## Guide - MITO Topology Import

### General Overview

This document provides a comprehensive guide for importing data into the **MITO Topology** system using Neo4j. The system manages network inventory through a graph model that represents configuration items (CiItems), their relationships, and logical organization.

**Key Features**

* **Database**: Neo4j (Graph Database)
* **Model**: Graph with nodes and relationships
* **Import**: Batch processing via CSV
* **Scalability**: Optimized for large data volumes
* **Integrity**: Constraints and indexes to ensure consistency

**Important Note on Sample Data**

The provided CSV files contain:

* **First row**: Header with required data model (DO NOT MODIFY)
* **Subsequent rows**: Sample data to be replaced with your real data

### System Architecture

**Main Entities**

| **Entity** | **Description** | **Primary Key** |
| --- | --- | --- |
| **Topology** | Logical groupings of network elements | name |
| **Layer** | Network abstraction levels | name |
| **ItemType** | Types of configurable elements | itemTypeName |
| **Account** | Ownership accounts | account\_name |
| **CiItem** | Configuration items | name |
| **Location** | Geographic positions | name |

**Main Relationships**

Topology -[HAS\_ELEMENT]-> CiItem

Layer -[IS\_LAYER\_OF]-> CiItem

Layer -[IS\_PARENT\_OF]-> Layer

Location -[IS\_LOCATION\_OF]-> CiItem

CiItem -[*business\_model\_relations*]-> CiItem

**Layer Hierarchical Model**

Service Circuit (Layer 3)

↓ IS\_PARENT\_OF

Ethernet (Layer 2)

↓ IS\_PARENT\_OF

Physical (Layer 1)

### Prerequisites

**Required Software**

* **Neo4j Desktop/Server** (version 4.4+)
* **APOC Plugin** installed and configured
* **Neo4j Spatial Plugin** for geographic coordinate management

**Neo4j Configuration**

# neo4j.conf

dbms.security.procedures.unrestricted=apoc.\*,spatial.\*

dbms.security.procedures.allowlist=apoc.\*,spatial.\*

**Directory Structure**

A screenshot of a computer

AI-generated content may be incorrect.

### Complete Summary Table

Refer to table.html

### CSV Data Models

#### PHASE 1 - Base Entities

* **layer.csv**

*name,mandatory*

**Required Fields**:

name: Unique layer name

**Optional Fields**:

mandatory: Y/N indicates if layer is mandatory

* **itemType.csv**

*itemTypeName,defaultLayer*

**Required Fields**:

* + itemTypeName: Unique element type name

**Optional Fields**:

* + defaultLayer: Default layer name**, can be a list separate by semicolon“;”**
* **account.csv**

*account\_name*

**Required Fields**:

* account\_name: Unique account name

#### PHASE 2 - Inventory

**locations.csv**

*location\_id,name,latitude,longitude*

**Required Fields**:

* location\_id: Unique numeric ID
* name: Unique location name

**Optional Fields**:

* latitude/longitude: Geographic coordinates (decimal format)

**ciItems.csv**

*item\_id,name,itemTypeName,status,accountName,importance,isLink,fromElement,toElement,latitude,longitude,itemIcon,style,url,propKey\_1,propVal\_1,propKey\_2,propVal\_2………………………*

**Required Fields**:

* item\_id: Unique numeric ID
* name: Unique element name
* itemTypeName: Must exist in itemType.csv
* accountName: Must exist in account.csv
* isLink: Y/N indicates if it's a link
* fromElement/toElement: IDs of connected elements (for links only)

**Optional fields Fields**:

* importance: Importance level
* url : callback url
* status: Element status

**Dynamic Properties**:

* propKey\_N / propVal\_N: Key-value pairs for additional properties
* N must be sequential (1,2,3,...)
* **topology.csv**

*name,description,createdAt,updatedAt,createdBy,modifiedBy*

**Required Fields**:

* + name: Unique topology name

**Optional Fields**:

* description: Text description
* createdAt/updatedAt: Timestamp in ISO 8601 format
* createdBy/modifiedBy: Creator/modifier user

#### PHASE 3 - Base Relations

* **layer\_rel.csv**

*layer\_name,ciitem\_name*

* **location\_rel.csv**

location\_name,ciitem\_name

#### PHASE 4 - Items Relations

* **ciitem\_rel.csv**

*source\_name,target\_name,relation*

* **topology\_rel.csv**

*topology\_name,ciitem\_name*

**Common Relation Types**:

* terminated\_at: Physical connection terminated on
* routed\_on: Routed through
* depends\_on: Depends on
* contains: Contains
* connects\_to: Connects to

