



UNIVERSITÀ DEGLI STUDI DI SALERNO



 MongoDB. vs  neo4j

## Relazione Progettuale

Photovoltaic Panels Recognition and Geospatial Analysis using  
MongoDB and Neo4j

**Antony Storti**

a.storti2@studenti.unisa.it

Matricola: 0622702353

DIEM

**Paola Saggiomo**

p.saggiomo1@studenti.unisa.it

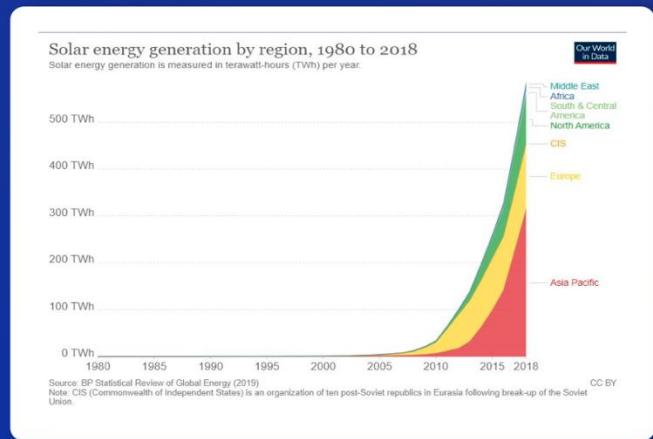
Matricola: 0622702380

DIEM

10 Dicembre 2024

## Motivation

- Solar energy growth
- Resource optimization for energy companies



## Goal

- Detect the presence of **solar panels**
- Approximate the **surface of the cells**

# Dominio di Interesse: «Photovoltaic Panels Recognition and Geospatial Analysis»



+

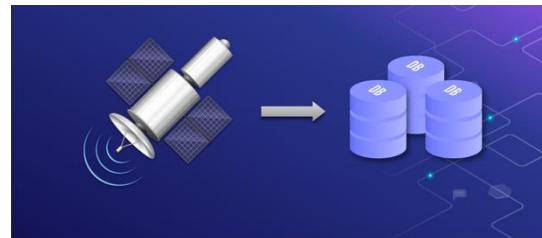


Infatti, i DBMS (Data Management Systems), nel nostro caso, MongoDB e Neo4j, sono strumenti essenziali per gestire ed analizzare i dati raccolti. Senza l'uso di un sistema di gestione dei dati adeguato, non sarebbe possibile organizzare, archiviare né tantomeno sfruttare in modo efficace le informazioni spaziali derivanti dai risultati del modello.

I DBMS permettono non solo di immagazzinare i dati, ma anche di effettuare analisi avanzate come: la ricerca di aree densamente popolate da pannelli solari, il calcolo di distanze tra pannelli e l'integrazione con altre informazioni geospaziali; ad esempio, per l'analisi del potenziale solare di una zona nell'ottica di ottimizzare le nuove installazioni. Nell'era dell'exploit dell'intelligenza artificiale, è doveroso ammettere che senza i progressi avuti nel campo della gestione dei dati nulla sarebbe stato possibile!

L'intelligenza artificiale si nutre di dati per addestrare modelli sempre più sofisticati, e senza l'evoluzione dei DBMS, che oggi supportano enormi volumi di dati distribuiti, query ad alte prestazioni e funzionalità integrate per il machine learning, sarebbe stato impossibile raggiungere gli attuali livelli di innovazione.

I moderni database, sia relazionali che NoSQL, hanno reso i dati più accessibili, scalabili e analizzabili, consentendo all'IA di sfruttare appieno il proprio potenziale. In questo contesto, la gestione efficace dei dati è diventata il pilastro su cui si fondano tutte le applicazioni avanzate di AI, dal riconoscimento di immagini e linguaggio, alla guida autonoma, all'analisi predittiva.



# Dominio di Interesse: «Photovoltaic Panels Recognition and Geospatial Analysis»

## 4.1 Dominio di Interesse

Essendo nell'ambito di un corso incentrato sui sistemi NoSQL, vi era la necessità di disporre di un grande quantitativo di dati (possibilmente non strutturati); motivo per cui abbiamo deciso di espandere quelli relativi ai pannelli con altri dati: semanticamente affini ed a quest'ultimi relazionabili. [ Così abbiamo circa **10 milioni** di documenti! ]

Nello specifico abbiamo deciso di raccogliere, per ognuna delle sei città, le seguenti informazioni:

- Informazioni anagrafiche e reddituali degli abitanti <sup>[9]</sup>.
- Valori degli inquinanti atmosferici (per Zona OMI)
- Consumi elettrici (per Zona OMI)

Purtroppo avere dati reddituali vecchi di vent'anni rende il tutto vano, ma, per questioni di privacy, non vi è modo di recuperare dati recenti!

### 4.1.1 Perchè questa scelta?

Questa scelta non è casuale, ma risponde a precise motivazioni legate agli obiettivi del nostro progetto e alle esigenze di comprensione dei fenomeni correlati alla diffusione e all'uso degli impianti fotovoltaici. Il nostro obiettivo non è solo monitorare la distribuzione degli impianti fotovoltaici, ma anche comprendere il loro impatto socio-economico ed ambientale. Raccogliere dati eterogenei e correlati permette di individuare le aree più bisognose di interventi.

- Le *informazioni anagrafiche e reddituali* degli abitanti sono fondamentali per comprendere il contesto socio-economico delle diverse zone urbane. Ad esempio, il reddito medio può influire sulla probabilità che una famiglia o un'impresa scelga di installare un impianto fotovoltaico. Le aree con redditi più elevati potrebbero avere una maggiore capacità di investimento iniziale, mentre le aree a reddito medio-basso potrebbero richiedere incentivi o finanziamenti per adottare tecnologie green. Questi dati ci permettono di formulare analisi di fattibilità e proposte mirate per favorire l'adozione delle energie rinnovabili.
- I *valori degli inquinanti atmosferici*, raccolti per Zona OMI (una suddivisione standardizzata delle città in aree omogenee), sono cruciali per valutare l'impatto ambientale degli impianti fotovoltaici. Una riduzione delle emissioni locali di CO<sub>2</sub> e altri inquinanti può essere correlata alla diffusione delle energie rinnovabili. Inoltre, comprendere quali aree soffrono di maggiore inquinamento può aiutare a individuare le zone dove l'installazione di impianti fotovoltaici potrebbe avere il maggior impatto positivo sulla qualità dell'aria e, di conseguenza, sulla salute pubblica.
- I *dati sui consumi elettrici*, anch'essi organizzati per Zona OMI, sono essenziali per calcolare il fabbisogno energetico delle diverse aree urbane. Questi dati ci permettono di stimare la capacità di produzione necessaria degli impianti fotovoltaici per soddisfare le esigenze locali. Confrontare i consumi con la distribuzione degli impianti installati ci consente inoltre di valutare il grado di autosufficienza energetica di ciascuna zona e di proporre soluzioni di miglioramento mirate.

# Project Design

Feature engineering  
vs  
Neural Networks



- Raw Satellite Data
- Google Maps Images
- Solar Panel Dataset

- Model comparison
- Result analysis



# Dataset di Addestramento: «ZENODO»

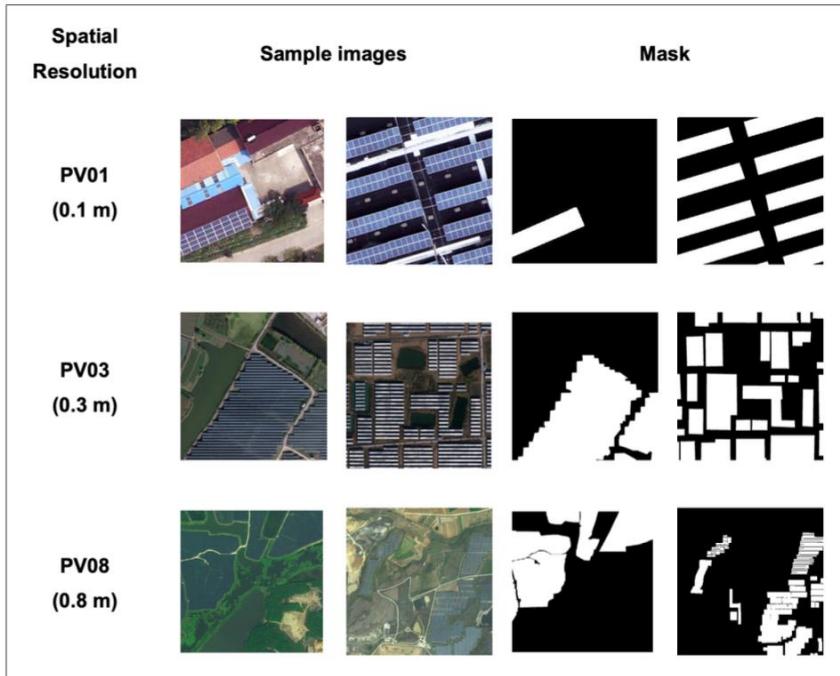


Figura 1: Contenuto del Dataset

Spatial Resolution	Ground	Rooftop	TOTAL
PV01 (0.1m)	0	645	645
PV03 (0.3m)	2122	186	2308
PV08 (0.8m)	673	90	763
3716			

Figura 2: Suddivisione per tipologia

## 2.1 Le immagini satellitari

L'idea iniziale che abbiamo avuto per ottenere le immagini è stata quella di usare l'API di Copernicus, che offre accesso libero e gratuito ai dati dei satelliti Sentinel.

Nelle fasi iniziali del progetto, è stato utilizzato il software *SNAP Toolbox* <sup>[1]</sup> per elaborare i dati provenienti dal satellite Sentinel-2. I dati risultanti includevano più immagini a diverse risoluzioni spaziali con le relative informazioni dello spettro elettromagnetico.

Il problema principale riscontrato nell'uso di questa fonte di dati è stata la mancanza di campioni di riferimento di pannelli solari nei dati geospaziali. Per addestrare un modello al riconoscimento degli oggetti, è necessario disporre di numerosi esempi per caratterizzare le caratteristiche comuni nel modello e garantire un rilevamento corretto.

Inoltre, la risoluzione dei dati elaborati non era sufficientemente elevata per garantire un rilevamento accurato in questo tipo di immagini. Pertanto, a causa del periodo limitato per l'implementazione del progetto e della specificità del compito, era fondamentale trovare un'alternativa migliore per ottenere i dati satellitari. Tale alternativa, *free* a causa di una falla nel sistema API di Google, si basa sull'uso combinato di QGIS e Google Maps. Per spiegazioni sul funzionamento dettagliato, si rimanda il lettore al Capitolo <sup>[3]</sup>.

## 2.2 Dataset di Addestramento

Per addestrare un modello in grado di rilevare i pannelli solari è necessario un campione relativamente ampio di immagini. Tuttavia, vi è una carenza di dataset che contengano immagini di pannelli solari ripresi da un punto di vista satellitare.

Nell'agosto 2021, però, la *School of Engineering of China* ha pubblicato uno dei dataset più completi di immagini satellitari ( $\approx 7GB$ ) contenenti celle fotovoltaiche <sup>[2]</sup>.

# Addestrare la Rete Neurale: «Che approccio usare?»

## 2.3.1 Approccio di estrazione delle Features [8]

Per il primo articolo, viene esaminato un approccio leggermente più datato che utilizza l'estrazione classica delle caratteristiche per elaborare le immagini ed estrarre i modelli, e una *Random Forest* (RF) per prevedere i livelli di confidenza. Un team di annotatori umani è incaricato di annotare manualmente gli array fotovoltaici (PV).

I dati utilizzati provengono dall'Aerial Dataset (Fresno, USA).

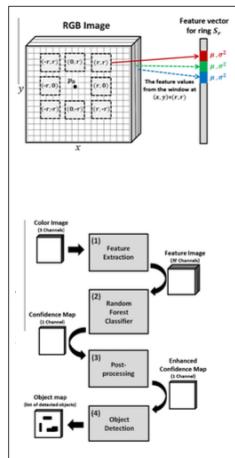


Figura 3: Estrazione delle Features con (*Random Forest*)

Il processo può essere riassunto come segue:

1. Estrazione delle caratteristiche dalle immagini attorno a ciascun pixel che caratterizzano tutti i modelli.
2. La RF assegna una probabilità o confidenza a ciascun pixel di appartenere a un array fotovoltaico.
3. Post-elaborazione per identificare i pixel ad alta confidenza (massimi locali).
4. Rilevamento degli oggetti: identificazione di gruppi di pixel ad alta confidenza contigui.

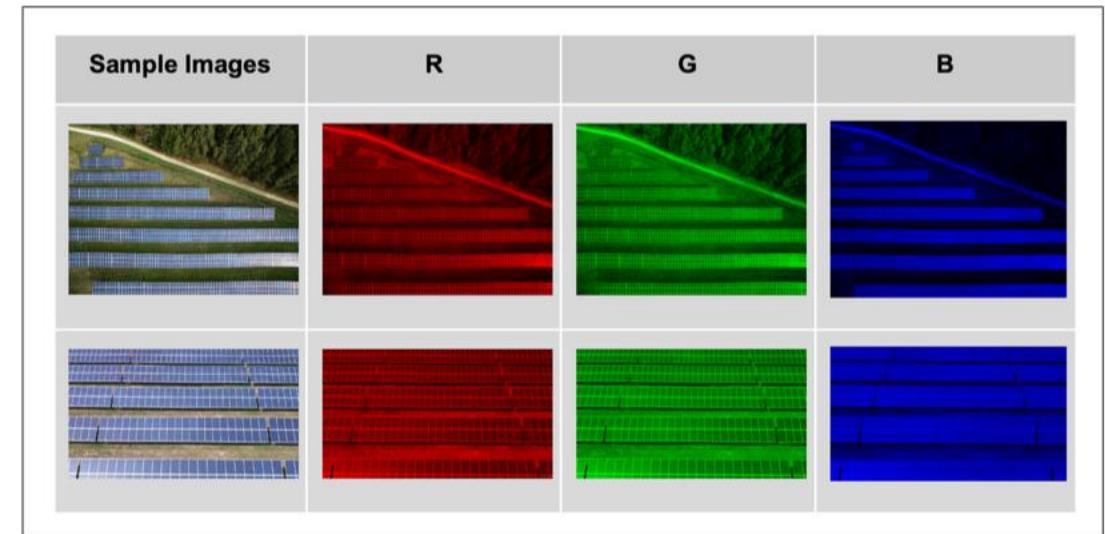


Figura 10: Preprocessing del dataset

# Addestrare la Rete Neurale: «Che approccio usare?»

## 2.3.2 Approccio R-CNN [4]

Questo articolo introduce uno schema di miglioramento basato su *Faster R-CNN* per il rilevamento di piccoli oggetti nelle immagini di telerilevamento, cercando di minimizzare il tempo di elaborazione. Il modello è stato testato per il rilevamento di veicoli e aerei.

Il funzionamento è il seguente:

1. **Uso di un modello pre-addestrato:** viene utilizzato un modello pre-addestrato (VGG16) per addestrare un numero limitato di dati etichettati.
2. **Online Hard Example Mining (OHEM):** questa tecnica viene utilizzata per selezionare i campioni difficili da classificare, basandosi sulla loro perdita, accumulando i gradienti e passando questi al network convoluzionale.
3. **Modifica della risoluzione:** per mantenere la risoluzione durante l'espansione del campo ricettivo, viene rimossa la layer pool4 dal modello pre-addestrato e viene estesa la dimensione dei filtri in Conv5 a 2.
4. **Rappresentazione multi-scala:** invece di utilizzare mappe di caratteristiche di dimensioni fisse dell'ultimo layer della parte CNN per estrarre le regioni candidate, vengono utilizzate le concatenazioni dei risultati da Conv3, Conv4 e Conv5 per generare un nuovo layer convoluzionale. In questo processo vengono applicati metodi di inizializzazione chiamati Xavier, Batch Normalization e ReLU. Il diagramma di flusso dettagliato è mostrato qui sotto.
5. **Ottimizzazione delle connessioni complete:** le due layer completamente connesse per eseguire la classificazione e la regressione vengono sostituite da una layer convoluzionale e una layer di pooling al fine di ridurre il tempo computazionale, diminuendo il numero di parametri generati.

