УТВЕРЖДЕН
RU.31465715.01000-01 34 01-ЛУ

# ПРОГРАММНЫЙ КОМПЛЕКС СЕРВИС ТОЧКА

## Руководство системного программиста

## Техническая архитектура. Установка и конфигурирование

**RU.31465715.01000-01 32 01-2**

**Листов 14**

2025

# АННОТАЦИЯ

Настоящий документ является руководством системного программиста на Программный комплекс Сервис Точка. Также документ распространяется на иные сервисы АО «Точка». В значении ПК Точка и иные сервисы АО «Точка» в тексте именуются как «программа».

Руководство системного программиста состоит из следующих частей:

– «Сведения о технических средствах, обеспечивающих выполнение программы. Аварийные ситуации»;

– «Техническая архитектура. Установка и конфигурирование».

Настоящая часть содержит сведения о технической архитектуре, описание установки и конфигурирования программ и предназначена для должностных лиц, занимающихся установкой и поддержанием их работоспособности.

Целью документа является ознакомление с методами решения задач администрирования и описание технологических процессов поддержки.

**СОДЕРЖАНИЕ**

## СОКРАЩЕНИЯ, ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

| | |
|---|---|
| API | Application Programming Interface (программный интерфейс приложения) |
| ELK | Стек проектов Elasticsearch, Logstash и Kibana |
| ESB | Enterprise Service Bus (корпоративная сервисная шина) |
| АО «Точка» | Акционерное общество «Точка» |
| БД | База данных |
| деплой | Развертывание программного обеспечения |
| ПК Точка | Программный комплекс Сервис Точка |
| ОС | Операционная система |
| ПО | Программное обеспечение |

*Примечание. Определения, не содержащиеся в настоящем разделе и использующиеся по тексту, имеют значения, установленные для таких определений в документе «Термины и определения», офертах https://tochka.com/offer/ib/ и сети интернет.*

# 1 ПЕРСОНАЛ

## 1.1 Требования к персоналу

Количество персонала, обеспечивающего штатную эксплуатацию программы, а также квалификация зависят от приобретенных лицензий и модулей.

Данный персонал должен обладать высоким уровнем квалификации и практическим опытом выполнения работ по установке, настройке и администрированию всей системы.

Главный системный администратор должен обладать уровнем компетенций, который позволяет управлять общими процессами, выполнять контроль, обеспечивать правильность выполнения работ, принимать архитектурные решения.

## 1.2 Обязанности персонала

Основными обязанностями главного системного администратора являются:

- контроль общего функционирования системы;
- анализ системных проблем;
- принятие архитектурных решений;
- консультирование системных администраторов программ;
- контроль и правильность выполнения работ системными администраторами программ;
- разработка предложений по развитию системы;
- установка, настройка и мониторинг работоспособности ПО.

Основными обязанностями системного администратора программы и администратора БД являются установка, настройка и мониторинг работоспособности программы, а также стороннего ПО, необходимого для установки и работы программы. Перечень стороннего ПО приведен в первой части руководства системного программиста «Сведения о технических средствах, обеспечивающих выполнение программы. Аварийные ситуации».

Системный администратор программы должен выполнять полное техническое обслуживание в части администрирования, вверенной ему программы. Роль

администратора БД и системного администратора программы могут быть совмещены в одну.

# 2 УСЛОВИЯ ВЫПОЛНЕНИЯ ПРОГРАММЫ

Для обеспечения функционирования программы необходимо выполнить действия:

– произвести установку, настройку и запуск программы, а также инфраструктуры (выполняется при первичной установке и при установке обновлений). Описание порядка действий приведено в пунктах ниже, см. 3.1 и см. 3.2;

– выполнить процедуру мониторинга работоспособности.

Для выполнения этих задач используется рабочее место с персональным компьютером, имеющее подключение к внутренней и внешней сети, а также к системе хранения контроля версий GitLab. На персональном компьютере должна быть предустановлена ОС Windows или Linux, а также следующее ПО:

– браузер Mozilla Firefox или Google Chrome версии не ранее 2019 г.;

– текстовый редактор с поддержкой Unicode (UTF-8/UTF-16).

# 3 ТЕХНИЧЕСКАЯ АРХИТЕКТУРА

## 3.1 Общее описание архитектуры

Техническая архитектура программ представляет собой совокупность компонентов программы и систем, обеспечивающих ее эффективное функционирование. Взаимодействие программ выполняется с использованием корпоративной сервисной шины или API в зависимости от типа обмена.

