



ENTERPRISE PRODUCT SPECIFICATION & ARCHITECTURE MANUAL

CEEZ Semantic Operations Platform

Source of Truth Foundation (SOTF Engine) — Architecture, Multi-Agent Deliberation, CI/CD Pipeline, Runtime Governance & API Reference

System:	CEEZ AI · Semantics (SOTF)
Document Version:	v2.4.0 (Enterprise Release)
Verification Status:	570/570 Automated Checks Passed
Authoring Team:	Google Antigravity & CEEZ Engineering
Date:	September 2026

CEEZ Semantic Operations Platform (SOTF Engine)

Comprehensive Architecture, Features, Functions, and API Reference Manual

1. Executive Summary & Core Mission

Foundational Thesis

2. Platform Architecture Overview

High-Level Component Interactions

3. Plane 1: CONNECT — Data Ingestion, Tiers & Security

3.1 PostgreSQL Engine & Read-Only Invariant

3.2 Tier 0 / Tier 1 / Tier 2 Data Access Tiers

3.3 Multi-Source Cross-Estate Indexing (`source_index.py`)

3.4 Unstructured Document Ingestion (`documents.py`)

4. Plane 2: UNDERSTAND — The Semantic Workforce & Adversarial Debate

4.1 Structural Graph Derivation (`sotf_engine.py`)

4.2 The Semantic Workforce (Standing Team A Roles)

4.3 The 4-Round Adversarial Debate Protocol

4.4 The Trust Ladder & Epistemic States

4.5 Temporal Investigator (`temporal.py`)

5. Plane 3: COMPILE — Business Intents & Executable Contracts

5.1 Business Intent Discovery & Signal Matching (`intent_engine.py` , `metrics.py`)

5.2 The 10-Point Semantic Validation Checklist

5.3 Assessment & Autonomy Ceilings

5.4 Executable Semantic Contracts (`contracts.py`)

6. Plane 4: GOVERN — The Release Pipeline & Governed Runtime

6.1 The 12-Stage Semantic CI/CD Pipeline (`release.py`)

6.2 The Governed Runtime Decision Engine (`runtime.py`)

7. Plane 5: IMPROVE — Continuous Sensing, Remapping & Learning

7.1 Two-Stage Watcher Loop (`watcher.py`)

7.2 Standing Team B: Impact Analysis & Remapping (`impact.py`)

7.3 Outcomes Recording & Root-Cause Walk (`outcomes.py`)

8. Multi-Source Federation & Contradiction Detection

8.1 Estate Federation (`federation.py`)

8.2 Cross-Source Contradictions (`contradictions.py`)

9. Consumption Interfaces: MCP Server, Ask Engine & Data Catalog

9.1 Model Context Protocol (MCP) Server (`mcp_server.py`)

9.2 The Natural Language Ask Engine (`ask.py` , `sqlguard.py`)

9.3 Data Catalog & Semantic Register (`catalog.py` , `dictionary.py`)

10. Frontend Architecture & User Journey

10.1 The 4-Step Cycle Navigation

10.2 Command Bar (⌘K)

-
- 11. Complete REST API Reference
 - 11.1 Release Pipeline Endpoints
 - 11.2 Catalog & Extraction Endpoints
 - 11.3 Intent & Lifecycle Endpoints
 - 11.4 Contracts & Governance Endpoints
 - 11.5 Runtime Decision & Outcome Endpoints
 - 11.6 Team B & Impact Endpoints
 - 11.7 Documents & Policy Endpoints
 - 11.8 Multi-Source Federation & Contradiction Endpoints
 - 11.9 Semantic Workforce & Agent Configuration
 - 11.10 Ask, Dictionary & Curation Endpoints
 - 12. Verification & Testing Framework
 - Key Verification Highlights
-

CEEZ Semantic Operations Platform (SOTF Engine)

Comprehensive Architecture, Features, Functions, and API Reference Manual

1. Executive Summary & Core Mission

The **CEEZ Semantic Operations Platform** (also known as the **SOTF Engine** — *Source of Truth Foundation*) is an enterprise semantic operations and governance platform. It solves the foundational vulnerability of enterprise AI agents: **agents hallucinating meaning, misinterpreting relational data schemas, and executing actions without semantic bounds or policy authority.**

Traditional metadata catalogs and vector databases only offer static schema dumps or naive vector embeddings. A conventional name-matcher will misclassify `sq_footage` as a temporal duration (due to substring “age”), `operation_id` as a financial ratio, or `capacity_tons_per_month` as a foreign key pointing to a `capacity` table. In production, such errors cause catastrophic failures, such as agents declaring victory on failing KPIs or executing unauthorized operations.

Foundational Thesis

“A single agent asserting meaning is worthless. Any name-matcher will tell you `capacity_tons_per_month` is a reference to a `capacity` table, and be confidently wrong. What makes a semantic layer usable is not a better guesser but an adversarial process: every claim must cite evidence, every claim gets attacked by an agent whose only job is to attack it, the author must answer or withdraw, and an adjudicator applies stated rules.”