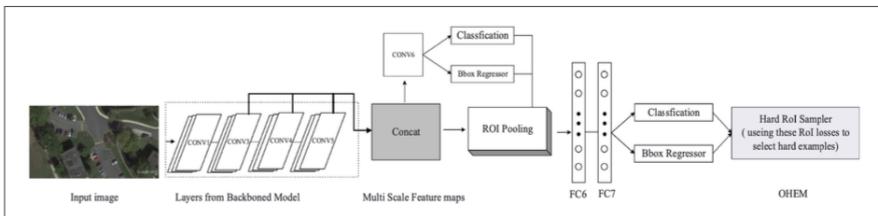


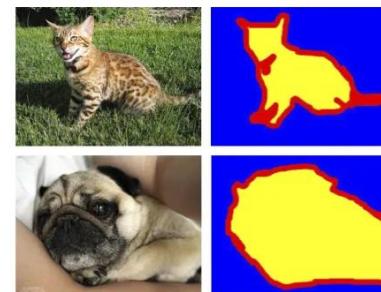
Figura 5: Funzionamento della R-CNN

## 2.3.3 Approccio di Segmentazione [5]

Questo articolo introduce un nuovo approccio al problema che vogliamo risolvere nella nostra ricerca. Esamina anche la segmentazione istantanea come primo passo per fornire una conoscenza affidabile sulla produzione di energia che una determinata area potrebbe avere. Troviamo quest'ultimo articolo di grande importanza per la quantità di diverse architetture che implementa, al fine di comprendere meglio i risultati ottenuti con l'approccio proposto.

Il funzionamento è il seguente:

1. **Implementazione di modelli con diversi encoder:** Lo studio implementa diversi modelli con vari encoder per trovare il metodo più affidabile, verificando anche l'efficienza in termini di tempo. Alcuni dei modelli testati sono Unet, Segnet, DilatedNet, PspNet, DeepLab v3+, Dilated Resnet, mentre gli encoder scelti sono diverse variazioni di VGGNet e Resnet. L'approccio proposto consiste nell'uso di Unet-Mobilenet con Mobilenet come encoder.
2. **Dati di input:** Vengono utilizzate 601 immagini RGB di alta qualità da 5000x5000 pixel, segmentate a 224x224 per catturare meglio i dettagli nelle immagini. Contemporaneamente, vengono ottenute le maschere binarie per utilizzarle come input.
3. **Implementazione della dice loss del layer:** Una delle caratteristiche chiave di questo articolo è l'implementazione di una dice loss layer. Questa layer affronta le bounding box in modo diverso rispetto a metodi più semplici come la cross entropy loss. Poiché questa layer è di grande importanza, la spiegheremo ulteriormente dopo aver mostrato i risultati dei vari modelli utilizzati.



# Addestrare la Rete Neurale: «Che approccio usare?»

Paper Number	Type of Approach	Number of Images in Train/Test	Type of Problem	Techniques	Evaluation Criteria	Performance
1	Classical approach	Training: 1780 (90km <sup>2</sup> ), Testing: 1014 (45km <sup>2</sup> ) + Annotations	Supervised (manually annotated)	Classical feature extraction + Random Forest	PR curves (precision - recall balance) + Jaccard index (for object detection)	Object (array) detection: Precision: 0.9 (J=0.1)
2	Object detection	1000 images with about 7000 aircrafts, 500 images with about 7000 cars	Supervised pre-trained model + Domain-specific	Faster R-CNN + OHEM + Multi-scale representation	Average Precision (AP) + Recall Score	Aircraft dataset: AP: 0.907, Recall: 0.9685 Car dataset: AP: 0.879, Recall: 0.8846
3	Instance segmentation	601 images of size 5000x5000 in subsets of 224x224	Supervised learning with masks	Unet-Mobilenet + Dice loss	Recall + Precision + Time	Solar panel dataset: Precision: 0.8498, Recall: 0.9595

Figura 8: Confronto fra i tre approcci

## IMAGE SEGMENTATION

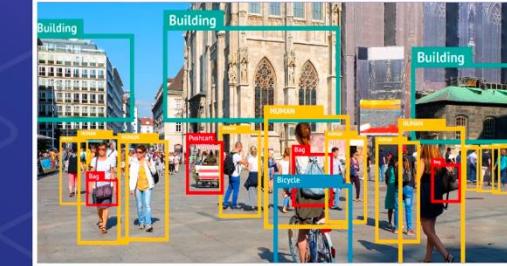
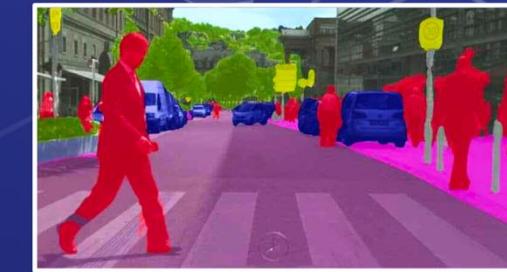
## OBJECT DETECTION

Most optimal  
**mask**  
around the object

Most optimal  
**bounding box**  
around the object

- 👍 **Shape** information of the object.
- 👎 **Slower** to deploy.

- 👍 Generally, **faster**.
- 👎 **No information** about the **shape**



# Pre-processing: «Ottenimento delle Immagini Satellitari»

Il primo passo è stato scaricare gli ShapeFile (.shp) contenenti i confini OMI delle città di interesse. Successivamente, è stata eseguita un'intersezione tra le aree geografiche definite dallo ShapeFile e le immagini satellitari, al fine di estrarre le immagini satellitari solo delle zone pertinenti. Per ottenere tali file abbiamo usato il servizio web gratuito messo a disposizione dall'*Agenzia delle Entrate* [\[7\]](#).

Attraverso questa operazione, è stato possibile ottenere un GeoTIFF contenente le immagini satellitari delle aree selezionate, pronte per essere utilizzate nel successivo processo di riconoscimento e geolocalizzazione dei pannelli fotovoltaici.

In questo capitolo, vengono descritti i passaggi chiave del processo: dalla configurazione iniziale di QGIS ed il collegamento ai provider di dati satellitari, alla selezione e al download delle immagini, fino alla loro preparazione per il *preprocessing*.

Gli ShapeFile sono un formato di dati spaziali usato principalmente nei GIS per rappresentare oggetti geografici e relativi attributi. Sviluppati da ESRI nel 1998, essi sono composti da diversi file interconnessi, di cui almeno tre sono essenziali:

- .shp: contiene le geometrie degli oggetti (punti, linee, poligoni).
- .shx: è l'indice spaziale, che facilita l'accesso ai dati geometrici.
- .dbf: è un file in formato tabellare che contiene gli attributi associati alle geometrie (simile ad un foglio di calcolo).

Oltre a questi, possono esserci file opzionali come:

- .prj: definisce il sistema di coordinate e la proiezione.
- .cpg: specifica la codifica dei caratteri.

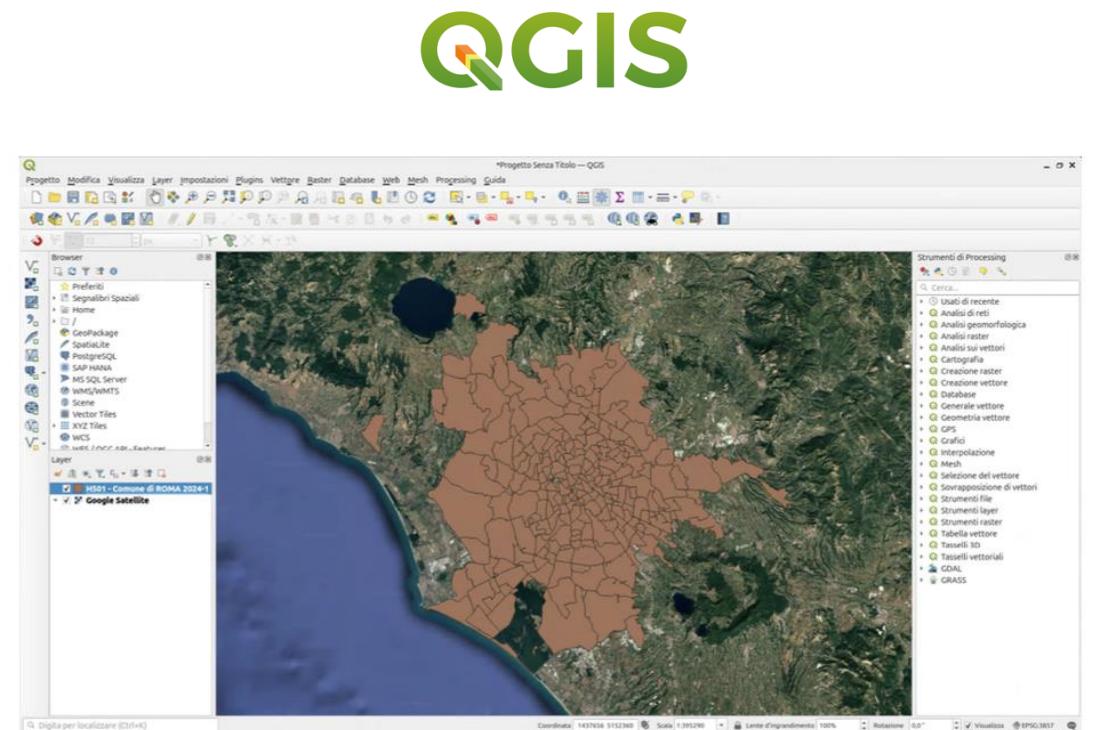


Figura 24: Intersezione dei Layers

# Pre-processing: «GeoTIFF to PNG»

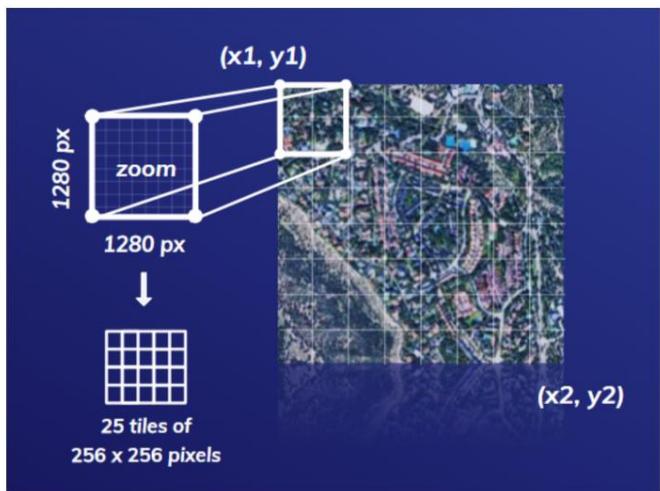


Figura 14: Processo di conversione in PNG

Abbiamo deciso di suddividere il file GeoTIFF in tiles PNH di dimensioni 256x256 pixel, un formato ideale per garantire che il modello di deep learning potesse concentrarsi sui dettagli delle immagini e rilevare con maggiore precisione i pannelli solari. Questo approccio, combinato con l'utilizzo di file associati contenenti informazioni geografiche, ha rappresentato un punto chiave del nostro processo di preprocessing. Ogni tile generato non era un'entità isolata; abbiamo creato file associati per ciascun blocco.

Questi file contenevano informazioni essenziali come:

- Le coordinate geografiche del tile.
- La proiezione e il sistema di riferimento del file GeoTIFF originale.
- I bordi (*bounding box*) del tile nel sistema di coordinate geografico.

Questa struttura ha garantito che ogni immagine mantenesse il suo contesto geospaziale originale. In pratica, i file associati ci hanno permesso di:

1. Mantenere la mappatura geografica: Le predizioni del modello, come la posizione dei pannelli solari, potevano essere ricollegate facilmente al mondo reale.
2. Garantire la replicabilità: Ogni tile e i suoi dati geografici erano tracciabili, consentendo verifiche o integrazioni successive.



Figura 15: Tile di 1024x1024



Figura 16: Tile di 256x256

# Image Segmentation: «UNET++»

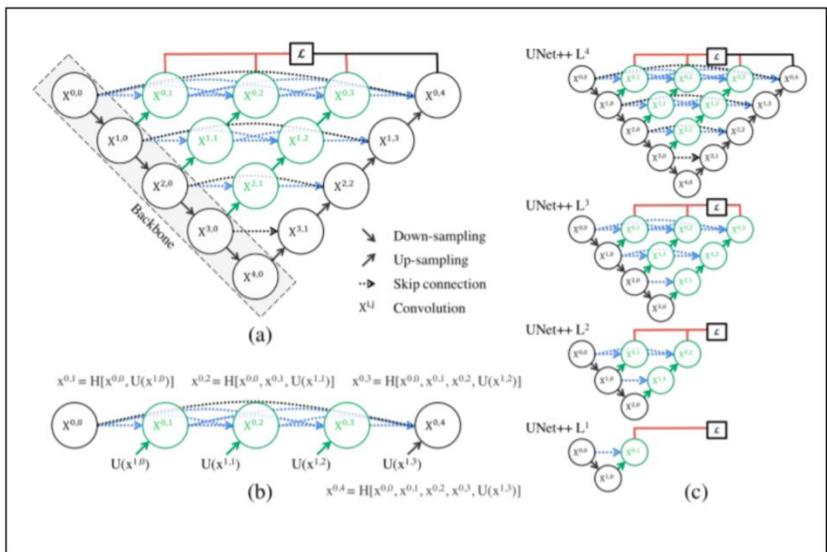


Figura 9: Architettura di UNET++

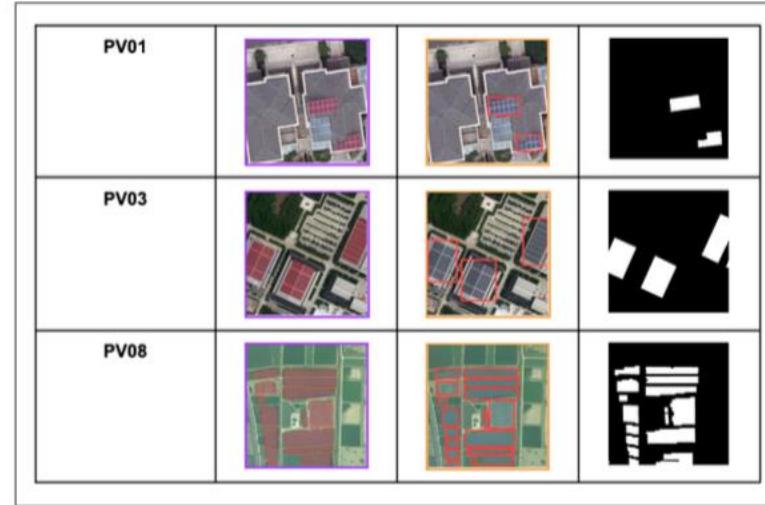


Figura 11: Funzionamento di UNET++

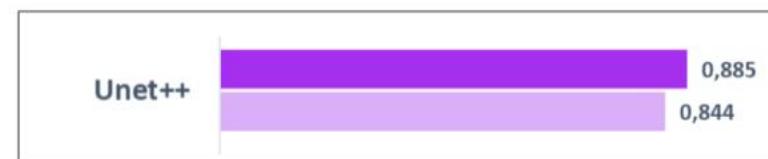


Figura 12: Accuracy di UNET++

# Object Detection: «YOLOv5»

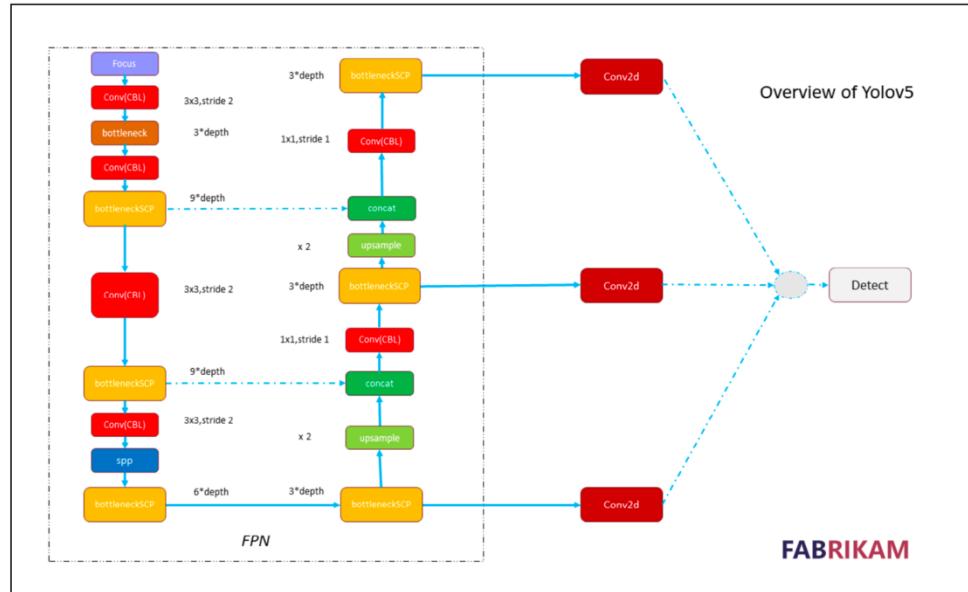


Figura 13: Architettura di YOLOv5

		Yolov5s			Yolov5m			
EPOCHS	Precision	Recall	mAP	F1 score	Precision	Recall	mAP	F1 score
25	0.869	0.778	0.825	0.821	0.879	0.772	0.821	0.813
50	0.893	0.794	0.836	0.841	0.904	0.763	0.829	0.827
100	0.918	0.786	0.841	0.847	0.918	0.786	0.841	0.847

