API-GATEWAY — Constructor de Estructuras

Documento maestro que organiza TODO el conocimiento recopilado del proyecto API-GATEWAY (NestJS + GraphQL) utilizando el molde cognition-module()constructor-de-estructuras.pdf. La salida se estructura en {contexto, nodes, metaforas} para integrarse en el grafo cognitivo. Este documento pretende no perder NINGUNA pieza relevante: arquitectura, flujos de ejecución, módulos, contratos de entrada/salida, seguridad, rendimiento, configuración, build/run, pruebas, operación y recomendaciones de preprocesado.

```
principio: Todo elemento nace estructurado en {contexto, nodes, metaforas}.
operador: E (Estructuración determinista)
formalizacion: CognitiveNode := E(Artifacts|StructuredText) ⇒ {contexto, nodes, metaforas}
```

Alcance

• Arquitectura completa del gateway

• Descripción detallada de módulos y sus contratos

• Interacciones HTTP/WS y PubSub

• Guías de seguridad, rendimiento, testing y operación

• Recomendaciones de preprocesado y extensión

Nodes (componentes y procesos)

ID: runtime.bootstrap

Rol: Arranque y composición raíz

Entrada: src/main.ts, src/app.module.ts

Salida: NestFactory inicia AppModule; GraphQL (Apollo), WS (graphql-ws), contexto unificado userInfo; aplica AuthenticationMiddleware; importa dominios (analytics, call-records, historical-mode, iam, live-mode, pub-sub).

detalles:

- HTTP: context(req.userInfo)

- WS: onConnect → IdpTokenService.verifyAsync(token) → extra.userInfo

ID: schema-first.pipeline

Rol: Workflow schema-first

Entrada: /*.graphql en cada dominio; GraphQLDefinitionsFactory

Salida: src/graphql.ts actualizado (tipos generados) consumidos por resolvers y services.

detalles:

- Convenciones: PascalCase tipos, camelCase campos

- Evitar drift contrato/código; usar watch

ID: module.iam

Rol: Autenticación y contexto

Entrada: accessToken (HTTP header / WS connectionParams)

Salida: userInfo en contexto (HTTP: req.userInfo, WS: extra.userInfo)

contratos:

- servicio: IdpTokenService.verifyAsync(token) → Promise

- env: IDP_PUBLIC_KEY, IDP_JWKS_URL

operacion:

- Fail-fast on invalid token

- Cache de JWKS opcional (TTL corto)

ID: module.pub-sub

Rol: Backbone de suscripciones

Entrada: Eventos de dominios (callRecordCreated/Updated, etc.)

Salida: AsyncIterator para resolvers (pubSub.asyncIterator('EVENT'))

detalles:

- Proveedor intercambiable: memoria (dev) / Redis|Rabbit (prod)

- Recomendación: backpressure/TTL para controlar fanout

ID: module.call-records

Rol: CRUD + eventos de Call Records

Entrada: ['CreateCallRecordInput, UpdateCallRecordInput', 'NotifyCallRecordCreatedInput, NotifyCallRecordUpdatedInput']

Salida: ['CallRecord, FreshCallRecord, TerminatedCallRecordsPage, Meta', 'Eventos a PubSub (created/updated)']

interacciones:

- Consumido por live-mode (stream) y analytics (insights)

operacion:

- Idempotencia en notify*

- Validar payloads extensos (transcription)

ID: module.live-mode

Rol: Streaming en tiempo real

Entrada: WS con accessToken, filtros opcionales

Salida: Subscriptions: callRecordCreated/Updated; freshCallRecords

detalles:

- Suscrito a eventos de PubSub

- Mantener keep-alives/timeouts; buffers para clientes lentos

ID: module.historical-mode

Rol: Consultas históricas paginadas

Entrada: QueryParameters (rango, filtros, paginación)

Salida: TerminatedCallRecordsPage + Meta

detalles:

- Límites de ventana temporal

- Preferir paginación por cursor

ID: module.analytics

Rol: Vistas analíticas (charts/carousel)

Entrada: range, granularity, pagination; filtros (agente/equipo/outcome)

Salida: Series tipadas y metadata para UI

operacion:

- Limitar tamaño y rango

- Cachear queries populares (mem/Redis)

estructura:

- charts/*/*.graphql (ai-operation-breakdown, call-frequency-outcome, call-volume, handling-overview, median-call-duration)

- carousel/agent-call-time/*.graphql

- common/dto/{granularity, pagination, range}.graphql

ID: module.common

Rol: Utilidades transversales

Entrada: type-utils/*, helpers

Salida: Funciones de soporte sin dependencias fuertes

reglas:

- Evitar ciclos de dependencia

- Exportar vía barrel para rutas limpias

ID: security.authz

Rol: Autorización

Entrada: context.userInfo

Salida: Permitir/denegar por rol/scope

detalles:

- Guards con @SetMetadata('roles', [...])

- Políticas por módulo o resolver

ID: observability

Rol: Logs y métricas

Entrada: Eventos de request/resolver/subscription

Salida: Logs estructurados (pino), métricas de rendimiento

detalles:

- request-id y correlación HTTP/WS
- Timers por resolver; contadores de subs activas

ID: performance.scalability

Rol: Rendimiento y escalado

Entrada: Cargas de consulta/subs; fanout

Salida: SLA estable; bajo N+1

detalles:

- Redis PubSub en multi-instancia
- Dataloader para nested fields
- Limits en historical windows

ID: configuration.env

Rol: Variables de entorno

Entrada: PORT, NODE_ENV, IDP_PUBLIC_KEY|JWKS, PUBSUB_BACKEND, LOG_LEVEL

Salida: Configuración validada (Joi recomendada)

defaults:

- PORT: 4000
- PUBSUB_BACKEND: memory
- LOG_LEVEL: info

ID: build.run

Rol: Ciclo de build y arranque

Entrada: npm run build; npm run gen-types; npm run start:dev

Salida: Servidor listo con graphql.ts sincronizado

checks:

- HTTP 200 en query simple
- WS connect con accessToken y recepción de eventos

ID: testing.strategy

Rol: Estrategia de pruebas

Entrada: Unit (services), Resolver (mapping/guards), e2e (HTTP/WS)

Salida: Confianza de contrato y regresión

detalles:

- Snapshot de src/graphql.ts para detectar drift
- Mocks en servicios; test handshake WS

ID: ops.runbook

Rol: Operación y soporte

Entrada: Eventos runtime, fallos, picos de subs

Salida: Runbook de salud y mitigación

procedimientos:

- Healthz o resolver de salud
- Throttle de subs ante fanout
- Inspección de lag en PubSub
- Rotación de llaves IdP; cache JWKS

ID: preprocessing.recs

Rol: Recomendaciones de preprocesado

Entrada: Artifacts.ndjson / structured.txt

Salida: Paquete óptimo para constructor de estructuras

detalles:

- Enfocar en src/, excluir dist/ y *.tsbuildinfo
- rels[] con evidencia de líneas (imports/exports)
- Etiquetar roles (module.ts, *.resolver.ts, *.service.ts)

ID: orchestrator.constructor

Rol: Orquestador del constructor de estructuras

Entrada:

Salida:

describe_how_nodes_interact:

- 1) runtime.bootstrap fija el contexto unificado (HTTP/WS).
- 2) schema-first.pipeline asegura tipado/update de graphql.ts.
- 3) Los módulos de dominio (iam, pub-sub, call-records, live-mode, historical-mode, analytics, common) se integran bajo AppModule.
- 4) Capas transversales (security.authz, observability, performance.scalability) garantizan políticas y SLAs.
- 5) build.run, testing.strategy y ops.runbook cierran ciclo de vida.
- 6) preprocessing.recs prepara la materia para futuros análisis cognitivos.

acceptance:

- Ninguna afirmación sin evidencia (cuando aplique).
- Preserva composición/adyacencia/orden entre módulos.
- La forma {contexto, nodes, metaforas} está completa y no vacía.

Metáforas

1. Imprenta técnica: cada módulo es un capítulo con portada (contexto), índice (nodes) y glosario (metáforas).
2. Molde industrial: ideas líquidas entran y salen con la misma geometría cognitiva.
3. Puerto estandarizado: cualquier barco (feature) atraca con el mismo protocolo y queda trazado.
4. Andamio: sostiene la forma mientras se completa el contenido, preservando composición/adyacencia/orden.