

Manual de construcción de aplicaciones

AliothStudio

Libro azul · Ruta operativa completa, desde la preparación del entorno hasta el despliegue y la publicación

Versión v1.1 · octubre de 2026 · Hangzhou Yuqi Technology Co., Ltd.

Versiones aplicables: modelo Alioth v10.0.44 · AliothStudio 0.1.x

Índice

1. Composición de la plataforma: dos cadenas
2. Preparación del entorno
3. Lado del modelo: centro de modelos y publicación del modelo
4. Construcción de aplicaciones: flujo de trabajo de AppAgent
5. Puntos clave del desarrollo de módulos, Block y Service
6. Extensiones: extensions/*.yaml
7. Compilación y despliegue
8. Consulta rápida de puertas de calidad
9. Práctica en línea: edición de aprendizaje SaaS

1. Composición de la plataforma: dos cadenas

AliothStudio se compone de dos cadenas con responsabilidades separadas; comprender sus límites es la premisa para usar la plataforma correctamente:

Cadena	Responsabilidad	Impulsor	Producto
Cadena de publicación del modelo	Evolución estructural de la base de datos (tablas/columnas/herencia/codificación de dimensiones)	Centro de modelos Meta (canal manual)	Paquete de publicación <code>Alioth/v*</code> (DDL · semillas · contrato legible por máquina · informe de verificación)
Cadena de construcción de aplicaciones	Ensamblar aplicaciones de negocio sobre la base congelada	Agente App-Agent + desarrolladores	Configuración de la aplicación, backend de Service, frontend de módulos, paquete de despliegue <code>Deploy/{ns}</code>

Línea roja: la estructura del esquema isahl está congelada. Cualquier solicitud de «añadir tabla / añadir columna / cambiar columna» en la cadena de construcción de aplicaciones debe volver a la cadena de publicación del modelo (decisión manual en el centro de modelos); se prohíbe sortearla mediante DDL, scripts de migración o escritura directa de metadatos.

2. Preparación del entorno

2.1 Dependencias previas

Dependencia	Uso
PostgreSQL 14+	Base de datos (el modelo v10.0 es una estructura de herencia de tablas)
mise	Punto de entrada unificado para tareas y entornos (<code>mise run <task></code>)
Rust toolchain / pnpm / bun	Compilación del backend, compilación del frontend, cadena de herramientas de scripts

2.2 Inicialización y arranque de la base de datos

```
# Inicialización de la base de datos de desarrollo (reproducida desde la instantánea de
publicación; hace DROP y recrea la base destino, solo para aliothstudio_dev)
bash scripts/db/reset-db.sh --dev

# Plataforma Meta (gestión de modelos + AppAgent): backend :4949, frontend :49494
mise run meta

# Entorno de desarrollo del espacio de nombres (Gateway + SS0), {ns} ∈ alioth | wz | avic | cosmic
| se | mat
mise run dev:{ns}
```

La base de datos tiene cuatro niveles de aislamiento: producción `aliothstudio` ← preproducción `aliothstudio_pre` → desarrollo `aliothstudio_dev` → pruebas `aliothstudio_test`. El entorno está vinculado obligatoriamente con `DATABASE_URL`; todas las consultas de esquema pasan siempre por `cd Meta/backend && mise run schema-info -- <command>` (la base de datos es la única fuente de verdad; se prohíbe inferir leyendo archivos DDL).

3. Lado del modelo: centro de modelos y publicación del modelo

3.1 Centro de modelos (Meta UI)

Toda la evolución estructural se realiza a través del centro de modelos del frontend Meta: crear colecciones (collection, enganchando una tabla padre para formar la herencia), registrar campos (fields) y mantener la codificación de dimensiones (escena/factor/función). El registro de colecciones y campos se sincroniza programáticamente como tablas físicas y enganches de herencia: el desarrollador no escribe ningún DDL de creación de tablas.

3.2 Publicación del modelo

Cuando el centro de modelos lanza una publicación, la cadena de procesamiento exporta el esquema `isahl` (`pg_dump --schema-only` + los datos de 268 tablas semilla + la semilla de dimensiones `seed-dimensions.sql`), lo posprocesa a forma pura CREATE/ALTER, lo escribe en `Alioth/v{version}/` y realiza una reconstrucción y verificación completas en una base de datos de verificación dedicada (ida y vuelta de estructura + ida y vuelta de semillas + unicidad de uid), con el informe volcado a disco en `verify-report.json`. La publicación más reciente actual es `v10.0.44`: 1022 tablas `isahl`, 1204 aristas de herencia, 268 tablas semilla con ida y vuelta tabla a tabla consistente, estado de verificación `verified`. El pipeline de publicación

produce 7 eventos de progreso (prepare / schema / isahl_dump / seed_dimensions / manifest / verify / done); el nivel de fila de las filas de colección lo porta la declaración `row_layer` (abstract / carry) del lado del modelo, y que una tabla padre no adjudicada contenga filas bloquea la publicación.

