EC3L Platform

Architecture & Workflow

Stateless Multi-Tenant Control Plane

February 2026

Contents

1. High-Level Architecture Diagram
2. Directory Structure
3. Request Flow: Client to Database
4. Service Layer Overview
5. Database Schema Map
6. RBAC Authorization Flow
7. Change Lifecycle State Machine
8. Workflow: Change Lifecycle (End-to-End)
9. Workflow: Execution Flow
10. Environment Release Flow
11. Agent System

12. Runner & Boundary Enforcement
13. Platform Layer
14. Key Architectural Invariants

1. High-Level Architecture Diagram

```
                          +----------------------------+
                          |      REACT CLIENT          |
                          |  Vite + TanStack Query     |
                          |  Tailwind + shadcn/ui      |
                          +-------------+--------------+
                                        |
                           HTTP JSON (x-tenant-id, x-user-id)
                                        |
                                        v
+-----------------------------------------------------------------------------------+
|                          EXPRESS SERVER (routes.ts)                                |
|                                                                                   |
|   +---------------------------+  +----------------+  +-------------------------+  |
|   | Tenant Resolution         |  |  RBAC Layer    |  |  Agent Guard            |  |
|   | slug -> UUID              |  |  authorize()   |  |  assertNotAgent()       |  |
|   | x-user-id / x-agent-id   |  |  audit logging |  |  blocks agent approvals |  |
|   +---------------------------+  +----------------+  +-------------------------+  |
|                                                                                   |
|   API Routes: /projects /changes /environments /workflows /record-types           |
|   /overrides /forms /templates /rbac /agent-proposals /admin                      |
+------+----------------------------+----------------------------------------------+
       |                            |
       v                            v
+----------------+    +---------------------------+    +------------------------+
| SERVICE LAYER  |    |    PATCH OP EXECUTOR      |    |   RUNNER BOUNDARY      |
|                |    |                           |    |                        |
| changeService  |    | 1. Load record types      |    | +--------------------+|
| patchOpService |    | 2. Transform in-memory    |    | | Boundary Guard     ||
| environmentSvc |    | 3. Snapshot + persist      |    | | tenant/module/     ||
| workflowSvc    |    |                           |    | | capability/path    ||
| workflowEngine |    | Ops: set_field add_field  |    | +--------------------+|
| triggerService |    |       remove_field         |    |                        |
| formService    |    |       rename_field         |    | Local or Remote       |
| rbacService    |    +---------------------------+    | Adapter + Telemetry   |
| ...12 more     |                                     +------------------------+
+-------+--------+
        |
        v
+----------------+         +---------------------------------------------+
| TENANT STORAGE |-------->|             POSTGRESQL (Drizzle ORM)        |
| getTenant-     |         |                                             |
| Storage(ctx)   |         |  tenants -> projects -> changes/modules/    |
|                |         |  environments/releases/targets/patch_ops    |
| All queries    |         |  record_types -> snapshots/fields/forms     |
| scoped by      |         |  workflows -> steps/executions/triggers     |
| tenantId       |         |  rbac -> roles/permissions/policies/audit   |
+----------------+         |  agents -> proposals/runs                   |
                           +---------------------------------------------+
```

2. Directory Structure

```
ec3l/
+-- client/          React SPA (Vite + TanStack Query + Tailwind/shadcn)
+-- server/          Express API server
|   +-- middleware/   Tenant resolution
|   +-- services/     Business logic (13 service files)
|   +-- executors/    Patch op executor
|   +-- execution/    Boundary guard
|   +-- __tests__/    Service and executor tests
+-- shared/          schema.ts (Drizzle ORM + Zod) + executionTypes.ts
+-- platform/        Domain modules
|   +-- audit/        Audit event sink
|   +-- cmdb/         Configuration Management Database
+-- runner/          Execution boundary layer
|   +-- adapters/     Local (in-process) + Remote (HTTP) adapters
+-- migrations/      Drizzle SQL migration files
+-- script/          Build + smoke-test scripts
+-- ai-context/      AI context documentation
```