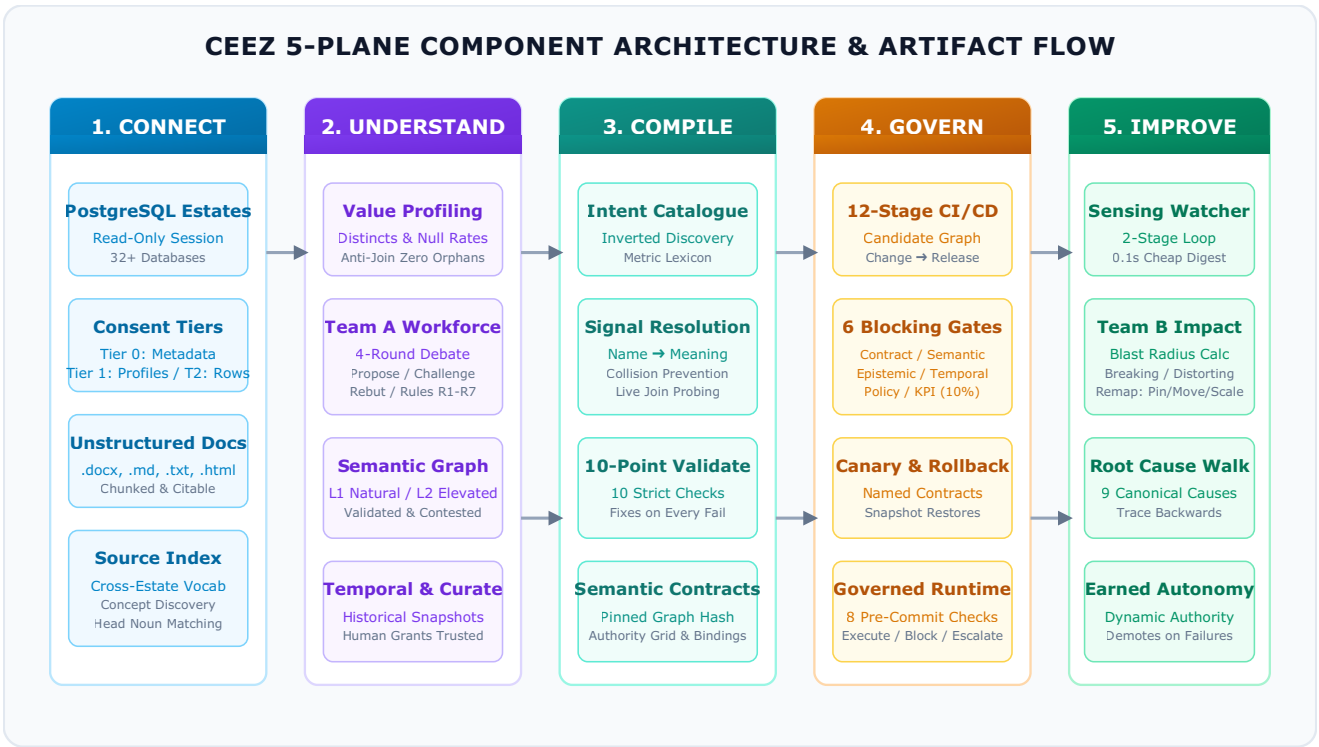
The platform connects to live enterprise databases, extracts relational topologies, profiles value distributions, hosts an adversarial multi-agent debate to adjudicate truth, compiles business intents into versioned **Executable Semantic Contracts**, enforces an automated **Semantic CI/CD Pipeline** with blocking gates, and governs runtime agent decision-making through **8 Pre-Commit Validation Checks**.

2. Platform Architecture Overview

The platform is designed around five architectural planes outlined in the CEEZ Product Blueprint:

CEEZ PLATFORM PLANES				
1. CONNECT	2. UNDERSTAND	3. COMPILE	4. GOVERN	5. IMPROVE
• PostgreSQL • Tier 0/1/2 Consent • Unstructured Doc Ingestion • Federation Index	• Team A Workforce • 4-Round Debate • Rules R1-R7 • Temporal Semantics • Curation (Trusted)	• Intent Discovery • 10-Point Validation • Semantic Contracts • Authority Matrix • Systems Bindings	• Release Pipeline • 6 Release Gates • Governed Runtime • 8 Pre-Commit Checks • Sandbox	• Team B Impact Analysis • Remapping Engine • 9 Root Causes • Continuous Earned Authority • Outcome Measurement • Drift & Distort Remedies

High-Level Component Interactions



3. Plane 1: CONNECT — Data Ingestion, Tiers & Security

3.1 PostgreSQL Engine & Read-Only Invariant

All database interactions are managed via `psycopg2`. Connections are strictly isolated to **read-only sessions**:

```
conn.set_session(readonly=True, autocommit=True)
```

The platform never creates, alters, updates, or deletes tables in client databases.

3.2 Tier 0 / Tier 1 / Tier 2 Data Access Tiers

The platform implements a formal data-consent model that balances commercial governance, privacy compliance (DPO/GDPR), and semantic depth:

Access Tier	Permissions & Operations	Data Exposure	Discovery Capability
Tier 0	Metadata & structural catalogs only (<code>information_schema</code> , <code>pg_class.reltuples</code>). No rows read.	Zero user row data.	Discovers tables, declared foreign keys, column names, estimated row counts. Fast estate indexing (32 DBs in 14s).
Tier 1	Value profiles: <code>COUNT(DISTINCT)</code> , null rates, numerical ranges, shape regexes, anti-join overlap checks. Up to 12 sample values for candidate evaluation.	Aggregate statistics; sample values displayed strictly in internal UI.	Unlocks undeclared foreign key discovery (zero-orphan anti-joins), identifies actual candidate keys, flags mostly-null columns.
Tier 2	Full row content interrogation, custom KPI measurements, deep query probing, and optional transmission of up to 4 example values per column to external LLMs.	Real row values sent to model if explicitly enabled by administrative toggle.	Enables rich LLM business interpretation, custom business rule verification, live metric computation.

3.3 Multi-Source Cross-Estate Indexing (`source_index.py`)

On startup, `source_index.py` inspects all accessible databases in the PostgreSQL cluster. It indexes: - **Estate Vocabulary**: Aggregates all business head nouns and tokens, ranked by frequency across sources. - **Concept Discovery**: Resolves generic business concepts (e.g. `customer` , `store` , `order`) against all connected databases, calculating match scores based on head nouns, signal column availability, and entity roles.

3.4 Unstructured Document Ingestion (`documents.py`)

Connects corporate policy documents to machine-executable semantics: - Supported formats: `.docx` , `.md` , `.txt` , `.html` . (PDF is rejected with an honest explanation rather than half-parsed). - Automatically splits documents into structural sections and numbered chunks. - Every extracted passage retains verbatim coordinates (`document_id` , `section` , `chunk`) for downstream citation auditing.

4. Plane 2: UNDERSTAND — The Semantic Workforce & Adversarial Debate

Understanding enterprise data is not left to a single probabilistic prompt. It is executed by **Standing Team A** through a structured, multi-round adversarial debate governed by explicit rules.

