Kenny Bastani (https://www.kennybastani.com/)

Escribo y hablo sobre la creación de software. También twitteo cosas @kennybastani

Siguenos en Twitter (http://www.twitter.com/kennybastani)

GitHub (http://www.github.com/kbastani)

LinkedIn (http://www.linkedin.com/in/kennybastani)

Análisis de sentimientos en datos de Twitter usando Neo4j y Google Cloud

Jueves 19 de septiembre de 2019

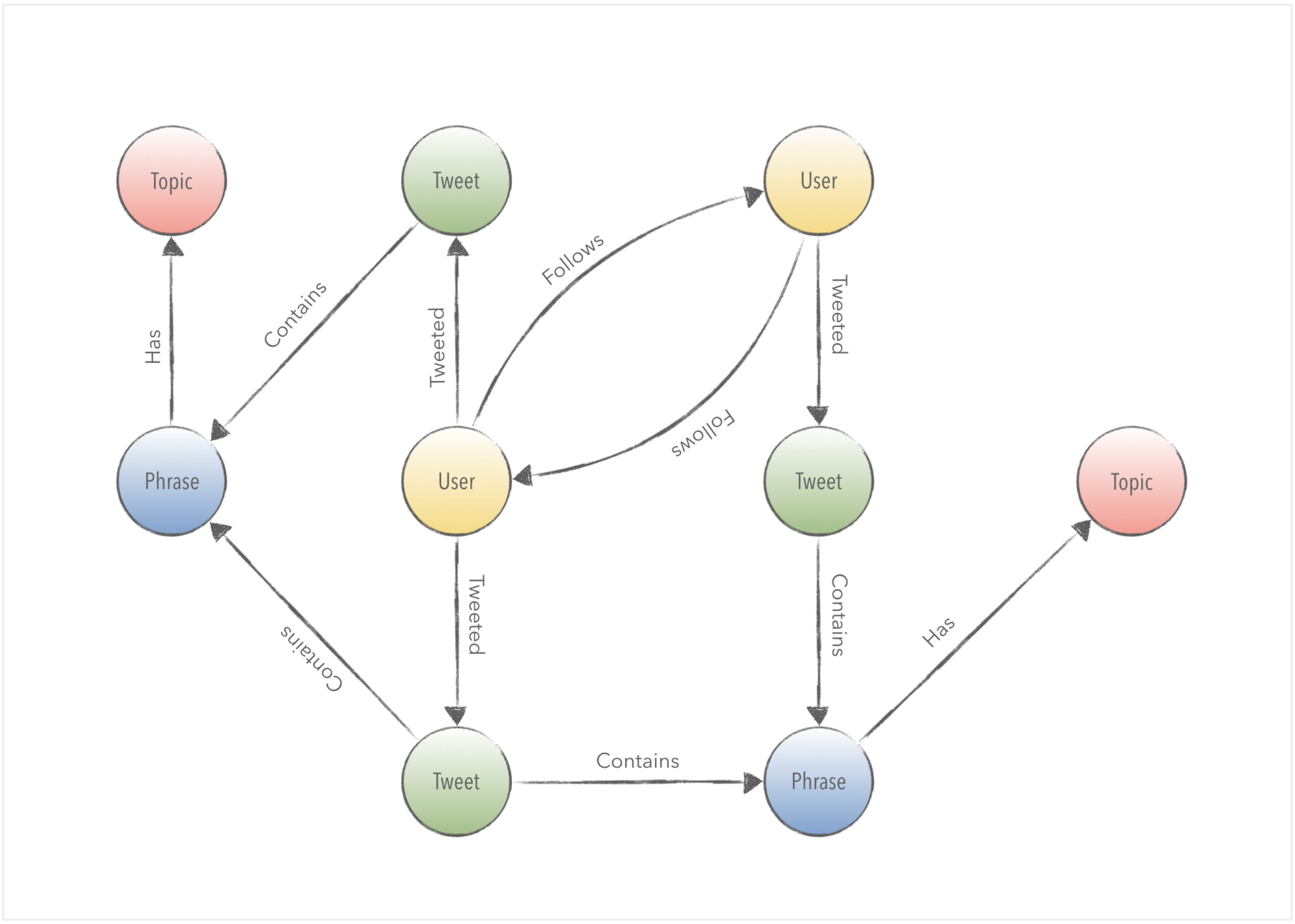
En esta publicación de blog, veremos cómo diseñar un algoritmo de procesamiento de gráficos (https://blog.acolyer.org/2015/05/26/pregel-a-system-for-large-scale-graph-processing/) sobre Neo4j (https://neo4j.com/developer/graph-database/) que descubra la influencia y el sentimiento de los tweets en su red de Twitter.

