

Руководство по построению приложений

AliothStudio

Синяя книга · Полный операционный путь от подготовки окружения до развёртывания и выпуска

Версия v1.1 · Октябрь 2026 · Hangzhou Yuqi Technology Co., Ltd.

Применимые версии: модель Alioth v10.0.44 · AliothStudio 0.1.x

Содержание

1. Состав платформы: две цепочки
2. Подготовка окружения
3. Сторона модели: центр моделей и выпуск модели
4. Построение приложения: рабочий процесс AppAgent
5. Ключевые моменты разработки модулей, Block и Service
6. Расширения: extensions/*.yaml
7. Сборка и развёртывание
8. Краткий справочник по гейтам качества
9. Онлайн-практика: обучающая SaaS-версия

1. Состав платформы: две цепочки

AliothStudio состоит из двух цепочек с разделённой ответственностью; понимание их границ — предпосылка правильного использования платформы:

Цепочка	Ответственность	Драйвер	Результат
Цепочка выпуска модели	Структурная эволюция основания данных (таблицы/столбцы/наследование/кодировки измерений)	Центр моделей Meta (ручной канал)	Пакет выпуска <code>Alioth/v*</code> (DDL · seed · машиночитаемый контракт · отчёт верификации)
Цепочка построения приложения	Сборка бизнес-приложений на замороженном основании	Агент App-Agent + разработчик	Конфигурация приложения, бэкенд Service, фронтенд модулей, пакет развёртывания <code>Deploy/{ns}</code>

Красная линия: структура схемы isahl заморожена. Любое требование «добавить таблицу / добавить столбец / изменить столбец» в цепочке построения приложения обязано возвращаться в цепочку выпуска модели (ручное решение в центре моделей); обходить это через DDL, миграционные скрипты или прямую запись метаданных запрещено.

2. Подготовка окружения

2.1 Предварительные зависимости

Зависимость	Назначение
PostgreSQL 14+	Основание данных (модель v10.0 — структура наследования таблиц)
mise	Единая точка входа для задач и окружения (<code>mise run <task></code>)
Rust toolchain / pnpm / bun	Сборка бэкенда, сборка фронтенда, инструментальная цепочка скриптов

2.2 Инициализация и запуск базы данных

```
# Инициализация базы разработки (воспроизведение из снимка выпуска; выполняет DROP и
пересоздание целевой БД, только aliothstudio_dev)
bash scripts/db/reset-db.sh --dev

# Платформа Meta (управление моделями + AppAgent): бэкэнд :4949, фронтенд :49494
mise run meta

# Окружение разработки пространства имён (Gateway + SS0), {ns} ∈ alioth | wz | avic | cosmic | se
| mat
mise run dev:{ns}
```

База данных имеет четыре уровня изоляции: продакшн `aliothstudio` ← предпродакшн `aliothstudio_pre` → разработка `aliothstudio_dev` → тестирование `aliothstudio_test`. Окружение жёстко привязано к `DATABASE_URL`; запросы к схеме выполняются исключительно через `cd Meta/backend && mise run schema-info -- <command>` (БД — единственный источник истины; выводить что-либо из чтения DDL-файлов запрещено).

3. Сторона модели: центр моделей и выпуск модели

3.1 Центр моделей (Meta UI)

Вся структурная эволюция выполняется через центр моделей фронтенда Meta: создание коллекции (`collection`, подключение родительской таблицы для формирования наследования), регистрация полей (`fields`), ведение кодировок измерений (сцена/фактор/функция). Регистрация коллекций и полей программно синхронизируется платформой в физические таблицы и подключения наследования — разработчик не пишет ни одного DDL-оператора создания таблиц.

