VOXAGENT

Описание архитектуры

Версия 1.0

Настоящая документация может быть использована только для поддержки работоспособности продуктов, поставленных платформой Voxagent.

Все примеры конфигураций, реквизитов и других данных в документе вымышлены и не относятся к реальным сущностям. Любое совпадение случайно.

Все встречающиеся в тексте товарные знаки и зарегистрированные товарные знаки являются собственностью их владельцев и используются исключительно для идентификации программного обеспечения или компаний.

Данная документация может не отражать некоторых модификаций программного обеспечения. Если вы заметили ошибки или опечатки, сообщите об этом по контактным данным ниже.

Россия

help@mail.voxagent.ru · voxagent.ru

Содержание

Связанные разделы

Описание архитектуры

Архитектура

Voxagent построен как набор слабо связанных сервисов, взаимодействующих через REST, WebSockets, WebRTC и шину событий Kafka. На этой странице описан каждый сервис системы и то, как они взаимодействуют между собой.

Сервисы, помеченные как **[внешний]**, — это сторонние SaaS-продукты, с которыми платформа интегрируется. Все остальные сервисы входят в состав платформы (разворачиваются самостоятельно или вместе с остальным стеком).

Основные backend-сервисы

ASP.NET Backend

Основной управляющий API на ASP.NET Core 9.0, построен по принципам Clean Architecture и паттерну CQRS (MediatR + FluentValidation). Использует PostgreSQL и хранит все доменные данные: пользователей, пространства, агентов, инструменты, номера телефонов, биллинг и вебхуки. Выдаёт и проверяет JWT-токены через Keycloak, предоставляет REST-эндпоинты для Angular-клиентов, а также публикует и читает события в Kafka.

ASP.NET Webhook Receiver

Отдельный .NET API, принимающий вебхуки от телефонии и медиа-провайдеров (LiveKit, Twilio, VoxImplant) и пересылающий их в Kafka для асинхронной обработки. Изоляция приёма вебхуков повышает надёжность и снимает нагрузку с основного бэкенда.

Python Backend

Worker AI-агента на **LiveKit Agents SDK** (Python 3.11). Зарегистрирован как LiveKit worker, получает запросы на запуск от LiveKit Server, подключается к комнатам и ведёт цикл разговора: вызывает LLM / STT / TTS-провайдеров, HTTP-инструменты и обрабатывает function calling. Отправляет OpenTelemetry-трейсы напрямую в Langfuse на каждое действие агента и дополнительно экспортирует их через OTLP в коллектор для аналитики в Kafka.

WebSocket Media Bridge

Сервис на Go, мостом соединяющий провайдеров без нативного SIP-транка (в первую очередь VoxImplant, а также Twilio Media Streams) с LiveKit. Конвертирует аудиокадры

PCM16 в base64 из WebSocket в WebRTC-треки внутри комнаты LiveKit, чтобы AI-агенты обрабатывали такие звонки по той же схеме, что и SIP-звонки.

Models Hosted

Self-hosted рантайм ML-моделей. Сейчас запускает Nvidia Parakeet для STT и может хостить дополнительные LLM/TTS-модели по gRPC/REST — используется Python backend, когда клиент не хочет отправлять данные внешним провайдерам.

Сообщения, телеметрия и хранилища

Kafka

Кластер Kafka — шина событий платформы. Передаёт события вебхуков, телеметрию агентов (OTLP Protobuf) и междусервисные доменные события.

OpenTelemetry Collector

Принимает OTLP-телеметрию от Python backend и worker-ов агентов, группирует её и выгружает в Kafka-топик `agent.telemetry.spans`, откуда её затем читают ClickHouse и Langfuse.

ClickHouse

Аналитическая БД, в которой хранится телеметрия агентов, записи звонков и метрики производительности, получаемые из Kafka. Является источником данных для аналитических дашбордов в Angular-клиенте.

MinIO

S3-совместимое объектное хранилище для записей разговоров, выгрузок и артефактов моделей. ASP.NET backend работает с ним через интеграцию `ObjectStorage`.

Redis

In-memory хранилище, используемое ASP.NET backend для кеширования, rate-limit и кратковременных сессионных данных.

Realtime-медиа и идентичность

LiveKit Server

Self-hosted WebRTC SFU, в котором проходят все живые разговоры с агентами. Любой звонок — из веб-виджета, телефона или SIP — разворачивается в комнату LiveKit, куда направляется Python worker агента. LiveKit также рассылает вебхуки (старт комнаты,

подключение участника, завершение egress), которые через Webhook Receiver попадают в Kafka.

Keycloak

Провайдер идентичности OpenID Connect. Выдаёт JWT-токены, которые использует каждый backend и клиент платформы, и предоставляет админку с кастомной темой для администраторов тенантов.

