



Getting Graphic: using Neo4j with Spring Data

Mark Heckler
Spring Developer Advocate, Pivotal
@mkheck

Jennifer Reif
Developer Relations Engineer, Neo4j
@jmhreif

Who is Mark?

- Author
- Speaker
- Architect & Developer
- Java Champion
- Seeker of a better way



Who is Jennifer?

- Neo4j Developer Relations Engineer
- Continuous learner
- Conference speaker
- Blogger





What is a graph database?

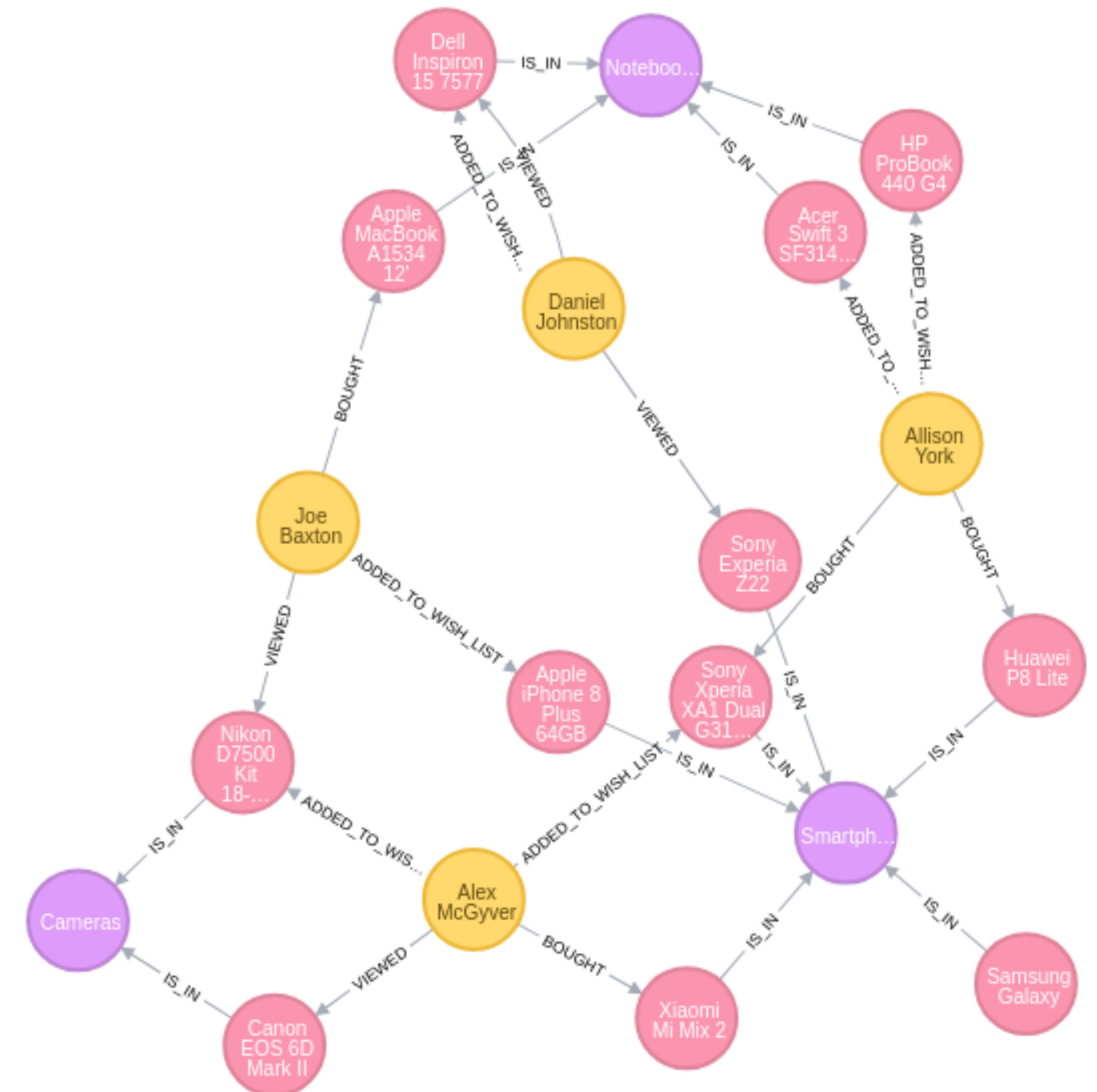
The world is a graph - everything is connected

- People, places, events
- Companies, markets, industries
- Countries, history, politics
- Criminals, fraudsters, divergent behavior
- Sciences, art, education
- Technology, networks, devices, apps, users
- Software, code, dependencies, architecture, deployments

