```
elemento: "programa-cognitivo@nodo-desde-codigo.alpha"
alias: "pc.nodo-cognitivo-desde-codigo"
version: "alpha.2"   # incrementa por cada ajuste del proceso

contexto:
  objetivo: "Dado un proyecto de código (o solo el contrato), producir un NODO COGNITIVO canónico que lo represente con evidencias y trazabilidad."
  entradas:
    - "repo?: árbol de archivos (opcional si se parte del contrato)"
    - "contrato_dump: schema JSONL requerido por la ingesta"
    - "reglas_invariantes: composición/adyacencia/orden/explicabilidad"
    - "catalogo_modular: {have↔need-bridge, code-dump-exporter, pre-procesar→estructura, constructor-de-estructuras, analisis-estructural, validador, publicador}"
  salidas:
    - "CognitiveNodePackage (documento canónico con {contexto, nodes, vistas, evidencias, provenance})"
    - "artefactos operativos: project-dump.jsonl, MANIFEST.json, grafo-dependencias.*, endpoints.*, metrics.log"
    - "cognitive-node.txt (descargable, formato estándar y determinista)"
  supuestos:
    - "El contrato (schema JSONL) es la pieza central."
    - "El pipeline debe ser reversible (adaptar lo que hay / generar lo que falta)."

orchestrator:
  name: "pc.nodo-desde-codigo.orchestrator"
  describe_how_nodes_interact:
    - "P0.planificar: modela HAVE/NEED como contratos y selecciona recorrido."
    - "P1.bridge: usa have↔need-bridge para decidir modo (adapt/generate) y componer el plan mínimo."
    - "P2.dump: si se requiere, ejecuta/instancia code-dump-exporter y emite JSONL + MANIFEST."
    - "P3.pre: aplica pre-procesar→estructura (normaliza, valida, orden determinista: path, chunkIndex, offsetBytes; dedupe; política de secretos)."
    - "P4.struct: ejecuta constructor-de-estructuras para producir el esqueleto canónico {nodes-por-archivo, edges, vistas-básicas}."
    - "P5.tesis: integra analisis-estructural para tesis, patrones y plantillas transferibles."
    - "P6.validar: corre validador (invariantes + cobertura + trazabilidad)."
    - "P7.publicar: empaqueta CognitiveNodePackage, genera cognitive-node.txt con plantilla estándar y registra provenance + métricas."
  policy:
    parcimonia: "min-pasos que satisfacen contrato"
    invariantes: ["composition","adjacency","ordering","explicabilidad","provenance"]
    reversibilidad: "todo paso tiene rollback"
    umbral_confianza: 0.7

modules:
  - id: "M.bridge"
    elemento: "have↔need-bridge@s1"
```

```
    rol: "Síntesis de brecha y plan"
    entradas: ["have{} (repo? exporter?)","need{} (schema JSONL)","catalogo_modular"]
    salidas: ["plan_conexion{steps[],modo}","confianza"]

  - id: "M.dump"
    elemento: "code-dump-exporter@1.0.0 (compatible)"
    rol: "Exportación JSONL"
    entradas: ["repo","opts_dump"]
    salidas: ["project-dump.jsonl","MANIFEST.json","metrics.dump"]

  - id: "M.pre"
    elemento: "pre-procesar→estructura@1.x"
    rol: "Normalización y chequeo de registros"
    entradas: ["project-dump.jsonl","schema_registro"]
    salidas: ["intake.ok.jsonl","report.pre"]

  - id: "M.struct"
    elemento: "constructor-de-estructuras@1.x"
    rol: "Esqueleto canónico"
    entradas: ["intake.ok.jsonl","reglas_invariantes"]
    salidas: ["skeleton.cn","graph.dependencies","views.basicas"]

  - id: "M.analysis"
    elemento: "analisis-estructural@1.x"
    rol: "Tesis + patrones + plantillas"
    entradas: ["skeleton.cn","graph.dependencies"]
    salidas: ["tesis.md","plantillas.json","views.avanzadas"]

  - id: "M.validate"
    elemento: "validador-invariantes@1.x"
    rol: "QA estructural"
    entradas: ["skeleton.cn","tesis.md","reglas_invariantes"]
    salidas: ["qa.report","score.confianza"]

  - id: "M.publish"
    elemento: "publicador-cn@1.x"
    rol: "Empaquetado y publicación"
    entradas: ["skeleton.cn","tesis.md","plantillas.json","qa.report","provenance"]
    salidas: ["CognitiveNodePackage","cognitive-node.txt","provenance.log","metrics.log"]

flows:
  A_HAVE_TO_NEED_adapt:
    - "P0.planificar"
    - "M.bridge (modo='adapt') → plan_conexion"
    - "M.dump (si hace falta exporter o ajustes según plan)"
    - "M.pre → intake.ok.jsonl"
    - "M.struct → skeleton.cn"
    - "M.analysis → tesis/plantillas"
```

```
    - "M.validate → qa.report (score≥umbral?)"