		Yolov5l			Yolov5x			
EPOCHS	Precision	Recall	mAP	F1 score	Precision	Recall	mAP	F1 score
25	0.895	0.782	0.831	0.834	0.895	0.782	0.831	0.834
50	0.899	0.775	0.828	0.832	0.893	0.794	0.836	0.841
100	0.922	0.775	0.836	0.842	0.921	0.784	0.840	0.847

Figura 18: Benchmarks delle reti

	Initial learning rate	Momentum	Weight_decay	hsv-h	hsv-s	hsv-v
For Baseline	0.01	0.937	0.0005	0.015	0.7	0.4
After Evolution	0.01162	0.9334	0.00035	0.01002	0.55308	0.31491

	Translate	Scale	Mosaic	Mixup	Copy-paste	Anchors
For Baseline	0.1	0.9	1	0.1	0.1	3
After Evolution	0.129	0.77614	0.82128	0.09367	0.0979	3.556

Figura 19: Iperparametri utilizzati

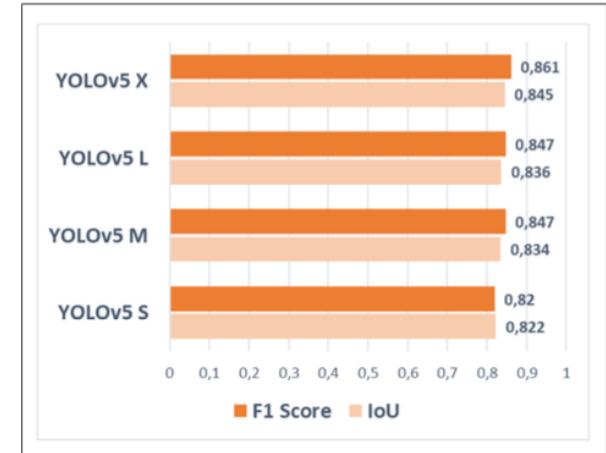


Figura 20: Risultati Complessivi

# Addestrare la Rete Neurale: «Approccio scelto!»

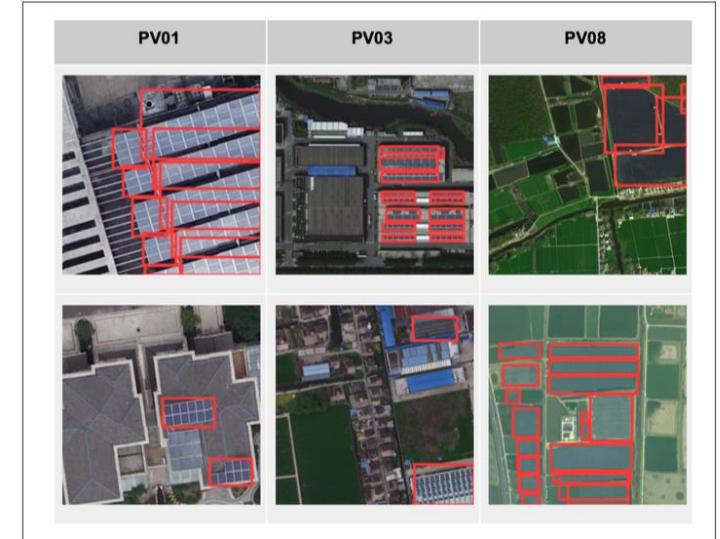
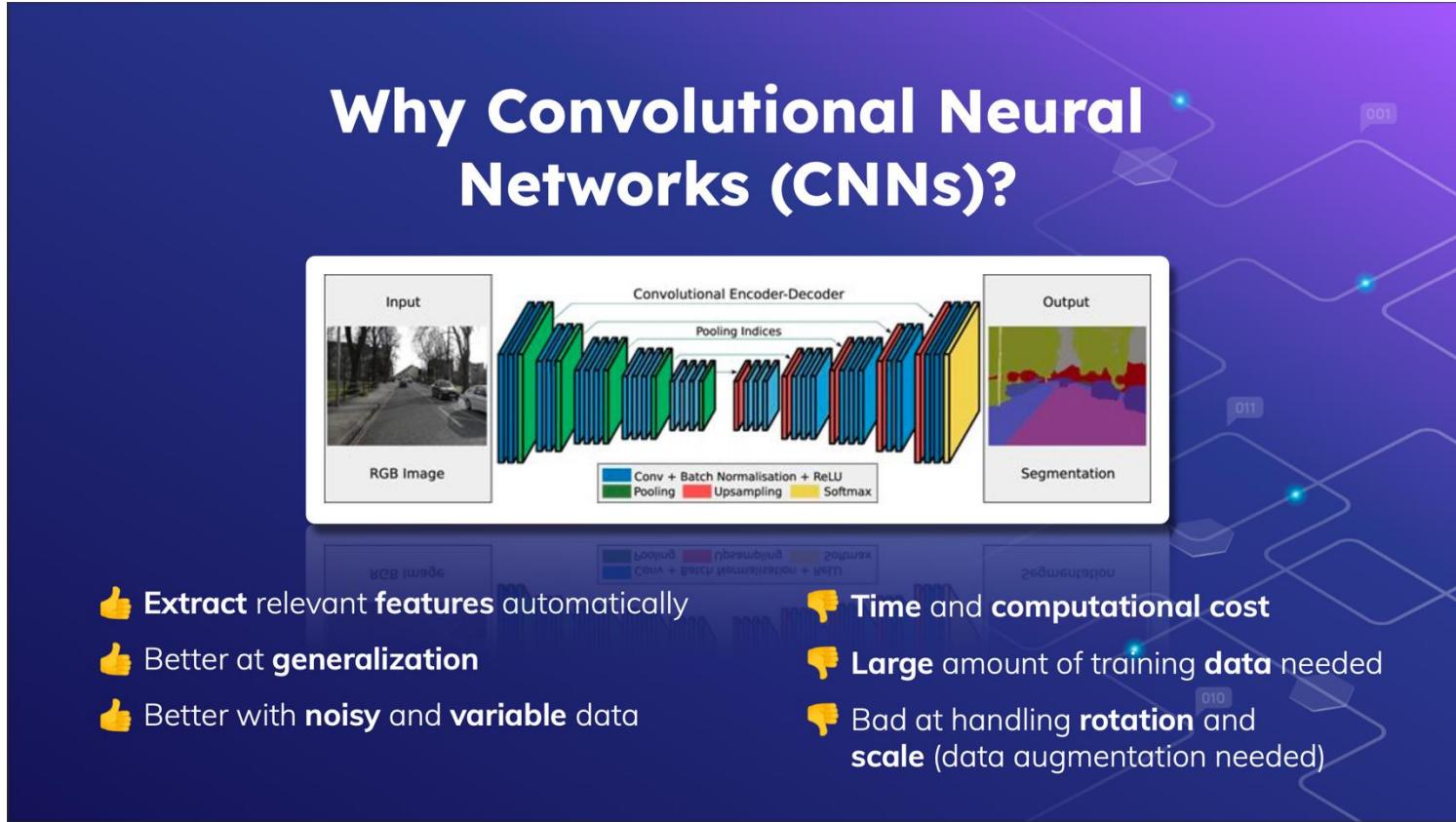


Figura 21: Funzionamento di YOLOv5

# Addestrare la Rete Neurale: «Conclusioni»

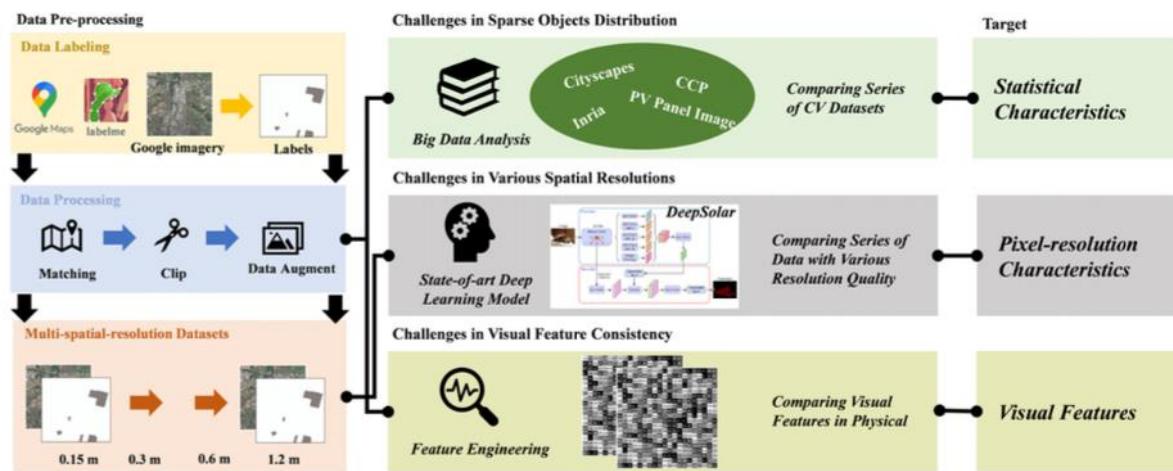


Figura 22: Processo di Addestramento

Siamo partiti dalle immagini satellitare in formato GeoTIFF, le abbiamo "spezzettate" e suddivise in tiles PNG di 256x256, siccome, abbiamo notato empiricamente che era la dimensione ideale per non sovraccaricare la rete con eccessive informazioni visive. Abbiamo addestrato due reti neurali che approcciavano il problema in maniere totalmente differenti: YOLOv5 e UNET++. Nonostante quest'ultima fosse quella con *accuracy* maggiore, essa era troppo lenta ed esosa di risorse computazionali; pertanto, abbiamo optato per usare la rete YOLOv5, che sfrutta il paradigma dell'*Object Detection*.

# CLUSTER MongoDB in Docker: «Focus»

Abbiamo creato una configurazione Docker composta da:

- Replica set per i *server di configurazione*: Tre nodi configurati per gestire la metadata e coordinare la distribuzione dei dati tra gli shard. Ogni nodo è stato eseguito in un contenitore separato, connesso a una rete Docker dedicata.
- Replica set per gli *shard*: Tre shard, ognuno con tre nodi replicati per garantire ridondanza e affidabilità dei dati. Gli shard sono stati configurati per suddividere i dati su più nodi fisici simulati.
- Router (*mongos*): Un componente centrale per la gestione delle query e la connessione agli shard. È stato configurato per comunicare con i server di configurazione e coordinare le operazioni di lettura e scrittura.

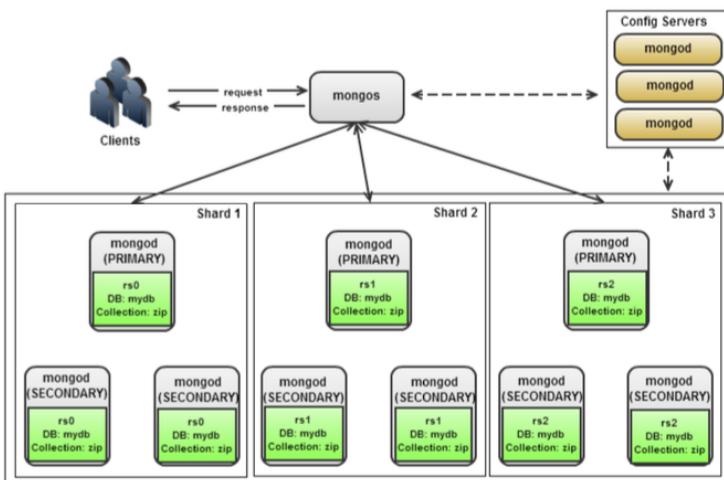
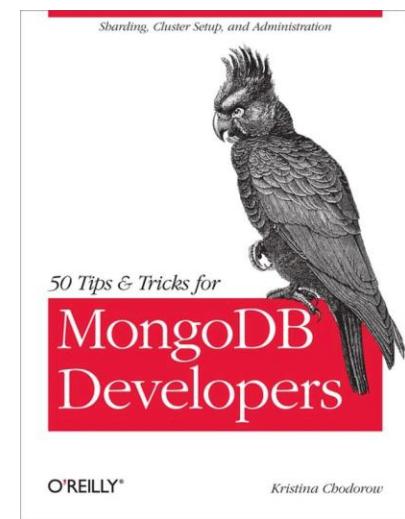


Figura 25: Struttura di un Cluster MongoDB



Ogni componente è stato definito utilizzando un *Docker Compose*, che ha facilitato la gestione dei contenitori e la simulazione dell'infrastruttura distribuita. I dettagli principali includono:

- *Port mapping*: Per ogni nodo, abbiamo mappato porte specifiche sulla macchina host per consentire il debug e l'accesso diretto.
- *Persistenza dei dati*: I volumi Docker sono stati utilizzati per salvare i dati in modo permanente, consentendo di preservare il contenuto tra i riavvii dei contenitori.
- *Rete dedicata*: Una rete Docker bridge è stata creata per garantire che tutti i contenitori potessero comunicare tra loro in modo isolato.

In ogni modo, la gestione di un cluster, seppur solo in simulazione, è stata un'esperienza molto formativa ed interessante!

# CLUSTER MongoDB in Docker: «XFS»

## 4.2.1 Il File System XFS

Vista la grande mole di dati movimentati è stato fondamentale, per il corretto funzionamento del cluster, garantire che tutti i containers avessero accesso a volumi formattati col File System XFS. Quest'ultimo è un file system *journaled*: è una tecnica utilizzata da molti file system moderni per preservare l'integrità dei dati da eventuali cadute di tensione. Derivata dal mondo dei database, il journaling si basa infatti sul concetto di transazione dove ogni scrittura su disco è interpretata dal file system come tale.

Quando un applicativo invia dei dati al file system per memorizzarli su disco, questo prima memorizza le operazioni che intende fare su un file di log e in seguito provvede a effettuare le scritture sul disco rigido, quindi registra sul file di log le operazioni che sono state effettuate. In caso di caduta di tensione durante la scrittura del disco rigido, al riavvio del sistema operativo il file system non dovrà far altro che analizzare il file di log per determinare quali sono le operazioni che non sono state terminate e quindi sarà in grado di correggere gli errori presenti nella struttura del file system.

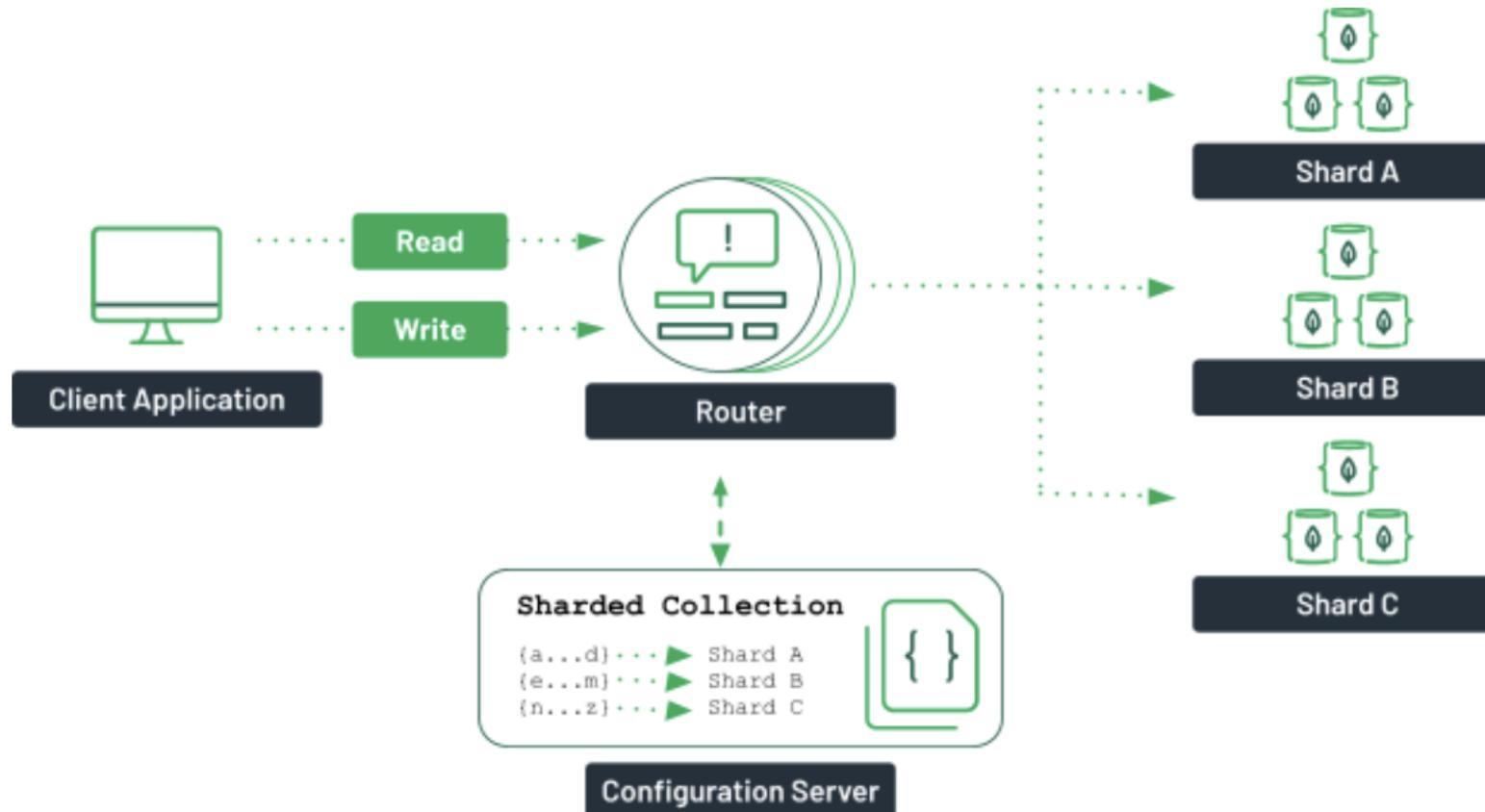
Poiché nel file di log vengono memorizzate solo le informazioni che riguardano la struttura del disco (metadati), un'eventuale caduta di tensione elimina i dati che si stavano salvando, ma non rende incoerente il file system.

