

SPEAKNODE

Руководство по установке

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 контактными данным ниже.

Все имущественные авторские права сохраняются за Voxagent в соответствии с действующим законодательством.

© Voxagent, 2026

Россия

help@mail.voxagent.app · voxagent.app

Содержание

Развертывание	4
Конфигурация	4
Секции	4
Критичные настройки	5
Профили	5
Масштабирование отдельных сервисов	6
Установка	6
1. Quickstart (один хост, Docker Compose)	6
2. Compose на прод-VM	6
3. Managed / Railway / Kubernetes	6
Что нужно перед стартом	6
Получение deployment-токена	7
Быстрый старт	7
1. Скачать бандл	7
2. Запросить токен для реестра	7
3. Настроить	8
4. Запустить стек	8
4a. Локально на ноуте (default)	8
4b. На виртуалке / удалённом сервере	8
4c. На виртуалке с HTTPS + реальным доменом	9
5. Дождаться angular-client	10
6. Создать пользователя	10
Включить подтверждение email	11
Другие точки входа	11
Что дальше	12
Требования	12
Программные зависимости	12
Ресурсы по сервисам	12
Калькулятор ресурсов	12
Замечания	12
Минимальный сайзинг машины	13

Руководство по установке

Развертывание

Как установить, настроить и запустить SpeakNode.

- [Быстрый старт](#) — `docker compose up` из публичного репозитория
- [Требования](#) — CPU, RAM, диск по сервисам и калькулятор ресурсов
- [Установка](#) — способы поставки, варианты развёртывания, как получить токен registry
- [Конфигурация](#) — переменные окружения и профили

Конфигурация

Вся конфигурация в одном `.env` файле рядом с `docker-compose.yml`. Копируем пример и заполняем поля под своё развёртывание:

```
cp .env.example .env
```

Эталонный `.env.example` опубликован в [публичном репо quickstart](#) рядом с compose-файлом.

Секции

Файл сгруппирован по сервисам. Каждый сервис читает **только** свой префикс — например, всё для asp-net бэкенда это `BACKEND_*`, для python-воркера это `WORKER_*` и т.п. Общие значения (хост Postgres, брокеры Kafka, пароль Redis) определены один раз в **базовых** секциях в начале файла и подставляются через `${VAR}` в секциях сервисов ниже.

Секция	Префикс	Что настраивает
Host ports	<code>*_EXTERNAL_PORT</code>	Хост-порты, на которых docker выставляет сервисы. Меняй если конфликтует.
Service URLs	<code>*_INTERNAL_URL</code> , <code>*_PUBLIC_URL</code>	Внутренние DNS-имена vs URL, видимые из браузера.
Postgres	<code>POSTGRES_*</code>	Креды БД. Один инстанс Postgres хостит несколько БД.
Redis	<code>REDIS_*</code>	Общий пароль. Сервисы используют разные logical DB.
Kafka	<code>KAFKA_*</code>	SASL-креды + bootstrap servers.
MinIO	<code>MINIO_*</code>	S3-совместимое хранилище для записей и счетов.
Keycloak	<code>KEYCLOAK_*</code>	Admin-пользователь, SMTP, OAuth, подтверждение email.
LiveKit	<code>LIVEKIT_*</code>	API-ключи, SIP-транк, WebRTC NAT-конфиг.
Backend	<code>BACKEND_*</code>	Настройки .NET API, ключи LLM, коды биллинговых планов.
Worker	<code>WORKER_*</code>	Python voice agent — ключи LLM/STT/TTS провайдеров.
Frontend	<code>FRONTEND_CLIENT_*</code> , <code>FRONTEND_WIDGET_*</code>	Runtime-конфиг для Angular-бандлов.
Lago	<code>GETLAGO_*</code>	Биллинговый движок: БД, Redis, креды платёжного провайдера, webhook URL.
Webhooks	<code>WEBHOOKS_*</code>	Приёмник входящих вебхуков (Lago, LiveKit).
Logging (opt-in)	<code>LOGGING_*</code>	Креды Fluent Bit / Benthos / Elasticsearch.

Критичные настройки

Часть значений **обязательно** заполнить перед первым запуском — дефолты не подойдут:

Переменная	Зачем
API-ключи для LLM / STT / TTS-провайдеров, которые вы включаете	Доступ к моделям для агентов (конкретные имена переменных зависят от вендора — смотрите <code>.env.example</code>)
<code>GETLAGO_MIGRATE_STRIPE_SECRET_KEY</code>	Биллинг (только если подключён платёжный провайдер)
<code>KEYCLOAK_SMTP_*</code>	Доставка email при регистрации / сбросе пароля
<code>LIVEKIT_API_KEY</code> / <code>LIVEKIT_API_SECRET</code>	Подпись WebRTC — сгенерируйте случайные 32-char секреты
<code>GETLAGO_RSA_PRIVATE_KEY</code>	Base64-кодированный RSA для подписи Lago-вебхуков
Encryption keys (<code>*_ENCRYPTION_*</code> , <code>SECRET_KEY_BASE</code>)	В проде заменить дефолты на случайные строки

В `.env.example` лежат рабочие placeholder'ы для локальной разработки, но **каждый** секрет должен быть заменён для прода.