Верхнеуровневая системная архитектура программ регламентируется сетевыми и доменными архитектурными решениями, а также правилами и способами взаимодействия компонентов, входящих в программу по архитектурным паттернам, описанным ниже.

## 3.2 Сетевая архитектура

В базовой сетевой архитектуре заложен принцип разделения программы на слои (среды), с явным запретом общения различных компонентов программы между разными слоями:

– продуктивная среда – предназначена для работы клиентов;

– препрод – среда, идентичная или максимально приближенная к продуктивной: имеет те же данные, аппаратно-программное окружение и производительность. Она используется, чтобы сделать финальную проверку программы в условиях максимально приближенным к реальности;

– среда тестирования – предназначена для тестирования и отладки взаимодействия систем на этапе разработки.

## 3.3 Доменная архитектура

Программы состоят из компонентов, декомпозированных по правилам принадлежности к бизнес-области, единой ответственности или с целью мультисистемного использования.

Взаимодействие программ осуществляется через систему ESB, реализующую одноименный архитектурный паттерн асинхронного взаимодействия.

### 3.4 Асинхронный обмен

Для централизованного и унифицированного асинхронного обмена сообщениями между программами используется ESB.

ESB использует Apache Camel в качестве интеграционного фреймворка и Apache Kafka в качестве внутреннего брокера сообщений.

ESB состоит из двух подсистем:

- узлы маршрутизации (получение сообщений от программ-отправителей, проверка разрешений, вычисление маршрутов, передача сообщений программам-получателям);
- внутренний брокер сообщений.

Для работы с ESB программы, как правило, используют промежуточный брокер сообщений RabbitMQ.

### 3.5 Правила формирования сообщений

При формировании сообщений используются следующие правила:

- формат сообщений: XML;
- стиль написания имен типов сообщений: lowerCamelCase;
- формат даты в сообщениях:
  - без времени: YYYY-MM-DD;
  - со временем: YYYY-MM-DDTHH:MI:SS[.FF3]TZ, где TZ – часовой пояс в формате [+-][0-9]{2}:[0-9]{2};
- состав сообщения:
  - заголовки – метаданные для идентификации и маршрутизации;
  - тело сообщения – данные, основная информационная часть;
  - размер одного сообщения: не более 512 Кб.

В ESB используются два подхода к передаче сообщений:

- уведомление о событиях;
- запрос/ответ.

### 3.5.1 Уведомление о событиях

Это сообщение о случившемся факте или распространение состояния объекта. Не предполагает ответа. В программе-отправителе нет информации о том, какие программы получат событие.

События одного типа отправляются только одной программой – той, которая имеет статус основной для хранения объекта (мастер-система). Получать события одного типа может неограниченное количество программ.

Рекомендуемые варианты имен событий:

– сущность, к которой относится событие (например: bank, counter, tariffPlan, userPermission);

– сущность и дополнительная информация о сущности или событии (например: accountBooked, deliveryStatusNotification, p2pTransactionComplete, workCalendarDayChange).

### 3.5.2 Запрос/ответ

Это отправка запроса, на который нужно получить ответ, или отправка команды на выполнение действия. Предполагает ответ. В программе-отправителе нет информации о том, какая программа ответит на запрос.

На запросы одного типа отвечает одна программа. Отправлять запросы одного типа может неограниченное количество программ.

Рекомендуемые варианты имен запросов:

– действие, которое необходимо выполнить (например: notifyClientWithReaction);

– действие, которое необходимо выполнить, и имя сущности (например: activateAccount, changeDirInfo, deleteTimeline, saveCustomer);

– действие, которое необходимо выполнить, имя сущности и дополнительная информация о сущности или действии (например: addFilesToDossier, createPaymentForSign, updateProductsFor1c).

### 3.6 Синхронный обмен

Для синхронного обмена сообщениями между компонентами программы используется сервис ApiBank и API самой программы, если подобный имеется.

# 4 УСТАНОВКА И КОНФИГУРИРОВАНИЕ

## 4.1 Развертывание программ с использованием технологии кластеризации

Типовая установка и развертывание в кластере Kubernetes, включая установку новой версии, основывается на принципах CI/CD, которые включают в себя автоматизацию всех процессов развертывания программы.