Inoltre, è progettato per funzionare in ambienti multithreaded, ottimizzando l'uso delle risorse del sistema. MongoDB, che supporta operazioni parallele tramite replica e distribuzione dei dati, beneficia enormemente di questa caratteristica di XFS. Le operazioni di lettura/scrittura concorrenti su più thread vengono gestite in modo molto più efficiente rispetto a filesystem meno ottimizzati per il multithreading.

Infine, XFS supporta la creazione di *snapshot*, che consente di fare copie istantanee dei dati senza interruzioni. Questo è particolarmente utile in un contesto MongoDB, dove i backup frequenti e coerenti sono essenziali per la protezione dei dati. Gli snapshot



# CLUSTER MongoDB in Docker: «Sharding»

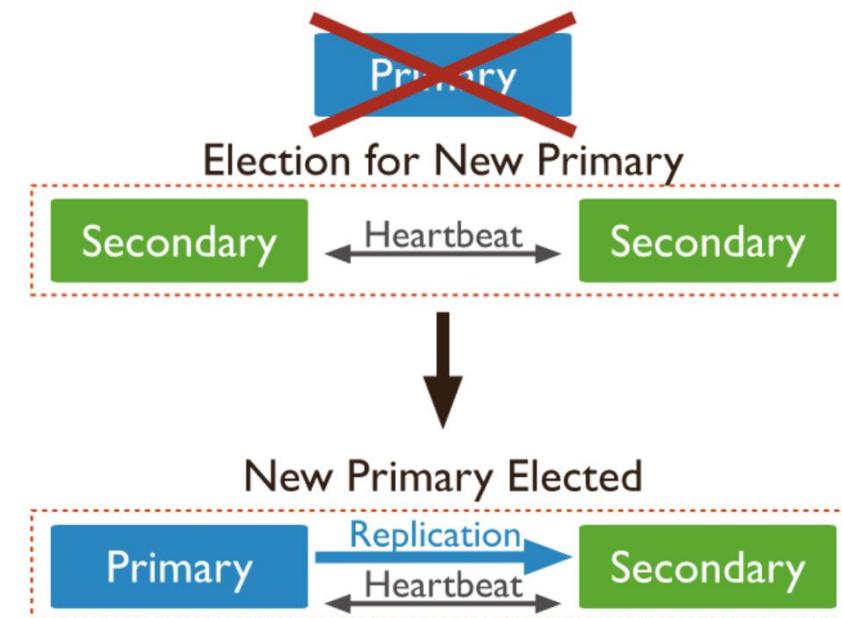
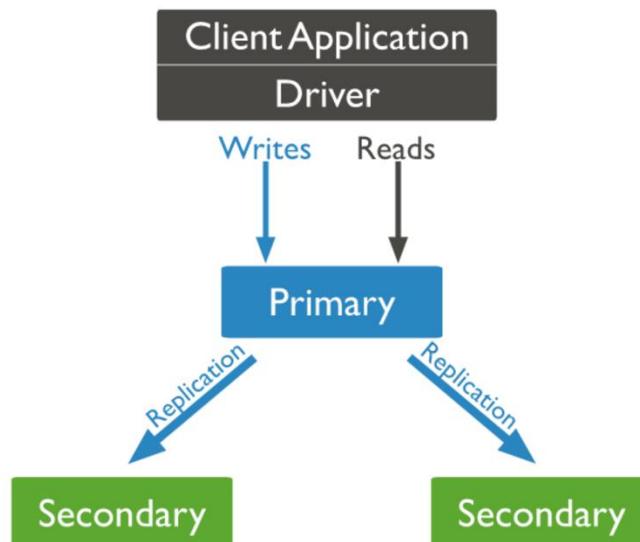
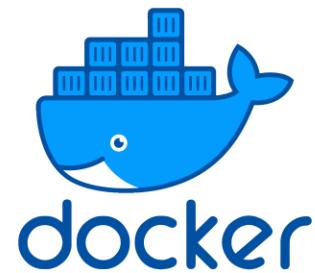


A typical sharded cluster in MongoDB.

# CLUSTER MongoDB in Docker: «Problematiche»

## Caratteristiche del sistema di votazione

1. **Imparzialità programmata:** Le elezioni si basano su parametri come latenza, priorità configurata dall'amministratore e stato del nodo (aggiornato o meno). Non c'è spazio per la manipolazione.
2. **Decentralizzazione:** Ogni nodo ha un peso di voto predeterminato, evitando il predominio arbitrario di un nodo.
3. **Automazione:** L'algoritmo è deterministico e segue regole fisse, evitando comportamenti arbitrari o "imbrogli".
4. **Meccanismi di sicurezza:** Utilizzano autenticazione e autorizzazione per evitare attacchi da nodi malevoli.



# CLUSTER MongoDB in Docker: «Sistema Elettorale»

Con 3 nodi, ne restano 2 se uno cade, ma un nodo non si auto-vota mai!!!

## Un ideale platonico?

Il sistema elettorale di MongoDB si avvicina al concetto platonico di **governo dei migliori**: non un'aristocrazia umana, ma una sorta di aristocrazia algoritmica, dove le decisioni non sono influenzate da passioni o appetiti, ma da un'intelligenza razionale e immutabile. I nodi si comportano come custodi ideali, servendo il bene comune del cluster senza mai deviare dal loro scopo.

Dal punto di vista filosofico, il processo di elezione dei nodi in un cluster MongoDB rappresenta un modello ideale di governance basata su regole invariabili, prive di ambiguità o interessi personali. Questo sistema si discosta radicalmente dalla fallibilità e dalla complessità etica che caratterizzano le votazioni umane.

## Ordine algoritmico e razionalità meccanica

Le elezioni tra i nodi si fondano su un algoritmo deterministico che riflette una forma di **razionalità meccanica**: ogni nodo agisce secondo regole prefissate, incapace di deviare da esse. Questo contrasta con la dimensione umana, dove decisioni e azioni sono spesso influenzate da emozioni, pregiudizi e conflitti d'interesse. Qui, i nodi non cercano il potere, non hanno preferenze né paure; eseguono semplicemente ciò per cui sono stati progettati.

## Assenza di inganno

Nell'ambito della filosofia morale, si potrebbe dire che i nodi vivono in uno stato di **trasparenza assoluta**. Non c'è spazio per la simulazione o la dissimulazione, qualità che in ambito umano sono spesso associate sia all'astuzia che all'inganno. I nodi non "imbrogliano" perché non possono volere altro se non ciò che il sistema richiede: eseguire il protocollo.

# CLUSTER MongoDB in Docker: «Shard Key & Chunk Size»

```
[direct: mongos] test> sh.status()
shardingVersion
{ _id: 1, clusterId: ObjectId('672b7de2350a14f13b3ffde6' ) }
---
shards
[
  {
    id: 'shard-1-replica-set',
    host: 'shard-1-replica-set/shard-1-node-a:27019,shard-1-node-b:27019,shard-1-node-c:27019',
    state: 1,
    topologyTime: Timestamp({ t: 1730903560, i: 11 }),
    replSetConfigVersion: Long('1')
  },
  {
    id: 'shard-2-replica-set',
    host: 'shard-2-replica-set/shard-2-node-a:27019,shard-2-node-b:27019,shard-2-node-c:27019',
    state: 1,
    topologyTime: Timestamp({ t: 1730903561, i: 8 }),
    replSetConfigVersion: Long('1')
  },
  {
    id: 'shard-3-replica-set',
    host: 'shard-3-replica-set/shard-3-node-a:27019,shard-3-node-b:27019,shard-3-node-c:27019',
    state: 1,
    topologyTime: Timestamp({ t: 1730903566, i: 9 }),
    replSetConfigVersion: Long('1')
  }
]
---
active mongoses
[ { '8.0.3': 1 } ]
---
autosplit
{ 'Currently enabled': 'yes' }
---
balancer
{
  'Currently running': 'no',
  'Currently enabled': 'yes',
  'Failed balancer rounds in last 5 attempts': 0,
  'Migration Results for the last 24 hours': 'No recent migrations'
}
```

Figura 26: Stato dello "Sharded Cluster"

# CLUSTER MongoDB in Docker: «Shard Key & Chunk Size»

```
'Impianti_Fotovoltaici.Roma.Redditi': {
    shardKey: { Soggetto: 1, 'Data di Nascita': 1 },
    unique: false,
    balancing: true,
    chunkMetadata: [
        { shard: 'shard-1 replica-set', nChunks: 1 },
        { shard: 'shard-2 replica-set', nChunks: 1 },
        { shard: 'shard-3 replica-set', nChunks: 1 }
    ],
    chunks: [
        { min: { Soggetto: MinKey(), 'Data di Nascita': MinKey() }, max: { Soggetto: "D'ALESSIO MANOLO", 'Data di Nascita': ISODate('1977-08-22T00:00:00.000Z') }, 'on shard': 'shard-1 replica-set', 'last modified': Timestamp({ t: 2, i: 0 }) },
        { min: { Soggetto: "D'ALESSIO MANOLO", 'Data di Nascita': ISODate('1977-08-22T00:00:00.000Z') }, max: { Soggetto: 'FUSCO ANNA MARIA', 'Data di Nascita': ISODate('1964-04-06T00:00:00.000Z') }, 'on shard': 'shard-2 replica-set', 'last modified': Timestamp({ t: 3, i: 0 }) },
        { min: { Soggetto: 'FUSCO ANNA MARIA', 'Data di Nascita': ISODate('1964-04-06T00:00:00.000Z') }, max: { Soggetto: MaxKey(), 'Data di Nascita': MaxKey() }, 'on shard': 'shard-3 replica-set', 'last modified': Timestamp({ t: 3, i: 1 }) }
    ],
    ...
}
```

Figura 27: Esempio di chiave di Shard

---

Nel nostro progetto, abbiamo scelto di impostare la dimensione dei chunk a 64MB, che si è dimostrata essere la soluzione migliore, in grado di affrontare e risolvere alcune problematiche legate alla distribuzione dei dati nel cluster.

# MongoDB: «Strutturazione Generale»

Compass

CLUSTER > Impianti\_Fotovoltaici

Sort by Collection Name

View

Open MongoDB shell Create collection Refresh

Impianti\_Fotovoltaici

Storage size: 122.88 kB Documents: 17 Avg. document size: 10.94 kB Indexes: 3 Total index size: 61.44 kB

Avellino\_OMI

Storage size: 2.06 MB Documents: 35 K Avg. document size: 267.00 B Indexes: 2 Total index size: 2.50 MB

Avellino\_Redditi

Consumi\_Elettrici

Genova\_OMI

Genova\_Redditi

Indirizzi

Inquinanti\_Aria

Milano\_OMI

Milano\_Redditi

Napoli\_OMI

Napoli\_Redditi

Pannelli

Roma\_OMI

Roma\_Redditi

Venezia\_OMI

Venezia\_Redditi

admin

config

Storage size: 177.25 kB Documents: 4.1 M Avg. document size: 142.55 B Indexes: 2 Total index size: 326.78 MB

Storage size: 790.53 kB Documents: 81 Avg. document size: 16.79 kB Indexes: 3 Total index size: 77.82 kB

Storage size: 47.73 kB Documents: 753 K Avg. document size: 269.33 B Indexes: 2 Total index size: 82.03 MB

Storage size: 108.44 kB Documents: 1.2 M Avg. document size: 216.33 B Indexes: 3 Total index size: 192.02 MB

Storage size: 8.02 MB Documents: 183 K Avg. document size: 302.00 B Indexes: 3 Total index size: 39.43 MB

Storage size: 241.66 kB Documents: 40 Avg. document size: 9.65 kB Indexes: 3 Total index size: 102.40 kB

Storage size: 65.43 MB Documents: 1M Avg. document size: 271.00 B Indexes: 2 Total index size: 109.04 MB

Storage size: 450.56 kB Documents: 65 Avg. document size: 11.86 kB Indexes: 3 Total index size: 73.73 kB

Storage size: 11.36 MB Documents: 175 K Avg. document size: 265.00 B Indexes: 2 Total index size: 13.86 MB

Storage size: 1.25 MB Documents: 234 Avg. document size: 9.34 kB Indexes: 3 Total index size: 122.88 kB

Storage size: 98.88 MB Documents: 1.6 M Avg. document size: 269.19 B Indexes: 2 Total index size: 154.54 MB

Storage size: 286.72 kB Documents: 35 Avg. document size: 13.71 kB Indexes: 3 Total index size: 81.92 kB

Storage size: 1.63 MB Documents: 24 K Avg. document size: 267.00 B Indexes: 2 Total index size: 1.90 MB

# MongoDB: «Struttura dei documenti»

```
_id: ObjectId('672c8fb99dc395b37ad8c8c4')
Soggetto : "ABATE ALFONSO"
Data di Nascita : 1977-10-04T00:00:00.000+00:00
Categoria di Reddito : "C"
Codice Attività : 4.33
Reddito Imponibile : 2
Imposta Netta : 582
Reddito d'Impresa / Lavoro Autonomo : 0
Volume d'Affari : 0
Tipo Modello : "730"
```

Figura 30: Document esemplificativo di *Redditi*

```
_id: ObjectId('67362a521959afae48fcda07')
geometry : Object
  type : "Point"
  coordinates : Array (2)
    0: 9.231667
    1: 45.478347
properties : Object
  Idstazione : 705
  Quota : 122
  Provincia : "MI"
  valore : 11.9
  Name : "Milano Pascal Città Studi"
  inquinante : "Ammoniaca"
  um : "µg/m³"
  data_ora : 2018-01-01T08:00:00.000+00:00
  DataStart : "2007-03-30T00:00:00"
  COMUNE : "Milano"
```

Figura 32: Document esemplificativo di *Inquinanti Aria*

```
_id: ObjectId('672ce4c07ba33fd6ec8f2eac')
geometry : Object
  type : "Polygon"
  coordinates : Array (1)
    0: Array (263)
properties : Object
  name : "AVELLINO - Zona OMI D8"
  description : Object
    @type : "html"
    value : "table border="1"><tr><td><b>Cod. Naz. Comune</b></td><td>A509</td></tr>
    styleUrl : "#style_160_0-153_0-0-0_1"
    fill-opacity : 0.6274509803921569
    fill : "#000099"
    stroke-opacity : 0.6274509803921569
    stroke : "#000000"
    stroke-width : 1
    LINKZONA : ""
    CODCOM : "A509"
    CODZONA : "D8"
```

Figura 31: Document esemplificativo di *OMI*

```
_id: ObjectId('673b600d8c9d2b997ffe6a89')
codzona : "D1"
città : "Avellino"
consumo_energetico : 1.99574
Unità Misura : "MWh"
data_ora : 2018-01-01T00:00:00.000+00:00
OMI : ObjectId('672ce4c07ba33fd6ec8f2eae')
```