Профили

Часть сервисов — opt-in и не стартует без профиля:

```
# Стек логирования (Elasticsearch + Kibana + Fluent Bit + Benthos)
docker compose --profile logging up -d

# Аналитика (ClickHouse)
docker compose --profile analytics up -d
```

Масштабирование отдельных сервисов

Чтобы запустить более одного инстанса session-bound сервиса (worker, livekit, bridge) — используйте реплики Docker Swarm или замените Compose на настоящий оркестратор (Kubernetes, Nomad, ECS). Все сервисы stateless, кроме backbone (Postgres, Redis, Kafka, MinIO, Keycloak).

Как подобрать размер под ожидаемую нагрузку — см. [Требования](#).

Установка

SpeakNode поставляется как набор Docker-образов в приватном container registry + публичный бандл `docker-compose.yml` + `.env.example`, который связывает все сервисы.

Поддерживаются три варианта установки:

1. Quickstart (один хост, Docker Compose)

Самый быстрый способ попробовать SpeakNode — один `docker compose up` поднимает весь стек на машине 4 vCPU / 8 GB. Пошаговый гайд — в [Быстром старте](#).

Подходит для: демо, локальной разработки, пилотов до ~10 одновременных сессий.

2. Compose на прод-VM

Тот же compose-файл что в quickstart, развёрнутый на правильно подобранной Linux-VM (см. [Требования](#)). Используется тот же публичный `docker-compose.yml`, но с секретами и увеличенными лимитами ресурсов.

Подходит для: небольшого/среднего прода до ~50 одновременных сессий, single-tenant.

3. Managed / Railway / Kubernetes

Каждый сервис доступен как отдельный контейнер — можно развернуть в Railway, ECS, Kubernetes или любом оркестраторе. В проде мы крутим SpeakNode на Railway с managed Postgres, Redis и Kafka.

Напишите нам, если нужны Helm-чарты или Terraform-модули — поможем настроить под ваше окружение.

Что нужно перед стартом

- Deployment-токен для container registry (см. ниже).
- Хост под [Требования](#).
- Аккаунты во внешних сервисах, которые планируете включить: LLM-провайдер, STT-провайдер, TTS-провайдер, платёжный провайдер из списка поддерживаемых биллинговым движком (для платных тарифов), [Mailgun](#) или аналог (email).

Получение deployment-токена

Образы лежат в `registry.gitlab.com/nodevoice/*`. Они не публичные — каждому клиенту выдаётся свой read-only токен.

Как запросить: откройте issue в [репо quickstart](#) или напишите нам — мы выдадим токен.

Залогиньтесь на хосте:

```
docker login registry.gitlab.com -u <имя-токена> -p <секрет-токена>
```

После этого `docker compose pull` скачает все нужные образы.

Быстрый старт

SpeakNode поставляется бандлом — `docker-compose.yml` и эталонный `.env.example` — прямо с этого сайта документации. Образы лежат в приватном container registry `registry.nodul.ru` — доступ выдаётся по read-only токenu (см. ниже).

1. Скачать бандл

```
mkdir SpeakNode && cd SpeakNode
curl -O https://docs.dev.voxagent.ru/cdn/docker-compose.yml
curl -O https://docs.dev.voxagent.ru/cdn/.env.example
```

- `docker-compose.yml` — полный стек сервисов (включает одноразовый сервис `postgres-init`, который создаёт дополнительные БД/пользователей после того как Postgres становится healthy)
- `.env.example` — эталонные переменные окружения

2. Запросить токен для реестра

Образы из `registry.nodul.ru/voxagent/nodevoice/*` требуют read-only токен. Токены выдаются **каждому клиенту по запросу** — напишите нам на `help@mail.voxagent.app`, и мы пришлём токен.

После этого залогиньтесь на хосте — docker сам спросит логин и пароль:

```
docker login registry.nodul.ru/voxagent/nodevoice
```

3. Настроить

```
cp .env.example .env
```

Базовый `.env.example` уже содержит всё необходимое для локального запуска — можно оставить как есть и поднимать стек. Ключи LLM/STT/TTS-провайдеров, SMTP для Keycloak, биллинг, телефония и прочие интеграции — опциональны и нужны только для соответствующего функционала.

Полный референс переменных — в [Конфигурации](#).