Бизнес-сервисы

Lago (GetLago)

Open-source платформа биллинга и метеринга, разворачиваемая через Docker Compose. ASP.NET backend отправляет в Lago события использования (минуты, символы, вызовы инструментов), а Lago отвечает за тарифы, подписки и счета.

Langfuse

Дашборд observability и трейсинга LLM. Потребляет телеметрию агентов и показывает трейсы по каждому разговору, распределение latency и атрибуцию стоимости по LLM-провайдерам.

Frontend и пользовательские интерфейсы

Angular Client

Основное SPA-приложение (Angular 20) — админ-дашборд для управления агентами, пространствами, инструментами, номерами, кампаниями, аналитикой и биллингом. Использует REST-клиент, сгенерированный из Swagger-спецификации ASP.NET backend.

Angular Widget

Встраиваемый веб-компонент (`<speaknode-agent>`), который клиенты подключают к своим сайтам. Загружает конфигурацию агента из бэкенда и напрямую из браузера подключается к комнате LiveKit для голосового или текстового диалога.

Angular Webhook Receiver Client

Отдельный Angular UI для Webhook Receiver — управление входящими путями вебхуков, правилами маршрутизации и аутентификацией.

Внешние провайдеры

Это сторонние API, с которыми работает ASP.NET backend (а для LLM/STT/TTS — Python backend). Платформа их **не разворачивает** — только интегрируется с ними.

Речь и голос

- **STT-провайдер [внешний]** — любой поддерживаемый сервис распознавания речи, выбирается администратором для каждого агента.
- **TTS-провайдер [внешний]** — любой поддерживаемый сервис синтеза речи, выбирается администратором для каждого агента.

LLM-провайдеры

- **LLM-провайдер [внешний]** — любая поддерживаемая LLM (напрямую или через агрегатор), выбирается администратором для каждого агента.

Телефония

- **Twilio [внешний]** — SMS и голосовой провайдер; входящие и исходящие звонки через SIP-транк или Media Streams.
- **VoxImplant [внешний]** — российский VoIP-провайдер. Не предоставляет прямого SIP-транка, поэтому звонки проходят через WebSocket Media Bridge.

Сообщения

- **SMTP / Email [внешний]** — внешний SMTP-провайдер для транзакционных писем.

Интеграции, настраиваемые пользователем

- **Webhook-инструменты [внешние]** — произвольные HTTP-эндпоинты, которые Python-worker вызывает как инструменты агента во время разговора. URL, метод, заголовки и аутентификация настраиваются администратором для каждого агента.
- **Исходящие вебхуки [внешние]** — HTTP-эндпоинты клиента, которые ASP.NET backend вызывает при наступлении выбранных событий платформы (звонок начат/завершён, подписка изменена, обновилась метрика использования и т. п.). URL и подписка на события настраиваются администратором.

Точки входа

Разговор с агентом можно начать одним из трёх способов. Во всех случаях разговор материализуется в виде комнаты LiveKit, в которую направляется worker Python backend —

различается только то, как в эту комнату попадают аудио и метаданные.

1. Раздел «Тестирование» на сайте

В админ-дашборде (Angular Client) есть раздел **Тестирование**, позволяющий оператору поговорить с агентом прямо из браузера. Поток:

1. Администратор открывает страницу тестирования в Angular Client и выбирает агента.
2. Angular Client вызывает ASP.NET backend, который создаёт комнату LiveKit и выдаёт токен подключения.
3. Браузер подключается к комнате по WebRTC напрямую к LiveKit Server.
4. ASP.NET backend просит Python backend направить worker в ту же комнату.
5. Worker ведёт диалог, обращаясь к LLM / STT / TTS-провайдерам.

Используется в основном для ручного QA, отладки промптов и smoke-тестов перед запуском.

2. Встраиваемый виджет на сайте клиента

Конечные пользователи общаются с агентом через Angular Widget, встроенный в сторонний сайт. Поток такой же, как у страницы тестирования, но инициируется анонимным пользователем, а не авторизованным администратором:

1. Посетитель открывает страницу, на которой размещён `<speaknode-agent>`.
2. Виджет загружает конфигурацию из ASP.NET backend и запрашивает токен LiveKit.
3. Браузер подключается к комнате LiveKit по WebRTC.
4. Python backend направляет worker-а агента в комнату.

Это основная точка входа для веб-сценариев общения с агентом.

3. Телефонный звонок (Twilio или VoxImplant)

Конечные пользователи звонят агенту на подключённый к платформе номер телефона.

- **Twilio** — поддерживает как прямой **SIP-транк** в LiveKit, так и **Media Streams** по WebSocket. В случае SIP звонок принимает встроенный в LiveKit SIP-компонент и помещает его в комнату. В случае Media Streams аудио идёт через WebSocket Media Bridge, который далее проксирует его в LiveKit как WebRTC.
- **VoxImplant** — полноценного SIP-транка нет, поэтому звонки **всегда** идут через WebSocket Media Bridge, преобразующий PCM16-кадры в WebRTC-треки LiveKit.

