АКС Комплекс

Описание технической архитектуры

Программный комплекс автоматизации и контроля процесса подбора и учета продукции «АКС Комплекс»

Версия: 1.0
Дата: 27.10.2026
Правообладатель: ИП Краснов Владимир Андреевич
ИНН: 781422796858 · **ОГРНИП:** 320784700096770
Свидетельство: № 2025687159

1. Общие сведения

Настоящий документ описывает техническую архитектуру программного комплекса автоматизации и контроля процесса подбора и учета продукции «АКС Комплекс» (далее — Комплекс).

Комплекс построен по микросервисному принципу: ядро диспетчеризации, пользовательский интерфейс и сервисы управления оборудованием являются независимыми компонентами, взаимодействующими по HTTP/REST и WebSocket. Все компоненты развёртываются в Docker-контейнерах в контуре заказчика.

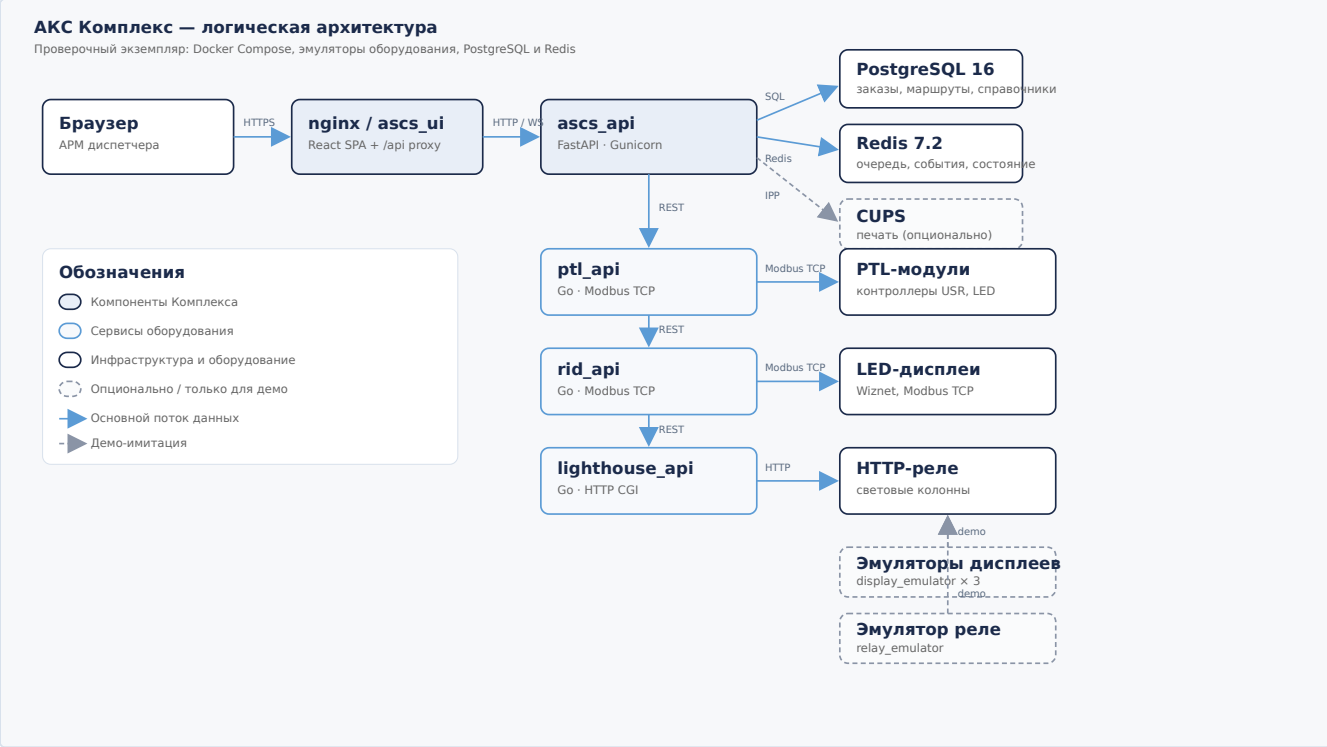


Рисунок 1. Логическая архитектура проверочного экземпляра

2. Компоненты

2.1. `ascs_api` — ядро Комплекса

Параметр	Значение
Язык	Python 3.11
Фреймворк	FastAPI
WSGI/ASGI	Gunicorn + Uvicorn workers
База данных	PostgreSQL 16
Кэш и очередь	Redis 7.2
API	REST/JSON, OpenAPI/Swagger