### Execution Procedure -ONLY FOR MITO TEAM

#### Step 1: Environment Preparation

# 1. Verify Neo4j is active

systemctl status neo4j # Linux

# or check from Neo4j Desktop

# 2. Copy CSV to import directory

cp \*.csv /var/lib/neo4j/import/

# 3. Verify permissions

chmod 644 /var/lib/neo4j/import/\*.csv

#### Step 2: Sequential Execution

##### PHASE 0: Database Setup and Configuration

**Purpose**: Initialize database with indexes, constraints, and spatial capabilities

**Dependencies**: None (first execution)

**Scripts**:

* F0\_SETUP.txt - Creates all indexes and constraints
* F0\_B.spatial\_layer\_F0.txt - Initializes spatial layer for geographic data

##### PHASE 1: Import Base Entities

**Purpose**: Import fundamental system entities and establish layer hierarchy

**Dependencies**: PHASE 0

**Required CSV Files**:

* layer.csv - Network layer definitions
* itemType.csv - Equipment and service type catalog
* account.csv - Account/ownership definitions
* layer\_hierarchy.csv - Parent-child layer relationships

**Scripts**:

* 1.F1.import\_basic.txt - Import Layer, ItemType, Account entities
* 2.F1.import\_layer\_hierarchy\_rel.txt - Create layer hierarchy relationships

##### PHASE 2: Import Inventory

**Purpose**: Import physical and logical network inventory

**Dependencies**: PHASE 0 + PHASE 1

**Required CSV Files**:

* locations.csv - Geographic locations with coordinates
* ciItems.csv - Network elements with properties and relationships
* topology.csv - Topology definitions and metadata

**Scripts**:

* 1.F2.import\_location.txt - Import locations with spatial indexing
* 2.F2.import\_items.txt - Import network elements with dynamic properties
* 3.F2.import\_topology.txt - Import topology definitions

##### PHASE 3: Import Base Relations

**Purpose**: Establish default layer assignments and location mappings

**Dependencies**: PHASE 0 + PHASE 1 + PHASE 2

**Required CSV Files**:

* layer\_rel.csv - Manual layer assignments (optional, supplements defaults)
* location\_rel.csv - Element-to-location mappings

**Scripts**:

* 1.F3.import\_defaultLayer\_rel.txt - Auto-assign elements to default layers
* 2.F3.import\_location\_rel.txt - Create location-element relationships

##### PHASE 4: Import Items Relations

**Purpose**: Create network interconnections and topology assignments

**Dependencies**: PHASE 0 + PHASE 1 + PHASE 2

**Required CSV Files**:

* ciitem\_rel.csv - Element-to-element business relationships
* topology\_rel.csv - Topology-to-element assignments

**Scripts**:

* 1.F4.import\_items\_rel.txt - Create network interconnections
* 2.F4.import\_topology\_rel.txt - Assign elements to topologies

#### Step 3: Import Verification

// Node count by type

MATCH (n)

RETURN labels(n)[0] as NodeType, count(n) as Count

ORDER BY NodeType;

// Relationship count by type

MATCH ()-[r]->()

RETURN type(r) as RelationType, count(r) as Count

ORDER BY RelationType;

// Verify referential integrity

MATCH (ci:CiItem)

WHERE NOT EXISTS((ci)<-[:IS\_LAYER\_OF]-(:Layer))

RETURN count(ci) as ItemsWithoutLayer;

// Verify elements with coordinates

MATCH (ci:CiItem)

WHERE ci.latitude IS NOT NULL AND ci.longitude IS NOT NULL

RETURN count(ci) as ItemsWithCoordinates;

#### Cleanup Scripts

**DELETE\_ALL.txt**

//DELETE ENTIRE GRAPH (node + relation)

//MATCH (n) DETACH DELETE n

//OR

//DELETE RELATIONS ONLY

//MATCH ()-[r]->() DELETE r;

**DELETE\_INDEXES.txt**

// WARNING: Use only if you need to remove indexes

/\*

DROP CONSTRAINT topology\_id\_unique;

DROP CONSTRAINT layer\_id\_unique;

DROP CONSTRAINT itemtype\_id\_unique;

DROP CONSTRAINT account\_id\_unique;

DROP INDEX topology\_name\_index;

DROP INDEX layer\_name\_index;

DROP INDEX itemtype\_name\_index;

DROP INDEX account\_name\_index;

DROP INDEX itemtype\_defaultlayer\_index;

DROP INDEX topology\_createdat\_index;

DROP INDEX topology\_updatedat\_index;