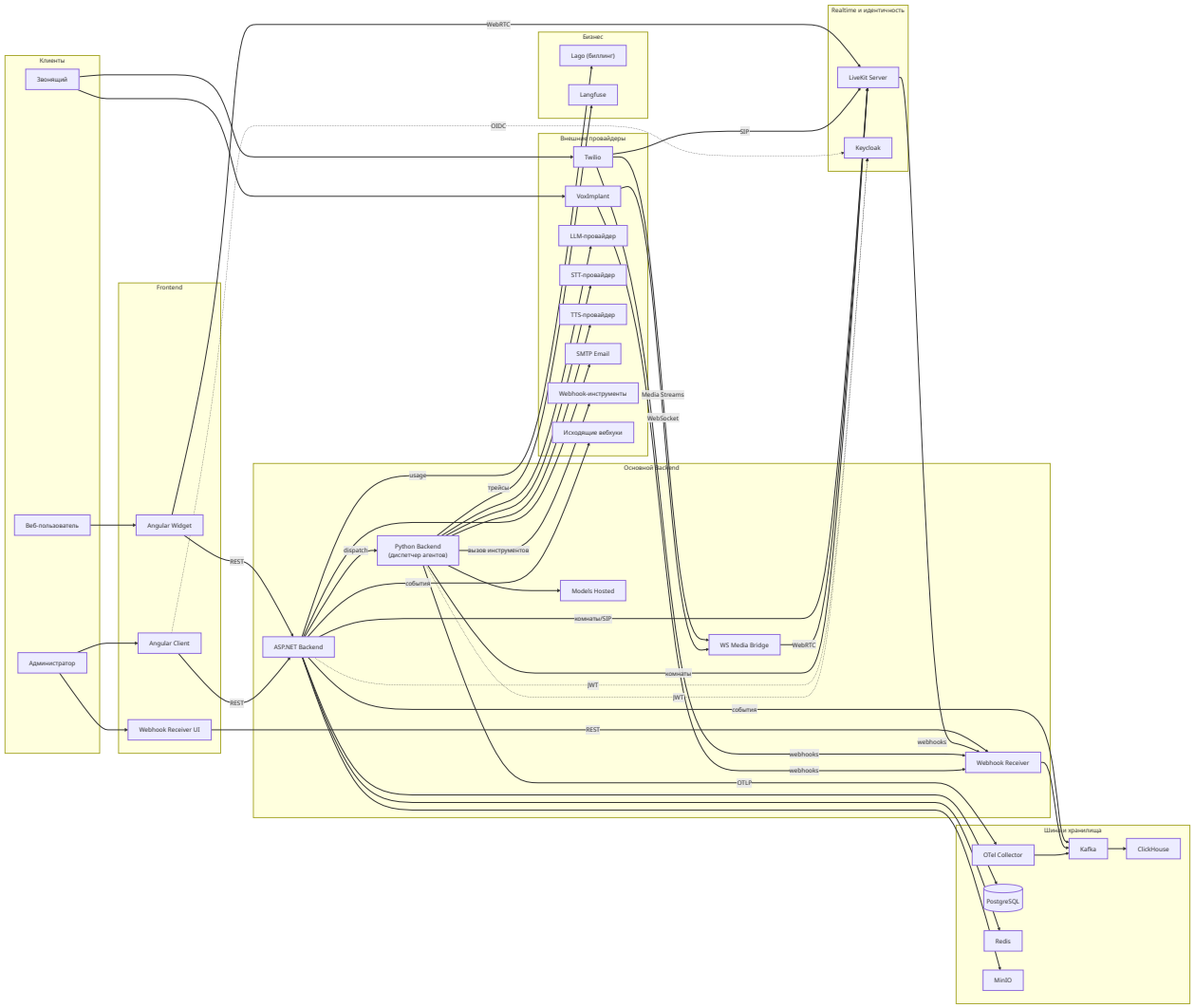
В обоих случаях:

1. Провайдер отправляет входящий вебхук в Webhook Receiver, который публикует его в Kafka.

2. ASP.NET backend читает событие, находит агента, привязанного к номеру, создаёт комнату LiveKit и просит Python backend направить worker-а.
3. Аудио (через SIP или через мост) попадает в ту же комнату LiveKit, и дальше разговор идёт точно так же, как при веб-сценариях.

Диаграмма взаимодействия сервисов

На диаграмме ниже показано, как типичный запрос проходит через платформу — от пользователя на телефоне или сайте, через realtime-медиа, к worker-у AI-агента и обратно через телеметрию и биллинг.



Связанные разделы

- Развертывание — как установить и запустить платформу
- Функционал — пользовательские возможности, построенные поверх этой архитектуры