COSMIC-TOOLS · Yuqi Technology

Manuel de construction d'applications AliothStudio

Livre bleu · parcours opérationnel complet, de la préparation de l'environnement au déploiement et à la mise en service

Version v1.1 · octobre 2026 · Hangzhou Yuqi Technology Co., Ltd.
Versions applicables : modèle Alioth v10.0.44 · AliothStudio 0.1.x

---

Sommaire

1. Composition de la plateforme : deux chaînes
2. Préparation de l'environnement
3. Côté modèle : centre de modèles et publication du modèle
4. Construction d'applications : flux de travail AppAgent
5. Points clés du développement des modules, Block et Service
6. Extensions : extensions/*.yaml
7. Compilation et déploiement
8. Aide-mémoire des portes qualité
9. Pratique en ligne : édition d'apprentissage SaaS

1. Composition de la plateforme : deux chaînes

AliothStudio se compose de deux chaînes aux responsabilités séparées ; comprendre leurs frontières est le préalable à un usage correct de la plateforme :

Chaîne	Responsabilités	Moteur	Produits
Chaîne de publication du modèle	Évolution structurelle du socle de données (tables/colonnes/héritage/codes de dimension)	Centre de modèles Meta (canal manuel)	Paquet de publication `Alioth/v*` (DDL · seed · contrat lisible par machine · rapport de vérification)
Chaîne de construction d'applications	Assembler des applications métier sur le socle figé	Agent App-Agent + développeurs	Configuration d'application, backend Service, frontend de module, paquet de déploiement `Deploy/{ns}`

> **Ligne rouge** : la structure du schéma isahl est figée. Toute demande d'« ajout de table / ajout de colonne / modification de colonne » dans la chaîne de construction d'applications doit revenir à la chaîne de publication du modèle (arbitrage humain au centre de modèles) ; il est interdit de contourner par DDL, script de migration ou écriture directe de métadonnées.

2. Préparation de l'environnement

2.1 Dépendances préalables

Dépendance	Usage
PostgreSQL 14+	Socle de données (le modèle v10.0 est une structure par héritage de tables)
mise	Point d'entrée unifié des tâches et de l'environnement (`mise run <task>`)
Rust toolchain / pnpm / bun	Compilation du backend, compilation du frontend, chaîne d'outils de scripts

2.2 Initialisation et démarrage de la base de données

```
# Initialisation de la base de développement (rejouée depuis l'instantané de publication ; DROP
puis recrée la base cible, réservé à aliothstudio_dev)
bash scripts/db/reset-db.sh --dev

# Plateforme Meta (gestion de modèle + AppAgent) : backend :4949, frontend :49494
mise run meta

# Environnement de développement d'un espace de noms (Gateway + SSO), {ns} ∈ alioth | wz | avic |
cosmic | se | mat
mise run dev:{ns}
```

> La base de données est isolée sur quatre niveaux : production `aliothstudio` ← préproduction `aliothstudio_pre` → développement `aliothstudio_dev` → test `aliothstudio_test`. L'environnement et DATABASE_URL sont liés de manière obligatoire ; toute consultation de schéma passe exclusivement par `cd Meta/backend && mise run schema-info -- <command>` (la base est l'unique source de vérité ; il est interdit de déduire en lisant les fichiers DDL).

3. Côté modèle : centre de modèles et publication du modèle

3.1 Centre de modèles (Meta UI)

Toute évolution structurelle passe par le centre de modèles du frontend Meta : création de collections (collection, rattachée à une table parente pour former un héritage), enregistrement de champs (fields), maintenance des codes de dimension (scène/facteur/fonction). L'enregistrement des collections et des champs est synchronisé programmatiquement par la plateforme vers des tables physiques et des rattachements d'héritage — le développeur n'écrit aucun DDL de création de table.

3.2 Publication du modèle

Une fois la publication lancée depuis le centre de modèles, le pipeline exporte le schéma `isahl` (`pg_dump --schema-only` + les données de 268 tables de seed et le seed de dimensions `seed-dimensions.sql`), les post-traite sous forme de CREATE/ALTER purs, les écrit dans `Alioth/v{version}/` et effectue une reconstruction complète de vérification dans une base de vérification dédiée (aller-retour de structure + aller-retour de seed + unicité des uid), le rapport étant déposé dans `verify-report.json`. Dernière publication en date `v10.0.44` : 1022 tables isahl, 1204 arêtes d'héritage, aller-retour conforme table par table pour

les 268 tables de seed, état verified. Le pipeline de publication produit 7 événements de progression (prepare / schema / isahl_dump / seed_dimensions / manifest / verify / done) ; le niveau de ligne des lignes de collection est porté par la déclaration `row_layer` (abstract / carry) côté modèle, toute table parente non arbitrée contenant des lignes bloquant la publication.