El aterrizaje del modelo y la superficie de derivación de la base de datos se acoplan en el mismo lote: `refresh-derived-artifacts.ts` es el único punto de entrada de refresco de los cuatro productos derivados `fk-index`, `relation-index`, `value-domain-columns` y `dk-projection-tables`, y ya está integrado en `reset-db.sh`; el diff de los productos es la superficie de revisión de la topología del modelo, y la protección contra su eliminación la garantiza la puerta W9.

3.3 Dimensiones y coordenadas

La semántica de negocio se localiza mediante coordenadas tridimensionales: `dk_scene` (escena), `dk_factor` (factor), `dk_function` (función). El código de dimensión se compone automáticamente con el código de categoría + el símbolo del número de orden (por ejemplo `JC` = gestión del sistema). Las coordenadas se interpretan localmente según el espacio de nombres; cada API de un Service vincula estáticamente una terna fija de coordenadas.

4. Construcción de aplicaciones: flujo de trabajo de AppAgent

La cadena de construcción de aplicaciones está impulsada por AppAgent; secuencia estándar de herramientas:

```
analyze_ontology      # requisitos/prototipo → IR OntologyModel (con validación dura de completitud de los cinco dominios)
build_module          # flow-plan → module.json (declaración del módulo)
build_block           # flow-plan → block.json (declaración del bloque de capacidades)
transfer_ontology     # alineación ontológica → alignment-graph.json + mapped-entities.json
generate_service_api  # entidad ontológica → backend de Service (Rust, vinculación estática de coordenadas)
compose_app           # ensamblaje de módulos → app.json + extensions/*.yaml + productos del prototipo
```

Ubicación de los productos: `Pre-Proc/{ns}/Apps/{app}/` (app.json, extensions, productos del flujo) y `Pre-Proc/{ns}/Sources/Apps/{Modules,Blocks,Services}/` (declaraciones y código de módulos/bloques/servicios). La forma de las entradas del producto de mapeo ontológico (`ontology-output.json`) está sujeta a la validación de una puerta de contratos, y las coordenadas deben ser códigos reales de la base de datos.

5. Puntos clave del desarrollo de módulos, Block y Service

Capa	Restricciones clave
Module	Contenedor de combinación Scene/Factor; los campos obligatorios de <code>module.json</code> deben estar completos; se prohíbe DashboardPage (salto directo desde la ruta raíz a la página de listado de entidades)
Block	Unidad de capacidad; los ID referenciados por <code>factors</code> / <code>services</code> deben existir (las referencias huérfanas tienen una puerta de control específica)
Backend de Service	Esqueleto estándar <code>handlers/models/repositories/services</code> ; vinculación estática de la terna de coordenadas (la misma tabla con coordenadas distintas se separa en APIs distintas); se prohíbe el middleware auth (NGAC unificado en el Gateway); se prohíbe el crate <code>tracing</code> dentro del módulo
Frontend	Gestión de estado siempre con Jotai v2; los campos de medida siguen el formato «valor + unidad de selección única»; integración contra el backend real, se prohíbe el mock; el acceso a datos se unifica a través del transport de nivel de recurso de <code>@alioth/api</code> (list/get/mutate, el sobre se desenvuelve con <code>items()/unwrap()</code>); el selector de referencias debe usar siempre el descriptor ReferenceSelectDescriptor y tiene prohibido extrapolar el contrato a partir del tipo de endpoint