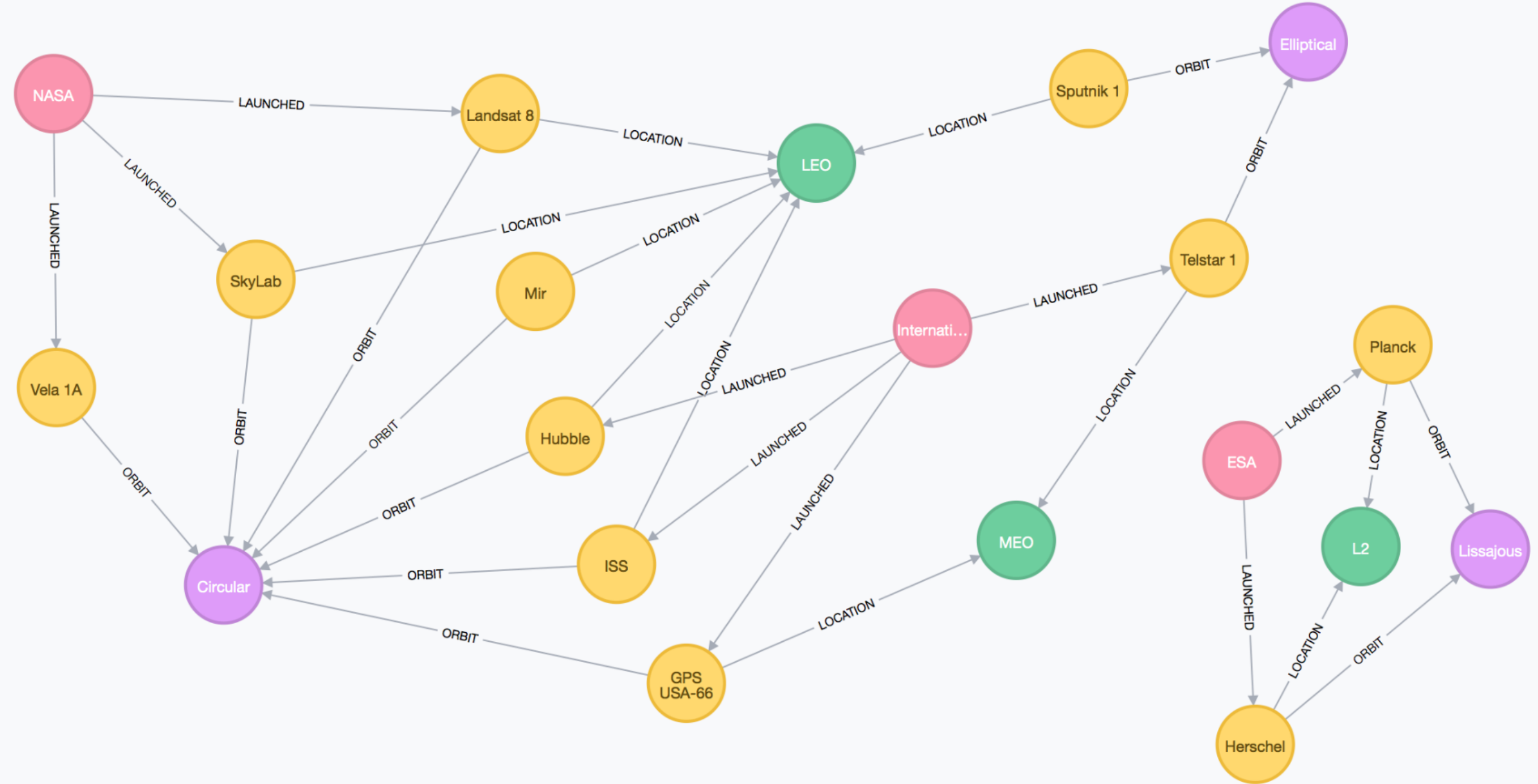
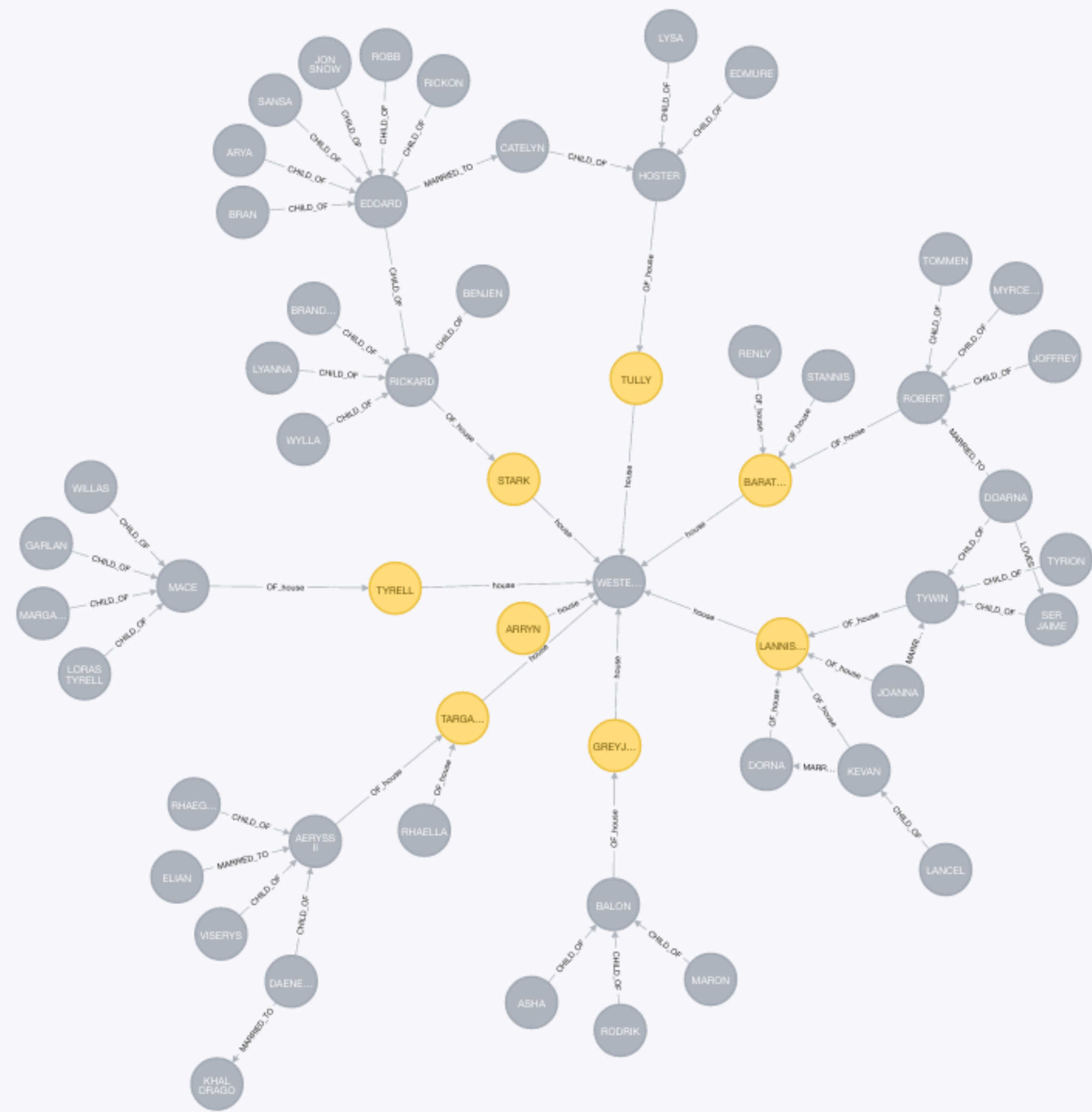