\*/

#### Diagnostic Queries

// 1. General graph statistics

CALL db.stats.retrieve('GRAPH') YIELD data

RETURN data;

// 2. Nodes by label

MATCH (n)

RETURN labels(n)[0] as Label, count(n) as Count

ORDER BY Count DESC;

// 3. Relations by type

MATCH ()-[r]->()

RETURN type(r) as Relation, count(r) as Count

ORDER BY Count DESC;

// 4. Items without location

MATCH (ci:CiItem)

WHERE NOT EXISTS((ci)<-[:IS\_LOCATION\_OF]-(:Location))

RETURN count(ci) as ItemsWithoutLocation;

// 5. Items without layer

MATCH (ci:CiItem)

WHERE NOT EXISTS((ci)<-[:IS\_LAYER\_OF]-(:Layer))

RETURN count(ci) as ItemsWithoutLayer;

// 6. Verify layer hierarchy

MATCH path = (parent:Layer)-[:IS\_PARENT\_OF\*]->(child:Layer)

RETURN parent.name, child.name, length(path) as depth

ORDER BY depth, parent.name;

// 7. Items with dynamic properties

MATCH (ci:CiItem)

WHERE keys(ci) <> ['item\_id', 'name', 'itemTypeName', 'status', 'accountId', 'importance', 'isLink', 'fromElement', 'toElement', 'latitude', 'longitude', 'itemIcon', 'style', 'url']

RETURN ci.name, [k IN keys(ci) WHERE NOT k IN ['item\_id', 'name', 'itemTypeName', 'status', 'accountId', 'importance', 'isLink', 'fromElement', 'toElement', 'latitude', 'longitude', 'itemIcon', 'style', 'url'] | k] as DynamicProperties

LIMIT 10;

#### Appendices

##### Appendix A: Configuration Parameters

**APOC Batch Processing Parameters**

{

batchSize: 10000, // Number of records per batch

parallel: false, // Parallel processing (recommended false)

concurrency: 5, // Number of concurrent threads

retries: 10 // Number of retries on error

}

**Volume Optimization**:

* **Small datasets** (<10K records): batchSize: 1000, concurrency: 1
* **Medium datasets** (10K-100K records): batchSize: 5000, concurrency: 3
* **Large datasets** (>100K records): batchSize: 10000, concurrency: 5

##### Appendix B: Date and Coordinate Formats

**Date Format**

ISO 8601: 2024-01-15T10:00:00

Timezone: 2024-01-15T10:00:00+01:00

**Coordinate Format**

Latitude: 45.4642 (decimal format)

Longitude: 9.1900 (decimal format)

Point: POINT(9.1900 45.4642) (WKT format)

##### Appendix C: Business Query Examples

**Network Queries**

// 1. Find all elements in a location

MATCH (loc:Location {name: 'Milano\_Centro'})-[:IS\_LOCATION\_OF]->(ci:CiItem)

RETURN loc.name, collect(ci.name) as Items;

// 2. Find path between two elements

MATCH path = shortestPath((source:CiItem {name: 'GE\_MIL\_ROM\_001'})-[\*]-(target:CiItem {name: 'SWRT\_Roma\_01'}))

RETURN path;

// 3. Elements by layer with hierarchy

MATCH (parent:Layer)-[:IS\_PARENT\_OF\*0..]->(layer:Layer)-[:IS\_LAYER\_OF]->(ci:CiItem)

WHERE parent.name = 'Service Circuit'

RETURN layer.name, collect(ci.name) as Items;

// 4. Connectivity analysis

MATCH (ci:CiItem {isLink: 'Y'})

MATCH (source:CiItem {item\_id: ci.fromElement})

MATCH (target:CiItem {item\_id: ci.toElement})

RETURN ci.name as Link, source.name as Source, target.name as Target;

// 5. Critical elements by topology

MATCH (t:Topology {name: 'Core\_Network'})-[:HAS\_ELEMENT]->(ci:CiItem)

WHERE ci.importance = 'Critical'

RETURN t.name, collect(ci.name) as CriticalItems;

##### Appendix D: Pre-Import Checklist

**Technical Checklist**

* [ ] Neo4j Server/Desktop installed and running
* [ ] APOC plugin installed and configured
* [ ] Neo4j Spatial plugin installed
* [ ] /import directory accessible with correct permissions
* [ ] Existing database backup (if applicable)
* [ ] Sufficient disk space (estimate 2-3x CSV size)
* [ ] Neo4j heap memory properly configured