4.1 Structural Graph Derivation (`sotf_engine.py`)

The extraction engine derives four graph tiers directly from PostgreSQL:

1. L1 Natural Nodes (Entity Classification)

Every table is assigned a structural role based on keys and cardinality: - **MASTER_DATA** : Core business entities (independent keys, persistent lifetime, referenced by transactions). - **TRANSACTION** : High-volume operational events (foreign keys pointing to multiple masters, event timestamps). - **REFERENCE** : Lookup tables, codes, statuses, taxonomies (low row counts, referenced across tables). - **BRIDGE** : Associative tables representing many-to-many relationships (composite primary keys formed of foreign keys). - **AUDIT** : System changelogs, queue logs, event tracking (created/updated timestamps, approval trails).

2. L2 Elevated Nodes (Elevation Score)

Categorical columns are elevated to first-class concepts based on the formula: $ES = 0.30 \cdot RF + 0.30 \cdot SR^* + 0.20 \cdot BRA + 0.15 \cdot ED + 0.05 \cdot CER$ - **RF (Reference Frequency)**: How widely referenced across queries/joins. - **SR* (Semantic Richness)**: Structural proxy for informational depth. - **BRA (Business Rule Activity)**: Frequency of appearance in predicates. - **ED (Entropy / Distinctness)**: Information density across distinct values. - **CER (Cardinality / Extremity Ratio)**: Balance of unique categories.

3. L3 Virtual Nodes

- **Event Nodes**: Synthetic virtual entities combining transactions joining several masters.
- **Path Nodes**: Permissible traversals and customer/operational journeys supported by key relationships.

4. Undeclared References Detection

Identifies unstated foreign keys across tables by executing value-overlap anti-joins: - Requires **zero orphans** (100% distinct value containment). - Enforces three anti-hallucination guards: 1. Generic tokens (`id` , `key` , `code` , `no`) are never sufficient evidence alone. 2. Physical measures are never references (e.g. `capacity_tons_per_month` cannot point to `capacity_id`). 3. Matches the **head noun**, not a modifier (e.g. `store_id` cannot point to `store_operations.operation_id`).

5. Graph Edge Classifications & Clear Refusals

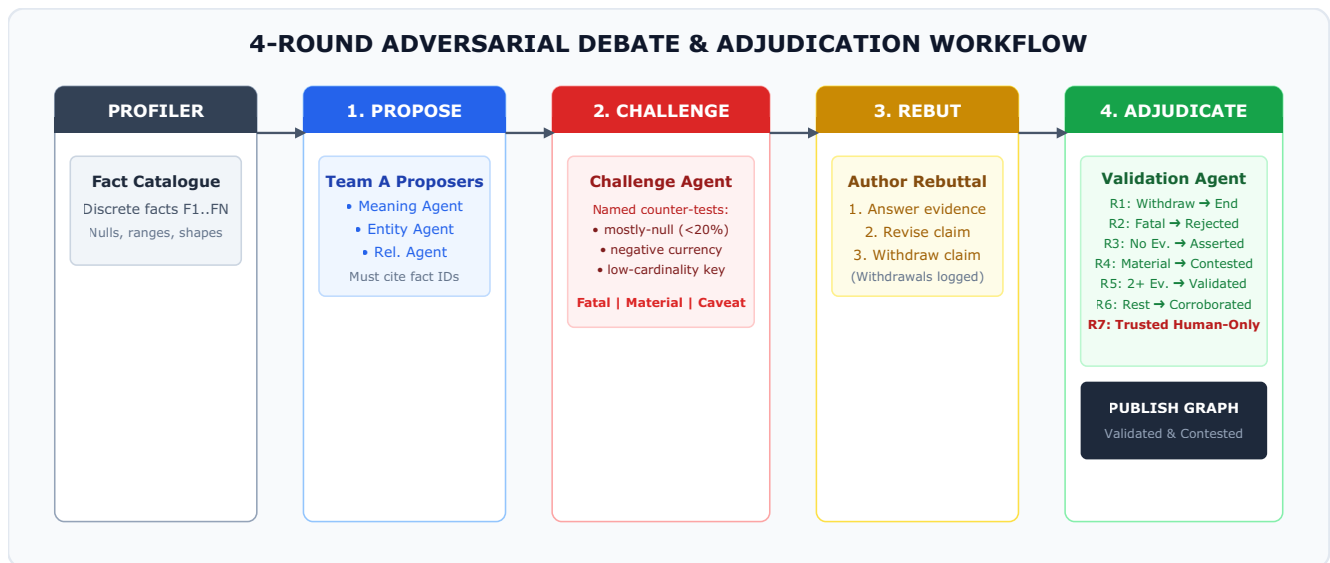
- **Edges Emitted**: **STRUCTURAL** (declared FK), **INFERRED_REFERENCE** (zero-orphan anti-join), **CLASSIFICATION** , **HIERARCHICAL** , **TEMPORAL** , **LIFECYCLE** .
- **Refused Edges**: The engine explicitly refuses to fabricate edges it cannot evidence:
 - **BEHAVIORAL** & **SIMILARITY** : Requires row-level machine learning; not attempted.
 - **CAUSAL** : Refused outright. Testing numerical pairs produces spurious correlations.
 - **DEPENDENCY** , **CAPABILITY** , **PROCESS** , **OPERATIONAL** : Absent from relational schemas; flagged as unevidenced.

