Техническая архитектура программного обеспечения Web-SCADA

Версия 1.0
Дата составления: 16.07.2024

1. Введение и архитектурные драйверы

1.1. Назначение документа

Настоящий документ описывает логическую и физическую архитектуру программного комплекса Web-SCADA – системы оперативного диспетчерского управления и мониторинга промышленных объектов.

1.2. Бизнес-цели архитектуры

Архитектура системы разработана для обеспечения:

- Гибкости и адаптивности: Возможности адаптации под уникальные технологические процессы, оборудование, стандарты (включая нормативные) и существующую инфраструктуру каждого конкретного объекта.
- Поэтапного внедрения: Четкого разделения этапов: монтаж полевого уровня → настройка сервера агрегации → разработка интерфейсов и аналитики → интеграция с ИТ-ландшафтом заказчика.
- Оперативности реакции: Обеспечения мгновенной локальной реакции на аварии (свето-звуковая сигнализация на контроллерах) и эшелонированного удаленного оповещения персонала.
- Интегрируемости как основной функции: Прямого двустороннего взаимодействия с внешними системами (ML/ИИ-сервисы, корпоративные ERP/MES, системы уведомлений) через выделенный слой интеграции.

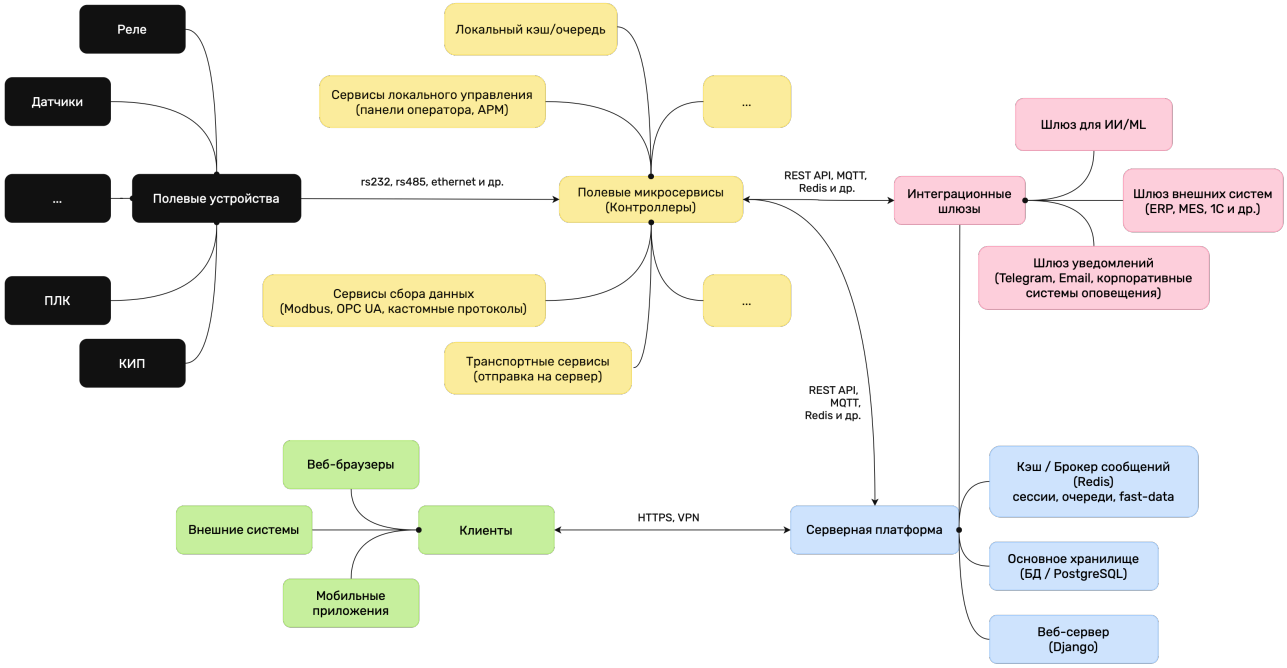
1.3. Ключевые архитектурные принципы

- Микросервисная модульность на полевом уровне: Каждая задача, от сбора данных до управления, выносится в отдельный, переиспользуемый и конфигурируемый сервис.
- Централизованное ядро с чёткими интерфейсами: Единый серверный узел отвечает за приём, нормализацию, хранение данных и выполнение базовой бизнес-логики, предоставляя стабильные API для полевого уровня и слоя интеграции.
- Выделенный слой бизнес-интеграции: Отдельный логический уровень для сложных расчётов, трансформации данных и взаимодействия с корпоративными ИТ-системами, изолирующий ядро от их специфики.
- Безопасность через изоляцию и минимальные привилегии: Активное использование изолированных локальных сетей объекта, белых списков IP-адресов, UUID для идентификации агентов и разделения протоколов (HTTP в LAN / HTTPS через VPN).