3.2 Выпуск модели

После запуска выпуска из центра моделей конвейер экспортирует схему `isah1` (`pg_dump --schema-only` + данные 268 seed-таблиц + seed измерений `seed-dimensions.sql`), постобрабатывает её до формы чистых `CREATE/ALTER`, записывает в `Alioth/v{version}/` и выполняет полную пересборку с верификацией в выделенной проверочной БД (round-trip структуры + round-trip seed + уникальность `uid`), а отчёт сохраняется в `verify-report.json`. Текущий последний выпуск `v10.0.44`: 1022 таблицы `isah1`, 1204 рёбер наследования, 268 seed-таблиц с попотабличной round-trip-

согласованностью, статус верификации — `verified`. Конвейер выпуска производит 7 событий прогресса (`prepare` / `schema` / `isahl_dump` / `seed_dimensions` / `manifest` / `verify` / `done`); уровень строки коллекции декларативно несётся объявлением `row_layer` (`abstract` / `carry`) на стороне модели, а неразрешённая родительская таблица, содержащая строки, блокирует выпуск.

Приземление модели и производная поверхность БД сцеплены одним пакетом: `refresh-derived-artifacts.ts` — единственная точка входа обновления четырёх производных артефактов `fk-index`, `relation-index`, `value-domain-columns`, `dk-projection-tables`, уже подключена к `reset-db.sh`; diff артефактов есть поверхность ревью топологии модели, а защита от удаления обеспечивается гейтом W9.

3.3 Измерения и координаты

Бизнес-семантика локализуется трёхмерной координатой: `dk_scene` (сцена), `dk_factor` (фактор), `dk_function` (функция). Код измерения автоматически составляется из кода категории + порядкового символа (например, `JC` = системное управление). Координаты интерпретируются локально в каждом пространстве имён; каждый API сервиса Service статически привязан к фиксированной тройке координат.

4. Построение приложения: рабочий процесс AppAgent

Цепочка построения приложения управляется AppAgent; стандартная последовательность инструментов:

```
analyze_ontology      # требования/прототип → OntologyModel IR (со строгой проверкой полноты пяти доменов)
build_module          # flow-plan → module.json (декларация модуля)
build_block           # flow-plan → block.json (декларация блока возможностей)
transfer_ontology     # онтологическое выравнивание → alignment-graph.json + mapped-entities.json
generate_service_api  # онтологическая сущность → бэкенд Service (Rust, статическая привязка координат)
compose_app           # сборка модулей → app.json + extensions/*.yaml + артефакты прототипа
```

Размещение артефактов: `Pre-Proc/{ns}/Apps/{app}/` (`app.json`, `extensions`, артефакты процесса) и `Pre-Proc/{ns}/Sources/Apps/{Modules,Blocks,Services}/` (декларации и код модулей/блоков/сервисов). Форма записей артефакта онтологического отображения (`ontology-output.json`) проверяется контрактным гейтом, а координаты обязаны быть реальными кодами из БД.

5. Ключевые моменты разработки модулей, Block и Service

Слой	Ключевые ограничения
Module	Контейнер комбинации Scene/Factor; обязательные поля <code>module.json</code> заполнены полностью; <code>DashboardPage</code> запрещён (прямой переход с корневого пути на страницу списка сущностей)
Block	Единица возможностей; ID, на которые ссылаются <code>factors / services</code> , обязаны существовать (для висячих ссылок есть отдельный гейт)
Бэкенд Service	Стандартный каркас <code>handlers/models/repositories/services</code> ; статическая привязка тройки координат (разные координаты одной таблицы разделяются на разные API); <code>middleware auth</code> запрещён (NGAC централизуется в Gateway); крейт <code>tracing</code> внутри модуля запрещён
Фронтенд	Управление состоянием исключительно на Jotai v2; поля измерения по форме «числовое значение + единица измерения одним выбором»; реальная интеграция с бэкендом, mock запрещён; доступ к данным унифицирован через ресурсный transport из <code>@alioth/api</code> (<code>list/get/mutate</code> , конверт распаковывается через <code>items() / unwrap()</code>); селектор ссылок обязан всегда использовать дескриптор <code>ReferenceSelectDescriptor</code> , экстраполяция контракта по типу эндпоинта запрещена