4.2 The Semantic Workforce (Standing Team A Roles)

`workforce.py` and `roles.py` organize the standing agents into defined charters:

Agent Role	Kind	Default Engine	Allowed Trust Writing	Charter
Discovery Agent	Propose	Rule	Asserted	Inventories tables and columns; flags empty or un-evidenced schemas.
Entity Resolution Agent	Propose	Rule	Asserted, Evidenced	Identifies primary keys, natural keys, and row identity columns.
Business Meaning Agent	Propose	LLM / Rule	Asserted, Evidenced	Assigns semantic types from value distributions and shapes, not bare names.
Relationship Agent	Propose	Rule	Asserted, Evidenced	Evaluates joins, measuring coverage %, fan-out ratios, and orphan counts.
Maintenance Agent	Propose	Rule	Asserted, Evidenced	Monitors schema fingerprints, row updates, and staleness counters.
Challenge Agent	Challenge	LLM / Rule	Contested	Attacks proposed claims with named, executable counter-tests.
Evidence Agent	Audit	Rule	Evidenced, Corroborated	Audits citation links; enforces the 90-day autophagy rule.
Validation Agent	Adjudicate	Rule	Validated, Contested, Corroborated, Asserted	Adjudicates claims by deterministic rules R1–R7. Cannot write Trusted.
Compilation Agent	Compile	Rule	Published	Compiles the adjudicated claims into the published semantic graph.
Context Agent	Package	Rule	Published	Packages semantic briefs and table summaries for consuming agents.

4.3 The 4-Round Adversarial Debate Protocol



- Round 1: Propose:** Proposers emit claims citing discrete, numbered facts (F1 , F2 ...) from the profiler. Uncited claims cannot advance past `Asserted`.
- Round 2: Challenge:** The Challenge Agent applies named counter-tests:
 - `mostly-null` : Value presence < 20% of rows.
 - `unconstrained-range` : Unbounded numeric spread inconsistent with type.
 - `negative-values` : Negative numbers on currency/quantity/count columns.
 - `measure-not-identifier` : High entropy float mistaken for an entity identifier.
 - `low-cardinality` : Categorical column mistaken for a unique primary key.
 - `no-matching-key` : Identifier with no matching target domain table.
 - `stale-data` : Table has ceased receiving event updates. Objections are graded as **fatal**, **material**, or **caveat**.
- Round 3: Rebut:** The authoring agent must answer the counter-test, **revise** the claim (e.g. converting `identifier` to `reference` or `code`), or **withdraw** the claim.
- Round 4: Adjudicate:** The Validation Agent evaluates the transcript against the Rule Book:
 - R1:** An author who withdraws ends the matter → `Withdrawn`.
 - R2:** A standing fatal objection rejects the claim → `Rejected`.
 - R3:** No independent evidence cited → capped at `Asserted`.
 - R4:** Standing material objections → capped at `Contested`.
 - R5:** ≥ 2 independent facts, weight ≥ 1.5, only caveats → `Validated`.
 - R6:** Any other evidenced claim rests at → `Corroborated`.
 - R7:** `Trusted` is not an agent's to grant. Any attempt by an agent to write `Trusted` is automatically capped to `Validated`.

4.4 The Trust Ladder & Epistemic States

The platform enforces two orthogonal state vectors:

```

Trust Ladder (Process Rigor):
Asserted → Evidenced → Corroborated → Validated → Published → Trusted (Human-only)
    |           |
    └─[Objection Material]─→ Contested (Published with caveats)
    └─[Objection Fatal]──→ Rejected / Withdrawn (Withheld)

Epistemic State (Operational Reality):
KNOWN → STALE (Validation window expired) → Revalidation Queue
UNKNOWN → Non-executable by construction
INVALIDATED → Root-cause failure confirmed; revoked
DEPRECATED → Superseded by human or policy update
CONTRADICTED > Disputed across systems; blocks execution

```

- **Publishing Contested Meaning:** Contested claims are published to the graph carrying their objections. This enables downstream systems (like Team B) to distinguish between a column whose scale changed versus one that vanished entirely.
- **Autophagy Prevention:** Evidence authored by the platform itself carries zero weight for 90 days (`AUTOPHAGY_DAYS = 90`), preventing self-reinforcing hallucination loops.
- **Human Curation (`curation.py`):** The only mechanism in the entire platform that can elevate a claim to `Trusted` is explicit human sign-off via `/api/curate`. Human overrides form a persistent overlay that surviving re-extractions cannot erase.

4.5 Temporal Investigator (`temporal.py`)

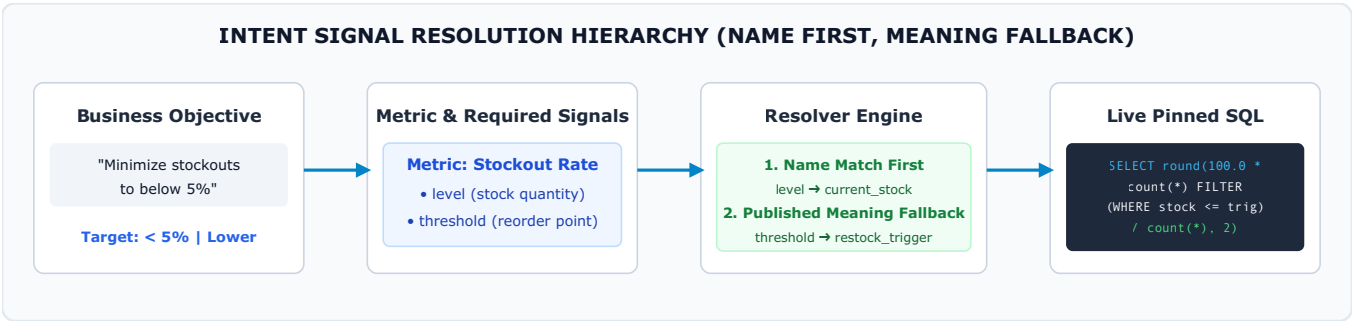
Relational schemas change over time. The Temporal Investigator maintains validity intervals: - **Meaning Projections:** Records snapshots of fields that define what an attribute is: `semantic_type`, `scale`, `unit`. (Confidence and trust do not trigger new temporal windows). - **Historical Queries:** `definition_at(db, attribute, timestamp)` recovers exact historical semantics. - **Honest Unverifiability:** If asked about a date prior to the snapshot store, the system answers `unverifiable` rather than guessing or pretending nothing changed.