2. Логическая архитектура (Диаграмма компонентов)

Система строится по четырехуровневой логической модели:

- Уровень 1 - Полевые устройства
- Уровень 2 - Полевые микросервисы
- Уровень 3 - Серверная платформа
- Уровень 4 - Слой интеграции и бизнес-логики



3. Детализация ключевых логических компонентов

3.1. Уровень 2 *(Полевые микросервисы)*:

- Назначение: Непосредственное взаимодействие с физическим оборудованием и выполнение команд в реальном времени.
- Сервисы сбора данных: Изолированные процессы, отвечающие за опрос датчиков и контроллеров по специфичным протоколам (Modbus TCP/RTU, OPC UA и др.).
- Транспортные сервисы: Отвечают за отправку собранных данных на сервер (Уровень 3) по REST API или MQTT и приём управляющих команд.
- Локальное управление и кэширование: Обеспечивают работу локальных панелей оператора и кэшируют данные в Redis на случай потери связи с сервером. Могут иметь автономную логику реакции на аварии (свето-звуковая сигнализация).

3.2. Уровень 3 *(Серверная платформа)*:

- Назначение: Агрегация, хранение, первичная обработка данных и предоставление базового веб-интерфейса.
- Backend (REST API): Единая точка приёма телеметрии. Выполняет аутентификацию агентов по UUID и IP из белого списка, валидацию, нормализацию, запись в БД и немедленную проверку аварийных уставок.
- *Frontend (Server-Side Rendering):* Формирует HTML на сервере. Динамическое обновление данных реализовано через Long Polling к REST API.

- Сервис мониторинга соединений: Отслеживает регулярность поступления данных от каждого полевого контроллера. При превышении настраиваемого таймаута генерирует событие "потеря связи".
- Хранилища данных:
    - PostgreSQL: Основное реляционное хранилище для исторических данных, конфигураций, пользователей, событий.
    - Redis: Многофункциональный слой данных в памяти: кэш "горячих" значений, хранилище сессий пользователей, брокер для внутренних очередей и Pub/Sub каналов.

3.3. Уровень 4 *(Слой интеграции и бизнес-логики)*:

- Назначение: Оркестрация сложных процессов, трансформация данных и безопасное взаимодействие с внешним миром.
- Шлюз внешних ИТ-систем:
    - Входящий поток: Принимает команды из ERP/MES. Преобразует их во внутренние команды и передаёт на Уровени 2 и 3, для исполнения на полевом уровне.
    - Исходящий поток: Агрегирует данные (выпуск, простои, параметры) и публикует их во внешние системы по расписанию или событию.
- Шлюз ML/ИИ-сервисов: Реализует асинхронный конвейер на базе Redis Lists. Клиенты (Уровень 2 и 3) создают задачи, воркеры забирают их, взаимодействуют с моделями ИИ и возвращают результаты. Уведомление клиента — через callback URL или механизм Long Polling.
- Шлюз уведомлений: Подписывается на поток аварий и событий (из Redis Pub/Sub или БД). Рассылает оповещения по всем настроенным каналам (Telegram Bot, Email, REST-hook в систему заказчика).
- Сервисы расширенной логики: Выполняют ресурсоёмкие или не связанные с реальным временем расчёты (сменные отчёты, OEE, предиктивная аналитика), часто запускаемые по расписанию или событию.

4. Архитектура безопасности

Безопасность реализована по принципу глубокой эшелонированной обороны:

1. Физическая и сетевая изоляция: Полевые контроллеры и сервер работают в изолированной локальной сети объекта. Выход во внешнюю сеть — только через контролируемые точки (VPN-шлюз, DMZ).
2. Контроль доступа на сетевом уровне:
    - Сервер ядра содержит белый список IP-адресов полевых контроллеров, имеющих право отправлять данные.
    - Внешний доступ к веб-интерфейсу из корпоративной сети осуществляется только по HTTPS или через VPN.
3. Аутентификация и авторизация:
    - Полевые агенты: Аутентифицируются по паре UUID устройства + IP-адрес. Пароли/ключи для доступа к оборудованию хранятся в зашифрованном виде в конфигурации settings.cfg.

- Пользователи веб-интерфейса: Стандартная аутентификация сессиями Django.
- Мобильные приложения: Аутентификация по JWT-токенам.
- Внешние системы: Интеграция через API-ключи или аутентификацию в корпоративной системе заказчика (настраивается в шлюзах Уровень 4).

4. Безопасность данных: При передаче за пределы изолированной ЛВС используется TLS/HTTPS. Конфиденциальные настройки хранятся в переменных окружения, а не в репозитории.

5. Аспекты эксплуатации

- Масштабируемость:
  - Вертикальная: Увеличение мощности сервера ядра (CPU, RAM, SSD).
  - Горизонтальная (полевой уровень): Добавление новых контроллеров с уникальными UUID.
  - Горизонтальная (Уровни 2 и 3): Разнесение PostgreSQL, Redis, Django и шлюзов на разные серверы; использование балансировщиков нагрузки.
- Развертывание и обновление:
  - Полевые контроллеры: Ручное обновление через выполнение скрипта Install.sh на каждом устройстве. Конфигурация — через файл settings.cfg или централизованно с сервера (по запросу контроллера).
  - Серверная часть: Обновление стандартными процедурами развертывания Django-приложений (обновление кода, миграции БД, перезапуск сервисов).
- Мониторинг работоспособности:
  - Системный: Контроль прихода данных от каждого полевого агента с генерацией события "потеря связи".
  - Инфраструктурный: Осуществляется средствами заказчика (мониторинг серверов, сети, СХД).