5.1 Железные правила записи данных

- **INSERT в листовую таблицу:** целевая таблица записи обязана быть листовой в частичном порядке наследования (гейт проверяет это фактически через `pg_inherits` и не принимает статически угаданные имена)
- **Состояние — через таблицу отношений:** состояние/категория/тег сущности записываются в таблицу отношений `zc_id_lifecycle_r_*`; создавать в сущности собственные столбцы состояния запрещено (`flag` используется только в строках словаря состояний)
- **Контракт записи класса:** запись в подтаблицу `lifecycle` обязана предоставлять `dk_function` одного из шести префиксов (`_f_ / _t_` выводятся автоматически, рукописные литералы запрещены)
- **Скалярные ссылки:** столбцы `qk_*` — указатели в пространство значений (`bigint`); арифметические операции над ними запрещены, чтение выполняется через разыменование `_refs`

- **Временная семантика:** будущие срезы идут в семейство task, прошлые — в семейство event; завершающее действие попадает в срез события (переворот среза)

6. Расширения: extensions/*.yaml

Расширения уровня приложения (constraints / rules / statemachines / workflows) размещаются в Pre-Proc/{ns}/Apps/{app}/extensions/ и загружаются при старте Gateway — это единственная поверхность, приводящая расширения в действие: проверки, правила и процессы можно настраивать без повторной генерации кода.

7. Сборка и развёртывание

```
# Единственная точка входа сборки (feature-гейтинг + изоляция target + автоматическая обработка подписи)
bash scripts/build-ns.sh <NAMESPACE> release    # или dev (режим debug)

# Выпуск (синхронизация Apps / seed / .env / apps.json; собственная сборка не выполняется)
bash scripts/deploy/release-to-namespace.sh <NAMESPACE>
```

Артефакты: Deploy/{ns}/bin/{ns}-server (для пространств имён с включённым OpenActivity дополнительно Deploy/{ns}/0pen/bin/openactivity-server). При выпуске на стадию production Gateway перезапускается автоматически. Приложения, не привязанные к пространству имён (Meta / SSO), идут через scripts/meta|sso/deploy.sh и попадают в Release/.

8. Краткий справочник по гейтам качества

Репозиторий принуждает качество через «субстрат исполнения спецификаций»: каждая проверяемая спецификация снабжена скриптом проверки + базовой линией + регистрацией в гейте (229 зарегистрированных гейтов: 137 на стадии commit / 92 на стадии push); библиотека спецификаций содержит 403 живые спецификации sarability; каждая клауза MUST/NEVER обязана декларировать место исполнения (id гейта / скрипт / путь теста) через структурированное поле «执行面» (поверхность исполнения); прозаические декларации более не принимаются. Часто встречающиеся позиции:

Шлюз	Что блокирует	Этап
<code>check-isahl-ddl-boundary</code>	Любая обходная запись структуры isahl	commit
<code>check-leaf-insert</code>	INSERT не в листовую таблицу / рукописные литералы <code>_f/_t_</code>	commit
<code>check-code-semantics</code>	Запись семантики типов/перечислений в столбец code	commit
<code>check-ontology-contract</code>	Форма артефакта онтологического отображения / подделка координат	commit
<code>check-bridge-host-domain</code>	Смещение домена-хоста на левом конце таблицы отношений	commit
<code>formal-proofs</code>	Падение сборки теорем Lean или наличие заглушки вместо доказательства	push
<code>check-all</code> (T1/T2)	Согласование версий / соответствие OpenSpec / i18n / DAG сервисов и пр.	push

При блокировке не обходите гейт: работайте по указанной гейтом поверхности исправления; существующие нарушения уходят в базовую линию (только уменьшение, не рост), а новые нарушения обязаны быть исправлены.

9. Онлайн-практика: обучающая SaaS-версия

Полноценную платформу можно попробовать без локального окружения: <https://cosmic-tools.shop/> предоставляет онлайн-обучающую SaaS-версию AliothStudio — облегчённое развёртывание для индивидуальных пользователей (ключевое моделирование, сборка приложений и линия выполнения полностью доступны; корпоративная оркестрация множества пространств имён и конвейер выпуска опущены). Рекомендуемый путь: зарегистрироваться и войти → последовательно пройти практику по главам 2–7 настоящего руководства → самостоятельно проверить артефакты по списку гейтов главы 8.

Сопутствующая литература: «Белая книга теории модели Alioth» — чтобы понять математическую структуру, лежащую в основе всех ограничений настоящего руководства.