> El código fuente de esta aplicación de referencia es de código abierto. Puedes encontrar el proyecto GitHub aquí (https://github.com/kbastani/sentiment-analysis-twitter-microservices-example) .

Modelado de datos gráficos

Lo primero que tendremos que hacer es diseñar un modelo de datos para analizar los sentimientos e influencias de los usuarios en Twitter. Este ejemplo itera de un ejemplo de procesamiento gráfico anterior (https://www.kennybastani.com/2016/01/spring-boot-graph-processing-microservices.html) descrito en otra publicación de blog. Recomiendo echar un vistazo a esa publicación para comprender mejor los conceptos de los que hablo en esta.

El siguiente diagrama es el modelo de datos gráficos (https://www.youtube.com/watch?v=3w_ih8_9rVY) que usaremos para importar, analizar y consultar datos de Twitter.

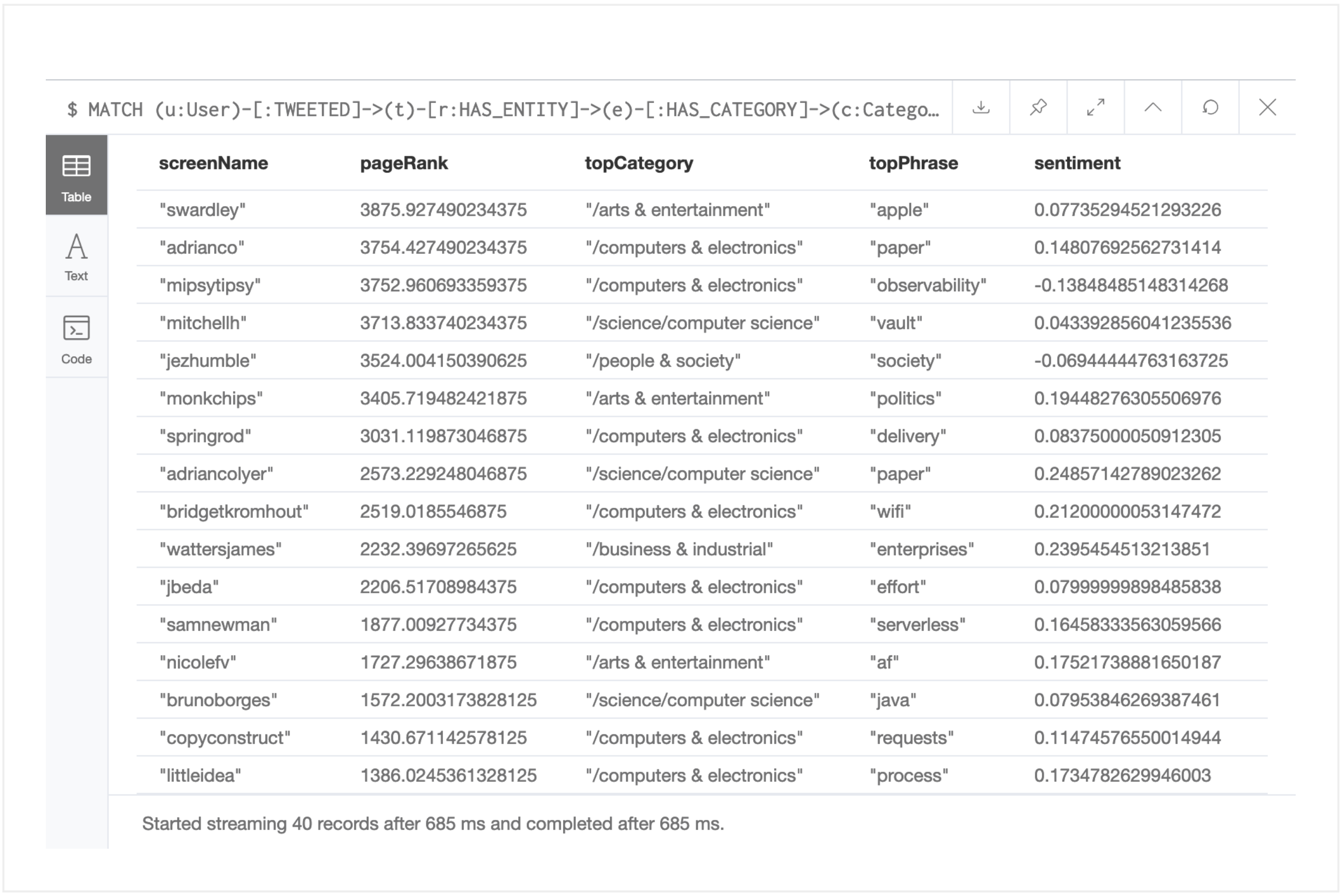


En el diagrama anterior, se describen las siguientes relaciones.

- `Users` seguir a otro `users`
- `Users` crear `tweets`
- `Tweets` Contiene `phrases`
- `Phrases` se clasifican en `topics`

Ranking de usuarios de Twitter

Para esta primera publicación de blog, nos centraremos en generar un rango de usuarios influyentes de Twitter en mi red social que me diga sobre qué temas tuitea un usuario.



```
$ MATCH (u:User)-[:TWEETED]->(t)-[r:HAS_ENTITY]->(e)-[:HAS_CATEGORY]->(c:Catego…
```

screenName	pageRank	topCategory	topPhrase	sentiment