3. Request Flow: Client to Database

```
React Client
  |  localStorage: { tenantId (slug), userId }
  |  Headers: x-tenant-id: "ec3l-labs", x-user-id: "test-user"
  v
Express routes.ts
  |
  +-- GET /api/tenants  (no tenant middleware)
  |
  +-- /api/*  -->  Tenant Resolution Middleware
                     |  1. Read x-tenant-id header (slug)
                     |  2. Query tenants table: slug -> UUID
                     |  3. Read x-user-id / x-agent-id headers
                     |  4. Attach req.tenantContext
                     v
                Route Handler
                     |  1. Zod schema validation on request body
                     |  2. RBAC: authorize(ctx, actor, PERMISSION)
                     |  3. Agent Guard: assertNotAgent(actor)  [if needed]
                     v
                Service Layer (e.g. changeService.ts)
                     |  1. Business logic & validation
                     |  2. State machine enforcement
                     |  3. Cross-service calls (e.g. executePatchOps on merge)
                     v
                getTenantStorage(ctx)
                     |  Closure with tenantId baked in
                     |  Every query: WHERE tenant_id = ?
                     |  Mutations auto-stamp tenantId
                     v
                PostgreSQL via Drizzle ORM
                     |  Returns typed rows
                     v
                JSON Response to Client
```

4. Service Layer Overview

All services receive a TenantContext and delegate to getTenantStorage(ctx) for tenant-scoped database access.

Service	Responsibility
changeService	Change CRUD + deterministic state machine + patch op execution on merge
patchOpService	Create/delete/list patch ops with field validation & duplicate guards
environmentSvc	Environment CRUD + createReleaseSnapshot() for immutable releases
workflowService	Workflow definition/step/execution CRUD
workflowEngine	Core execution loop: assignment, approval, notification, decision, mutation, lock
triggerService	Create/enable/disable triggers; fire manual; emit record events
intentDispatcher	Dequeue pending intents -> build module context -> execute workflow
schedulerService	60-second poll loop; evaluate cron/interval triggers -> create intents
formService	Record type/form CRUD; form compilation; vibe patch AI; form overrides
recordTypeSvc	Record type CRUD with key-based lookup
rbacService	Permission checking, role/policy evaluation, audit logging
agentProposalSvc	Agent proposal lifecycle: draft -> submitted -> accepted/rejected
projectService	Project CRUD (tenant-scoped)
moduleService	Module CRUD (tenant-scoped)
overrideService	Module override lifecycle: draft -> active -> retired
installService	Install templates into tenants; emit install audit events
templateService	Read-only template access (system context)
agentGuardSvc	assertNotAgent() - blocks agent actors from human-only operations

5. Database Schema Map

```
tenants (id, name, slug, plan)
  |
  +-- projects (id, tenantId, name, githubRepo, defaultBranch)
  |     |
  |     +-- modules (id, projectId, name, type, rootPath, capabilityProfile)
  |     +-- environments (id, projectId, name, isDefault)
  |     +-- change_records (id, projectId, title, status, moduleId, environmentId)
  |           |
  |           +-- change_targets (id, changeId, type, ref)
  |           +-- change_patch_ops (id, changeId, targetId, opType, payload, executedAt)
  |           +-- change_events (id, changeId, eventType)  [append-only audit]
  |           +-- agent_runs (id, changeId, intent, status, skills, logs)
  |           +-- agent_proposals (id, changeId, agentId, proposalType, status)
  |
  +-- environment_releases (id, projectId, environmentId, createdBy)
  |     +-- environment_release_changes (releaseId, changeId)  [composite PK]
  |
  +-- environment_deployments (id, environmentId, releaseId, promotedFromReleaseId)
  |
  +-- record_types (id, tenantId, key, projectId, schema, baseType, status)
  |     +-- record_type_snapshots (id, recordTypeId, changeId, previousSchema)
  |     +-- field_definitions (id, recordTypeId, name, type)
  |
  +-- choice_lists -> choice_items
  +-- form_definitions -> form_sections -> form_field_placements
  |                                    -> form_behavior_rules
  |
  +-- workflow_definitions (id, tenantId, name, status)
  |     +-- workflow_steps (id, definitionId, type, orderIndex, config)
  |     +-- workflow_triggers (id, definitionId, type, config, status)
  |
  +-- workflow_executions (id, definitionId, status, pausedAtStepId)
  |     +-- workflow_step_executions (id, executionId, stepId, output)
  |
  +-- workflow_execution_intents (id, definitionId, idempotencyKey, status)
  |
  +-- rbac_roles -> rbac_role_permissions -> rbac_permissions
  +-- rbac_user_roles (userId, roleId, tenantId)
  +-- rbac_policies (roleId, resourceType, resourceId, effect: allow/deny)
  +-- rbac_audit_logs (actorId, permission, decision, resourceType, resourceId)
  |
  +-- record_locks (id, recordType, recordId, lockedBy)
  +-- execution_telemetry_events

templates -> template_modules  (global, not tenant-scoped)
installed_apps -> installed_modules -> module_overrides
installed_app_events
```