Figura 33: Document esemplificativo di *Consumi Elettrici*

```
_id: ObjectId('67334e6ee02f8025b7b9dab9')
Soggetto : Array (4)
  0: ObjectId('672c97fd9dc395b37aec8394')
  1: ObjectId('672c98509dc395b37aeee43ed')
  2: ObjectId('672c97b9dc395b37a03da9d')
  3: ObjectId('672c9ad39dc395b37afb579d')
Indirizzo : "Sentiero uliveto"
Civico : 31
Città : "Roma"
location : Object
  type : "Point"
  coordinates : Array (2)
    0: 12.538325
    1: 41.8422703
```

Figura 34: Document esemplificativo di *Indirizzi*

```
_id: ObjectId('6745e0d5c8f82b1ef0dad455')
Indirizzo : "Via della Meccanica"
Civico : 12
Città : "Venezia"
location : Object
  type : "Point"
  coordinates : Array (2)
    0: 12.2125064
    1: 45.4373791
```

Figura 35: Document esemplificativo di *Pannelli*

# MongoDB: «Riferimenti Esterni»

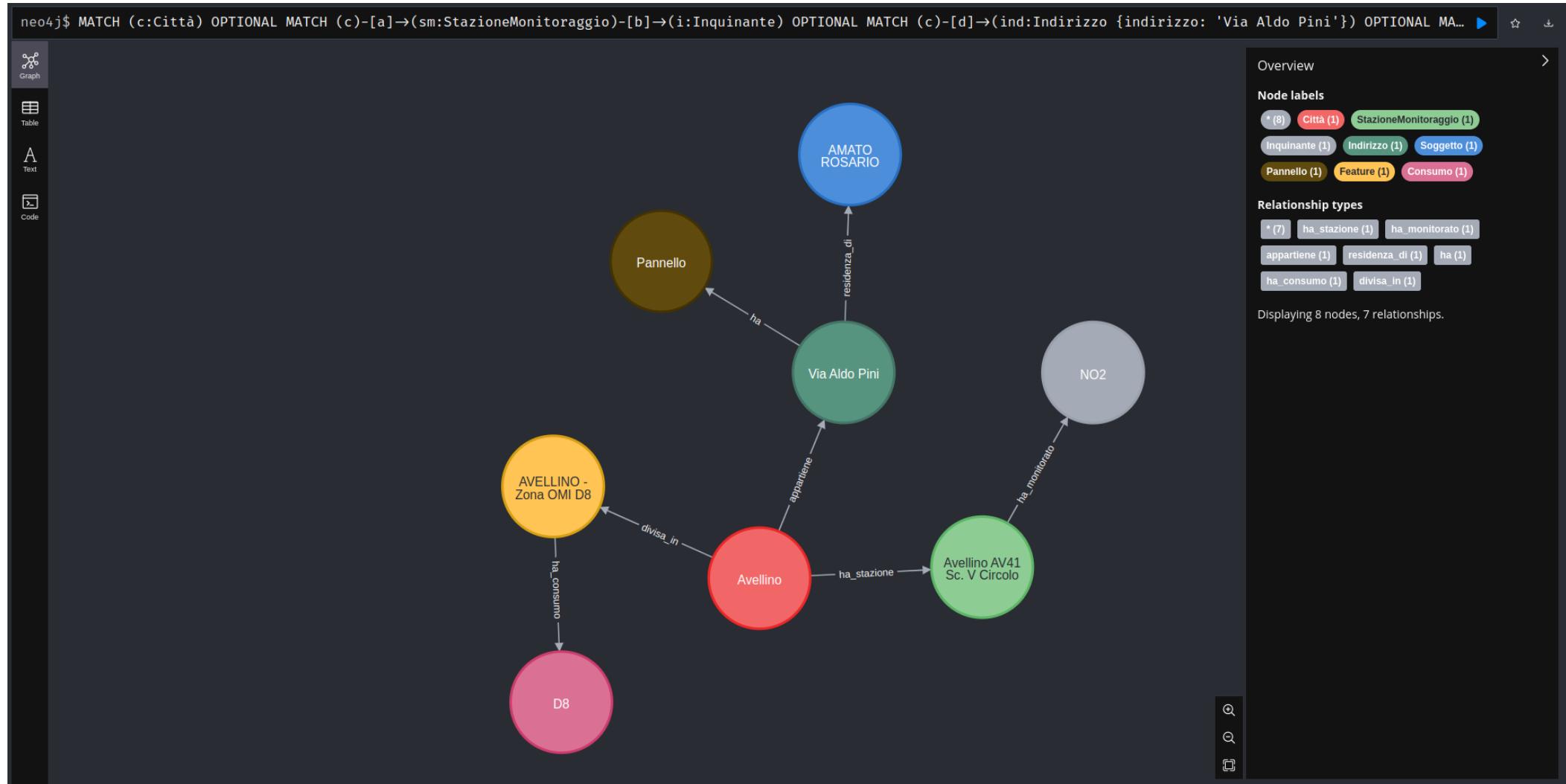
## Motivo della Scelta

L'uso di riferimenti esterni è stato dettato dalla necessità di gestire un'elevata quantità di dati strutturati, in cui le relazioni tra entità giocano un ruolo centrale. Incorporare direttamente i dati (*embedding*) avrebbe potuto complicare le operazioni di aggiornamento e sovraccaricare la memoria del database, soprattutto nel caso di relazioni uno-a-molti o molti-a-molti.

## Relazioni Implementate

- **Soggetti e Indirizzi:** Abbiamo collegato i documenti relativi ai soggetti (persone fisiche) agli indirizzi attraverso riferimenti esterni. Questa relazione ci ha permesso di ricostruire con precisione i nuclei familiari e le loro rispettive residenze. L'associazione è stata utilizzata anche per correlare i dati demografici ai consumi energetici e agli impianti fotovoltaici, rendendo possibili analisi di tipo socio-economico.
- **Consumi e Zone OMI:** I dati sui consumi energetici sono stati associati alle relative zone OMI (Osservatorio del Mercato Immobiliare). Questa relazione ha facilitato l'analisi georeferenziata dei consumi, consentendo di valutare variazioni e trend energetici in base alla posizione geografica e alle caratteristiche delle aree urbane.

# Neo4j: «Struttura del Grafo»



# Neo4j: «Processo di Migrazione da MongoDB»

## 5.1 Strutturazione dei Dati

La migrazione dei dati da MongoDB a Neo4j è stata affrontata come un processo manuale, senza utilizzare strumenti automatici come APOC (*A Procedure On Cypher*) [12].

Questa scelta è stata dettata dalla necessità di preservare e ricostruire la struttura complessa e le relazioni annidate che caratterizzavano il nostro database da migrare.

In MongoDB, le informazioni erano organizzate in documenti JSON-like, spesso con relazioni implicite o gerarchiche tra i dati. Ad esempio, un documento relativo ad un inquinante poteva contenere direttamente informazioni sulla zona OMI associata o un riferimento a un altro documento correlato. Questa struttura non è facilmente traducibile in un grafo senza un'analisi accurata della semantica stessa che vi è nei dati!

In Neo4j, abbiamo adottato un approccio che mette al centro il grafo come modello concettuale. I dati sono stati completamente riorganizzati, in nodi e corrispondenti relazioni.

La differenza principale che salta all'occhio è quella di aver elevato ad entità, quello che in MongoDB siamo riusciti a modellare col costrutto della *Collections*, ovvero: le **Città**. E' partendo da questa entità che siamo riuscito ad aver un unico grande grafo interconnesso, un esempio è visibile nelle immagini sottostanti:

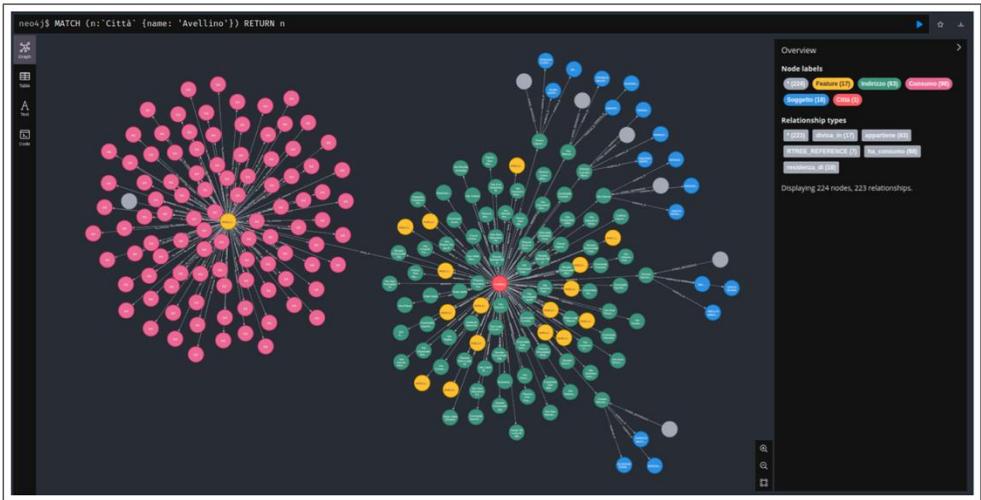


Figura 48: Visione d'insieme della città di Avellino

### 5.1.6 I Pannelli

Questo nodo non contiene alcun dato (*Empty Node*) poiché la sua semantica è restituita solo se relato all'indirizzo corrispondente alla sua locazione spaziale.

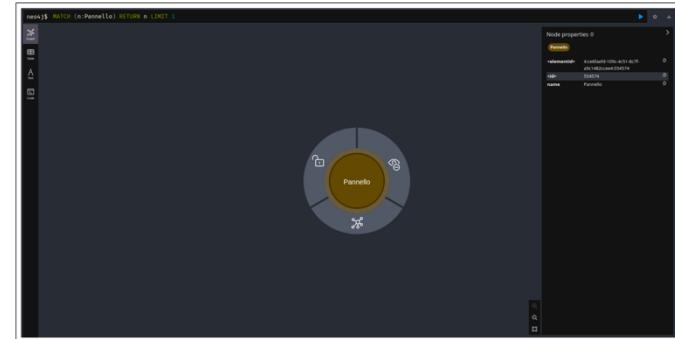


Figura 54: Il Nodo *Pannello*

### 5.1.3 Gli Inquinanti Atmosferici

Questo nodo rappresenta l'inquinante misurato e ha come attributo la stazione che lo ha misurato. La stazione di monitoraggio è stata elevata ad entità autonoma ed il rispettivo valore misurato è divenuto attributo della relazione fra quest'ultimi.

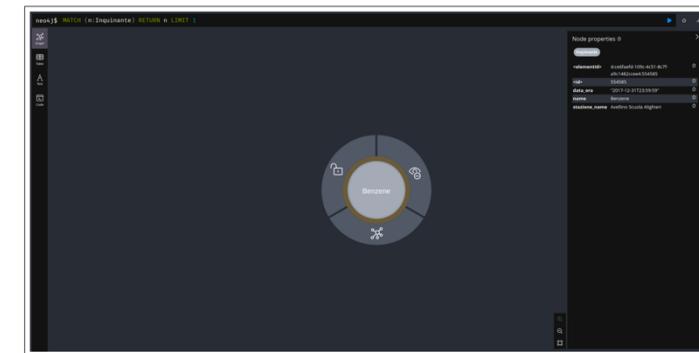
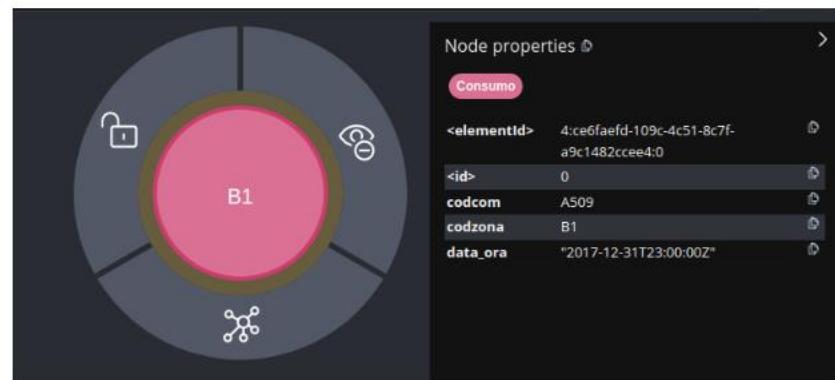
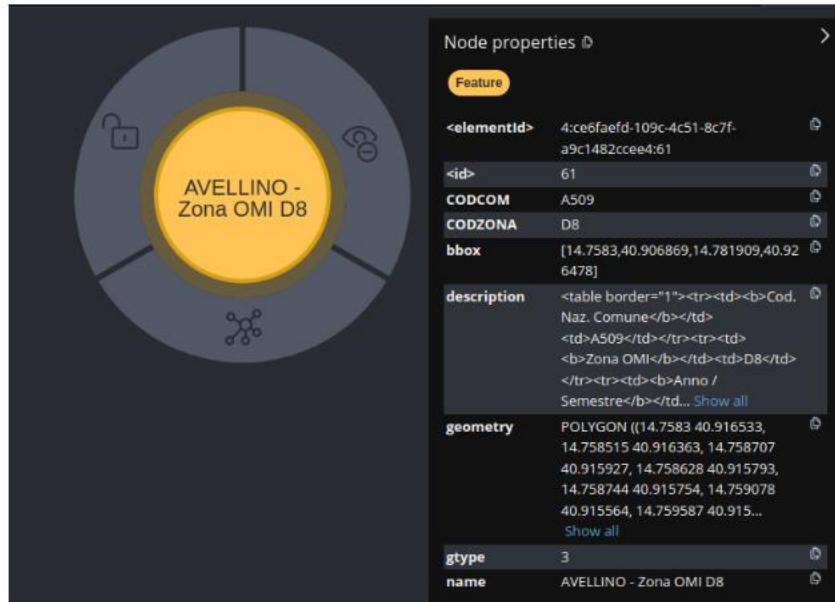
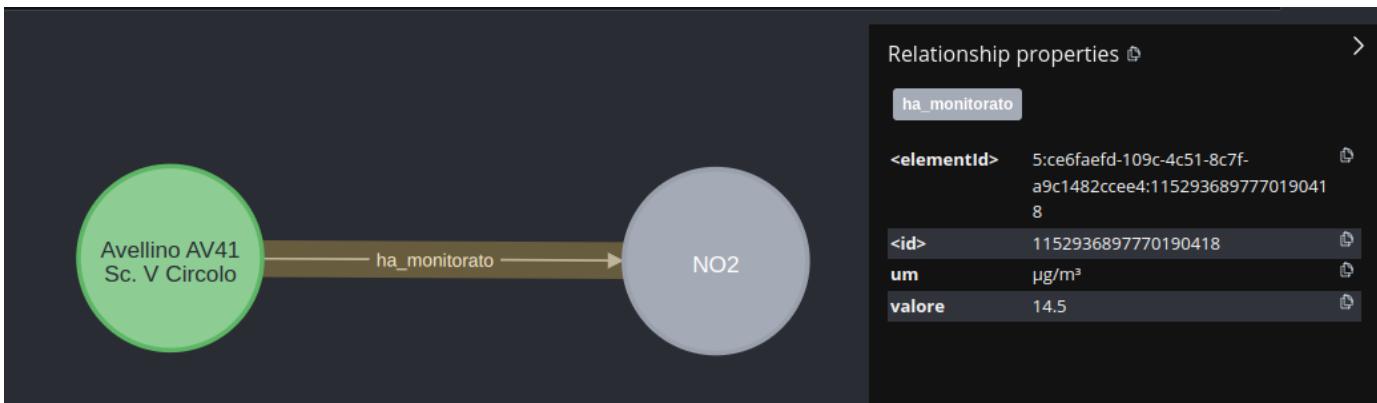
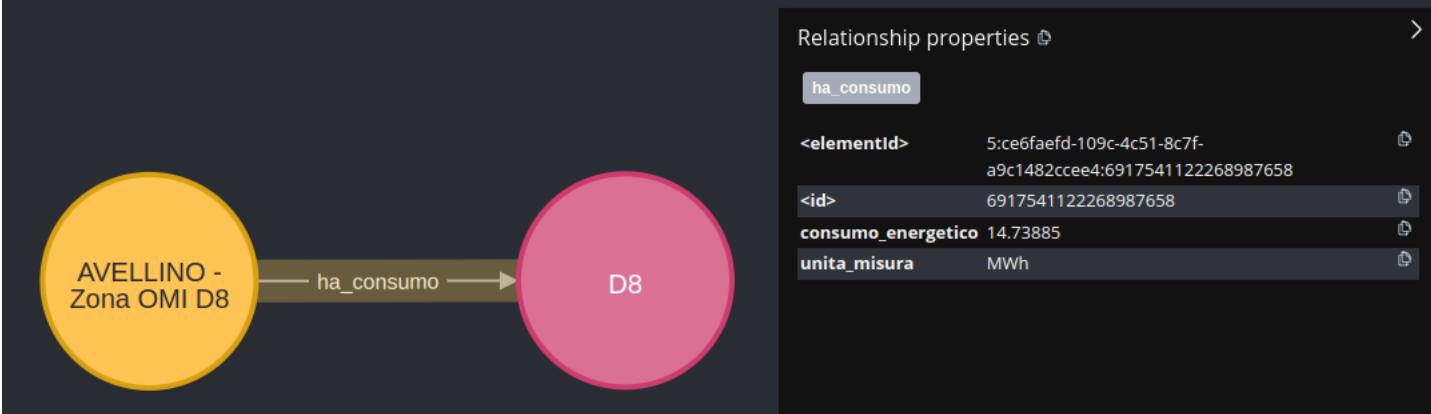


Figura 51: Il Nodo *Inquinanti Atmosferici*

# Neo4j: «Struttura dei Nodi»



# Neo4j: «Relazioni con proprietà»



# Neo4j: «Problematiche»

## 4. Gestione della Memoria:

- **MongoDB:** MongoDB è progettato per essere molto efficiente in operazioni di lettura e scrittura su grandi set di dati. L'uso di C++ consente un controllo diretto della memoria, riducendo l'overhead e permettendo l'elaborazione più rapida dei dati. Inoltre, MongoDB usa il sistema di **memory-mapped files**, che migliora ulteriormente l'efficienza di accesso ai dati.
- **Neo4j:** Pur avendo configurazioni avanzate di gestione della memoria, Neo4j può comunque subire un certo overhead rispetto a MongoDB. Java è progettato per la portabilità e la gestione automatica della memoria, ma questo viene a scapito di un controllo più diretto. Il garbage collector della JVM, che gestisce la memoria, può introdurre pause occasionali durante l'elaborazione, rallentando operazioni particolarmente intensive come il caricamento massivo di dati. Inoltre, la gestione della memoria in Neo4j dipende molto dalla configurazione di **heap memory** e **page cache**, e se queste non sono ottimizzate correttamente, le prestazioni possono degradare, soprattutto su set di dati molto grandi.