5. Plane 3: COMPILE — Business Intents & Executable Contracts

The platform inverts the traditional intent paradigm: instead of asking “*here is my intent, where is the data?*”, it asks: > “**Here is the data, what business intents can it honestly support?**”

5.1 Business Intent Discovery & Signal Matching (`intent_engine.py`, `metrics.py`)

`intent_catalogue.py` ships an extensible catalogue of enterprise business intents (e.g. Stockout Prevention, Margin Leakage, Promotional ROI, Customer Churn).



Signal Resolution Hierarchy

- 1. Name Match:** Searches for conventional identifier names (`current_stock` , `reorder_point`).
- 2. Published Meaning Match:** If renamed (e.g. `reorder_point` → `restock_trigger`), resolves via the workforce-published semantic type (`quantity` , `threshold`).
- 3. Collision Guards:** A single column cannot satisfy two slots (e.g. preventing `current_stock <= current_stock` , which would produce 100% false stockout rates).

5.2 The 10-Point Semantic Validation Checklist

Before any intent can advance to execution, it must pass 10 rigid checks (`intent_lifecycle.validate`):

#	Validation Check	Failure Condition	Remediation Fix Provided
1	Measurable Outcome	Objective lacks metric, direction, or target.	State direction (lower/higher) and target threshold.
2	Required Concepts Sourced	Any required concept lacks an assigned table.	Select a candidate source table from the estate.
3	Signal Columns Present	Assigned tables lack the columns the metric requires.	Select a table carrying necessary level/threshold columns.
4	Sources Connected	Selected table resides in an unlinked database.	Connect the database or federate sources.
5	Concepts Connected in Data	Join path across concepts yields 0 rows.	Declare missing join or resolve foreign keys.
6	Metric Computes Today	Pinned measurement SQL fails to execute.	Fix syntax or resolve incompatible data types.
7	Target Worth Pursuing	Target is already met, or delta exceeds single-cycle limit.	Set an unachieved, realistic operational target.
8	Granted Access Sufficient	Required access tier exceeds customer grant.	Escalate consent from Tier 1 to Tier 2.
9	Actions Defined	Intent specifies no executable actions (“Report, not intent”).	Define at least one operational action in catalogue.
10	Owner Named	No business owner or accountable email assigned.	Assign a named person; unowned intents cannot reach agents.

5.3 Assessment & Autonomy Ceilings

Intents that pass validation are assigned an evidence-backed autonomy ceiling: - **L0 (Observe Only)**: Validation failures or missing signals present. - **L1 (Recommend)**: Tier below 2; agent may suggest actions to a human queue. - **L2 (Human Approved)**: Reservations or caveats remain; execution requires explicit human authorization. - **L3 (Policy Bounded)**: Clean validation; agent may act autonomously within pre-compiled policy parameters. - **L4/L5 (High / Autonomous)**: Requires earned operational track record over time.

5.4 Executable Semantic Contracts (`contracts.py`)

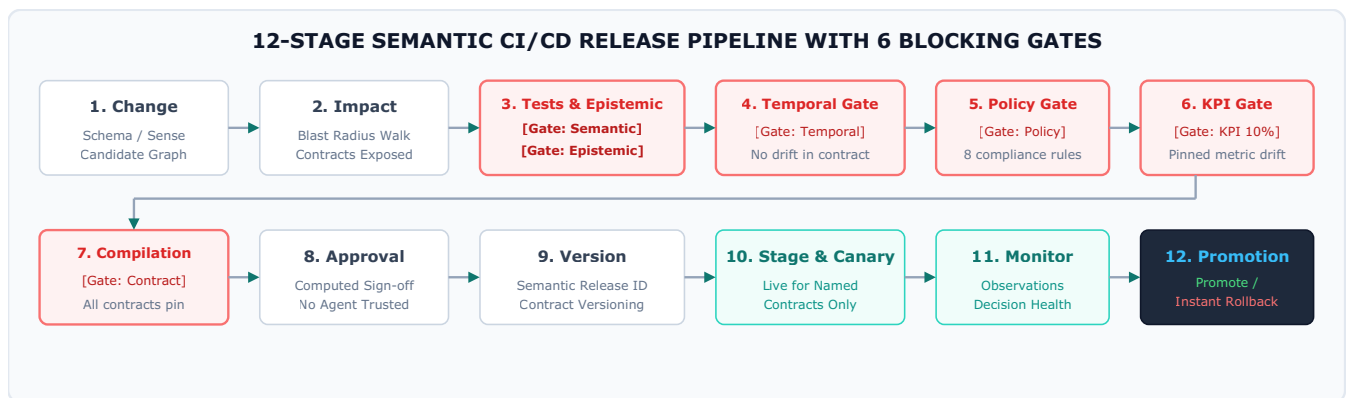
When an intent is implemented, it compiles into a permanent, versioned contract stored in `contracts.json` : - **Graph Fingerprint Pinning**: A SHA-256 hash of all node semantic types, scales, and edge topologies. If the underlying database changes, the contract does not silently drift — it becomes `STALE` . - **The Four-Way Authority Grid**: - `know` : Concepts the agent is authorized to read. - `infer` : Relationships the agent is authorized to traverse. - `decide` : Decision classes the agent is authorized to evaluate (with autonomy level). - `act` : Physical actions the

agent is permitted to propose. - **System Bindings Registry**: Declares systems the agent can touch (e.g. `postgres`, `register`, `mcp`, `notification`). Unregistered bindings are blocked. - **Derived Expiry**: Checked against `valid_until` dynamically; expired contracts lose authority immediately.

6. Plane 4: GOVERN — The Release Pipeline & Governed Runtime

6.1 The 12-Stage Semantic CI/CD Pipeline (`release.py`)

A new debate produces a **Candidate Graph**. It does not become live until it passes all twelve pipeline stages:



The 6 Blocking Gates

- Contract Compilation Gate (`contract`)**: Verifies every active contract compiles cleanly against the candidate graph.
- Semantic Assertions Gate (`semantic`)**: Evaluates assertions generated directly from contracts (e.g. column types, join coverage $\geq 90\%$).
- Epistemic State Gate (`epistemic`)**: Confirms no input concept is `UNKNOWN`, `STALE`, or `INVALIDATED`.
- Temporal Gate (`temporal`)**: Confirms no core definition shifted under a signed contract.
- Policy Gate (`policy`)**: Runs 8 compliance checks verifying that proposed agent actions do not violate ingested corporate policies.
- KPI Regression Gate (`kpi`)**: Executes pinned measurement queries against live data; flags any move exceeding a 10% relative tolerance.