Параметр	Значение
Real-time	WebSocket
Порт	8000 (внутри Docker-сети)

Основные функции:

- приём и импорт заказов (CSV, REST);
- маршруты, очередь и слоты наборки;
- бизнес-логика секций, акцепторов и опта;
- управление PTL, дисплеями, колоннами и принтерами;
- формирование печатных документов;
- журнал событий, статистика, тепловая карта;
- интеграционные webhooks и API для WMS/ERP.

2.2. `ascs_ui` — APM диспетчера

Параметр	Значение
Язык	TypeScript
Фреймворк	React 19
Сборка	Vite
Сервер	nginx
Порт	80 (внутри контейнера), 8080 на хосте
API	same-origin `/api` → `ascs_api`
Real-time	WebSocket

nginx выполняет две функции: раздаёт статику SPA и проксирует `/api` (включая WebSocket) на `ascs_api`. Благодаря same-origin схема UI работает на любом IP без пересборки.

2.3. `ptl_api` — управление PTL

Параметр	Значение
Язык	Go
Протокол оборудования	Modbus TCP, порт 502
Состояние	Redis
API	REST/JSON, WebSocket
Метрики	Prometheus

Параметр	Значение
Порт	8000

Функции: включение/выключение модулей, цвета и числа, массовые операции, назначение адресов, приём нажатий, сценарии, демо-режим.

2.4. `rid_api` — управление дисплеями

Параметр	Значение
Язык	Go
Протокол оборудования	Modbus TCP, порт 502
Состояние	Redis
API	REST/JSON, WebSocket
Метрики	Prometheus
Порт	8000

Функции: вывод текста, цвет, яркость, автообнаружение, CSV-импорт, история, демо-программы.

2.5. `lighthouse_api` — управление световыми колоннами

Параметр	Значение
Язык	Go
Протокол оборудования	HTTP CGI
Состояние	Redis
API	REST/JSON, WebSocket
Метрики	Prometheus
Порт	8010

Функции: управление цветами, привязка к секциям, desired-state reconciliation, мониторинг реле, история, демо-режим.

2.6. Эмуляторы

Компонент	Назначение
`display_emulator`	Имитация LED-дисплея по Modbus TCP и HTTP UI
`relay_emulator`	Имитация HTTP-реле с 16 выходами и HTTP UI

Эмуляторы используются только в проверочном экземпляре и демо-стендах; в промышленной эксплуатации вместо них подключается реальное оборудование.

2.7. Инфраструктура

Компонент	Назначение
PostgreSQL 16	Основная база данных
Redis 7.2	Очередь, события, состояние оборудования, кэш
nginx	Раздача UI и reverse proxy
CUPS	Печать документов (опционально)
Prometheus	Сбор метрик (опционально, внешняя система)

3. Протоколы и порты

От	К	Протокол	Порт
Браузер	nginx / `ascs_ui`	HTTPS/HTTP	8080
nginx	`ascs_api`	HTTP, WebSocket	8000
`ascs_api`	PostgreSQL	PostgreSQL wire protocol	5432
`ascs_api`	Redis	RESP	6379
`ascs_api`	`ptl_api`	HTTP/REST	8000
`ascs_api`	`rid_api`	HTTP/REST	8000
`ascs_api`	`lighthouse_api`	HTTP/REST	8010
`ptl_api`	PTL-контроллеры	Modbus TCP	502
`rid_api`	LED-дисплеи	Modbus TCP	502
`lighthouse_api`	HTTP-реле	HTTP CGI	80/8080
`ascs_api`	CUPS	IPP	631
`ptl_api` / `rid_api` / `lighthouse_api`	Prometheus	HTTP	8000/8010

4. Потоки данных

4.1. Импорт заказов

1. WMS/ERP выгружает CSV или вызывает REST API.
2. `ascs_api` валидирует данные и создаёт маршруты, заказы и позиции.
3. Позиции связываются с ячейками через `product_sku` → `trays`.

4. Формируется очередь операций `queue_ops`.