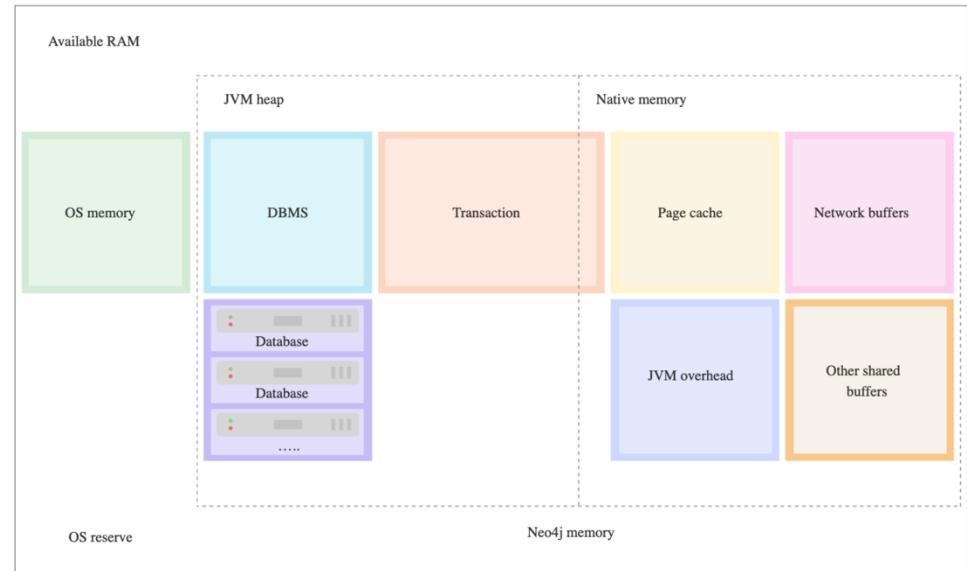


Figura 73: La gestione della memoria in *Neo4j*

Anche se abbiamo aumentato i parametri di memoria di Neo4j, non siamo riusciti a caricare tutti i dati che avevamo in MongoDB, e questa difficoltà può essere attribuita a diversi fattori legati alle differenze architettonali e implementative tra i due database, in particolare, per quanto riguarda la base su cui ciascuno è costruito: **Neo4j** è scritto in **Java**, mentre **MongoDB** è scritto in **C++**.

# Neo4j: «Problematiche»

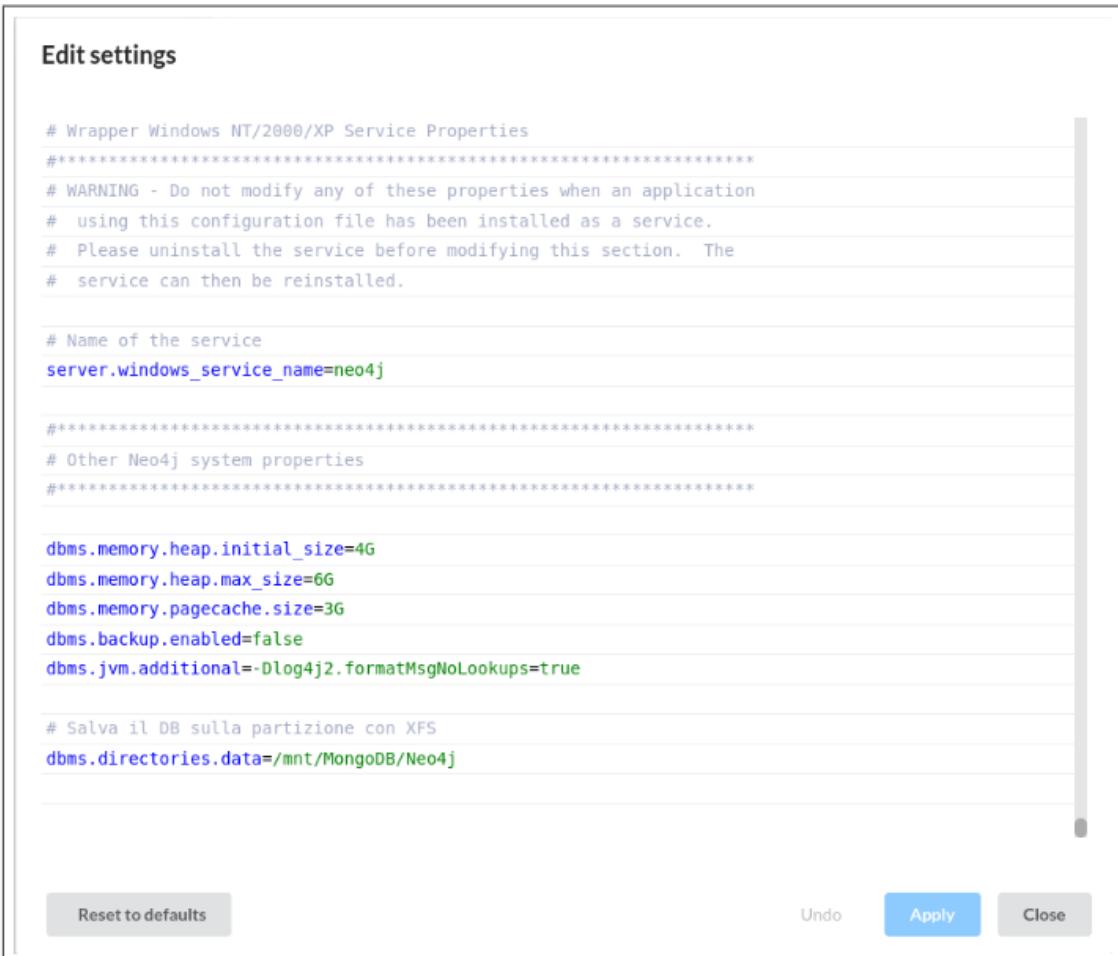
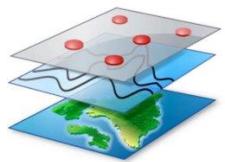


Figura 74: Personalizzazione dei parametri della JVM

Nonostante l'ottimizzazione apportata alla memoria della JVM, abbiamo: aumentato la dimensione della **page cache**, della **heap memory** e fatto puntare il DBMS in una partizione formattata con XFS [vedi Figura 74], la differenza intrinseca tra i due sistemi ha un impatto significativo sul processo di caricamento dei dati:

- **Neo4j:** Con la sua struttura grafica e la gestione complessa delle relazioni tra i nodi, Neo4j richiede una gestione accurata della memoria per evitare che il sistema venga rallentato. Inoltre, i **garbage collector di Java** possono introdurre pause nel flusso di lavoro, impedendo al sistema di gestire un volume molto elevato di dati in modo fluido.
- **MongoDB:** Grazie al suo modello più semplice e alla scrittura ottimizzata per set di dati non relazionati, MongoDB riesce a caricare grandi quantità di dati con minor overhead. In più, la gestione diretta della memoria in C++ consente di sfruttare meglio le risorse hardware, risultando più veloce durante l'importazione massiva di dati.

# Neo4j: «Query Geospaziali»



Neo4j Spatial  
An Introduction

Il plugin **Neo4j-Spatial** è una libreria che estende le funzionalità di Neo4j per permettere di lavorare con **dati geospaziali** in modo efficiente. Con esso, è possibile memorizzare e interrogare dati geografici come punti, linee, poligoni, e utilizzare operazioni spaziali per eseguire ricerche e analisi geospaziali direttamente nel grafo.

## Cosa fa Neo4j-Spatial?

Neo4j-Spatial consente di memorizzare dati spaziali nel grafo e di eseguire query spaziali come:

- **Punti (Point)**: Rappresentazione di coordinate geografiche in un sistema di riferimento spaziale, ad esempio, latitudine e longitudine.
- **Poligoni (Polygon)**: Geometrie definite da un insieme di coordinate, come aree geografiche delimitate.
- **Linee (LineString)**: Rappresentazione di linee geografiche, ad esempio per tracciare strade o percorsi.

Il plugin rende anche possibile eseguire operazioni come la ricerca di nodi o poligoni all'interno di una determinata area geografica, calcolare distanze tra punti, identificare l'intersezione tra geometrie, e molto altro.

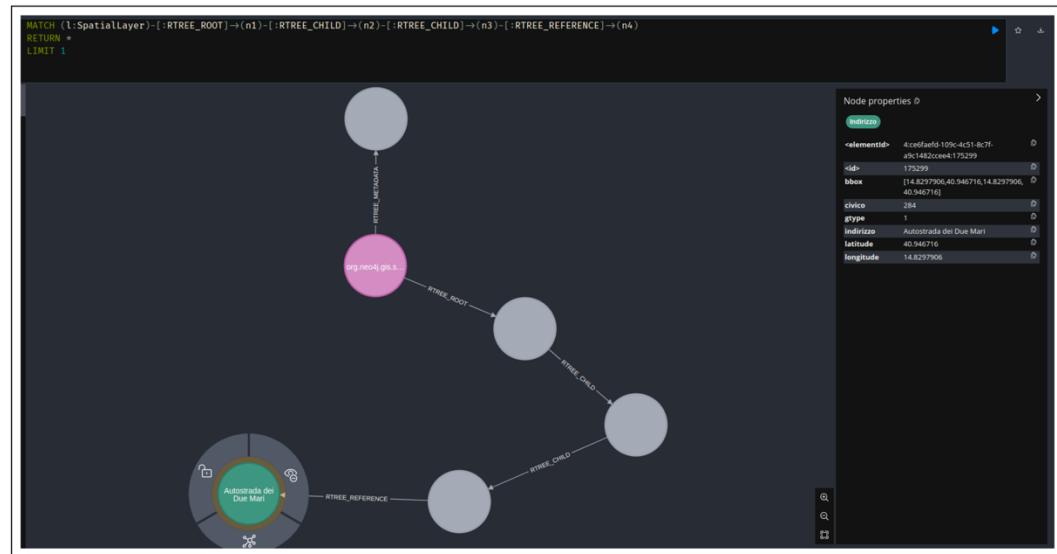


Figura 57: Struttura del "Layer Spaziale" in Neo4j

## Come funziona Neo4j-Spatial?

1. **Aggiungere dati spaziali al grafo:** Puoi memorizzare dati spaziali come proprietà su nodi o relazioni. Ad esempio, puoi avere un nodo di tipo **Città** con una proprietà **location** che rappresenta la latitudine e la longitudine della città come punto geospaziale.
2. **Creare indici spaziali:** Per ottimizzare le query geospaziali, puoi creare indici spaziali sui nodi che contengono dati geospaziali. Questo accelera notevolmente le ricerche spaziali.
3. **Query spaziali con Cypher:** Neo4j-Spatial si integra perfettamente con il linguaggio di query **Cypher**, il che significa che puoi utilizzare le funzionalità spaziali direttamente nelle tue query. Ad esempio, puoi usare la funzione **spatial.intersects()** per trovare nodi che si trovano all'interno di un poligono, o **spatial.distance()** per calcolare la distanza tra due punti.

# Neo4j: «Gli R-Tree»

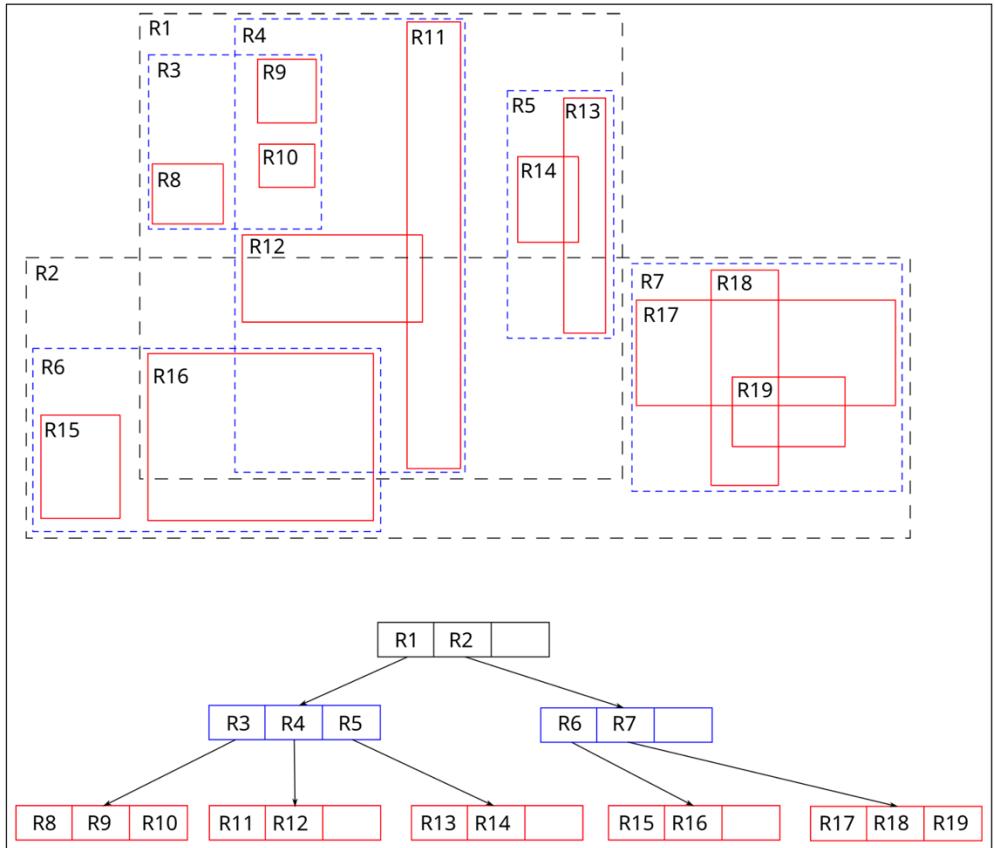


Figura 56: La struttura dati "R-Tree"

## Cos'è un R-Tree?

Un R-Tree è un albero bilanciato progettato per l'indicizzazione di oggetti spaziali multidimensionali, come rettangoli, poligoni, cerchi e altre forme geometriche. La struttura dell'R-Tree è particolarmente adatta per applicazioni geospaziali, dove è fondamentale lavorare con oggetti che occupano aree nello spazio e la loro posizione deve essere determinata rispetto ad altre geometrie.

In termini semplici, un R-Tree è un albero dove:

- Ogni nodo contiene un rettangolo (spesso chiamato “bounding box”) che rappresenta l’area geografica coperta dal nodo.
- Ogni foglia dell’albero rappresenta un oggetto spaziale effettivo, come un punto, una linea o un poligono.
- I nodi non fogliare rappresentano aree “raggruppate” di oggetti spaziali, e sono utili per la ricerca efficiente, riducendo il numero di confronti tra oggetti spaziali.