Canary Scoping & Rollback Invariants

- Named Contract Canary**: In a semantic layer, traffic cannot be split by percentage. The canary runs the candidate graph for **specifically named contracts** while leaving all others on the live graph.
- Restoration, Not Inverse-Replay**: Releases store the complete candidate graph. Rollback restores the prior graph snapshot and contract states intact.

6.2 The Governed Runtime Decision Engine (`runtime.py`)

When an autonomous agent proposes an action at runtime, `runtime.evaluate()` executes **8 Pre-Commit Validation Checks in strict sequence**:

8 PRE-COMMIT RUNTIME CHECKS

1. Contract Check	: An active, unexpired contract exists
2. Semantics Check	: Meaning is settled (no contradictions)
3. Evidence Check	: Inputs meet minimum trust & are not UNKNOWN / INVALIDATED
4. Freshness Check	: Data is within max_staleness_hours
5. Temporal Check	: Contract applies to current window
6. Policy Check	: Action satisfies all bound policies
7. Decision Check	: Agent has decision authority level
8. Action Check	: Action is in catalogue & bindings

Runtime Outcomes

- `execute` : All checks pass cleanly; action is authorized.
- `escalate` : Action exceeds autonomy ceiling or touches funds requiring two-person sign-off; queued for human approval.
- `downgrade` : Non-fatal warning or partial evidence; action restricted to recommendation.
- `block` : Fails any pre-commit check; action execution is halted.

Sandbox Simulation & Revalidation Boundary

- `runtime.simulate()` : Produces a deterministic report of what the action would execute, without performing external calls.
- `runtime.revalidate()` : Enforces a semantic transaction boundary — revalidates all 8 checks immediately prior to final execution commit.

7. Plane 5: IMPROVE — Continuous Sensing, Remapping & Learning

7.1 Two-Stage Watcher Loop (`watcher.py`)

Schema drift breaks analytics silently. The Watcher monitors databases on a background thread: 1. **Cheap Stage (0.1s)**: Reads `information_schema` table counts, `reltuples`, and computes a structural hash. If counters and structure are unmoved, it exits as a no-op. 2. **Expensive Stage**: If and only if a structural hash moves, it triggers profiling and targeted re-debate for the affected tables.

7.2 Standing Team B: Impact Analysis & Remapping (`impact.py`)

When Team A updates the semantic graph, Team B calculates the operational blast radius on active contracts:

Notice Severity	Definition	Example Scenario	System Response
breaking	The metric can no longer compute.	A required column was dropped or renamed without fallback.	Suspends contract authority; queues urgent remap proposal.
distorting	The query computes, but the number now means something else.	A unit scale moved from 0–100 to 0–1 (e.g. <code>waste_pct</code>).	Flags 100x error; proposes <code>rescale</code> factor in SQL.
advisory	Trust or caveats shifted, but computation remains sound.	Confidence dropped or a new non-fatal caveat was attached.	Records notice in governance queue.
opportunity	A new column or join was added that improves precision.	A dedicated <code>discount_amount</code> column was created.	Recommends upgrading the intent's signal resolution.

Remapping Proposals

- `pin` : Switch signal resolution to an alternate column.
- `move` : Shift concept sourcing to a different table.
- `rescale` : Inject mathematical scaling factors into measurement SQL.
- **Tombstones**: Rejections are permanently stored so dismissed notices never recur.

7.3 Outcomes Recording & Root-Cause Walk (`outcomes.py`)

Post-execution results are tracked against the contract's pinned baseline. When an outcome is judged `poor` , the platform walks backward along the provenance chain across **9 Canonical Root Causes**: 1. `identity` : Incorrect cross-source entity resolution. 2. `missing` : Required business concept was absent from the graph. 3. `relationship` : An assumed join carried unintended fan-out or orphans. 4. `stale` : Definition was outside its valid trust window. 5. `policy` : Corporate policy shifted after contract compilation. 6. `evidence` : Inputs failed the contract's trust threshold. 7. `temporal` : Definition did not apply to the historical time window. 8. `authority` : Agent was granted excessive autonomy for the evidence. 9. `intent` : The business objective itself was poorly specified.

Continuous Effective Authority Function

Authority is not static. If an agent incurs policy violations, unaddressed contradictions, or degraded accuracy, `outcomes.effective_authority()` **automatically demotes the agent's autonomy level** (e.g. from L3 down to L2 or L1).

8. Multi-Source Federation & Contradiction Detection

8.1 Estate Federation (`federation.py`)

Reads multiple disparate databases simultaneously, creating source-qualified keys (`database.table.column`) and establishing candidate cross-source entity identities.

8.2 Cross-Source Contradictions (`contradictions.py`)

Detects enterprise disagreements across sources (e.g. CRM vs Billing vs Support on `churn`):

- **definition** : Incompatible business definitions for the same concept.
- **type** : Same-named attribute carries contradictory types across systems (e.g. `audit_time` vs `event_time`).
- **grain** : Sources aggregate the same entity at conflicting granularities.
- **Execution Blocking**: A `CONTRADICTED` concept cannot be used in active contracts until resolved by a person.