5.1 Reglas de oro para la escritura de datos

- **INSERT en la tabla hoja:** el destino de escritura debe ser una tabla hoja del orden parcial de herencia (la puerta de control lo mide realmente con `pg_inherits`; no se aceptan nombres conjeturados estáticamente)
- **El estado va a tablas de relación:** el estado, la categoría y las etiquetas de la entidad se escriben en las tablas de relación `zc_id_lifecycle_r_*`; se prohíbe que la entidad cree su propia columna de estado (`flag` se usa solo en las filas del diccionario de estados)
- **Contrato de escritura de categoría:** la escritura en subtablas lifecycle debe aportar el `dk_function` de uno de los seis prefijos (`_f_` / `_t_` se derivan automáticamente; se prohíbe escribir literales a mano)
- **Referencia escalar:** las columnas `qk_*` son punteros al espacio de valores (bigint); se prohíbe operar aritméticamente sobre ellas y la lectura pasa por la resolución `_refs`
- **Semántica temporal:** los cortes futuros van a la familia task y los cortes pasados a la familia event; la acción de completar aterriza en el corte de evento (inversión de corte)

6. Extensiones: extensions/*.yaml

Las extensiones a nivel de aplicación (`constraints` / `rules` / `statemachines` / `workflows`) se colocan en `Pre-Proc/{ns}/Apps/{app}/extensions/` y el Gateway las carga al arrancar; es la única superficie de activación de las extensiones: se pueden ajustar validaciones, reglas y flujos sin necesidad de regenerar código.

7. Compilación y despliegue

```
# Único punto de entrada de compilación (feature gating + aislamiento de target + tratamiento
automático de firmas)
bash scripts/build-ns.sh <NAMESPACE> release    # o dev (modo debug)

# Publicación (sincronización de Apps / seed / .env / apps.json; no compila por sí misma)
bash scripts/deploy/release-to-namespace.sh <NAMESPACE>
```

Productos: `Deploy/{ns}/bin/{ns}-server` (los espacios de nombres con OpenActivity habilitado disponen además de `Deploy/{ns}/Open/bin/openactivity-server`). Al publicar en la etapa production se reinicia automáticamente el Gateway. Las aplicaciones que no son de espacio de nombres (Meta / SSO) siguen `scripts/meta|sso/deploy.sh` y aterrizan en `Release/` .

8. Consulta rápida de puertas de calidad

El repositorio impone la calidad mediante un «sustrato de ejecución de especificaciones»: cada especificación verificable cuenta con un script de comprobación + una línea base + un registro de puerta de control (actualmente 229 puertas registradas: 137 en commit / 92 en push). La biblioteca de especificaciones contiene 403 especificaciones vivas de capability; cada cláusula MUST/NEVER debe declarar su punto de ejecución (id de puerta / script / ruta de test) mediante el campo estructurado «执行面» (superficie de ejecución); las declaraciones en prosa ya no se admiten. Elementos de contacto frecuente:

Puerta de control	Objeto bloqueado	Fase
<code>check-isahl-ddl-boundary</code>	Cualquier escritura estructural isahl que sortee el canal previsto	commit
<code>check-leaf-insert</code>	INSERT que no aterriza en la tabla hoja / literales <code>_f/_t</code> escritos a mano	commit
<code>check-code-semantics</code>	Escritura de semántica de tipos/enumeraciones en la columna code	commit
<code>check-ontology-contract</code>	Forma del producto de mapeo ontológico / falsificación de coordenadas	commit
<code>check-bridge-host-domain</code>	Desalineación del dominio anfitrión en el extremo izquierdo de la tabla de relación	commit
<code>formal-proofs</code>	Fallo en la compilación de los teoremas Lean o presencia de marcadores de prueba pendientes	push
<code>check-all</code> (T1/T2)	Alineación de versiones / conformidad con OpenSpec / i18n / DAG de Services, etc.	push

Cuando hay un bloqueo, no lo sortee: trátelo según la superficie de corrección que indica la salida de la puerta de control; las infracciones preexistentes siguen la línea base (solo pueden disminuir, nunca aumentar) y toda infracción nueva debe corregirse.

9. Práctica en línea: edición de aprendizaje SaaS

Puede experimentar la plataforma completa sin entorno local: <https://cosmic-tools.shop/> ofrece la edición en línea de aprendizaje práctico SaaS de AliothStudio, un despliegue simplificado para usuarios individuales (el modelado central, el ensamblaje de aplicaciones y la línea de ejecución están completos y disponibles; se omiten la orquestación empresarial multi-espacio de nombres y la cadena de publicación). Ruta recomendada: registrarse y entrar → practicar en orden siguiendo los capítulos 2 a 7 de este manual → autoverificar los productos con la lista de puertas de control del capítulo 8.

Bibliografía complementaria: «Libro blanco de la teoría del modelo Alioth» — para comprender la estructura matemática que subyace a todas las restricciones de este manual.

© 2025–2026 Hangzhou Yuqi Technology Co., Ltd. (CosmicTools.ltd).