"swardley"	3875.927490234375	"/arts & entertainment"	"apple"	0.07735294521293226
"adrianco"	3754.427490234375	"/computers & electronics"	"paper"	0.14807692562731414
"mipsytipsy"	3752.960693359375	"/computers & electronics"	"observability"	-0.13848485148314268
"mitchellh"	3713.833740234375	"/science/computer science"	"vault"	0.043392856041235536
"jezhumble"	3524.004150390625	"/people & society"	"society"	-0.06944444763163725
"monkchips"	3405.719482421875	"/arts & entertainment"	"politics"	0.19448276305506976
"springrod"	3031.119873046875	"/computers & electronics"	"delivery"	0.08375000050912305
"adriancolyer"	2573.229248046875	"/science/computer science"	"paper"	0.24857142789023262
"bridgetkromhout"	2519.0185546875	"/computers & electronics"	"wifi"	0.21200000053147472
"wattersjames"	2232.39697265625	"/business & industrial"	"enterprises"	0.2395454513213851
"jbeda"	2206.51708984375	"/computers & electronics"	"effort"	0.07999999898485838
"samnewman"	1877.00927734375	"/computers & electronics"	"serverless"	0.16458333563059566
"nicolefv"	1727.29638671875	"/arts & entertainment"	"af"	0.17521738881650187
"brunoborges"	1572.2003173828125	"/science/computer science"	"java"	0.07953846269387461
"copyconstruct"	1430.671142578125	"/computers & electronics"	"requests"	0.11474576550014944
"littleidea"	1386.0245361328125	"/computers & electronics"	"process"	0.1734782629946003

```
Started streaming 40 records after 685 ms and completed after 685 ms.
```

