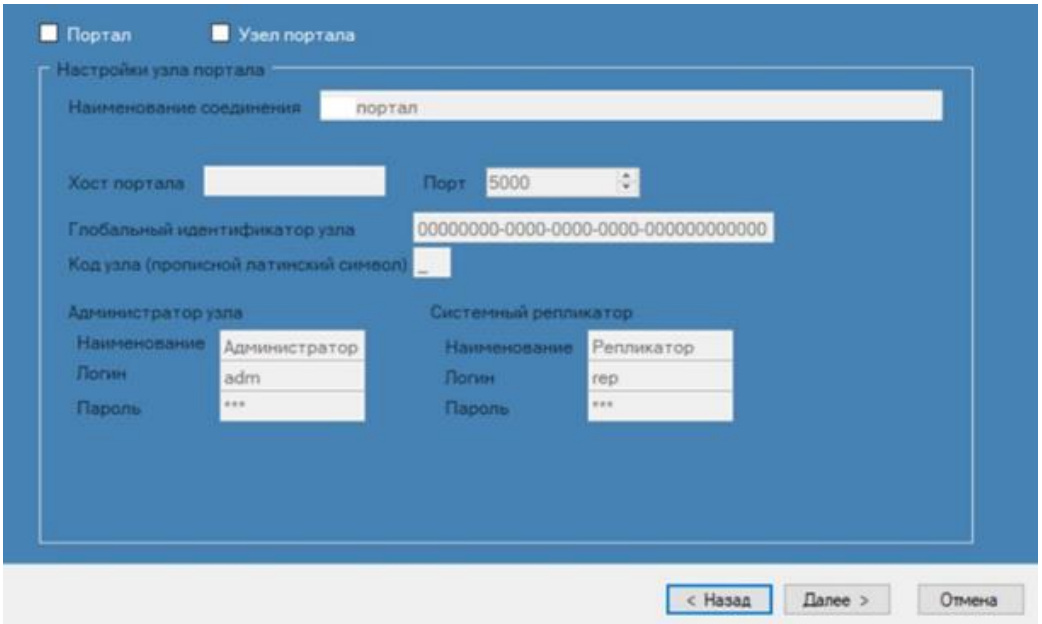


Рисунок 14

Далее программа установки предложит возможность настройки режима тихой установки клиентов Системы. В случае выбора варианта подготовки режима тихой установки клиентов следует нажать кнопку «Настройка» и

выполнить настройку параметров: только в этом случае станет доступна для нажатия кнопка «Далее». Тихая установка клиентов выполняется на клиентских машинах путем запуска клиентского инсталлятора при помощи командной строки и не требует взаимодействия с пользователем, что позволяет увеличить скорость развертывания клиентов в сети. Отчеты о выполнении тихой установки записываются в журнал событий операционной системы и файлы отчетов.

В следующем окне инсталлятор сообщит о готовности к процессу установки. Для запуска процесса установки необходимо нажать кнопку «Установить».

Процесс установки отображается в окне с указанием производимых действий и прогресса их выполнения. По окончании установки серверной части инсталлятор проинформирует об этом и предложит нажать кнопку «Готово» для выхода (Рисунок 15).