4. Запустить стек

****Apple Silicon / ARM64.**** Наши образы собраны под `linux/amd64`. Чтобы docker гарантированно тянул и запускал именно эту платформу (на M1/M2/M3 через Rosetta), везде ниже ставится префикс `DOCKER_DEFAULT_PLATFORM=linux/amd64`. На Intel-машинах префикс безвреден — можно оставить.

Сначала подтяните актуальные образы — чтобы не работать с устаревшим `:latest`, закешированным при прошлом `docker login`:

```
DOCKER_DEFAULT_PLATFORM=linux/amd64 docker compose pull
```

4a. Локально на ноуте (default)

Стек работает «из коробки» на `http://localhost:4200` и других портах — никаких дополнительных настроек не нужно:

```
DOCKER_DEFAULT_PLATFORM=linux/amd64 docker compose up -d
```

4b. На виртуалке / удалённом сервере

Если запускаете compose **на VM** (cloud-инстанс, LAN-сервер) и заходить будете с другого устройства — браузер по умолчанию пойдёт на `localhost` своего ноута, а не на VM. Нужно регенерировать `.env` так, чтобы публичные URL'ы указывали на адрес VM:

```
# узнать IP VM (на самой VM):
curl -4 https://ifconfig.me           # cloud VM с публичным IP
hostname -I | awk '{print $1}'        # LAN-сервер (на Linux)

# сгенерировать .env с этим адресом:
DOCKER_DEFAULT_PLATFORM=linux/amd64 PUBLIC_HOST=10.0.0.5 \
  docker compose run --rm env-gen

# теперь запускать как обычно:
DOCKER_DEFAULT_PLATFORM=linux/amd64 docker compose up -d
```

`env-gen` — отдельный one-shot контейнер. На `docker compose up` он **не** запускается (сидит за профилем `tools`), вызывается явно одной командой.

4с. На виртуалке с HTTPS + реальным доменом

Если хочешь обслуживать стек через `https://demo-app.dev.example.io`, `https://demo-api.dev.example.io` и т.д. с автоматическим TLS (внутренний CA от Caddy или Let's Encrypt) — нужно прописать DNS-записи (см. ниже) и сгенерировать `.env` в TLS-режиме.

DNS — что на что смотрит

Все поддомены смотрят на **один и тот же IP** — IP вашей VM. Caddy внутри стека маршрутизирует по поддоменам сам. Два варианта DNS-конфигурации:

Вариант А — wildcard (рекомендуется)

Тип	Имя	Значение
A	<code>*.dev.example.io</code>	<code><IP вашей VM></code>
A	<code>dev.example.io</code>	<code><IP вашей VM></code> (опционально)

Вариант В — отдельные A-записи на каждый поддомен

Тип	Имя	Сервис
A	demo-app.dev.example.io	Angular Client (приложение)
A	demo-widget.dev.example.io	Angular Widget (виджет)
A	demo-api.dev.example.io	ASP.NET Backend (REST API)
A	demo-identity.dev.example.io	Keycloak (логин / OAuth)
A	demo-livekit.dev.example.io	LiveKit (WebRTC + WebSocket)
A	demo-docs.dev.example.io	Документация
A	demo-s3.dev.example.io	MinIO S3 API
A	demo-s3-console.dev.example.io	MinIO Web Console
A	demo-lago.dev.example.io	Lago Billing UI
A	demo-lago-api.dev.example.io	Lago Billing API
A	demo-webhooks.dev.example.io	Webhook Receiver
A	demo-kafka-ui.dev.example.io	Kafka UI (опционально, devops)

Подождите DNS-пропагации (обычно 5-30 минут, проверить: `dig +short demo-app.dev.example.io`).

Запуск стека

```
DOCKER_DEFAULT_PLATFORM=linux/amd64 PUBLIC_HOST=dev.example.io PUBLIC_MODE=tls \
  docker compose run --rm env-gen

# поднять стек + caddy edge (профиль tls):
DOCKER_DEFAULT_PLATFORM=linux/amd64 COMPOSE_PROFILES=tls docker compose up -d

# один раз — доверить root CA для local-certs:
docker compose exec caddy caddy trust
```

Доступ: `https://demo-app.dev.example.io`, `https://demo-identity.dev.example.io`, `https://demo-api.dev.example.io` и т.д.

TLS-сертификаты. По умолчанию Caddy использует локальный CA — cert'ы доверены только на машине где выполнили `caddy trust`. Для публично-доверенных Let's Encrypt: прокомментируйте `local_certs` в `caddy/Caddyfile`, поставьте валидный `CADDY_LE_EMAIL=...` в `.env` и убедитесь что 80/443 на VM доступны из интернета.

Первый запуск занимает несколько минут — Postgres инициализируется, Keycloak импортирует realm, Lago накатывает миграции и сидит биллинг-организацию.