**Data Checklist**

* [ ] All CSV files present in import directory
* [ ] CSV headers verified and compliant with model
* [ ] Sample data replaced with real data
* [ ] UTF-8 encoding verified for all CSV
* [ ] Unique names verified within each entity
* [ ] Cross-CSV references verified (e.g., itemTypeName in ciItems.csv must exist in itemType.csv)
* [ ] Geographic coordinates in decimal format
* [ ] Dates in ISO 8601 format

**Dependencies Checklist**

* [ ] PHASE 0: Setup scripts ready
* [ ] PHASE 1: Base entity CSV prepared ( layer, itemType, account,layer hierarchy)
* [ ] PHASE 2: Inventory CSV prepared (locations, ciItems, topology)
* [ ] PHASE 3: Base relations CSV prepared (, \*\_rel.csv)
* [ ] PHASE 4: Items and Topology relations CSV prepared (ciitem\_rel.csv, topology\_rel.csv)

##### Appendix E: Performance Tuning Parameters

**Neo4j Configuration for Import**

# neo4j.conf - Import optimizations

dbms.memory.heap.initial\_size=2G

dbms.memory.heap.max\_size=4G

dbms.memory.pagecache.size=1G

# Disable logging during import (optional)

dbms.logs.query.enabled=false

dbms.track\_query\_cpu\_time=false

**Pre-Import Script**

// Temporarily disable constraints (only if necessary)

// CALL apoc.schema.assert({},{});

// Optimize for import

CALL dbms.checkpoint();

**Post-Import Script**

// Rebuild statistics

CALL db.resampleIndex('your\_index\_name');

// Garbage collection

CALL dbms.checkpoint();

**Appendix F: Monitoring and Logging**

**Monitoring Queries During Import**

// Verify active processes

CALL dbms.listTransactions()

YIELD transactionId, username, currentQuery, elapsedTimeMillis

WHERE currentQuery CONTAINS 'LOAD CSV'

RETURN transactionId, username, currentQuery, elapsedTimeMillis;

// Verify memory

CALL dbms.queryJmx('java.lang:type=Memory')

YIELD attributes

RETURN attributes.HeapMemoryUsage, attributes.NonHeapMemoryUsage;

// Verify I/O

CALL dbms.queryJmx('java.lang:type=OperatingSystem')

YIELD attributes

RETURN attributes.ProcessCpuLoad, attributes.SystemCpuLoad;

**Log Files to Monitor**

# Main Neo4j log

tail -f /var/log/neo4j/neo4j.log

# Debug log (if enabled)

tail -f /var/log/neo4j/debug.log

# Query log (if enabled)

tail -f /var/log/neo4j/query.log

##### Appendix G: Rollback Scenarios

**Complete Rollback**

// 1. Preventive backup (before import)

CALL apoc.export.cypher.all("backup\_pre\_import.cypher", {});

// 2. In case of error - selective deletion

// Delete only data imported in the last session

MATCH (n)

WHERE n.createdAt >= datetime('2024-01-15T00:00:00')

DETACH DELETE n;

// 3. Restore from backup

CALL apoc.cypher.runFile("backup\_pre\_import.cypher");

**Partial Rollback by Phase**

// Rollback PHASE 4 - delete only items relations

MATCH (ci1:CiItem)-[r]->(ci2:CiItem)

WHERE type(r) IN ['terminated\_at', 'routed\_on', 'depends\_on', 'contains', 'connects\_to']

DELETE r;

// Rollback PHASE 3 - delete base relations

MATCH ()-[r:HAS\_ELEMENT|IS\_LAYER\_OF|IS\_LOCATION\_OF|IS\_PARENT\_OF]->()

DELETE r;

// Rollback PHASE 2 - delete inventory

MATCH (n:CiItem) DETACH DELETE n;

MATCH (n:Location) DETACH DELETE n;

// Rollback PHASE 1 - delete base entities

MATCH (n:Topology|Layer|ItemType|Account) DETACH DELETE n;

##### Appendix J: Maintenance Scripts

**Periodic Cleanup**

// Remove orphaned nodes (without relationships)

MATCH (n)

WHERE NOT (n)--()

DELETE n;

// Update statistics

CALL db.resampleOutdatedIndexes();

// Compact storage

CALL dbms.checkpoint();

**Batch Update**

// Template for mass updates

CALL apoc.periodic.iterate(

"MATCH (ci:CiItem) WHERE ci.status = 'Active' RETURN ci",

"SET ci.last\_checked = datetime(), ci.health\_status = 'OK'",

{batchSize: 1000}

);

**Incremental Backup**

// Export only recent changes

MATCH (n)

WHERE n.updatedAt >= datetime('2024-01-15T00:00:00')