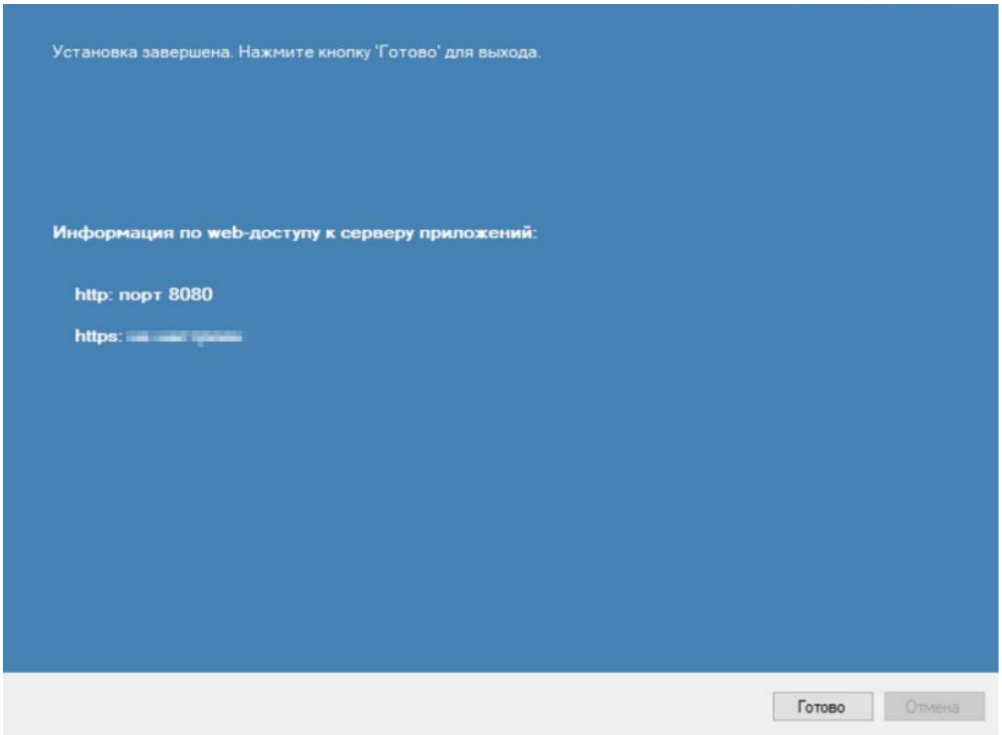


Рисунок 15

7.3 Установка компонента интеграции с CAD

Установка компонента интеграции с CAD включает установку серверной части и установку клиентской части.

После установки серверной части Системы можно выполнить установку серверной части компонента интеграции с CAD.

Установка серверной части компонента интеграции с CAD выполняется путем запуска установочного файла из состава инсталляционного пакета.

В процессе установки инсталлятор предложит указать папку для установки.

Папка, предлагаемая по умолчанию, является оптимальной с точки зрения последующей установки клиентской части, поэтому не рекомендуется изменять ее без крайней необходимости. После указания папки установки необходимо выбрать устанавливаемы компоненты.

После всех необходимых приготовлений программа будет готова установить серверную часть компонента интеграции с CAD.

Процесс установки отображается в окне с указанием производимых действий и прогресса их выполнения.

По окончании установки серверной части инсталлятор проинформирует об этом и предложит нажать кнопку «Готово» для выхода.

Клиентская часть компонента интеграции с CAD устанавливается после установки серверной части.

Внимание! Предполагается, что на клиентском ПК установлена CAD-система (КОМПАС-3D или Siemens NX).

Установка клиентской части компонента интеграции с CAD производится аналогично установке серверной части.

Программа установки предложит указать папку для установки.

После указания папки установки необходимо выбрать устанавливаемы компоненты.

Далее необходимо указать папку для создаваемых ярлыков.

После всех необходимых приготовлений программа будет готова установить клиентскую часть компонента интеграции с CAD.

Процесс установки отображается в окне с указанием производимых действий и прогресса их выполнения.

По окончании установки клиентской части инсталлятор проинформирует об этом и предложит нажать кнопку «Готово» для выхода.

8 Удаление программного обеспечения

8.1 Деинсталляция ПО выполняется при запуске файла Uninstall.exe из папки текущей установки ПО (Рисунок 16).