Исходные коды программ опубликованы в репозитории GitLab. Сборка выполняется автоматически при обновлении исходных кодов в определенные ветки репозитория.

Результатом выполнения сборки является docker-image. Для настройки автоматизации необходимо:

– создать конфигурационный файл .gitlab-ci.yml в репозитории программы (адрес уточнить у системного администратора соответствующей программы), содержащий настройки сборки, опубликования, создания докер-образа, деплоя;

– создать докер-файл Dockerfile, содержащий инструкцию построения используемого для контейнеризации образа программы (расположен в папке /var/lib/docker/);

– задать параметры публикации докер-образа в Harbor в setting/CI/CD/Variables (запросить у системного администратора), см. Рисунок 1;

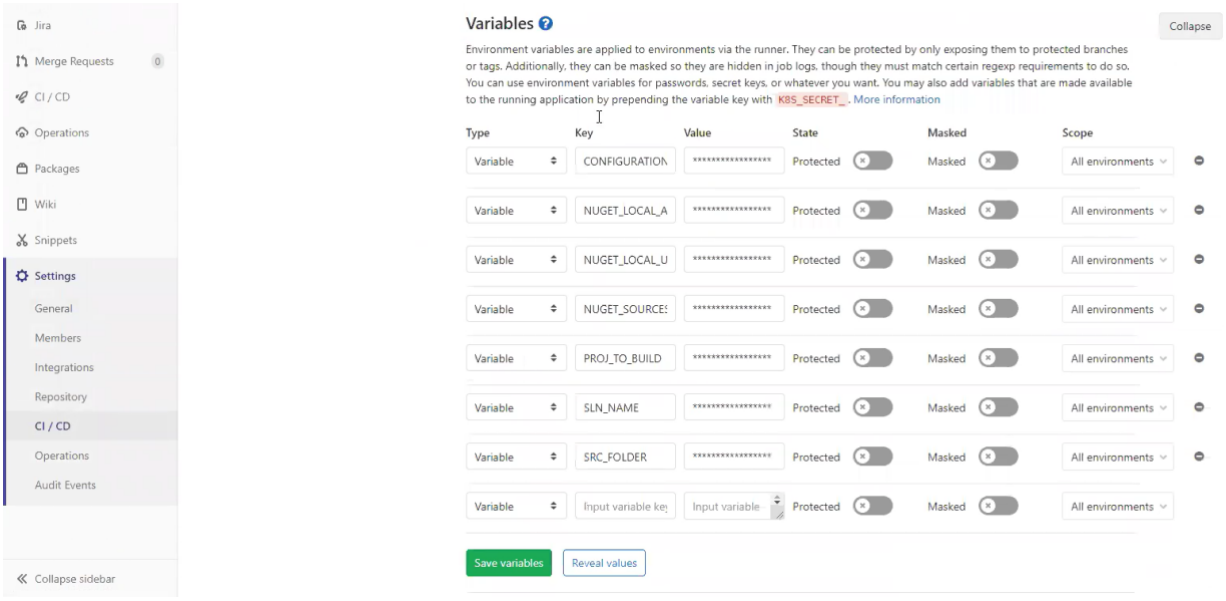


Рисунок 1 – Переменные CI/CD

– создать в необходимой папке файлы chart.yml (инструкция по установке), values-dev.yml (переменные окружения, балансировщик и т. д.), папку templates, создать в ней шаблоны ingress.yml (события, происходящие после развертывания балансировщика), deploent.yml (процесс деплоя для Kubernetes), service.yml (порты подключения);

– в settings/CI/CD/general pipelines определить, что автоматическая установка и развертывание начинается по команде merge request, и указать путь к конфигурационному файлу .gitlab-ci.yml), см. Рисунок 2.