9. Consumption Interfaces: MCP Server, Ask Engine & Data Catalog

9.1 Model Context Protocol (MCP) Server (`mcp_server.py`)

Provides stdio-based MCP connectivity for external agents (e.g. Claude Code, Cursor):

MCP Tool	Token Footprint	Function
<code>semantic_brief</code>	~ 1.6k tokens	High-level summary of database domain, table representations, valid joins, and unanswerable questions.
<code>describe_table</code>	~ 1.7k tokens	Full table inspection: attribute meanings, trust levels, relationships, and pre-computed findings.
<code>what_does_it_mean</code>	~ 200 tokens	Specific attribute definition, scale, unit, trust, and metric capability.
<code>find_data_for</code>	Dynamic	Maps a natural-language business objective onto verified tables and columns.
<code>list_findings</code>	Dynamic	Returns pre-computed analytical findings to prevent redundant query computation.
<code>query</code>	Dynamic	Read-only gated SQL query tool (strictly disabled unless <code>SOTF_MCP_QUERY=1</code>).

9.2 The Natural Language Ask Engine (`ask.py` , `sqlguard.py`)

Answers ad-hoc business questions from published semantics:

1. **Deterministic Concept Resolution**: Maps question tokens against the published knowledge base and metric definitions.
2. **Epistemic Guard**: Refuses immediately if required concepts are `UNKNOWN` or `INVALIDATED` .
3. **Investigation Agent**: Generates SQL restricted exclusively to the resolved allow-list.
4. **SQLGuard AST Validation**: Enforces single-statement `SELECT` , blocks system tables (`pg_*` , `information_schema`), applies row limits and execution timeouts.
5. **Auditable Trace**: Returns the answer alongside execution SQL, cited evidence, and confidence.

9.3 Data Catalog & Semantic Register (`catalog.py` , `dictionary.py`)

- **Data Catalog** (`/api/catalog`): Tracks every estate database through six progression stages: `unavailable` → `discovered` → `profiled` → `interpreted` → `reviewed` → `published` .
- **Exportable Records** (`catalog/db/v0001.json`): Emits versioned catalog records using stable identifiers (`sid`) formatted as `database:table:column` .
- **Semantic Register** (`/register`): Self-contained HTML data dictionary detailing every attribute, its mathematical proof, debate history, and rule usage.

10. Frontend Architecture & User Journey

The web frontend (`ui.html`) is built in vanilla JavaScript with custom design tokens, zero external framework dependencies, and zero build step.

10.1 The 4-Step Cycle Navigation

CEEZ CONSOLE NAVIGATION	
PINNED	
• Catalog	: Multi-source overview and readiness
• This Source	: Source overview, coverage & health
THE CYCLE	
Step 1: Catalogue it	
• Run it	: Automated extraction & publishing
• The record	: Exportable published catalog record
• Structure	: Raw ontology, L1/L2/L3 nodes
• Deliberation	: Team A debate transcript & watcher
Step 2: Refine	
• What it represents	: Business understanding & curation
• Entities	: Entity list & drill-down views
• Attributes	: Published attribute meanings
• Relationships	: Measured joins, fan-out & orphans
• Metrics	: Metric lexicon & signal bindings
• History	: Temporal validity timeline
Step 3: Ask	
• Ask a question	: Governed semantic question engine
• Answers	: Pre-computed analytical findings
Step 4: Automate	
• Intents	: Registered intents & KPIs
• Discover intents	: Catalogue-supported intents
• Build an intent	: Plain-language intent builder
• Contracts	: Versioned Executable Contracts
• Releases	: 12-Stage CI/CD pipeline & gates
• Checks	: Semantic test assertions
• Policy compl.	: Document policy compliance checks
• Changes	: Team B notice queue & remapping
• Execution	: Live runtime decisions & 8 checks
• Outcomes	: Outcome tracking & root cause walk
ELSEWHERE (Drawer)	
• Business Rules, Policies, Documents, Federation, Contradictions, Connect Source, Agents & Endpoint	

10.2 Command Bar (⌘K)

Features a unified search index over 250+ objects spanning sources, entities, attributes, metrics, rules, contracts, and views, falling back directly to the semantic Ask engine.

11. Complete REST API Reference

All endpoints are served by `server.py` on default port `8899`.

11.1 Release Pipeline Endpoints

- `GET /api/releases?db=<name>` : Returns pipeline stages, active releases, and canary status.
- `GET /api/release?id=<id>` : Full details of a specific release candidate and gate results.
- `POST /api/release/gate` : Runs the 6 blocking release gates against live data.
 - Body: `{"id": "<release_id>", "by": "<user>"}`
- `POST /api/release/canary` : Deploys candidate to named contracts.
 - Body: `{"id": "<release_id>", "contracts": ["<id>"], "by": "<user>"}`
- `POST /api/release/observe` : Observes canary metrics and decision health.
- `POST /api/release/waive` : Waives a failing gate with mandatory reason.
 - Body: `{"id": "<release_id>", "gate": "<name>", "by": "<user>", "reason": "<str>"}`
- `POST /api/release/approve` : Human sign-off on a staged release.
- `POST /api/release/promote` : Promotes release candidate to the live graph.
- `POST /api/release/rollback` : Restores previous graph snapshot and resets contract statuses.
 - Body: `{"db": "<name>", "to": "<release_id>", "reason": "<str>", "by": "<user>"}`
- `POST /api/release/settings` : Updates auto-promotion and gate failure behavior.

11.2 Catalog & Extraction Endpoints

- `GET /api/catalog?db=<name>` : Returns estate catalog stages and readiness.
- `GET /api/catalog/record?db=<name>&version=<v>` : Draft vs published catalog record.
- `GET /api/catalog/published?db=<name>&version=<v>` : Machine-readable published record.
- `POST /api/catalog/publish` : Publishes current draft record.
- `POST /api/extract` : Kicks off background extraction and debate pipeline.
 - Body: `{"db": "<name>", "granted_tier": 2, "with_model": true, "max_tables": 50}`
- `GET /api/extract/status?db=<name>` : Polling status of background extraction.