WITH collect(n) as recentNodes

CALL apoc.export.cypher.data(recentNodes, [], "incremental\_backup.cypher", null)

YIELD file, source, format, nodes, relationships, properties, time, rows, batchSize, batches, done, data

RETURN file, nodes, relationships, time;

##### Appendix K: Data Validation Scripts

**Pre-Import Validation**

// Check CSV data integrity before import

LOAD CSV WITH HEADERS FROM 'file:///ciItems.csv' AS row

WITH row

WHERE row.itemTypeName IS NULL OR row.itemTypeName = ''

RETURN count(\*) as InvalidItemTypeNames;

// Check for duplicate names

LOAD CSV WITH HEADERS FROM 'file:///ciItems.csv' AS row

WITH row.name as itemName, count(\*) as cnt

WHERE cnt > 1

RETURN itemName, cnt as DuplicateCount;

// Validate coordinate ranges

LOAD CSV WITH HEADERS FROM 'file:///locations.csv' AS row

WHERE toFloat(row.latitude) < -90 OR toFloat(row.latitude) > 90

OR toFloat(row.longitude) < -180 OR toFloat(row.longitude) > 180

RETURN row.name as InvalidLocation, row.latitude, row.longitude;

**Post-Import Validation**

// Validate all items have required relations

MATCH (ci:CiItem)

OPTIONAL MATCH (ci)<-[:IS\_LAYER\_OF]-(l:Layer)

OPTIONAL MATCH (ci)<-[:HAS\_ELEMENT]-(t:Topology)

WHERE l IS NULL OR t IS NULL

RETURN ci.name,

CASE WHEN l IS NULL THEN 'Missing Layer' ELSE 'OK' END as LayerStatus,

CASE WHEN t IS NULL THEN 'Missing Topology' ELSE 'OK' END as TopologyStatus;

// Check link integrity

MATCH (ci:CiItem {isLink: 'Y'})

WHERE ci.fromElement IS NOT NULL AND ci.toElement IS NOT NULL

OPTIONAL MATCH (source:CiItem {item\_id: ci.fromElement})

OPTIONAL MATCH (target:CiItem {item\_id: ci.toElement})

WHERE source IS NULL OR target IS NULL

RETURN ci.name as BrokenLink,

CASE WHEN source IS NULL THEN 'Missing Source' ELSE 'OK' END as SourceStatus,

CASE WHEN target IS NULL THEN 'Missing Target' ELSE 'OK' END as TargetStatus;

##### Appendix L: Performance Benchmarking

**Import Performance Metrics**

// Time import phases

CALL apoc.util.sleep(0) // Start timer

WITH datetime() as startTime

// Your import script here

CALL apoc.periodic.iterate(

"LOAD CSV WITH HEADERS FROM 'file:///your\_file.csv' AS row RETURN row",

"CREATE (n:YourNode) SET n = row",

{batchSize: 10000}

)

WITH startTime, datetime() as endTime

RETURN duration.between(startTime, endTime) as ImportDuration;

**Query Performance Testing**

// Profile complex queries

PROFILE

MATCH (t:Topology)-[:HAS\_ELEMENT]->(ci:CiItem)-[:terminated\_at]->(target:CiItem)

WHERE t.name = 'Core\_Network' AND ci.importance = 'Critical'

RETURN t.name, count(target) as ConnectedTargets;

// Explain query execution plan

EXPLAIN

MATCH path = shortestPath((source:CiItem)-[\*..5]-(target:CiItem))

WHERE source.name = 'SWRT\_Milano\_01' AND target.name = 'SWRT\_Roma\_01'

RETURN path;

#### Conclusions

This comprehensive guide provides all the necessary tools to successfully import data into the MITO Topology system. By following the procedures in the specified order and using the proposed verification queries, the team will be able to:

1. **Configure** the Neo4j environment correctly
2. **Import** all data respecting dependencies
3. **Verify** import integrity
4. **Handle** errors and common issues
5. **Optimize** performance for large volumes
6. **Maintain** the system over time

**Key Points to Remember**

* **Sequentiality**: Phases must be executed in the specified order
* **Integrity**: Always verify references between CSV before import
* **Performance**: Adapt batch parameters to data size
* **Backup**: Always backup before important imports
* **Monitoring**: Keep logs and performance under control during import

**Support and Resources**

* **Neo4j Logs**: /var/log/neo4j/neo4j.log
* **APOC Documentation**: <https://neo4j.com/labs/apoc/>
* **Neo4j Community**: <https://community.neo4j.com/>

The MITO Topology system, once properly imported, will provide a powerful platform for network inventory analysis and management through a scalable and performant graph model.