KgSyncService

September 4, 2026

KgSyncService

Kind: Service

Source: `atloria-monorepo/apps/api/src/graph/services/kg-sync.service.ts`

Knowledge Graph Sync Service (Neo4j Implementation)

Handles bulk sync operations from CLI dynamic mode.
Provides efficient batch upsert for entities and relationships in Neo4j.

Neo4j is the ideal storage for knowledge graph data because:

- Optimized for relationship traversals
- Native graph queries with Cypher
- Better performance for complex dependency analysis
- Built-in graph algorithms

`KgSyncService` is a NestJS service that performs high-throughput, bulk synchronization of knowledge-graph data into Neo4j, optimized for CLI “dynamic mode” runs. It batches upserts for both entities (nodes) and relationships (edges) to efficiently keep the graph consistent and queryable for dependency analysis and traversal-heavy workloads.

Methods

Method	Signature	Returns	Description
`syncEntities`	`syncEntities(projectId: string, entities: SyncEntityDto[])`	`Promise<SyncEntitiesResponseDto>`	Bulk sync entities for a project
`syncRelationships`	`syncRelationships(projectId: string, relationships: SyncRelationshipDto[])`	`Promise<SyncRelationshipsResponseDto>`	Bulk sync relationships for a project
`getEntities`	`getEntities(projectId: string, type: string)`	`unknown`	Get all entities for a project
`getEntityById`	`getEntityById(projectId: string, entityId: string)`	`unknown`	Get a single entity by ID
`getRelationships`	`getRelationships(projectId: string)`	`unknown`	Get all relationships for a project
`getRelationshipsForEntity`	`getRelationshipsForEntity(projectId: string, entityId: string)`	`unknown`	Get relationships for a specific entity

Dependencies

- `Neo4jService`
- `PrismaService`

Where it refuses work

- `KgSyncService` stops the work with `BadRequestException` when `!project`.
- `KgSyncService` stops the work with `Error` when `entity.filePath.includes('..')`.
- `KgSyncService` stops the work with an early return when `results.length === 0`.
- `KgSyncService` stops the work with an early return when `results.length === 0 || !results[0].relationships`.

When something fails

- `KgSyncService` handles failure in 4 places: it logs it and continues in all 4.

Diagram

```
sequenceDiagram
  autonumber
  participant CLI as CLI Dynamic Mode
  participant API as NestJS API
  participant KG as KgSyncService
  participant N4J as Neo4j

  CLI->>API: Trigger bulk sync (entities + relationships)
  API->>KG: sync(payload)
  KG->>KG: Validate/normalize input\n(chunk into batches)
  loop For each batch
    KG->>N4J: Upsert nodes (MERGE)\n+ set properties
    KG->>N4J: Upsert relationships (MERGE)\n+ set properties
  end
  KG-->>API: Summary (counts/errors)
  API-->>CLI: Sync result
```

Usage

```
// Example NestJS command handler / controller usage
import { Injectable } from '@nestjs/common';
import { KgSyncService } from './graph/services/kg-sync.service';

type KgEntityUpsert = {
  id: string;
  type: string; // e.g. "Package" | "File" | "Symbol"
  props?: Record<string, unknown>;
};

type KgRelationshipUpsert = {
  fromId: string;
  toId: string;
  type: string; // e.g. "DEPENDS_ON" | "IMPORTS" | "REFERENCES"
  props?: Record<string, unknown>;
};

@Injectable()
export class KgSyncRunner {
  constructor(private readonly kgSync: KgSyncService) {}
```

```
  async run() {
    const entities: KgEntityUpsert[] = [
      { id: 'pkg:api', type: 'Package', props: { name: 'api' } },
      { id: 'file:src/main.ts', type: 'File', props: { path: 'src/main.ts' } },
    ];

    const relationships: KgRelationshipUpsert[] = [
      {
        fromId: 'file:src/main.ts',
        toId: 'pkg:api',
        type: 'BELONGS_TO',
        props: { inferred: true },
      },
    ];

    // The service typically performs batched upserts into Neo4j.
    // Adjust method name/shape to match the implementation in your repo.
    const result = await this.kgSync.sync({
      entities,
      relationships,
      mode: 'dynamic',
      batchSize: 1000,
    });

    return result; // e.g. { upsertedNodes, upsertedRels, errors }
  }
}
```

AI Coding Instructions

- Preserve the “batch upsert” pattern (chunk inputs, run Cypher `MERGE` with parameters); avoid per-entity queries, which will destroy performance in dynamic CLI runs.
- Keep node/relationship identity stable (consistent `id` /keys and relationship endpoints); changing identity fields can cause duplicates instead of updates.
- Ensure writes are parameterized Cypher (no string concatenation) and prefer `UNWIND`-based queries to insert/update many rows per transaction.
- Integrate cleanly with the Neo4j driver/session lifecycle (transactions, session close) and surface a compact sync summary (counts + failures) for CLI reporting.
- When adding new entity/relationship types, update the mapping/labeling logic and add minimal indexes/constraints in Neo4j to support the new keys.

Relationships

- DEPENDS_ON → `Neo4jService`
- DEPENDS_ON → `PrismaService`

Referenced By

- `GraphModule` (MODULE_PROVIDES)
- `GraphModule` (MODULE_EXPORTS)
- `KgSyncController` (DEPENDS_ON)
- `SyncController` (DEPENDS_ON)