La mise en œuvre du modèle et la surface dérivée de la base de données sont couplées dans un même lot : `refresh-derived-artifacts.ts` est l'unique point d'entrée de rafraîchissement des quatre produits dérivés `fk-index`, `relation-index`, `value-domain-columns` et `dk-projection-tables`, déjà intégré à `reset-db.sh` ; le diff des produits constitue la surface de revue de la topologie du modèle, et la protection contre le retrait est assurée par la porte W9.

3.3 Dimensions et coordonnées

La sémantique métier est localisée par des coordonnées tridimensionnelles : `dk_scene` (scène), `dk_factor` (facteur), `dk_function` (fonction). Le code de dimension est composé automatiquement du code de catégorie et d'un symbole de numéro d'ordre (par exemple `JC` = administration système). Les coordonnées sont interprétées localement par espace de noms ; chaque API d'un Service est statiquement liée à un triplet de coordonnées fixe.

4. Construction d'applications : flux de travail AppAgent

La chaîne de construction d'applications est pilotée par AppAgent, selon la séquence d'outils standard suivante :

```
analyze_ontology     # besoin/prototype → IR OntologyModel (avec validation stricte de la
complétude des cinq domaines)
build_module         # flow-plan → module.json (déclaration de module)
build_block          # flow-plan → block.json (déclaration de bloc de capacité)
transfer_ontology    # alignement ontologique → alignment-graph.json + mapped-entities.json
generate_service_api # entité ontologique → backend Service (Rust, liaison statique des
coordonnées)
compose_app          # assemblage des modules → app.json + extensions/*.yaml + produits de
prototype
```

Emplacement des produits : `Pre-Proc/{ns}/Apps/{app}/` (app.json, extensions, produits de flux) et `Pre-Proc/{ns}/Sources/Apps/{Modules,Blocks,Services}/` (déclarations et code des modules/blocs/services). La forme des entrées du produit de mappage ontologique

(`ontology-output.json`) est contrôlée par une barrière de contrat, et les coordonnées doivent être des codes réels de la base.

5. Points clés du développement des modules, Block et Service

Couche	Contraintes clés
Module	Conteneur de combinaison Scene/Factor ; champs obligatoires de `module.json` au complet ; DashboardPage interdit (saut direct depuis le chemin racine vers la page de liste d'entités)
Block	Unité de capacité ; les ID référencés par `factors` / `services` doivent exister (les références orphelines font l'objet d'une barrière dédiée)
Backend Service	Squelette standard `handlers/models/repositories/services` ; liaison statique du triplet de coordonnées (une même table avec des coordonnées différentes donne des API séparées) ; middleware auth interdit (NGAC unifié par la Gateway) ; crate `tracing` interdite dans un module
Frontend	Gestion d'état uniformément en Jotai v2 ; champs de mesure au format « valeur + unité en choix unique » ; intégration avec le vrai backend, mock interdit ; l'accès aux données passe uniformément par le transport de niveau ressource de `@alioth/api` (`list/get/mutate`, enveloppe déballée via `items()` / `unwrap()`) ; le sélecteur de référence doit toujours utiliser le descripteur `ReferenceSelectDescriptor`, interdiction d'extrapoler le contrat à partir du type d'endpoint

5.1 Règles d'or de l'écriture de données

- **INSERT dans la table feuille** : la cible d'écriture doit être la table feuille de l'ordre partiel d'héritage (la barrière le mesure réellement via pg_inherits ; aucune déduction statique du nom n'est acceptée)
- **L'état passe par une table de relation** : l'état de l'entité, la catégorie et l'étiquette s'écrivent dans la table de relation `zc_id_lifecycle_r_*` ; il est interdit de créer une colonne d'état dans l'entité (le `flag` n'est utilisé que par les lignes du dictionnaire d'états)
- **Contrat de classe à l'écriture** : l'écriture dans une sous-table lifecycle doit fournir un `dk_function` parmi les six préfixes (`_f_` / `_t_` sont dérivés automatiquement ; les littéraux écrits à la main sont interdits)
- **Référence scalaire** : les colonnes `qk_*` sont des pointeurs d'espace de valeurs (bigint) ; toute opération arithmétique sur celles-ci est interdite, la lecture passe par la résolution