6. RBAC Authorization Flow

Permissions Catalog

Permission	Description
form.view	View compiled forms
form.edit	Edit forms, create form overrides
workflow.execute	Execute workflows
workflow.approve	Approve/reject workflow approval steps
override.activate	Activate module overrides
change.approve	Approve changes (Ready/Merged transitions)
admin.view	View Admin Console
environment.release_create	Create environment release snapshots

Default Roles

Role	Permissions
Admin	All 8 permissions
Editor	form.view, form.edit, workflow.execute, override.activate
Viewer	form.view

Authorization Flow

```
authorize(ctx, actor, permission, resourceType?, resourceId?)
  |
  1. Validate permission name against known set (reject unknown)
  2. If SystemContext: allow system actors, deny others
  3. Require actorId to be non-null
  4. Load all active roles for (actorId, tenantId) pair
  5. Check if any role grants the target permission
  6. If resourceType provided:
  |   a. Load all policies for those roles
  |   b. Check for explicit DENY policies (deny wins)
  |   c. Check for ALLOW policies
  7. Record audit log entry (always, regardless of outcome)
  |
  Result: allow or throw RbacDeniedError
```

Actor Resolution

actorFromContext(ctx): requires userId header -> { actorType: 'user', actorId }
resolveActorFromContext(ctx): prefers agentId if present -> { actorType: 'agent' }, else userId
systemActor(): { actorType: 'system', actorId: null }

7. Change Lifecycle State Machine

```
          create
            |
            v
      +-----------+
+-----+   Draft   +<-----------+
|     +-----+-----+            |
|           |                   |
|     status -> Implementing    |
|           |                   |
|     +-----v----------+        |
|     | Implementing   +--------+
|     +-----+----+-----+  status -> Draft
|           |    |
|   status ->    status ->
|  Workspace     Validating
|   Running           |
|     +--------+      |
|     |Workspace|     |
|     |Running  |     |
|     +--+--+--+      |
|        |  |         |
|   back to  status -> |
|   Impl.  Validating  |
|        |  |         |
|        |  v         v
|        | +------------+
|        | | Validating |
|        | +--+------+--+
|        |    |      |
|        |  pass    fail
|        |    |      |
|        |    v      v
|        | +-----+ +------------------+
|        | |Ready| |ValidationFailed  |
|        | +--+--+ +---+----------+---+
|        |    |        |          |
|        |  merge   back to    retry
|        |    |    Implementing Validating
|        |    v
|        | +--------+
|        | | Merged |  <-- executePatchOps() runs here
|        | |(final) |      success -> Merged
|        | +--------+      failure -> ValidationFailed
```

Allowed Transitions

From Status	Allowed Targets
Draft	Implementing
Implementing	WorkspaceRunning, Validating, Draft
WorkspaceRunning	Validating, Implementing
Validating	Ready, ValidationFailed

ValidationFailed	Implementing, Validating
Ready	Merged
Merged	(terminal - no transitions)

Mutability Rules

Mutable statuses (can add/delete patch ops): Draft, Implementing, WorkspaceRunning, ValidationFailed

Immutable statuses (patch set frozen): Validating, Ready, Merged

8. Workflow: Change Lifecycle (End-to-End)

```
Developer/Agent                   Control Plane                        Database
---------------                   -------------                        --------

POST /api/changes
{ projectId, title }
  |
  +----------------------------> changeService.createChange()
  |                                +- validate project exists
  |                                +- resolve module (by ID or path)
  |                                +- resolve default environment
  |                                +- ts.createChange() ------------>  INSERT change_records
  |                                                                    status = "Draft"
  |<-----------------------------  { id, status: "Draft" }

POST /api/changes/:id/targets
{ type: "record_type" }
  |
  +----------------------------> changeTargetService.createTarget()
  |                                +- validate change is mutable
  |                                +- ts.createChangeTarget() ----->  INSERT change_targets
  |<-----------------------------  { id, type, ref }

POST /api/changes/:id/status     updateChangeStatus()
{ status: "Implementing" }         +- assertTransition(Draft->Implementing) OK
                                   +- ts.updateChangeStatus() ----->  UPDATE + INSERT event

POST /api/changes/:id/patch-ops  patchOpService.createPatchOp()
{ targetId, opType:"add_field",    +- validate change is mutable
  payload:{recordType:"task",      +- validate target belongs to change
  field:"priority",                +- validate target type = record_type
  definition:{type:"choice"}}}     +- validateAddFieldPayload()
                                   +- duplicate field guard
                                   +- ts.createChangePatchOp() --->  INSERT change_patch_ops

[Walk through: Implementing -> Validating -> Ready]

POST /api/changes/:id/status     updateChangeStatus()
{ status: "Merged" }               +- assertTransition(Ready->Merged) OK
                                   +- executePatchOps(ctx, changeId)
                                   |    +- Load: fetch ops, cache record types
                                   |    +- Transform: apply ops in-memory
                                   |    +- Persist:
                                   |         createRecordTypeSnapshot() -> INSERT snapshots
                                   |         updateRecordTypeSchema()   -> UPDATE record_types
                                   |         stamp executedAt on ops    -> UPDATE patch_ops
                                   |    +- return { success: true }
                                   +- ts.updateChangeStatus("Merged")-> UPDATE + INSERT event

POST /api/environments/:envId/release
                                 environmentService.createReleaseSnapshot()
                                   +- RBAC: authorize(environment.release_create)
                                   +- validate environment exists
                                   +- getEligibleChanges(envId)
                                   |    Merged AND env matches AND not already released
                                   +- ts.createEnvironmentRelease() -> INSERT releases
                                   +- ts.createReleaseChanges()     -> INSERT release_changes
```