5. Дождаться `angular-client`

Проверить статус сервисов:

```
docker compose ps -a --format "table {{.Service}}\t{{.Status}}\t{{.Ports}}"
```

Как только строка `angular-client` покажет `Up ... (healthy)` — приложение готово, открывайте <http://localhost:4200>.

6. Создать пользователя

На странице входа нажмите **Регистрация** и создайте учётную запись.

В поставке для `docker compose` подтверждение email по умолчанию **отключено** (`KEYCLOAK_VERIFY_EMAIL=false`), регистрация моментальная.

Включить подтверждение email

Если хотите, чтобы пользователи подтверждали почту перед первым входом, в `.env`:

1. Включите флаг:

```
KEYCLOAK_VERIFY_EMAIL=true
```

2. Заполните SMTP-реквизиты Keycloak (с них уходят письма верификации и сброса пароля):

```
KEYCLOAK_SMTP_HOST=smtp.example.com
KEYCLOAK_SMTP_PORT=465
KEYCLOAK_SMTP_FROM=no-reply@example.com
KEYCLOAK_SMTP_USER=<smtp-логин>
KEYCLOAK_SMTP_PASSWORD=<smtp-пароль>
KEYCLOAK_SMTP_SSL=true # 465
KEYCLOAK_SMTP_STARTTLS=false # переключите на true для порта 587
```

3. Перезапустите Keycloak, чтобы применить:

```
docker compose up -d --force-recreate keycloak
```

Другие точки входа

Когда стек полностью поднялся:

Сервис	URL
SpeakNode app	http://localhost:4200
Keycloak admin	http://localhost:8081
Backend Swagger	http://localhost:8040/swagger
Lago billing UI	http://localhost:4203
MinIO console	http://localhost:9001
Kafka UI	http://localhost:8084

Порты настраиваются в `.env` в секции **HOST PORTS** — меняйте любой, если на вашей машине он уже занят.

Что дальше

- [Требования](#) — подобрать хост под ожидаемую нагрузку.
- [Конфигурация](#) — полный референс переменных.
- [Установка](#) — другие варианты развёртывания (managed, Kubernetes).

Требования

SpeakNode — модульная платформа. Каждый сервис работает в своём контейнере, поэтому компоненты можно масштабировать независимо. Перед развёртыванием подберите железо под ожидаемое **число одновременных сессий** — остальное (бэкбон) остаётся примерно постоянным.

Программные зависимости

- Docker 24+ и Docker Compose v2.24+
- 64-битный Linux-хост (Ubuntu 22.04 LTS или аналог). macOS подойдёт для локальной разработки.
- Исходящий интернет (registry, LLM/STT/TTS провайдеры, STUN/TURN).
- Порты, доступные из браузеров пользователей: фронтенд, API бэкенда, Keycloak, LiveKit WebRTC (TCP 7881, UDP 50000–60000).

Ресурсы по сервисам

В таблице ниже — CPU / RAM / Disk для каждого сервиса при типовых нагрузках по одновременным сессиям. Значения — эмпирические замеры из прода.

Как читать:

- **Сервисы, зависящие от сессий** (`livekit-server` , `python-worker` , `egress` , `websocket-media-bridge`) растут линейно по числу активных звонков. Их и нужно планировать.
- **Backbone-сервисы** (Postgres, Kafka, Keycloak, Redis, MinIO и т.п.) имеют в основном фиксированный футпринт — разворачиваются один раз и переиспользуются между тенантами.
- **Диск** в основном растёт в Postgres (транскрипты, счета), Kafka (retention событий) и MinIO (записи звонков). Планируйте хранилище с учётом retention.

Калькулятор ресурсов

Оцените общий футпринт application-слоя под пиковую нагрузку:

Замечания

- Числа **только для application-слоя**. CPU/RAM/диск под LLM, STT и TTS провайдеров (или self-hosted модели) **оплачиваются отдельно** у вендоров и здесь не учтены. Про расчёт стоимости моделей — в отдельной странице позже.
- Добавьте запас 20–30% сверху на headroom, пики и оверхед ОС.

- Для HA-прода удвойте backbone (реплика Postgres, Kafka с replication factor ≥ 2 и т.п.).

Минимальный сайзинг машины

Нагрузка	Рекомендуемая VM
Демо / dev (1 сессия)	4 vCPU · 8 GB RAM · 50 GB SSD
Small (10 сессий)	6 vCPU · 16 GB RAM · 100 GB SSD
Medium (50 сессий)	12 vCPU · 32 GB RAM · 200 GB SSD
Large (250 сессий)	32 vCPU · 96 GB RAM · 500 GB SSD (multi-node)
XL (500 сессий)	64 vCPU · 192 GB RAM · 1 TB SSD (multi-node)

Для всего выше 50 одновременных сессий рекомендуем вынести `livekit-server` + `python-worker` на отдельные хосты, чтобы медиа и агентская нагрузка не мешали backbone.