It's all about relationships!

- A graph database shows how entities/objects in our data relate to one another
- Property Graph Data Model components:
 - Nodes
 - Relationships
 - Labels
 - Properties



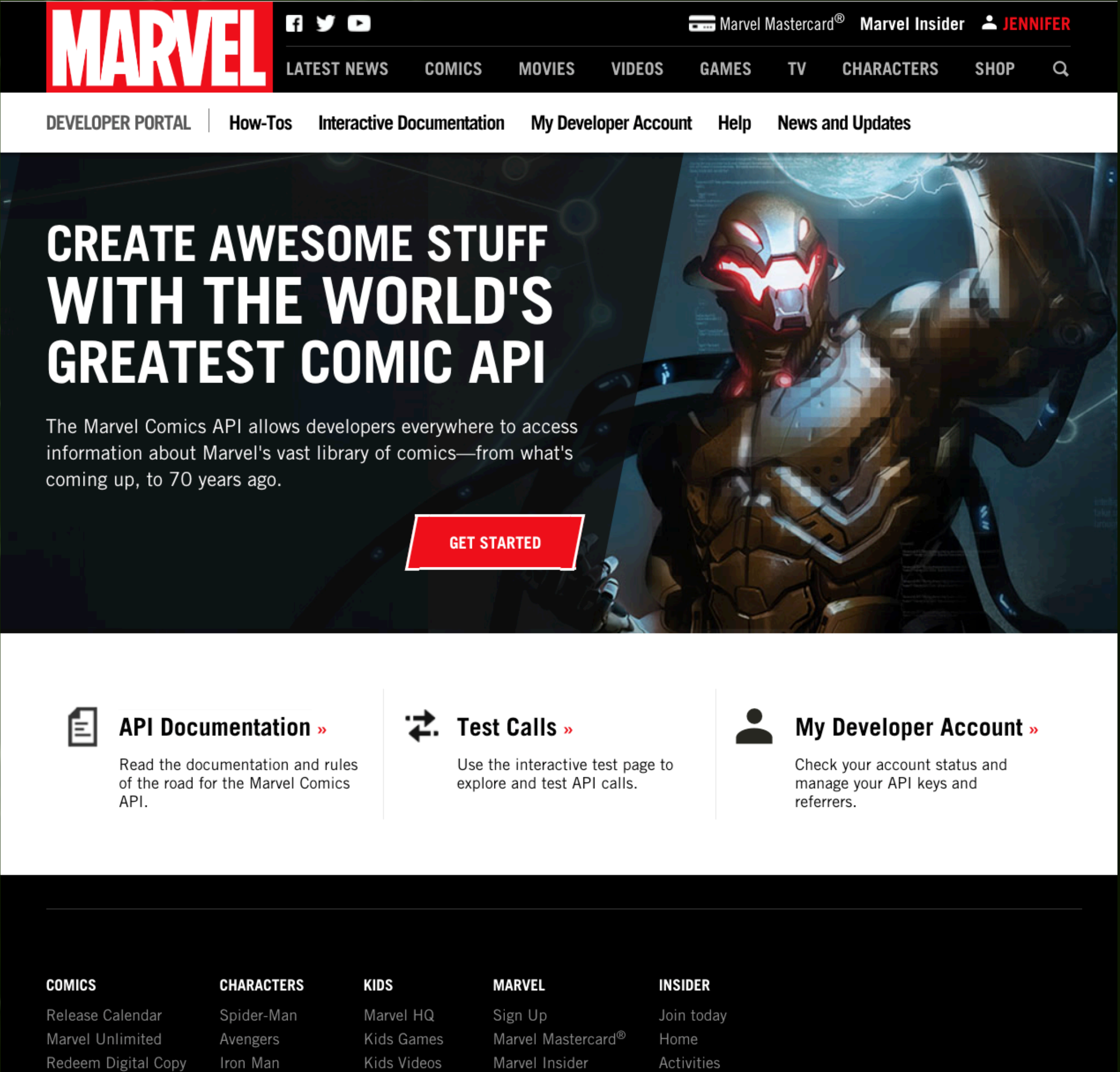
Game of Thrones Family Tree



NASA space satellite data

How to load the data

- developer.marvel.com
- Create an account and an API key
- Read the docs
- Determine the data model
- Create Cypher statements
- Import all the things!