9. Workflow: Execution Flow

```
Trigger Source            Intent Queue            Engine                   Runner
--------------            ------------            ------                   ------

record_event /
schedule /
manual fire
  |
  +- createWorkflow-
  |  ExecutionIntent()
  |  (idempotencyKey)
  +--------------------> INSERT intent
  |                       status: "pending"
  |                             |
  |                      dispatchPendingIntents()
  |                             |
  |                      +------v--------+
  |                      |dispatchIntent |
  |                      | resolve module|
  |                      | build context |
  |                      +------+--------+
  |                             |
  |                      executeWorkflow()
  |                      +- validate def "active"
  |                      +- load steps
  |                      +- createExecution() -----------------> INSERT execution
  |                      |                                        status: "running"
  |                      +- runStepsFromIndex(0)
  |                      |   |
  |                      |   +- Step 1: assignment
  |                      |   |  createStepExec() --------------> INSERT step_exec
  |                      |   |  --------------------------------> runner.executeStep()
  |                      |   |                                     boundaryGuard OK
  |                      |   |                                     telemetry emit
  |                      |   |  <-------------------------------- { result }
  |                      |   |
  |                      |   +- Step 2: approval
  |                      |   |  result: "awaiting_approval"
  |                      |   |  pauseExecution() ---------------> UPDATE execution
  |                      |   |                                     status: "paused"
  |                      |   +- HALT
  |                      |
  |                      +- updateIntentDispatched() -----------> UPDATE intent
  |                                                                  status: "dispatched"
  |
Human: POST /resume
{ stepExecutionId,
  approved: true }
  |
  +----------------------------------------> resumeWorkflow()
  |                      +- RBAC: workflow.approve
  |                      +- assertNotAgent()
  |                      +- runStepsFromIndex(nextStep)
  |                      |   +- Step 3: notification -> emit
  |                      |   +- Step 4: record_mutation -> mutate
  |                      |   +- completeExecution() --------> UPDATE execution
  |                      |                                     status: "completed"
```

10. Environment Release Flow

The environment release is an immutable snapshot that captures all merged-but-not-yet-released changes for a given environment. Releases are append-only and follow a 1:N model (one release can contain many changes).

```
POST /api/environments/:envId/release
  |
  +-> Tenant Resolution Middleware
  +-> resolveActorFromContext(req.tenantContext) -> ActorIdentity
  +-> authorize(ctx, actor, "environment.release_create")
  |
  +-> environmentService.createReleaseSnapshot(ctx, environmentId, actor)
       |
       1. Validate environment exists (tenant-scoped JOIN)
       |
       2. Query eligible changes:
       |     SELECT * FROM change_records
       |     INNER JOIN projects ON projects.id = change_records.project_id
       |     WHERE change_records.status = 'Merged'
       |       AND change_records.environment_id = :envId
       |       AND projects.tenant_id = :tenantId
       |       AND change_records.id NOT IN (
       |           SELECT change_id FROM environment_release_changes
       |           INNER JOIN environment_releases
       |           WHERE environment_releases.environment_id = :envId
       |       )
       |
       3. If no eligible changes -> 409 "No eligible merged changes"
       |
       4. INSERT INTO environment_releases
       |     (project_id, environment_id, created_by)
       |
       5. INSERT INTO environment_release_changes
       |     (release_id, change_id) for each eligible change
       |
       6. Return { id, environmentId, projectId, createdBy, createdAt, changeIds[] }
```