`_refs`

- **Sémantique temporelle** : les tranches futures vont dans la famille task, les tranches passées dans la famille event, et l'action d'achèvement se dépose dans la tranche événementielle (bascule de tranche)

6. Extensions : extensions/*.yaml

Les extensions au niveau application (`constraints` / `rules` / `statemachines` / `workflows`) sont placées dans `Pre-Proc/{ns}/Apps/{app}/extensions/` et chargées au démarrage de la Gateway ; c'est la seule surface de pilotage par laquelle une extension prend effet — il est possible d'ajuster validations, règles et flux sans régénérer le code.

7. Compilation et déploiement

```
# Point d'entrée unique de compilation (barrière de feature + isolation de cible + traitement automatique des signatures)
bash scripts/build-ns.sh <NAMESPACE> release    # ou dev (mode debug)

# Publication (synchronisation des Apps / seed / .env / apps.json ; pas de compilation propre)
bash scripts/deploy/release-to-namespace.sh <NAMESPACE>
```

Produits : `Deploy/{ns}/bin/{ns}-server` (les espaces de noms où OpenActivity est activé disposent en outre de `Deploy/{ns}/Open/bin/openactivity-server`). La publication en phase production redémarre automatiquement la Gateway. Les applications hors espace de noms (Meta / SSO) passent par `scripts/meta|sso/deploy.sh` et aboutissent dans `Release/`.

8. Aide-mémoire des portes qualité

Le dépôt impose la qualité au moyen d'un « substrat d'exécution des spécifications » : chaque spécification vérifiable est dotée d'un script de contrôle + d'une base de référence + d'un enregistrement de barrière (actuellement 229 portes enregistrées : 137 en commit / 92 en push). La bibliothèque de spécifications compte 403 spécifications vivantes de capability ; chaque clause MUST/NEVER doit déclarer son point d'exécution (id de porte / script / chemin de test) via le champ structuré « 执行面 » (surface d'exécution) ; les déclarations en prose ne sont plus admises. Éléments les plus fréquemment rencontrés :

Porte	Objet intercepté	Phase
`check-isahl-ddl-boundary`	Toute écriture de structure isahl par contournement	commit
`check-leaf-insert`	INSERT ne tombant pas dans une table feuille / littéraux _f_/_t_ écrits à la main	commit
`check-code-semantics`	Écriture de sémantique de type/énumération dans une colonne code	commit
`check-ontology-contract`	Forme du produit de mappage ontologique / falsification de coordonnées	commit
`check-bridge-host-domain`	Décalage du domaine hôte à l'extrémité gauche d'une table de relation	commit
`formal-proofs`	Échec de compilation des théorèmes Lean ou présence d'emplacements de preuve	push
`check-all` (T1/T2)	Alignement de version / conformité OpenSpec / i18n / DAG des Services, etc.	push

> En cas de blocage, ne pas contourner : traiter le problème selon la surface de correction indiquée par la barrière ; les violations existantes passent par la base de référence (à la baisse uniquement) et les nouvelles violations doivent être corrigées.

9. Pratique en ligne : édition d'apprentissage SaaS

Il est possible d'essayer la plateforme complète sans environnement local : https://cosmic-tools.shop/ propose une édition d'apprentissage SaaS en ligne d'AliothStudio — un déploiement allégé destiné aux utilisateurs individuels (modélisation de base, assemblage d'applications et chaîne d'exécution pleinement utilisables, l'orchestration multi-espaces de noms et le pipeline de publication de niveau entreprise étant omis). Parcours recommandé : s'inscrire et entrer → suivre les chapitres 2 à 7 de ce manuel dans l'ordre → auto-vérifier les produits avec la liste de barrières du chapitre 8.

Document complémentaire : « Livre blanc de la théorie du modèle Alioth » — pour comprendre la structure mathématique qui sous-tend toutes les contraintes de ce manuel.

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