The screenshot shows the Marvel Developer Portal homepage. At the top is the Marvel logo and navigation links for social media and various content types. Below this is a secondary navigation bar for developer resources. The main hero section features a large image of Iron Man and a call to action to create stuff with the world's greatest comic API. Below the hero section are three main links: API Documentation, Test Calls, and My Developer Account. The footer contains links organized by category: Comics, Characters, Kids, Marvel, and Insider.

MARVEL [f](#) [t](#) [v](#) [Marvel Mastercard®](#) [Marvel Insider](#) [JENNIFER](#)


[LATEST NEWS](#) [COMICS](#) [MOVIES](#) [VIDEOS](#) [GAMES](#) [TV](#) [CHARACTERS](#) [SHOP](#) [Q](#)


[DEVELOPER PORTAL](#) | [How-Tos](#) [Interactive Documentation](#) [My Developer Account](#) [Help](#) [News and Updates](#)


CREATE AWESOME STUFF WITH THE WORLD'S GREATEST COMIC API

The Marvel Comics API allows developers everywhere to access information about Marvel's vast library of comics—from what's coming up, to 70 years ago.

[GET STARTED](#)

 **API Documentation »**
Read the documentation and rules of the road for the Marvel Comics API.

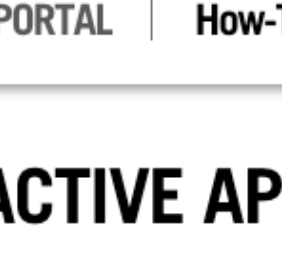
 **Test Calls »**
Use the interactive test page to explore and test API calls.




 **My Developer Account »**
Check your account status and manage your API keys and referrers.


COMICS	CHARACTERS	KIDS	MARVEL	INSIDER
Release Calendar	Spider-Man	Marvel HQ	Sign Up	Join today
Marvel Unlimited	Avengers	Kids Games	Marvel Mastercard®	Home
Redeem Digital Copy	Iron Man	Kids Videos	Marvel Insider	Activities

Tools

- **Marvel Interactive Documentation**
- **Neo4j Desktop/Neo4j Browser**
- **Cypher query language**
- **APOC standard library**
- **My brain! :)**





Marvel Mastercard®

Marvel Insider

JENNIFER

LATEST NEWS

COMICS

MOVIES

VIDEOS

GAMES

TV

CHARACTERS

SHOP

Q

DEVELOPER PORTAL

How-Tos

Interactive Documentation

My Developer Account


Help

News and Updates

INTERACTIVE API TESTER

The panel below displays documentation all endpoints, parameters and error messages available to the Marvel API. For a more detailed explanation of API structure, please read the full [documentation](#).

If you have an API key, you can also test API calls directly from this panel. Just log-in to your Marvel account and your key will be pre-filled. (If you don't have a key, [get one now](#).)



public :