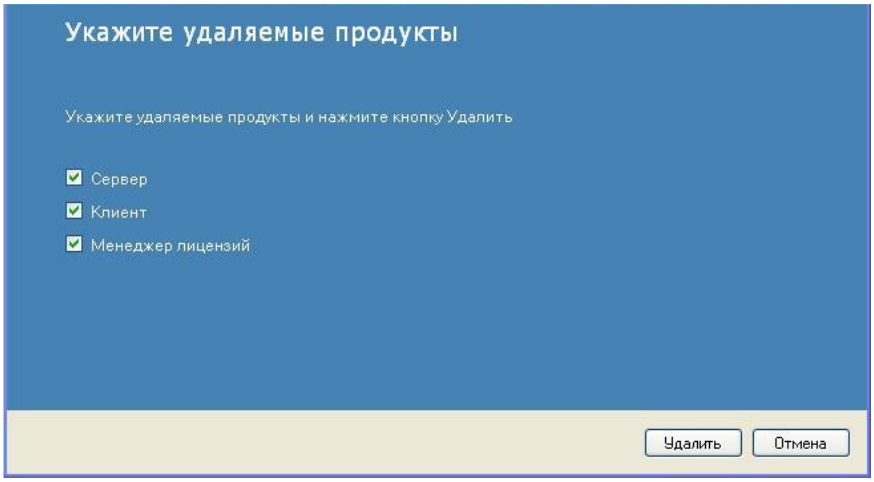


Рисунок 16

8.2 Программа деинсталляции спросит, что делать с общими файлами, установленными при развертывании ПО (Рисунок 17). Если на данной машине больше нет необходимых компонентов, то можно ответить «Да для всех».

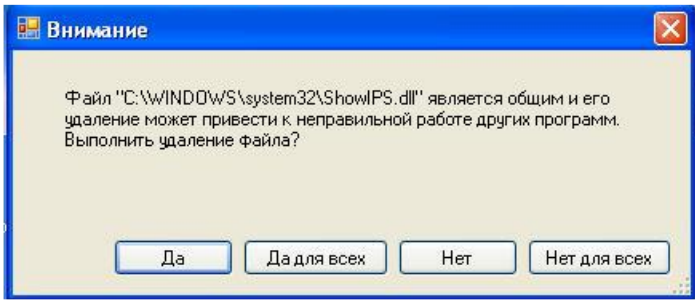


Рисунок 17

8.3 По окончании процесса программа деинсталляции выдаст уведомление.

8.4 Нажмите кнопку «Готово» и процесс удаления ПО будет завершен.

9 Техническая поддержка

9.1 Техническая поддержка включает:

- гарантийную поддержку;
- дополнительную техническую поддержку.

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

9.3 Регламенты технической поддержки описаны в документе «Корпоративная система управления и обмена данными. ПЛАТФОРМА – Конструкторская подготовка производства (КСУОД.ПЛАТФОРМА-КПП). ».

9.4 При возникновении вопросов по установке, требующих консультации специалистов технической поддержки, обращения принимаются:

- Телефон: +7 495 514-14-14
- Электронная почта: ssa@ksuod.ru

Приложение 1. Методика расчёта характеристик серверов

П1.1 Расчет количества ядер

Значение N необходимого количества ядер (типа Xeon Gold, Platinum или их аналогов), выделяемого для виртуальной машины, на которой развертывается сервер приложений и сервер БД, определяется по следующей формуле:

$$N = P * S,$$

где:

P – количество одновременно работающих пользователей в системе.

S – количество ядер на одного пользователя системы.

Значение P рассчитывается на основании значений общего количества пользователей, зарегистрированных в системе (Робщ), и коэффициента одновременного использования (k):

$$P = Робщ * k,$$

где k = 0,5…1,

k=1 – если систему одновременно используют 100% пользователей;

k=0,5 – если систему одновременно используют 50% пользователей.

Значение S определяется исходя из следующего правила:

- для сервера приложений:
- оптимальное значение S=0,2 (1 ядро на 5 пользователей).
- минимальное значение S=0,07 (1 ядро на 15 пользователей).
- для сервера БД:
- оптимальное значение S=0,05 (1 ядро на 20 пользователей).

Пример расчета.

Дано:

Робщ=1000;

Максимальное значение одновременного использования системы пользователями составляет 60% (k=0,6).

Расчет:

Оптимальное количество ядер для сервера приложений:

$$Nрек = P * S = Робщ * k * S = 1000 * 0{,}6 * 0{,}2 = 120 \text{ ядер}$$

Минимальное количество ядер:

$$Nmin = P * S = Pобщ * k * S = 1000 * 0{,}6 * 0{,}07 = 42 \text{ ядра}$$

Оптимальное количество ядер для сервера БД:

$$Nрек = P * S = Pобщ * k * S = 1000 * 0{,}6 * 0{,}05 = 30 \text{ ядер}$$

П1.2 Расчет объема оперативной памяти

Расчетное значение объема оперативной памяти (в ГБ) определяется по следующей формуле:

$$V = P * R,$$

где:

P – количество одновременно работающих пользователей в системе.