11.3 Intent & Lifecycle Endpoints

- `GET /api/intents?db=<name>&tier=<0|1|2>` : Scored catalogue intents against the estate.
- `GET /api/intent/compose?db=<name>&q=<text>&tier=<0|1|2>` : Drafts intent from text.
- `GET /api/intent/test?db=<name>&id=<intent_id>&tier=<tier>` : Runs SQL probes for intent.
- `POST /api/intent/discover` : Ranks sources and checks signal columns for an objective.
- `POST /api/intent/build` : Compiles and tests an engineered intent join path.
- `POST /api/lifecycle/list` : Lists all registered intents.
- `POST /api/lifecycle/save` : Creates or updates an intent definition.
- `POST /api/lifecycle/validate` : Runs the 10-point semantic validation suite.
- `POST /api/lifecycle/assess` : Calculates work required, risks, and autonomy ceiling.
- `POST /api/lifecycle/implement` : Compiles validated intent into an Executable Semantic Contract.
- `POST /api/lifecycle/delete` : Removes an intent.

11.4 Contracts & Governance Endpoints

- `GET /api/contracts?db=<name>` : Returns all contracts, effective statuses, and staleness.
- `GET /api/contract?id=<id>` : Complete contract detail, authority grid, and version diff.
- `POST /api/contract/status` : Updates contract status (`active` , `suspended` , `draft`).

11.5 Runtime Decision & Outcome Endpoints

- `GET /api/decisions?db=<name>&outcome=<filter>` : Feed of runtime decisions.

- `POST /api/decide` : Evaluates proposed agent action against 8 pre-commit checks.
 - Body: `{"intent_id": "<id>", "action": "<name>", "params": {}, "by": "agent"}`
- `POST /api/decisions/approve` : Authorizes an escalated decision.
- `POST /api/decisions/revalidate` : Re-runs checks at transaction boundary.
- `GET /api/outcomes?intent=<id>` : Records of decision outcomes and KPI movements.
- `POST /api/outcomes/record` : Enters actual measured outcome for a decision.
- `POST /api/outcomes/investigate` : Traces poor outcome backward across 9 root causes.
- `GET /api/authority?intent=<id>&db=<name>` : Calculates continuous effective authority.

11.6 Team B & Impact Endpoints

- `GET /api/changes?db=<name>&state=<pending|decided>` : Returns notice queue.
- `POST /api/changes/analyse` : Forces sensing cycle and diffs published graph.
- `POST /api/changes/decide` : Accepts or rejects a proposed remap notice.
- `POST /api/changes/suspend-exposed` : Suspends contracts impacted by breaking notices.

11.7 Documents & Policy Endpoints

- `GET /api/documents` : Lists ingested policy documents and unreadable formats.
- `POST /api/documents/ingest` : Ingests and chunks `.docx`, `.md`, `.txt`, `.html`.
- `POST /api/documents/forget` : Purges document and removes associated policies.
- `GET /api/policies?state=<filter>` : Lists extracted policy rules with clause citations.
- `POST /api/policies/extract` : Runs Policy Investigator to extract rules from document.
- `POST /api/policies/decide` : Approves or rejects an extracted policy rule.
- `GET /api/policy-tests?db=<name>` : Runs 8 policy compliance checks against contracts.

11.8 Multi-Source Federation & Contradiction Endpoints

- `GET /api/federation?source=<db1>&source=<db2>` : Cross-source entity view.
- `POST /api/federation/set` : Saves named source set.
- `GET /api/contradictions?state=<filter>` : Returns open cross-source contradictions.
- `POST /api/contradictions/detect` : Scans federated estate for conflicting definitions.
- `POST /api/contradictions/decide` : Resolves or accepts cross-source contradictions.

11.9 Semantic Workforce & Agent Configuration

- `GET /api/agents` : Role configurations, kinds, schemas, and LLM endpoint status.
- `POST /api/llm/config` : Saves LLM endpoint, model, and authentication settings.
- `POST /api/llm/probe` : Tests connection and reports latency and token counts.
- `POST /api/roles/save` : Creates or modifies an agent role (refuses `Trusted`).
- `POST /api/roles/reset` : Restores default shipped agent system prompts.
- `GET /api/workforce?db=<name>` : Team A roster, debate history, rule book usage.
- `POST /api/workforce/debate` : Triggers live adversarial debate.
- `POST /api/workforce/cycle` : Executes single sense-and-maybe-debate cycle.
- `POST /api/workforce/watch` : Starts or stops background watcher thread.

11.10 Ask, Dictionary & Curation Endpoints

- `POST /api/ask` : Answers business question from published semantics with SQL trace.
 - Body: `{"db": "<name>", "question": "<text>"}`

- `GET /register?db=<name>` : Renders self-contained data dictionary HTML.
- `GET /reference.md?db=<name>&version=<v>&raw=<0|1>` : Markdown reference document.
- `GET /api/temporal?db=<name>&attribute=<table>.<col>&at=<timestamp>` : Temporal definitions.
- `GET /api/curation?db=<name>` : Human curation summary.
- `POST /api/curate` : Human accept, correct, or reject semantic claim (grants `Trusted`).
- `GET /api/rules?db=<name>` : Business rules predicates.
- `POST /api/rules/save` : Validates and saves a business rule.

12. Verification & Testing Framework

The platform includes a test suite in `test_all.py` comprising **over 570 automated checks**:

```
# Full test suite (deterministic engines, gates, contracts, runtime)
PGPASSWORD=... python3 test_all.py

# Complete suite including live LLM model debate
PGPASSWORD=... python3 test_all.py --llm

# Rapid verification skipping scratch database creation
PGPASSWORD=... python3 test_all.py --quick
```

Key Verification Highlights

- **165 Base Engine Assertions**: Tests extraction, profiling, debate rules R1–R7, SQL generation, and MCP protocol.
- **Contract & CI/CD Pipeline**: Verifies gate enforcement, canary isolation, automated risk scoring, and zero-loss snapshot rollbacks.
- **Governed Runtime & 8 Checks**: Asserts that unauthorized actions are blocked, escalated actions halt for human sign-off, and simulations match live queries.
- **Temporal Invariants**: Validates that historical definitions are preserved across snapshot intervals and pre-store epochs report `unverifiable` .
- **Scratch DB Isolation**: Destructive schema drift scenarios (e.g. dropping columns, 100x unit rescaling) run strictly within temporary scratch databases that are created and destroyed during the test run.