## Struttura dell'R-Tree

Un R-Tree è costituito da vari livelli di nodi:

- **Nodi foglia:** Contengono i dati spaziali effettivi (ad esempio, i poligoni o i punti) e le loro proprietà. In pratica, ogni foglia ha un bounding box che rappresenta l’area spaziale dell’oggetto.
- **Nodi non foglia:** Contengono bounding boxes che racchiudono i bounding box dei nodi figli. Ogni nodo non foglia rappresenta una “superficie” che racchiude altre superfici.
- **Radice:** È il nodo più alto dell’albero e contiene il bounding box che racchiude tutti gli oggetti nell’albero. La radice è il punto di partenza per la ricerca spaziale.

Ogni nodo dell’albero è generalmente bilanciato in modo che l’albero non diventi troppo profondo, migliorando così le prestazioni nelle operazioni di ricerca e aggiornamento.

# Neo4j: «Gli R-Tree»

## Vantaggi dell'R-Tree

- **Efficienza nelle ricerche spaziali:** L'R-Tree riduce significativamente il numero di comparazioni necessarie durante le ricerche spaziali. Poiché i nodi non foglia contengono informazioni sugli oggetti che racchiudono, è possibile eliminare ampie porzioni del grafo senza doverle esplorare nel dettaglio.
- **Adattabilità a geometrie complesse:** L'R-Tree è molto flessibile e si adatta bene a dati spaziali di vario tipo e complessità, come poligoni, linee e punti. È particolarmente utile per applicazioni geospatiali che richiedono operazioni di ricerca su grandi quantità di dati.
- **Bilanciamento automatico:** Il bilanciamento dell'albero evita che la struttura diventi troppo sbilanciata, il che potrebbe rallentare le ricerche. In pratica, l'albero cresce e si adatta in modo dinamico ai dati, mantenendo buone prestazioni.
- **Ottimizzazione per l'area:** Gli oggetti spaziali che sono vicini tra loro fisicamente sono facilmente raggruppati insieme, riducendo la necessità di fare confronti tra oggetti distanti tra loro.

## Operazioni di ricerca

Quando si eseguono operazioni di ricerca, come la ricerca di intersezioni spaziali o la determinazione di distanze tra oggetti, l'R-Tree sfrutta il principio di “propagazione” delle intersezioni tra i nodi:

- Inizia la ricerca dalla radice dell'albero e naviga verso il basso, visitando solo i nodi che potrebbero contenere oggetti che soddisfano la condizione di ricerca.
- I nodi vengono esplorati in base alla coincidenza dei loro bounding box con l'area di ricerca. Poiché i nodi sono organizzati in modo spaziale, questo approccio riduce notevolmente il numero di confronti da fare, limitando le ricerche solo ai nodi che effettivamente potrebbero essere rilevanti.
- Quando si arriva ai nodi foglia, viene eseguita la ricerca effettiva sugli oggetti spaziali (ad esempio, calcolando l'intersezione tra il poligono ricercato e gli altri oggetti).

# QUERY 1: «Reddito medio per città»

## Obiettivo

Questa query ha l'obiettivo di ottenere una media globale del reddito imponibile per tutti i soggetti presenti nella collezione `Roma_Reddit`. Non vengono utilizzati gruppi specifici (ad esempio, per città, anno o altre caratteristiche) in quanto l'intento è calcolare la media complessiva di tutti i dati nella collezione.

The screenshot shows the MongoDB Query Planner interface. On the left, there is a code editor with the following query:

```
1 {  
2   _id: null,  
3   media_reddito: {  
4     $avg: "$Reddito Imponibile"  
5   }  
6 }
```

To the right of the code editor is a panel titled "Output after \$group stage (Sample of 1 document)" which displays the result of the query:

```
_id: null  
media_reddito : 2558.61396
```

Figura 36: QUERY 1: Struttura

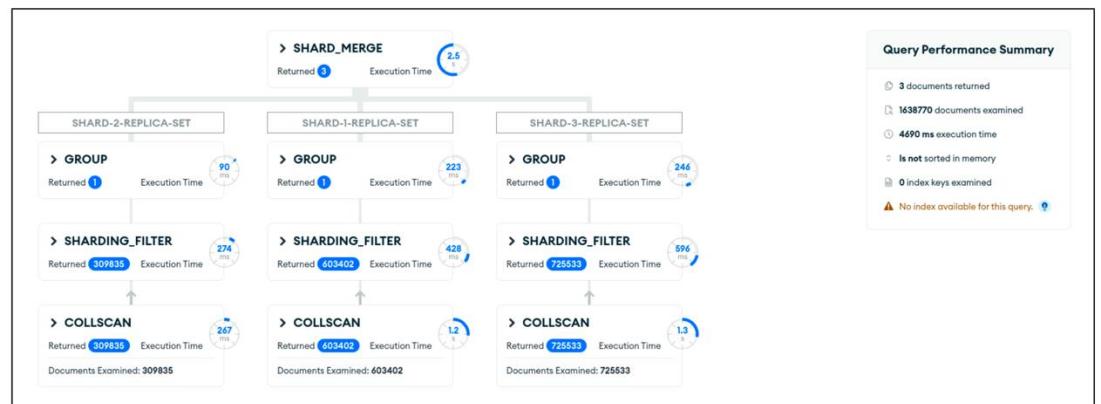


Figura 37: QUERY 1: Esecuzione senza indice

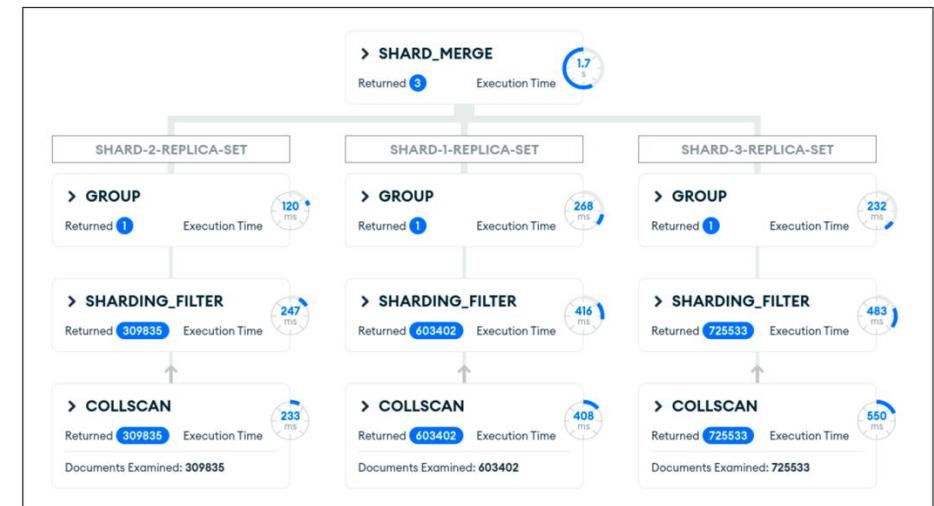


Figura 38: QUERY 1: Esecuzione con indice

# QUERY 1: «Reddito medio per città»

```
1 MATCH (c:Città)-[:appartiene]→(i:Indirizzo)-[:residenza_di]→(s:Soggetto)
2 RETURN c.name AS città, AVG(s.reddito_imponibile) AS media_reddito
3 ORDER BY media_reddito DESC
```

città	media_reddito
"Avellino"	2094.110570659258
"Venezia"	1726.9606016896805

```
neo4j@neo4j:~$ PROFILE
MATCH (c:Città)-[:appartiene]→(i:Indirizzo)-[:residenza_di]→(s:Soggetto)
RETURN c.name AS città, AVG(s.reddito_imponibile) AS media_reddito
ORDER BY media_reddito DESC;
```

città	media_reddito
"Avellino"	2094.110570659258
"Venezia"	1726.9606016896805

```
Plan | Statement | Version | Planner | Runtime | Time | DBHits | Rows | Memory (Bytes)
-----+-----+-----+-----+-----+-----+-----+-----+-----+
"PROFILE" | "READ ONLY" | "5" | "COST" | "PIPELINED" | 1075 | 452809 | 2 | 1664
```

```
Cypher 5
Planner COST
Runtime PIPELINED
Runtime version 5.25
Batch size 1024
```

Operator	Id	Details	Estimated Rows	Rows	DB Hits	Memory (Bytes)	Page Cache Hits/Misses	Time (ms)	Ordered by	Pipeline
+ProduceResults	0	città, media_reddito	242	2	0	0	0/0	0.155		
+Sort	1	media_reddito DESC	242	2	0	512	0/0	0.058	media_reddito DESC	In Pipeline 2
+EagerAggregation	2	cache{c.name} AS città, AVG(s.reddito_imponibile) AS media_reddito	242	2	117608	1032				
+Filter	3	s:Soggetto	58804	58804	117608					
+Expand(All)	4	(i)-[anon_i:residenza_di]->(s)	58804	58804	98457					
+Filter	5	i:Indirizzo	39653	39653	79360					
+Expand(All)	6	(c)-[anon_e:appartiene]->(i)	39653	39653	39655					
+CacheProperties	7	cache{c.name}	2	2	118					
+NodeByLabelScan	8	c:Città	10	2	3	376	81089/0	255,588		Fused in Pipeline 0

Total database accesses: 452809, total allocated memory: 1664  
2 rows  
Ready to start consuming query after 773 ms, results consumed after another 302 ms

# QUERY 2: «Tasso di Pannelli per Zona OMI»

## Obiettivo

Questa query ha l'obiettivo di determinare il numero di pannelli fotovoltaici presenti in ogni zona OMI della città di Roma. I pannelli sono mappati rispetto alle zone utilizzando un'interrogazione geospatial basata sulla relazione `$geoIntersects`, che verifica quali geometrie (i pannelli) si trovano all'interno delle geometrie delle zone OMI definite nel database. Questo tipo di analisi è utile per comprendere la distribuzione degli impianti nelle diverse aree territoriali.

```
46 # Iteriamo attraverso tutti i pannelli
47 for pannello in pannelli:
48     # Controlliamo in quale zona OMI si trova il pannello
49     omi_zones = db.Roma_OMI.find({
50         "geometry": {
51             "$geoIntersects": {
52                 "geometry": pannello["location"]
53             }
54         }
55     })
56
57     # Per ogni zona OMI trovata, incrementiamo il conteggio dei pannelli in quella zona
58     for omi_zone in omi_zones:
59         zona_name = omi_zone["properties"]["name"]
60         if zona_name not in zona_pannelli:
61             zona_pannelli[zona_name] = 0
62             zona_pannelli[zona_name] += 1
63
64
```

Figura 39: QUERY 2: Struttura

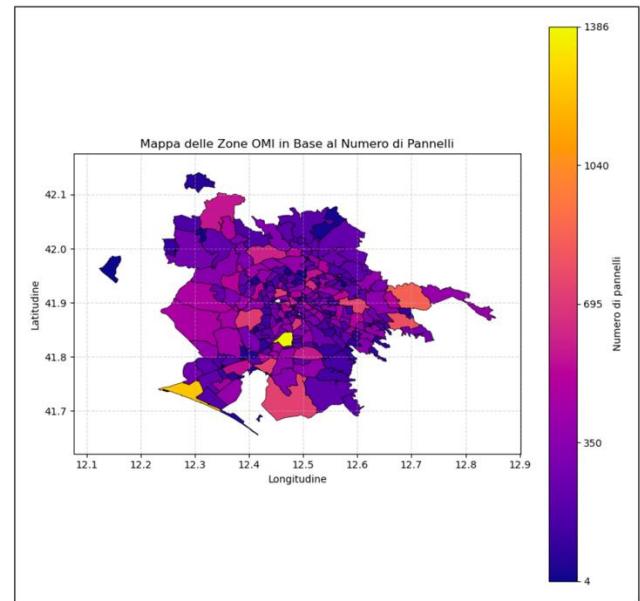


Figura 40: QUERY 2: Risultato

```
Totale documenti esaminati: 113775
Totale tempo di esecuzione (ms): 25754
```

Figura 41: QUERY 2: Esecuzione

# QUERY 2: «Tasso di Pannelli per Zona OMI»

```
1 MATCH (c:Città {name: 'Avellino'})-[r]-(i:Indirizzo)-[ha]→(p:Pannello)
2 WITH i.latitude AS lat, i.longitude AS lon, i
3 WITH point({latitude: lat, longitude: lon}) AS indirizzo_point, i
4 CALL spatial.intersects('zone_omi', indirizzo_point) YIELD node AS polygon
5 WITH polygon.name AS zone_name, COUNT(i) AS pannelli_count
6 RETURN zone_name, pannelli_count
```

Figura 67: QUERY 2: Struttura

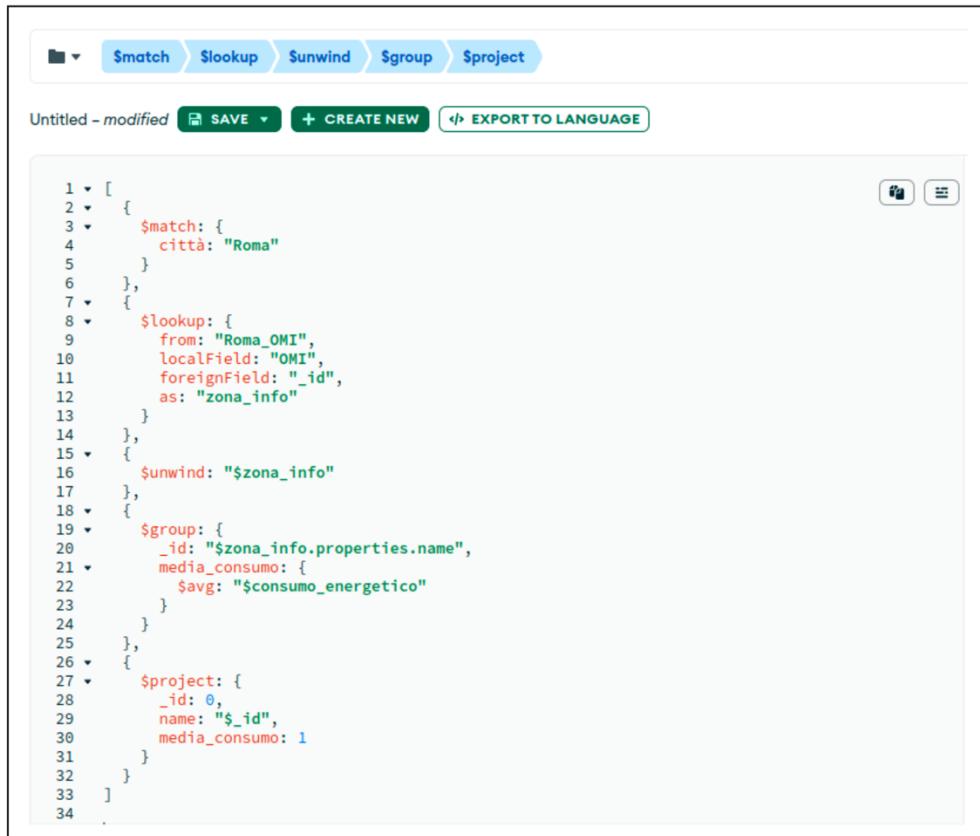
Cypher 5										
Planner COST										
Runtime PIPELINED										
Runtime version 5.25										
Batch size 1024										
+ProduceResults	0	zone_name, pannelli_count				117	17	0	0	0/0
+EagerAggregation	1	polygon.name AS zone_name, COUNT(i) AS pannelli_count				117	17	6180	4120	
+ProcedureCall	2	spatial.intersects(\$autostring_1, indirizzo_point) :: (polygon :: NODE)				13645	3994		0	
+Projection	3	point({latitude: lat, longitude: lon}) AS indirizzo_point				1365	3257			
+Projection	4	cache[i.latitude] AS lat, cache[i.longitude] AS lon				1365	3257			
+CacheProperties	5	cache[i.latitude], cache[i.longitude]				1365	3257	6510		
+Filter	6	NOT ha = r AND p:Pannello				1365	3257	75580		
+Expand(All)	7	(i)-[ha]->(p)				4323	37796	53180		
+Filter	8	i:indirizzo				1986	15388	36810		
+Expand(All)	9	(c)-[r]->(i)				1986	15407	15404		
+Filter	10	c.name = \$autostring_0				1	1	6		
+NodeByLabelScan	11	c:città				10	2		376	257989/0 7776,401 Fused in Pipeline 0
Total database accesses: 107606 + ?, total allocated memory: 4592										
17 rows ready to start consuming query after 1376 ms, results consumed after another 7815 ms										