R – объем оперативной памяти на одного пользователя системы.

Значение R определяется исходя из следующего правила:

- для сервера приложений:
- оптимальное значение R=0,2ГБ (1ГБ на 5 пользователей).
- минимальное значение R=0,07ГБ (1ГБ на 15 пользователей).
- для сервера БД:
- оптимальное значение R=0,05ГБ (1ГБ на 20 пользователей).

Пример расчета.

Дано:

Робщ=1000;

Максимальное значение одновременного использования пользователями составляет 60% (k=0,6).

Расчет:

Оптимальное значение объема оперативной памяти для сервера приложений:

$$Vрек = P * R = Pобщ * k * R = 1000 * 0{,}6 * 0{,}2 = 120 \text{ Гб}$$

Минимальное значение объема оперативной памяти:

$$Vmin = P * R = Pобщ * k * R = 1000 * 0{,}6 * 0{,}07 = 42 \text{ Гб}$$

Оптимальное значение объема оперативной памяти для сервера БД:

$$Vрек = P * R = Pобщ * k * R = 1000 * 0{,}6 * 0{,}05 = 30 \text{ Гб}$$

Приложение 2. Настройка рабочих узлов

П6.1 На всех узлах необходимо:

– Установить уровень безопасности «Орел»;
– Настроить статический IP;
– Настроить разрешение DNS;
– Отключить Firewall;
– Настроить маршрут по умолчанию.

П6.2 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
```

П6.3 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
```

П6.4 Установить ***Docker*** и ***Docker Compose*** командами:

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

П6.5 Перелогиниться в текущей сессии, (чтобы *Docker*-команды работали без *sudo*) следующим образом:

```
sh
logout
```

П6.6 Выполнить вход в частный реестр для каждого узла кластера (ETCD, Control Plane, Worker):

```
sh
```

Docker login Docker-plm.ru

П6.7 Ввести логин и пароль для подключения к реестру.

Приложение 3. Настройка портов

П3.1 Порт 6443 узла сервера ***Rancher*** должен быть доступен всем узлам.

П3.2 Все узлы должны быть доступны друг другу по протоколу UDP и порту 8472. Данный порт используется модулем CNI (Container Network Interface) Flannel и Canal.

П3.3 Если используется сервер метрик, нужно на всех узлах открыть порт 10250.

П3.4 Проверить firewall:

sh
sudo ufw status

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

sh
sudo ufw disable

П3.5 Для того чтобы открыть порты нужно отредактировать файл настроек *iptables*:

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

П3.6 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

П3.7 Сохранить изменения. Затем выполнить команду:

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

П3.8 Входящие правила для узлов ***Rancher Server*** приведены в **Ошибка! Источник ссылки не найден.**, исходящие – в Таблица П3.**2**.

Таблица П3.1 – 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`kubelet` ***Rancher***	

Таблица П3.2 – Исходящие правила для узлов ***Rancher Server***

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

П3.9 Порты узлов сервера ***Rancher Server***

Для трафика между узлами ***Rancher Server*** используются порты в соответствии с **Ошибка! Источник ссылки не найден.** П3.0.

Таблица П3.3. – 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

П3.10 Узлы кластера ***Kubernetes***

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

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

П6.1 Порты, открываемые на узлах ***Kubernetes***, приведены в таблице П3.0.

Таблица П3.4. – 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)

Приложение 4. Настройка репозитория манифестов

П4.1 Автоматическое развёртывание ресурсов инфраструктуры и приложений обеспечивается репозиторием манифестов. Манифест – специальный файл, содержащий описание ресурсов.

П4.2 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.

П4.3 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.

П4.4 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 П4.1.

Таблица П4.1. – 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

П4.5 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.

П4.6 Каталог пакета ***Fleet***

П4.6.1 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***.