    - "M.publish → CognitiveNodePackage + cognitive-node.txt"
  B_NEED_TO_HAVE_generate:
    - "P0.planificar"
    - "M.bridge (modo='generate') → plan_conexion"
    - "M.dump.generate (scaffold de exporter desde contrato) → project-dump.jsonl (demo o
real)"
    - "M.pre → intake.ok.jsonl"
    - "M.struct → skeleton.cn"
    - "M.analysis → tesis/plantillas"
    - "M.validate → qa.report"
    - "M.publish → CognitiveNodePackage + cognitive-node.txt"

interfaces:
  provides:
["CognitiveNodePackage","cognitive-node.txt","artefactos_intermedios*","provenance","metri
cs"]
  requires: ["contrato_dump","reglas_invariantes","catalogo_modular","politica_confianza"]

metricas_clave:
  cobertura: "archivos_emitidos/legibles"
  fidelidad: "Σ(sizeBytes_emitidos)≈sizeFS"
  calidad_estructura: "violaciones_invariantes=0"
  explicabilidad: "100% afirmaciones con evidencia"
  ttx: "tiempo_extremo_a_extremo"
  reutilizacion_plantillas: "conteo plantillas aplicadas en otros repos"

errores_estandar:
  - { code: "E_SCHEMA", tip: "el JSONL no cumple el contrato_dump" }
  - { code: "E_INVAR",  tip: "violación de composición/adyacencia/orden" }
  - { code: "E_CONF",   tip: "score de confianza < umbral" }
  - { code: "E_IO",     tip: "fallo de lectura/escritura en etapas" }

provenance:
  registra: ["hashes de entradas","versiones de módulos","decisiones del bridge","evidencias
por archivo/línea"]
  rollback: "snapshot por etapa (dump, intake, skeleton)"

#
————————————————————————————————————————————————————————————————————————
————
# SECCIÓN INTEGRADA: Formato del TXT descargable (cognitive-node.txt)
txt_descargable:
  version: "1.0.0"
  generator: "refine-from-jsonl.cjs"
  estructura: |
    Cognitive Node — Generated from JSONL
    version: 1.0.0
```

```
schemaVersion: 1.0.0
generatedAt: <ISO-8601>
sourceFile: <jsonl_filename>
generator: refine-from-jsonl.cjs

# 0) IDENTIFICATION
id: <namespace>/<name>@<semver>
kind: <Application|Profile|Library|Service|Dataset>
title: <Human friendly title>
purpose: <One-line purpose>

# 1) EVIDENCE SUMMARY (from JSONL)
records: <int_total_lines>
unique_records: <int_after_dedup>
total_size_bytes: <sum_of_original_file_sizes>
time_range_utc: <first_mtime> → <last_mtime>
media_counts:
  - <mediaTypeA>: <count>
lang_counts:
  - <langA>: <count>
types_counts:
  - file: <count>
  - chunk: <count>
top_dirs:
  - <dir_path_1>  <count>
sanity:
  dupes: <int>
  secrets: <int>
  validation_errors: <int>
policy:
  evidence_only: true
  deterministic_order: [path, chunkIndex, offsetBytes]
  composition_adjacency_ordering: preserved
  provenance: [sha256, mtime]

# 2) EXPORT SURFACE (Boundary)
PROVIDES:
  - <capability_or_artifact>@<version>
REQUIRES:
  - <dependency>@<version_or_contract>
CONSTRAINTS:
  - evidence-only
  - determinism(same-input→same-output)
  - composition|adjacency|ordering preserved
  - provenance: sha256+mtime