Figura 68: QUERY 2: Esecuzione

# QUERY 3: «Consumi per Zona OMI»

## Obiettivo

La query ha l'obiettivo di calcolare la media del consumo energetico per ciascuna zona OMI (Osservatorio del Mercato Immobiliare) della città di Roma, raggruppando i dati per zona e associandoli al consumo medio registrato nella collezione dei consumi energetici.



```
1 [  
2 {  
3   $match: {  
4     città: "Roma"  
5   },  
6   {  
7     $lookup: {  
8       from: "Roma_OMI",  
9       localField: "OMI",  
10      foreignField: "_id",  
11      as: "zona_info"  
12    },  
13  },  
14  {  
15    $unwind: "$zona_info"  
16  },  
17  {  
18    $group: {  
19      _id: "$zona_info.properties.name",  
20      media_consumo: {  
21        $avg: "$consumo_energetico"  
22      }  
23    }  
24  },  
25  {  
26    $project: {  
27      _id: 0,  
28      name: "$_id",  
29      media_consumo: 1  
30    }  
31  }  
32 ]  
33 .
```

Figura 42: QUERY 3: Struttura

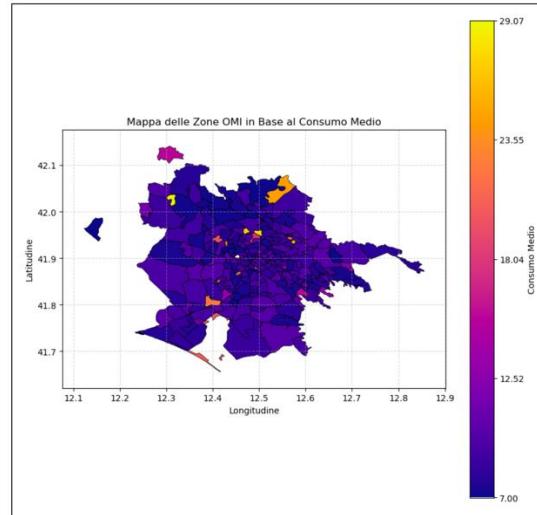
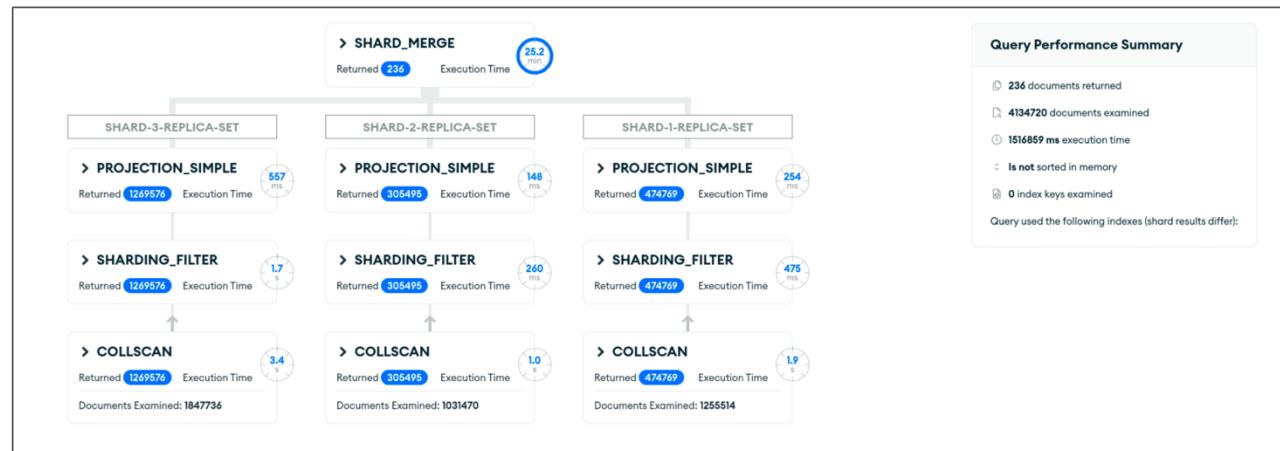


Figura 43: QUERY 3: Risultato



# QUERY 3: «Consumi per Zona OMI»

```
1 MATCH (c:Città)-[:divisa_in]→(z:Feature)-[r:ha_consumo]→()
2 WITH z, AVG(r.consumo_energetico) AS media_consumo
3 RETURN z.name AS zona, media_consumo
```

Figura 69: QUERY 3: Struttura

Cypher 5										
Planner COST										
Runtime PIPELINED										
Runtime version 5.25										
Batch size 1024										
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	Operator   Id   Details     Estimated Rows   Rows   DB Hits   Memory (Bytes)   Page Cache Hits/Misses   Time (ms)   Pipeline	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	+ProduceResults   0   zona, media_consumo     675   52   0   0   0/0   1,900	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	+Projection   1   z.name AS zona     675   52   104     11/0   1,675   In Pipeline 1	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	+EagerAggregation   2   z, AVG(r.consumo_energetico) AS media_consumo     675   52   455503   11384	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	+Expand(All)   3   (z)-[r:ha_consumo]→(anon_1)     455503   455503   455556	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	+Filter   4   z:Feature     53   53   994	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	+Expand(All)   5   (c)-[anon_0:divisa_in]→(z)     53   53   55	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+	+NodeByLabelScan   6   c:Città     10   2   3   376   3623/0   385,320   Fused in Pipeline 0	+-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+ +-----+								
Total database accesses: 912215, total allocated memory: 11840										
52 rows										
ready to start consuming query after 688 ms, results consumed after another 355 ms										

Figura 70: QUERY 3: Esecuzione

# QUERY 4: «Inquinanti per Zona OMI»

## Obiettivo

La query ha l'obiettivo di recuperare i dati relativi agli inquinanti atmosferici presenti nella città di Avellino, associandoli alle zone OMI (Osservatorio del Mercato Immobiliare). Inoltre, calcola la media dei valori degli inquinanti per ciascuna zona OMI, raccogliendo anche informazioni sulle performance della query, come il tempo di esecuzione e il numero di documenti esaminati, tramite l'operazione `explain`.

```
# Iteriamo attraverso tutti gli inquinanti
for inquinante in inquinanti:
    # Controlliamo in quale zona OMI si trova l'inquinante
    omi_zones = db.Avellino_OMI.find({
        "geometry": {
            "$geoIntersects": {
                "$geometry": inquinante["geometry"]
            }
        }
    })

    # Per ogni zona OMI trovata, aggiungiamo il valore dell'inquinante (e sommiamo i valori)
    for omi_zone in omi_zones:
        zona_name = omi_zone["properties"]["name"]
        if zona_name not in zona_inquinanti:
            zona_inquinanti[zona_name] = {} # Dizionario per contenere gli inquinanti per zona

        # Ottieni il valore dell'inquinante e aggiorna il dizionario
        inquinante_name = inquinante["properties"]["inquinante"]
        valore_inquinante = inquinante["properties"].get("valore", 0)

        if inquinante_name not in zona_inquinanti[zona_name]:
            zona_inquinanti[zona_name][inquinante_name] = []

        zona_inquinanti[zona_name][inquinante_name].append(valore_inquinante)
```

Figura 45: QUERY 4: Struttura

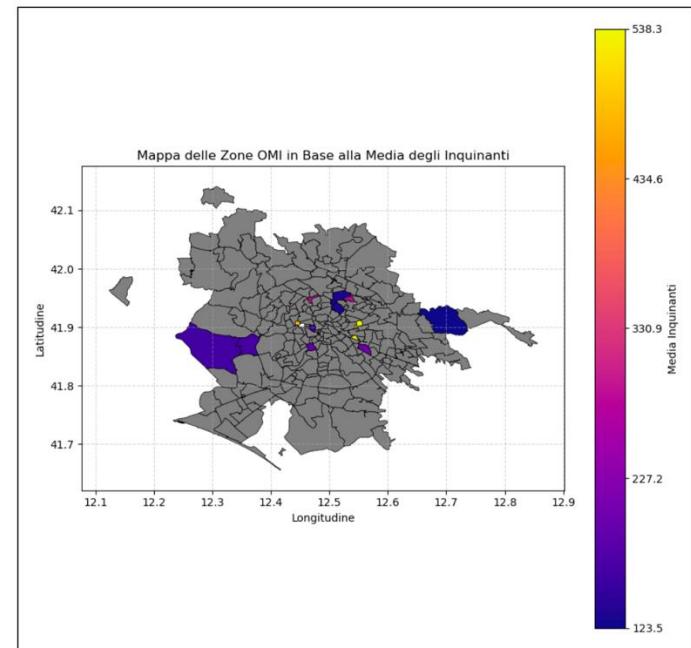


Figura 46: QUERY 4: Risultato

```
Totale documenti esaminati: 99820
Totale tempo di esecuzione (ms): 20335
```

Figura 47: QUERY 4: Esecuzione

# QUERY 4: «Inquinanti per Zona OMI»

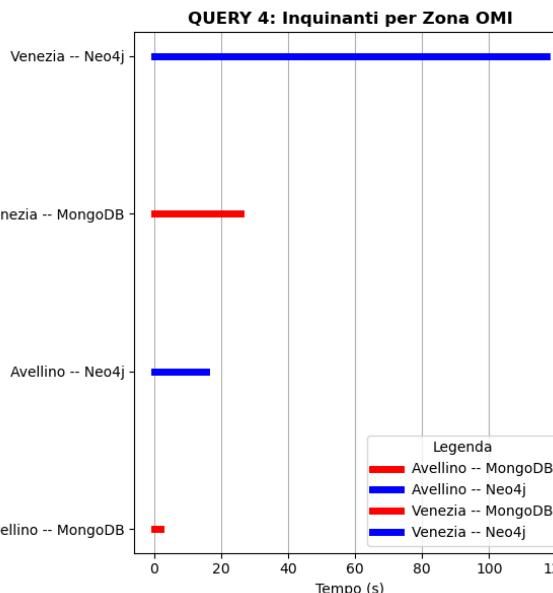
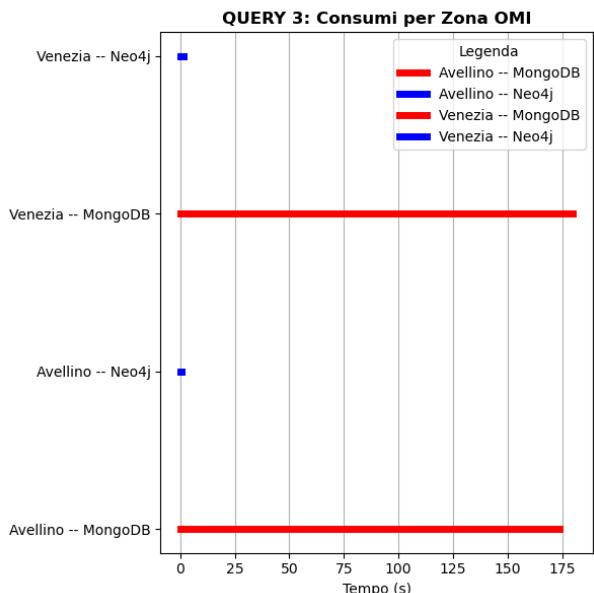
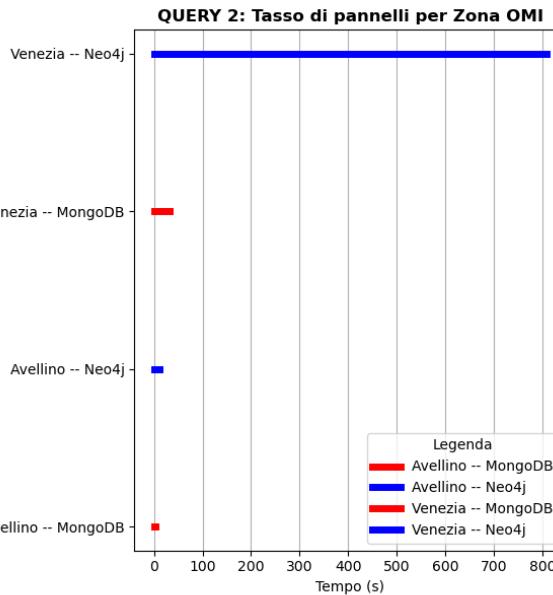
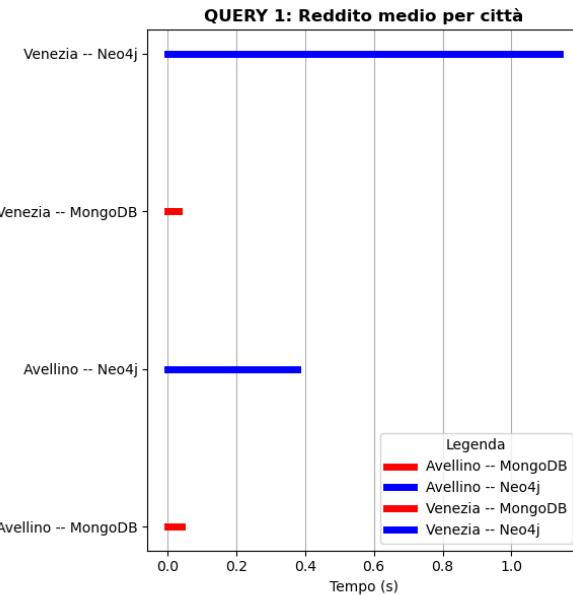
```
1 MATCH (c:Città)-[:ha_stazione]-(s:StazioneMonitoraggio)-[r:ha_monitorato]-(i:Inquinante)
2 WITH s, i, r.valore AS valore_inquinante, s.name AS station_name, s.latitude AS latitude, s.longitude AS longitude
3 WITH point({latitude: latitude, longitude: longitude}) AS stazione_coord, s, i, valore_inquinante
4 CALL spatial.intersects('zone_omi', stazione_coord) YIELD node
5 WITH node.name AS zona_name, i.name AS inquinante_name, valore_inquinante
6 WITH zona_name, inquinante_name, collect(valore_inquinante) AS valori_inquinante
7 UNWIND valori_inquinante AS valore
8 RETURN zona_name, inquinante_name, avg(valore) AS media_inquinante
```

Figura 71: QUERY 4: Struttura

Cypher 5		Planner COST							Runtime PIPELINED			Runtime version 5.25		Batch size 1024	
Operator	Id	Details			Estimated Rows		Rows	DB Hits	Memory (Bytes)	Page Cache	Hits/Misses	Time (ms)	Pipeline		
+ProduceResults	0	zona_name, inquinante_name, media_inquinante			68		41	0	0	0/0		1,111	In Pipeline 3		
+EagerAggregation	1	1   zona_name, inquinante_name, avg(valore) AS media_inquinante			68		41	0	10896						
+Unwind	2	2   valori_inquinante AS valore			4600		30235	0	893746	0/0		18,770	Fused in Pipeline 2		
+EagerAggregation	3	3   zona_name, inquinante_name, collect(valore_inquinante) AS valori_inquinante			468		41	0	871736						
+Projection	4	4   node.name AS zona_name, cache[i.name] AS inquinante_name			211645		30235	66470							
+ProcedureCall	5	5   spatial.intersects(\$autoString, \$stazione_coord) :- (node :: NODE)			211645		30235		0						
+Projection	6	6   point({latitude: latitude, longitude: longitude}) AS stazione_coord			21165		30235								
+Projection	7	7   cache[s.latitude] AS latitude, cache[s.longitude] AS longitude, cache[s.name] AS station_name, r.valore AS valore_inquinante			21165		30235	30285							
+CacheProperties	8	8   cache[i.name]			21165		30235	69440							
+Filter	9	9   i:Inquinante			21165		30235	69440							
+Expand(All)	10	10   (s)-[r:ha_monitorato]-(i)			21165		30235	30285							
+CacheProperties	11	11   cache[s.latitude], cache[s.longitude], cache[s.name]			7		7	70							
+Filter	12	12   c:città			7		7	42							
+Expand(All)	13	13   (s)-[anon_0:ha_stazione]-(c)			7		7	1							
+NodeByLabelScan	14	14   s:StazioneMonitoraggio			10		7	1	376	2133914/0	82706,995	Fused in Pipeline 6			
Total database accesses: 241880 + 7, total allocated memory: 911132															
41 rows ready to start consuming query after 997 ms, results consumed after another 82766 ms															

Figura 72: QUERY 4: Esecuzione

# CONFRONTO PRESTAZIONALE: «MongoDB vs Neo4j»



In breve, mentre MongoDB offre ottime prestazioni su query semplici e operazioni che non richiedono la gestione di complesse relazioni tra i dati, Neo4j eccelle in scenari complessi in cui le relazioni tra le entità sono un aspetto centrale, riducendo drasticamente il tempo necessario per operazioni che in un database documentale richiederebbero passaggi multipli e costosi.

# GRAZIE PER L'ATTENZIONE