Show/Hide

List Operations

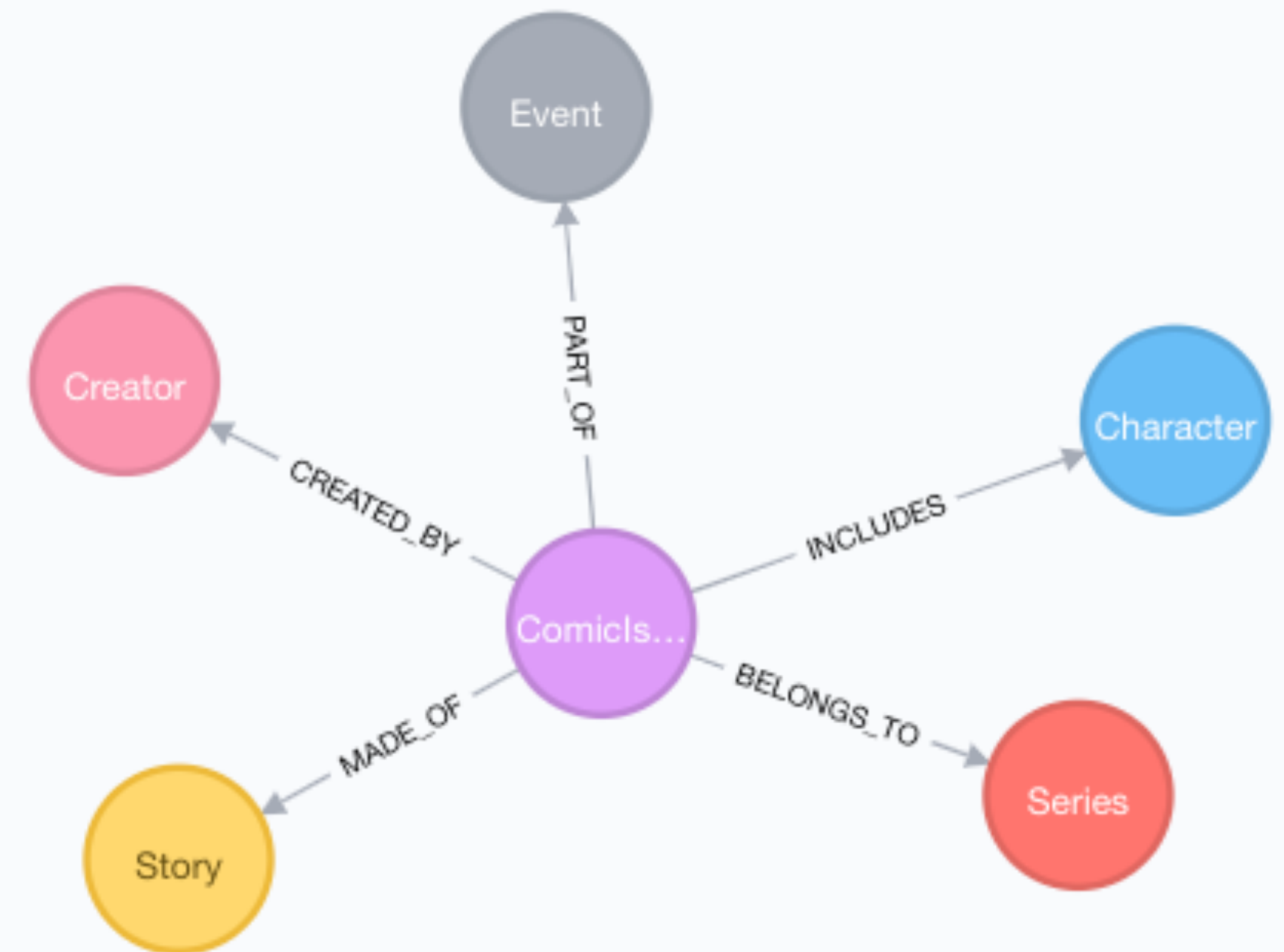
Expand Operations

Raw

GET	/v1/public/characters	Fetches lists of characters.
GET	/v1/public/characters/{characterId}	Fetches a single character by id.
GET	/v1/public/characters/{characterId}/comics	Fetches lists of comics filtered by a character id.
GET	/v1/public/characters/{characterId}/events	Fetches lists of events filtered by a character id.
GET	/v1/public/characters/{characterId}/series	Fetches lists of series filtered by a character id.
GET	/v1/public/characters/{characterId}/stories	Fetches lists of stories filtered by a character id.
GET	/v1/public/comics	Fetches lists of comics.
GET	/v1/public/comics/{comicId}	Fetches a single comic by id.
GET	/v1/public/comics/{comicId}/characters	Fetches lists of characters filtered by a comic id.
GET	/v1/public/comics/{comicId}/creators	Fetches lists of creators filtered by a comic id.
GET	/v1/public/comics/{comicId}/events	Fetches lists of events filtered by a comic id.

Data model & choices

- Marvel comics are complex
- Can't import it all, choose which information is important
- This data is retrieved as JSON (probably stored relationally), so we must create the graph model version
- Certain end points only give so much data, must make later calls to hydrate additional fields
- Some of the data is missing or messy