TYPES:
  JSONLRecord: {...}
  EvidenceBundle: {...}
```

```
  ALIASES:
    dump.jsonl → EvidenceBundle
    contrato_dump.schema.json → JSONLRecord[] spec

  # 3) INTEGRATION PROFILE (Orchestrator-level)
  provides:
    - { name: "<name>", kind: "<artifact|metrics|service>", version: "<x.y.z>", types_used:
["<TypeA>"] }
  requires:
    - { name: "<name>", kind: "<artifact|service>", version_range: ">=1.0.0", types_expected:
["<TypeX>"] }
  adapters:
    edge_aliases: { "<external_name>": "<internal_name>" }
    type_aliases: { "<external_type>": "<internal_type>" }
    data_mappers:
      - { from: "<TypeX>", to: "<TypeY>", lossless: true }
  negotiation: discover→check-compat→alias-map→validate→adapt→merge→report
  merge_policy:
    upsert_only: true
    conflict: prefer-specificity
    id_collision: namespaced-rename
    provenance: record

  # 4) PROFILE MODEL (if kind == Profile)
  summary: <short narrative>
  must_haves:
    - <item>
  nice_to_haves:
    - <item>
  anti_patterns:
    - <item>
  weights:
    typescript: <0..1>
    react: <0..1>
    redux: <0..1>
    mui: <0..1>
    hooks: <0..1>
    nestjs: <0..1>
    mongoose: <0..1>
    iam_auth: <0..1>
    testing: <0..1>
    devops_config: <0..1>
  signals:
    - name: strict_equality_rate
      target: ">=0.98"
    - name: default_export_ratio
      target: ">=0.6"
    - name: hook_density
```

```
    target: "2–6/100 LOC"
  - name: nestjs_layering
    target: "present"
  - name: env_hygiene
    target: "clean"

# 5) FUNCTIONAL MAP (optional)
areas:
  - name: <area_name>
    paths:
      - <prefix_path>
    notes: <responsibility>

# 6) QUALITY SIGNALS (computed)
frontend:
  react: <true|false>
  redux: <true|false>
  mui: <true|false>
  url_state: <true|false>
typescript:
  union_literals: <true|false>
  satisfies: <true|false>
backend:
  nestjs: <true|false>
  guards: <true|false>
  interceptors: <true|false>
  filters: <true|false>
  dto: <true|false>
  mongoose: <true|false>
testing:
  present: <true|false>
security_env:
  env_clean: <true|false>

# 7) PROVENANCE
files_sample:
  - path: <path/to/file.ts>
    size: <bytes>
    sha256: <hex>
    mtime: <ISO-8601>
hashes_summary:
  total_unique_hashes: <int>
  repeated_hashes: <int>

# 8) REFINEMENT REPORT
input_files:
  - <jsonl_filename>
signals_detected:
```

```
    - <signal>: <true|false>
  weight_updates:
    - <weight_name>: <new_value>
  notes:
    - <diagnostic>

  # 9) NEXT STEPS
  - <actionable_1>
  - <actionable_2>

  # End of Cognitive Node TXT

notes:
  - "El TXT es parte del artefacto de publicación (M.publish) y debe generarse siempre que haya evidence válida."
  - "El orquestador asegura orden determinista y política de secretos antes de computar señales."
  - "Cada archivo del repo se modela como nodo atómico; las vistas (feature/layer/hotspots) no destruyen la granularidad archivo=nodo."
```