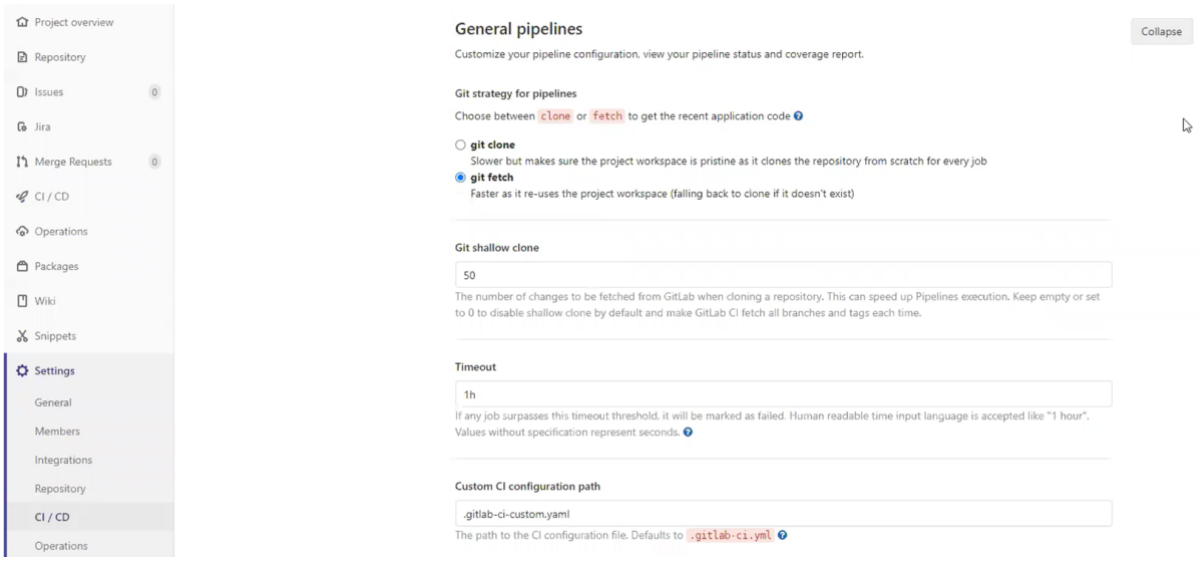


Рисунок 2 – Настройка pipelines

Собранные образы расположены в папке /var/lib/docker/volumes.

Для сборки и деплоя приложений в различные кластеры можно использовать различные ветки репозитория или использовать автоматический деплой.

Также возможна ручная установка и развертывание. Для этого необходимо:

– перейти в репозиторий программы в GitLab, см. Рисунок 3;


Рисунок 3 – Интерфейс типовой программы в GitLab

– перейти в раздел Pipelines, выбрать необходимую ветку и запустить сборку командой Run Pipeline, см. Рисунок 4.

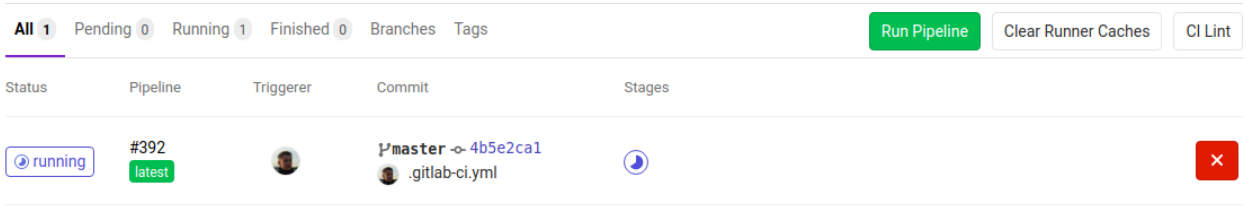


Рисунок 4 – Выбор ветки в Pipelines

## 4.2 Мониторинг работоспособности

Мониторинг работоспособности и отсутствия ошибок осуществляется путем анализа логов работы программы на платформе Graylog (в отдельных случаях используется ELK). Graylog собирает и извлекает важные данные из логов сервера, которые обычно отправляются с помощью протокола Syslog. Также Graylog позволяет искать и визуализировать логи в веб-интерфейсе.

В зависимости от характера сообщения подразделяются на следующие уровни:

– Emergency (0 уровень);
– Alert (1 уровень);
– Critical (2 уровень);
– Error (3 уровень);
– Warning (4 уровень);
– Notice (5 уровень);
– Informational (6 уровень);
– Debug (7 уровень).

Системному администратору стоит обратить внимание на ошибки с первого по четвертый уровень.

Для обращения к платформе необходимо запросить у системного администратора адрес и перейти по нему. Далее необходимо авторизоваться, используя доменные данные. Для поиска сообщений работы программы определенного уровня необходимо в строке поиска написать запрос «facility:<название программы> AND level:n», где n – уровень сообщения.