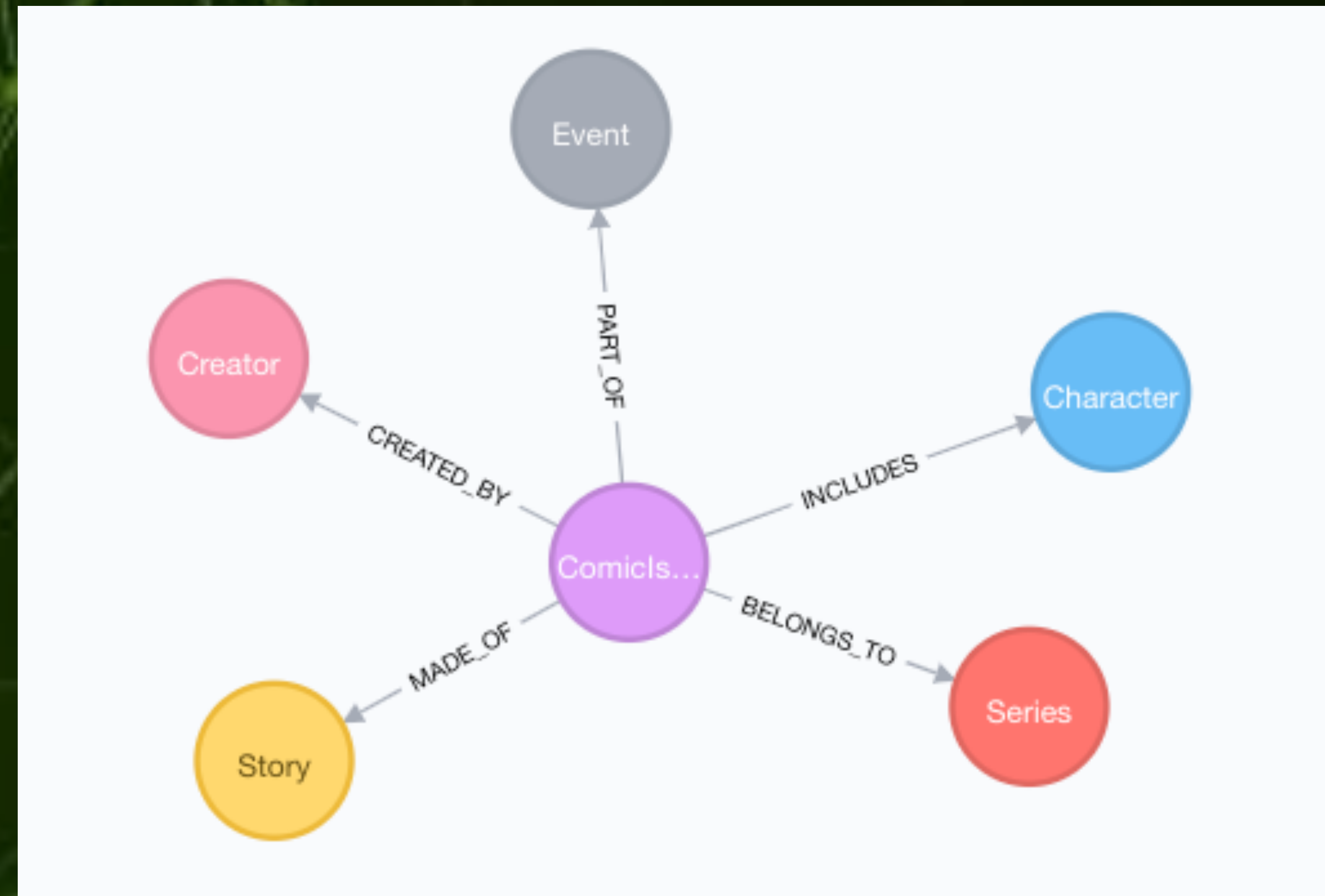
Other things to consider

- **Marvel doesn't fully support or maintain the system**
 - Some data is null or missing
 - Some servers randomly collapse mid-call
- **Limits on number of results you can return**
 - only up to 100!
- **Limits on the number of calls in 1 day**
 - Each user allowed 3,000 calls per day

**All about....
maximum information
within 100 results
and under 3,000 call limit!**

Importing sections

- **Set up parameters, indexes, and constraints**
 - **Parameters:** API keys, url strings
 - **Constraints:** disallow duplicate data on IDs
 - **Indexes:** create fast lookups for common query properties (like Character name and resource URIs)
- **Find the smallest entry point (trial-and-error)**
 - **Characters:** segment by letters of the alphabet (only 26 calls)
 - **# of character:** each letter of the alphabet had around 100 characters




```
//Load chunks of Characters by name starting with each letter of the alphabet
WITH apoc.date.format(timestamp(), "ms", 'yyyyMMddHHmmss') AS ts
WITH "&ts=" + ts + "&apikey=" + $marvel_public + "&hash=" + apoc.util.md5([ts,$marvel_private,$marvel_public]) as
suffix
CALL apoc.periodic.iterate('UNWIND split("ABCDEFGHIJKLMNOPQRSTUVWXYZ","") AS letter RETURN letter',
'CALL apoc.load.json("http://gateway.marvel.com/v1/public/characters?
nameStartsWith="+letter+"&orderBy=name&limit=100"+$suffix) YIELD value
UNWIND value.data.results as results
WITH results, results.comics.available AS comics
WHERE comics > 0
MERGE (char:Character {id: results.id})
  ON CREATE SET char.name = results.name, char.description = results.description, char.thumbnail =
results.thumbnail.path+"."+results.thumbnail.extension,
  char.resourceURI = results.resourceURI',
{batchSize: 1, iterateList:false, params:{suffix:suffix}})
```


//Load chunks of Characters by name starting with each letter of the alphabet

WITH **apoc.date.format**(timestamp(), "ms", 'yyyyMMddHHmmss') **AS** ts

WITH "&ts=" + ts + "&apikey=" + \$marvel_public + "&hash=" + apoc.util.md5([ts,\$marvel_private,\$marvel_public])
as suffix

CALL **apoc.periodic.iterate**('UNWIND split("ABCDEFGHIJKLMNOPQRSTUVWXYZ", "") **AS** letter **RETURN** letter',

'**CALL** **apoc.load.json**("http://gateway.marvel.com/v1/public/characters?

nameStartsWith="+letter+"&orderBy=name&limit=100"+\$suffix) **YIELD** value

UNWIND value.data.results **as** results

WITH results, results.comics.available **AS** comics

WHERE comics > 0

MERGE (char:Character {id: results.id})

ON CREATE SET char.name = results.name, char.description = results.description, char.thumbnail =

results.thumbnail.path+"."+results.thumbnail.extension,

char.resourceURI = results.resourceURI',

{batchSize: 1, iterateList:false, params:{suffix:suffix}})

//Load chunks of Characters by name starting with each letter of the alphabet
WITH **apoc.date.format**(timestamp(), "ms", 'yyyyMMddHHmmss') AS ts
WITH "&ts=" + ts + "&apikey=" + \$marvel_public + "&hash=" + apoc.util.md5([ts,\$marvel_private,\$marvel_public]) as
suffix

CALL apoc.periodic.iterate('UNWIND split("ABCDEFGHIJKLMNOPQRSTUVWXYZ","") AS letter RETURN letter',
'CALL **apoc.load.json**("http://gateway.marvel.com/v1/public/characters?
nameStartsWith="+letter+"&orderBy=name&limit=100"+\$suffix) YIELD value