Release Model

```
environment_releases (immutable snapshot header)
  |-- id              (PK, auto-generated UUID)
  |-- project_id      (FK -> projects)
  |-- environment_id  (FK -> environments, NOT NULL)
  |-- created_by      (actor who created the release)
  |-- status          (enum: created/deploying/deployed/failed)
  |-- created_at

environment_release_changes (join table)
  |-- release_id      (FK -> environment_releases)
  |-- change_id       (FK -> change_records)
  |-- PRIMARY KEY (release_id, change_id)

environment_deployments (head pointer)
  |-- id
  |-- environment_id
  |-- release_id
  |-- promoted_from_release_id  (tracks promotion chain)
```

11. Agent System

Agent Proposal Lifecycle

```
Agent: createProposal()  ->  status: "draft"
  |  proposalType: form_patch | workflow_change | approval_comment
  |  linked to a change (must be Draft status)
  |
Human: submitProposal()  ->  status: "submitted"    [assertNotAgent]
  |
Human: reviewProposal()  ->  status: "accepted"     [assertNotAgent + change.approve]
                          or  status: "rejected"
```

Agent Guard - Protected Operations

The agentGuardService.assertNotAgent() function blocks agent actors from performing human-only operations. This enforces the principle: agents propose, humans decide.

Protected Operation	Endpoint
Approve changes	POST /changes/:id/status (Ready/Merged)
Check-in changes	POST /changes/:id/checkin
Merge changes	POST /changes/:id/merge
Execute workflows	POST /workflow-definitions/:id/execute
Resume workflow executions	POST /workflow-executions/:id/resume
Fire manual triggers	POST /workflow-triggers/:id/fire
Activate overrides	POST /overrides/:id/activate
Submit proposals	POST /agent-proposals/:id/submit
Review proposals	POST /agent-proposals/:id/review

12. Runner & Boundary Enforcement

The runner is an execution boundary layer between the control plane and compute resources. It validates every execution request at the boundary before delegating to the actual runner service.

Adapter Selection

```
RUNNER_ADAPTER env var:
  "local"  (default) -> LocalRunnerAdapter  (in-process, SimulatedRunnerService)
  "remote"           -> RemoteRunnerAdapter (HTTP to RUNNER_URL, default :4001)
```

Boundary Guard Checks

Check	What It Validates
Tenant Context	tenantId non-empty, source is 'header' or 'system'
Module Context	moduleId, moduleRootPath, capabilityProfile all present
Tenant Immutability	tenantId + source match between top-level and nested context
Capability Validation	Every requested capability in moduleExecutionContext.capabilities
Path Validation	No absolute paths, no '..' traversal, within moduleRootPath

Capability Profiles

Profile	Use Case
CODE_MODULE_DEFAULT	Standard code module execution
WORKFLOW_MODULE_DEFAULT	Workflow step execution
READ_ONLY	Read-only operations
SYSTEM_PRIVILEGED	System-level operations

13. Platform Layer

The platform layer contains standalone domain modules not yet wired into the main Express server. These are designed as independent, storage-agnostic, tenant-isolated services.

CMDB (Configuration Management Database)

```
platform/cmdb/
  +-- graph/     Core types: CINode, CIEdge, CMDBGraph
  |                CINode: ciId, ciType, tenantId, attributes,
  |                        lifecycleState (planned/active/deprecated/retired),
  |                        source (manual/discovery/integration/agent)
  |                CIEdge: directed relationships between CI items
  |
  +-- store/     GraphStore interface, InMemoryGraphStore implementation
  |
  +-- service/   CMDBService: getNode, listNodes, upsertNode, deleteNode,
  |                           getEdge, listEdges, upsertEdge, deleteEdge
  |              All operations take TenantContext + optional GovernanceContext
  |
  +-- core/      createDevCMDB() factory for dev environment

platform/audit/
  +-- AuditEvent.ts         AuditEventType: CMDB_NODE_UPSERTED, _DELETED, etc.
  +-- AuditSink.ts          Interface for audit event sinks
  +-- InMemoryAuditSink.ts  In-memory implementation for dev/testing
```

14. Key Architectural Invariants

Principle	Enforcement
Stateless multi-tenant	No sessions/cookies. Tenant resolved from x-tenant-id slug on every request
Tenant isolation	getTenantStorage(ctx) closes over tenantId; every query scoped
Deterministic state machine	ALLOWED_TRANSITIONS map checked on every status update
Merge = execute	Transitioning to Merged always runs patchOpExecutor
Agents cannot approve	assertNotAgent() at every human-only decision point
Snapshot before mutation	Record type snapshots captured before schema changes
Idempotent intents	Workflow intents carry idempotencyKey to prevent double-firing
Immutable releases	Release snapshots append-only; released changes excluded from future
Capability-gated exec	Runner boundary validates tenant, module, capabilities, path