П4.6.2 Файл ***fleet.yml*** содержит настройки всех кластеров, в которые должны устанавливаться ресурсы, описанные в манифестах ***Kustomize***, находящихся в том же каталоге и его подкаталогах.

П4.6.3 Каждый каталог, в котором находится файл ***fleet.yml***, является пакетом ***Fleet*** (***Fleet Bundle***).

П4.6.4 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***.

П4.6.5 Некоторые пакеты ***Fleet*** развёртываются через менеджер пакетов ***Helm***. В этом случае каждый каталог пакета ***Helm*** (***Helm Chart***) является отдельным пакетом (bundle) ***Fleet***.

П4.6.6 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***.

П4.7 Настройка кластера в репозитории ***ksuod-ci***

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***.

П4.8 Редактирование настроек кластера (применительно к пакетам Fleet):

П4.8.1 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).

П4.8.2 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

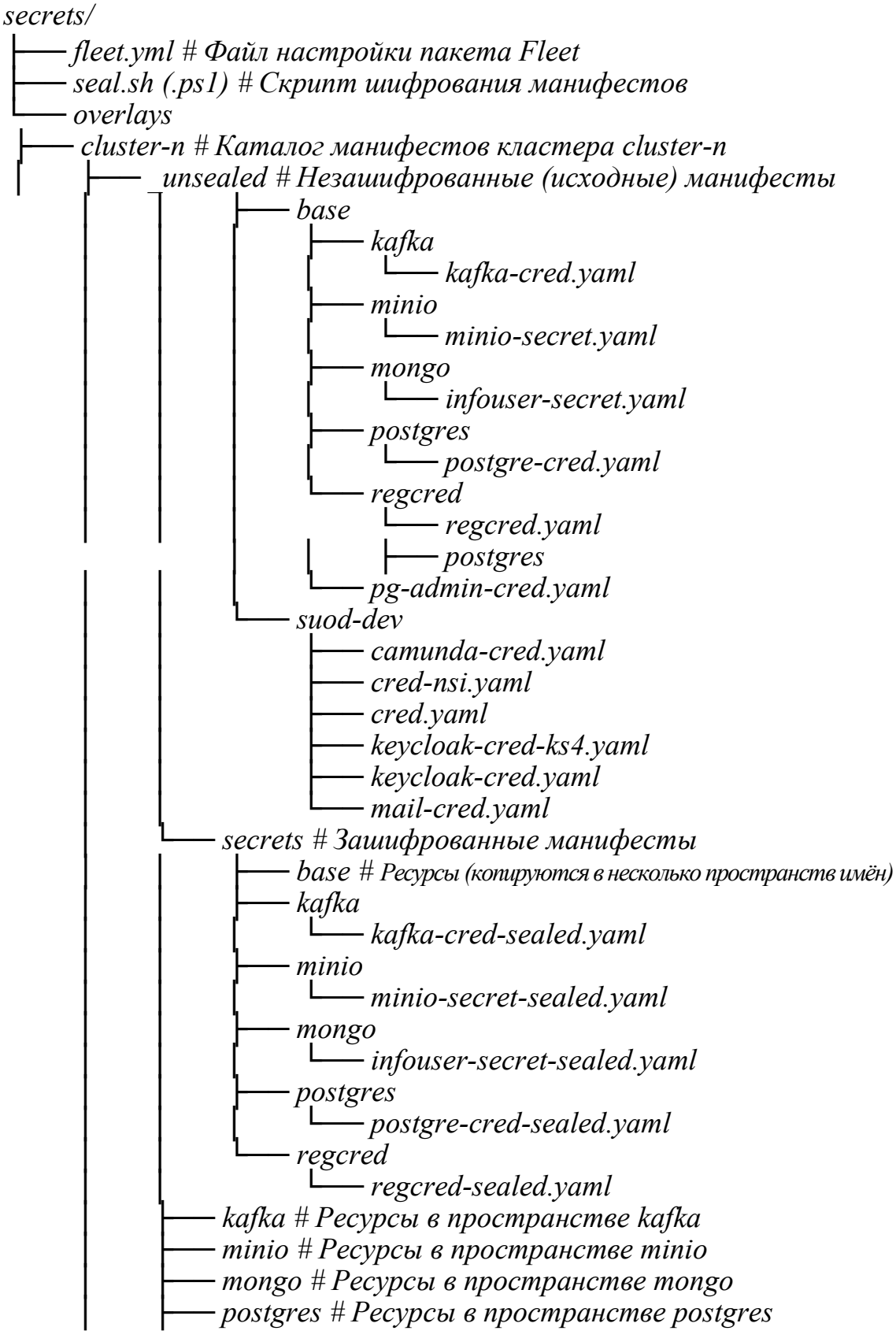
7. Назначить права (*chmod* 755) на все каталоги томов и назначить владельцев для томов с множественным доступом:

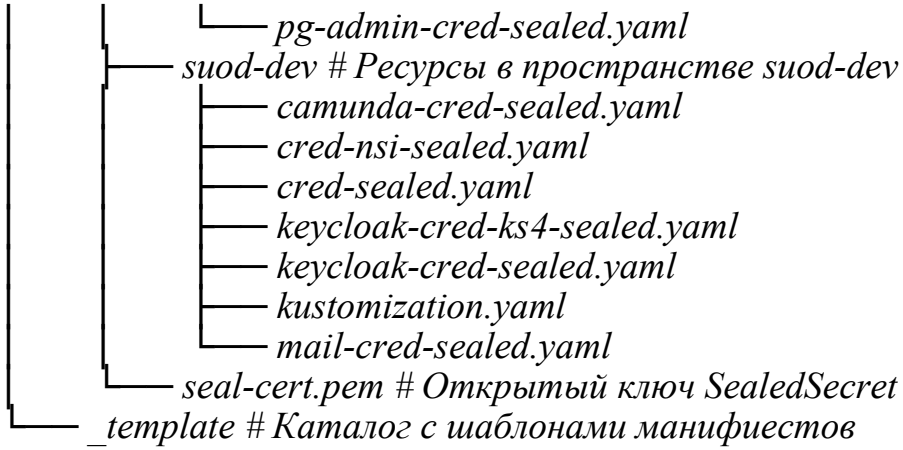
chown:2000 -R /mnt/volumes/dev-tcfms
chown:2000 -R /mnt/volumes/dev-tcfms-direct/
chown:101 -R /mnt/volumes/dev-camunda/

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

П4.8.3 secrets

Структура каталогов пакета *secrets*:





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 5).

Таблица 5. – 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.

П4.8.4 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***.

П4.8.5 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.

П4.8.6 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

П4.8.7 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

П4.9 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].

Приложение 5. Настройка реестра образов Docker

П5.1 Развернутый реестр образов ***Docker*** должен быть доступен для всех узлов кластера ***Kubernetes***.

П5.2 Сбор и загрузка образов в реестр ***Docker*** выполняется из установочного архива.

П5.3 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*

П5.4 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

П5.5 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.

Приложение 6. Установка и настройка *Kubernetes*

П6.2 Подготовка каталога с образами

П6.1.1 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***

П6.1.2 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 `#`.

П6.3 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.

П6.4 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`.*

Приложение 7. Установка и настройка Rancher

П7.1 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

П7.2 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.kalashnikovconcern.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.

П7.3 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.kalashnikovconcern.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`

Приложение 8. Развертывание кластера Kubernetes

П8.1 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.

П8.2 На каждом узле кластера ***Kubernetes*** (master/worker) выполнить создание директории:

sudo mkdir -p /mnt/volumes/volume

П8.3 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.

П8.4 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.

Приложение 9. Установка контроллера Sealed Secrets

П9.1 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.

П9.2 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.

Приложение 10. Установка утилиты Kubeseal

П10.1 Утилита ***Kubeseal*** предназначена для шифрования секретов в ***Kubernetes***

П10.2 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:

П10.3 Распаковать архив `*kubeseal-0.29.0-windows-amd64.tar.gz*` и скопировать утилиту в любой доступный каталог.

П10.4 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

П10.5 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.*

Приложение 11. Настройка GitOps Fleet

П6.5 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***.

П6.6 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.

П6.7 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