UNWIND value.data.results as results

WITH results, results.comics.available AS comics

WHERE comics > 0

MERGE (char:Character {id: results.id})

ON CREATE SET char.name = results.name, char.description = results.description, char.thumbnail =
results.thumbnail.path+"."+results.thumbnail.extension,

char.resourceURI = results.resourceURI',
{batchSize: 1, iterateList:false, params:{suffix:suffix}})


```
//Load chunks of Characters by name starting with each letter of the alphabet
WITH apoc.date.format(timestamp(), "ms", 'yyyyMMddHHmmss') AS ts
WITH "&ts=" + ts + "&apikey=" + $marvel_public + "&hash=" + apoc.util.md5([ts,$marvel_private,$marvel_public]) as
suffix
CALL apoc.periodic.iterate('UNWIND split("ABCDEFGHIJKLMNOPQRSTUVWXYZ","") AS letter RETURN letter',
'CALL apoc.load.json("http://gateway.marvel.com/v1/public/characters?
nameStartsWith="+letter+"&orderBy=name&limit=100"+$suffix) YIELD value
```

```
UNWIND value.data.results as results
```

```
WITH results, results.comics.available AS comics
```

```
WHERE comics > 0
```

```
MERGE (char:Character {id: results.id})
```

```
ON CREATE SET char.name = results.name, char.description = results.description, char.thumbnail =  
results.thumbnail.path+"."+results.thumbnail.extension,  
char.resourceURI = results.resourceURI',
```

```
{batchSize: 1, iterateList:false, params:{suffix:suffix}})
```



```
//For Characters just loaded, load all related ComicIssue, Series, Story, Event, and Creator objects.
WITH ....
CALL apoc.periodic.iterate('MATCH (c:Character) WHERE c.resourceURI IS NOT NULL AND NOT exists((c)-[:INCLUDES]-()) RETURN c LIMIT 100',
'CALL apoc.util.sleep(2000)
CALL apoc.load.json(c.resourceURI+"/comics?format=comic&formatType=comic&limit=100"+$suffix)
YIELD value WITH c, value.data.results as results WHERE results IS NOT NULL
UNWIND results as result
MERGE (comic:ComicIssue {id: result.id})
  ON CREATE SET comic.name = result.title, comic.issueNumber = result.issueNumber, comic.pageCount =
result.pageCount, comic.resourceURI = result.resourceURI, comic.thumbnail =
result.thumbnail.path+"."+result.thumbnail.extension
WITH c, comic, result
MERGE (comic)-[r:INCLUDES]->(c)
WITH c, comic, result WHERE result.series IS NOT NULL
UNWIND result.series as comicSeries
MERGE (series:Series {id: toInt(split(comicSeries.resourceURI,"/")[ -1])})
  ON CREATE SET series.name = comicSeries.name, series.resourceURI = comicSeries.resourceURI
WITH c, comic, series, result
MERGE (comic)-[r2:BELONGS_TO]->(series)
...<Creator>...
...<Story>...
...<Event>...',
{batchSize: 20, iterateList:false, retries:2, params:{suffix:suffix}})
```


//For Characters just loaded, load all related ComicIssue, Series, Story, Event, and Creator objects.
WITH

```
CALL apoc.periodic.iterate('MATCH (c:Character) WHERE c.resourceURI IS NOT NULL AND NOT exists((c)-[:INCLUDES]-()) RETURN c LIMIT 100',
```

```
'CALL apoc.util.sleep(2000)
```

```
CALL apoc.load.json(c.resourceURI+"/comics?format=comic&formatType=comic&limit=100"+$suffix)
```

```
YIELD value WITH c, value.data.results as results WHERE results IS NOT NULL
```

```
UNWIND results as result
```

```
MERGE (comic:ComicIssue {id: result.id})
```

```
ON CREATE SET comic.name = result.title, comic.issueNumber = result.issueNumber, comic.pageCount =  
result.pageCount, comic.resourceURI = result.resourceURI, comic.thumbnail =  
result.thumbnail.path+"."+result.thumbnail.extension
```

```
WITH c, comic, result
```

```
MERGE (comic)-[r:INCLUDES]->(c)
```

```
WITH c, comic, result WHERE result.series IS NOT NULL
```

```
UNWIND result.series as comicSeries
```

```
MERGE (series:Series {id: toInt(split(comicSeries.resourceURI,"/")[1])})
```

```
ON CREATE SET series.name = comicSeries.name, series.resourceURI = comicSeries.resourceURI
```

```
WITH c, comic, series, result
```

```
MERGE (comic)-[r2:BELONGS_TO]->(series)
```

```
...<Creator>...
```

```
...<Story>...
```

```
...<Event>...',
```

```
{batchSize: 20, iterateList:false, retries:2, params:{suffix:suffix}})
```



```
//For Characters just loaded, load all related ComicIssue, Series, Story, Event, and Creator objects.
WITH ....
CALL apoc.periodic.iterate('MATCH (c:Character) WHERE c.resourceURI IS NOT NULL AND NOT exists((c)<-[:INCLUDES]-()) RETURN c LIMIT 100',
'CALL apoc.util.sleep(2000)
CALL apoc.load.json(c.resourceURI+"/comics?format=comic&formatType=comic&limit=100"+$suffix)
YIELD value WITH c, value.data.results as results WHERE results IS NOT NULL
UNWIND results as result MERGE (comic:ComicIssue {id: result.id})
  ON CREATE SET comic.name = result.title, comic.issueNumber = result.issueNumber, comic.pageCount =
result.pageCount, comic.resourceURI = result.resourceURI, comic.thumbnail =
result.thumbnail.path+"."+result.thumbnail.extension
WITH c, comic, result
MERGE (comic)-[r:INCLUDES]->(c)
WITH c, comic, result WHERE result.series IS NOT NULL
UNWIND result.series as comicSeries
MERGE (series:Series {id: toInt(split(comicSeries.resourceURI,"/")[-1])})
  ON CREATE SET series.name = comicSeries.name, series.resourceURI = comicSeries.resourceURI
WITH c, comic, series, result
MERGE (comic)-[r2:BELONGS_TO]->(series)
...<Creator>...
...<Story>...
...<Event>...',
{batchSize: 20, iterateList:false, retries:2, params:{suffix:suffix}})
```