4.2. Запуск маршрута

1. Диспетчер нажимает «Пуск».
2. `ascs_api` назначает `queue_seq` заказам и активирует первый заказ.
3. Определяется секция и операция подбора.
4. `ascs_api` отправляет команды в `ptl_api`, `rid_api`, `lighthouse_api`.
5. Оборудование включает индикаторы, дисплеи и колонны.
6. Состояние сохраняется в Redis и PostgreSQL.

4.3. Подбор

1. Комплектовщик нажимает кнопку PTL.
2. `ptl_api` публикует событие.
3. `ascs_api` классифицирует событие и обновляет `pick_units`.
4. При завершении секции `ascs_api` включает акцептор и переходит к следующей секции.
5. После завершения заказа формируется отчёт и печатные документы.

4.4. Real-time

- `ascs_api` → браузер: WebSocket `/api/ws/queue/playlist`, `/api/ws/indicators`, `/api/ws/system/state`.
- `ptl_api` → `ascs_api` → браузер: события нажатий и состояния индикаторов.

5. Схема развёртывания

Проверочный экземпляр разворачивается одной Docker Compose-конфигурацией:

- сеть `172.28.0.0/24`;
- статические IP для эмуляторов и реле;
- тома `aks_pgdata` и `aks_logs`;
- конфигурации в `configs/`;
- демо-данные в `seed/`.

Сервис	Внутренний адрес	Внешний порт
`ascs_ui`	172.28.0.4:80	8080
`ascs_api`	172.28.0.10:8000	—
`ptl_api`	172.28.0.31:8000	8001
`rid_api`	172.28.0.32:8000	8002
`lighthouse_api`	172.28.0.33:8010	8010

Сервис	Внутренний адрес	Внешний порт
`display1`	172.28.0.11:502/8080	8091
`display2`	172.28.0.12:502/8080	8092
`display3`	172.28.0.13:502/8080	8093
`relay`	172.28.0.20:8080	8094
`db`	172.28.0.2:5432	—
`redis`	172.28.0.3:6379	—

6. Требования к ресурсам

Ресурс	Минимум	Рекомендуется
CPU	2 ядра	4 ядра
RAM	4 ГБ	8 ГБ
Диск	20 ГБ	40 ГБ
Сеть	100 Мбит/с	1 Гбит/с

Для промышленной эксплуатации с большим числом контроллеров и дисплеев ресурсы уточняются по результатам пилотного внедрения.

7. Масштабирование и отказоустойчивость

- Горизонтальное масштабирование сервисов оборудования возможно по количеству контроллеров и дисплеев.
- `ascs_api` масштабируется по числу одновременных рабочих мест; состояние хранится в PostgreSQL и Redis.
- PostgreSQL и Redis могут быть вынесены на отдельные хосты.
- Для повышения доступности рекомендуется резервирование БД и регулярное резервное копирование.
- При недоступности оборудования `ascs_api` фиксирует событие и может поставить наборку на паузу (настраиваемый таймаут).

8. Безопасность

- Аутентификация в АРМ: роль + пароль, JWT.
- Пароли хранятся в виде bcrypt-хэшей.
- Роли диспетчера и оператора разграничены.
- API и БД не публикуются наружу; доступ через reverse proxy.
- Рекомендуется TLS-терминация на reverse proxy.

- Секреты передаются через переменные окружения.
- Регулярное обновление базовых образов и зависимостей.
- Журналирование входов и ключевых операций.

9. Мониторинг и диагностика

Средство	Назначение
`GET /_health` (`ascs_api`)	Проверка PostgreSQL, Redis, PTL, RID
`GET /_ping` (`ptl_api`, `rid_api`, `lighthouse_api`)	Проверка доступности сервисов
Prometheus-метрики	Контроллеры, дисплеи, реле, ошибки, задержки
Журнал событий	Действия пользователей и оборудование
Логи контейнеров	`docker compose logs`
Телеметрия	Обезличенная статистика установок (отключаемая)

10. Ограничения

- Проверочный экземпляр использует эмуляторы; реальные контроллеры и дисплеи не входят в поставку.
- CUPS и физический принтер в демо-экземпляре не обязательны; доступны формирование, предпросмотр и скачивание документов.
- Для промышленной эксплуатации требуется настройка сети, резервного копирования и регламента доступа.
- Архитектура не требует облачных сервисов; все данные остаются в контуре заказчика.