//For Series just loaded, hydrate nodes with additional properties
WITH **apoc.date.format**(timestamp(), "ms", 'yyyyMMddHHmmss') AS ts
WITH "&ts=" + ts + "&apikey=" + \$marvel_public + "&hash=" + apoc.util.md5([ts,\$marvel_private,\$marvel_public]) as
suffix

**CALL apoc.periodic.iterate('MATCH (s:Series) WHERE s.resourceURI IS NOT NULL AND not exists(s.startYear)
RETURN s LIMIT 100',**

'CALL apoc.util.sleep(2000)

CALL apoc.load.json(s.resourceURI+"?limit=100" + \$suffix) YIELD value

WITH value.data.results as results WHERE results IS NOT NULL

UNWIND results as result

MERGE (series:Series {id: result.id})

**SET series.startYear = result.startYear, series.endYear = result.endYear, series.rating = result.rating,
series.thumbnail = result.thumbnail.path+"."+result.thumbnail.extension',
{batchSize: 20, iterateList: false, params: {suffix:suffix}})**

//For Events loaded, hydrate nodes with additional properties
WITH **apoc.date.format**(timestamp(), "ms", 'yyyyMMddHHmmss') AS ts
WITH "&ts=" + ts + "&apikey=" + \$marvel_public + "&hash=" + apoc.util.md5([ts,\$marvel_private,\$marvel_public]) as
suffix

CALL **apoc.periodic.iterate('MATCH (event:Event) WHERE event.resourceURI IS NOT NULL AND NOT
exists(event.start) RETURN DISTINCT event LIMIT 100',**

'CALL **apoc.util.sleep**(2000) CALL **apoc.load.json**(event.resourceURI+"?limit=100"+\$suffix) YIELD value
UNWIND value.data.results as result

MERGE (e:Event {id: result.id})

SET e.start = result.start, e.end = result.end', {batchSize: 20, iterateList:false, params: {suffix:suffix}})

PROJECTS : SPRING DATA

Spring Data Neo4j



This project provides Spring Data support for the [Neo4j](#) Graph Database, including annotated POJOs, SD-Repositories and Neo4j-Template.

[QUICK START](#)


Spring Data Neo4j offers advanced features to map annotated entity classes to the Neo4j Graph Database. The template programming model is equivalent to other Spring templates and builds the basis for interaction with the graph and is also used for the Spring Data repository support. Spring Data Neo4j is core part of the Spring Data project which aims to provide convenient data access for NoSQL databases.

Features

- Support for property graphs (nodes connected via relationships, each with arbitrary properties)
- Object-Graph-Mapping of annotated POJO entities
- extensive Spring Data Commons Repositories Support, including annotated and derived finder methods

Spring Data Neo4j

RELEASE

DOCUMENTATION

5.2.0 SNAPSHOT

5.1.1 SNAPSHOT

5.1.0 CURRENT GA

[Reference](#) | [API](#)

5.0.11 SNAPSHOT

5.0.10 GA

[Reference](#) | [API](#)



spring-projects / spring-data-neo4j

Watch ▾

126

★ Star

517

🍴 Fork

521

↔ Code

🔗 Pull requests 1

📁 Projects 0

📖 Wiki

📊 Insights

Provides support to increase developer productivity in Java when using the neo4j graph database. Uses familiar Spring concepts such as a template classes for core API usage and provides an annotation based programming model using AspectJ

<http://www.springsource.org/spring-da...>

📦 628 commits

🌿 27 branches

🏷 126 releases

👤 17 contributors

📄 Apache-2.0

Capabilities

- Full Spring Boot support (obviously!)
- Spring Initializr
- Interface-based repository support
- Persistence lifecycle events
- Annotation-based OGM for POJOs
- Connect to Neo4j with Bolt, HTTP, or embedded

Let's CODE!!!



Helpful Resources

- **Spring Data Neo4j:** <https://spring.io/projects/spring-data-neo4j>
- **Neo4j:** <https://neo4j.com/developer/get-started/>
- **Spring (projects, guides, blog):** <https://spring.io>
- **Source code:** <https://github.com/HecklerReifCollab/sdn-marvelcomics>
- **Contact:**
 - **Twitter/DMs:** @mkheck, @jmhreif
 - **Email:** mheckler@pivotal.io, jennifer.reif@neo4j.com

Thanks for coming!