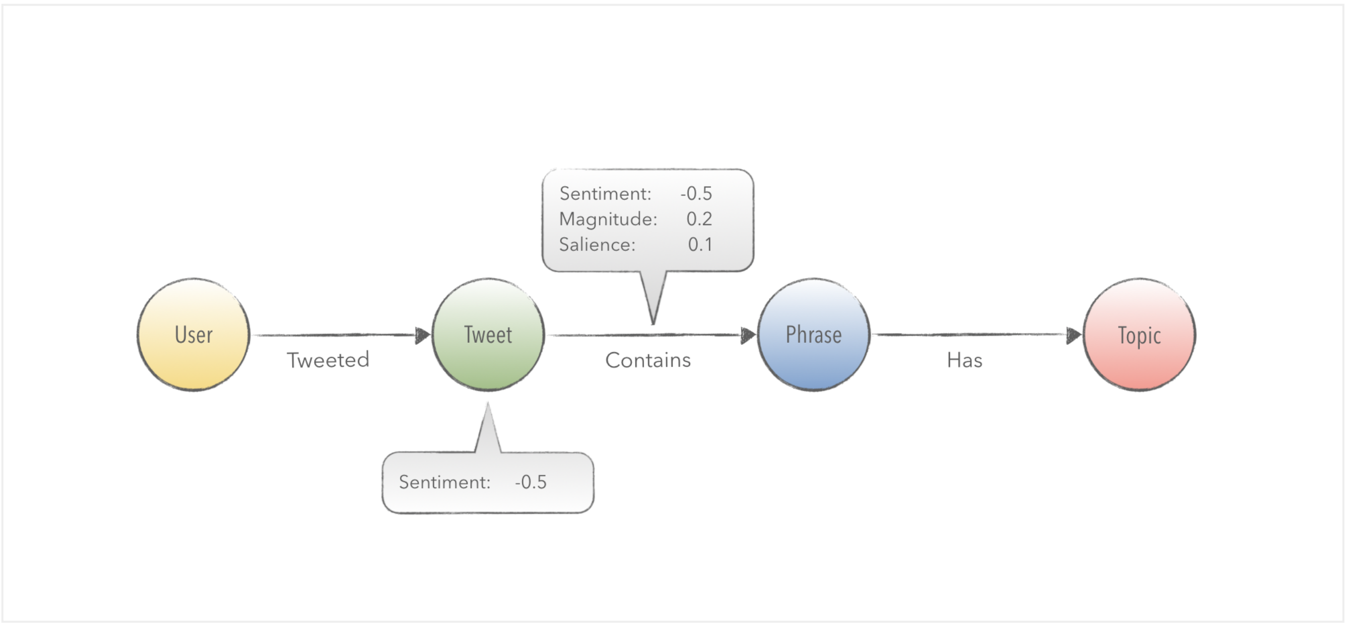
The screenshot above is from the results of a Neo4j cypher query (https://neo4j.com/developer/cypher-query-language/). Here we find a list of Twitter users that were discovered using a crawling algorithm (https://en.wikipedia.org/wiki/Web_crawler) based on PageRank (https://en.wikipedia.org/wiki/PageRank). This output is similar to the dashboard that was created in an earlier blog post, but adds in a top category, top phrase, and a sentiment score.

Let’s figure out how graph processing on Neo4j is used to generate this ranking of users.

Natural Language Processing

We’re going to use sentiment analysis (https://en.wikipedia.org/wiki/Sentiment_analysis) to enhance the graph data model described earlier. We will use Google Cloud’s Natural Language API (https://cloud.google.com/natural-language/) to do this. Every time a user’s tweet is fetched from the Twitter API, its text is submitted to multiple Natural Language API endpoints, which enhances data in our graph data model.

The diagram below is an example pathway between a `User` and a `Topic`.

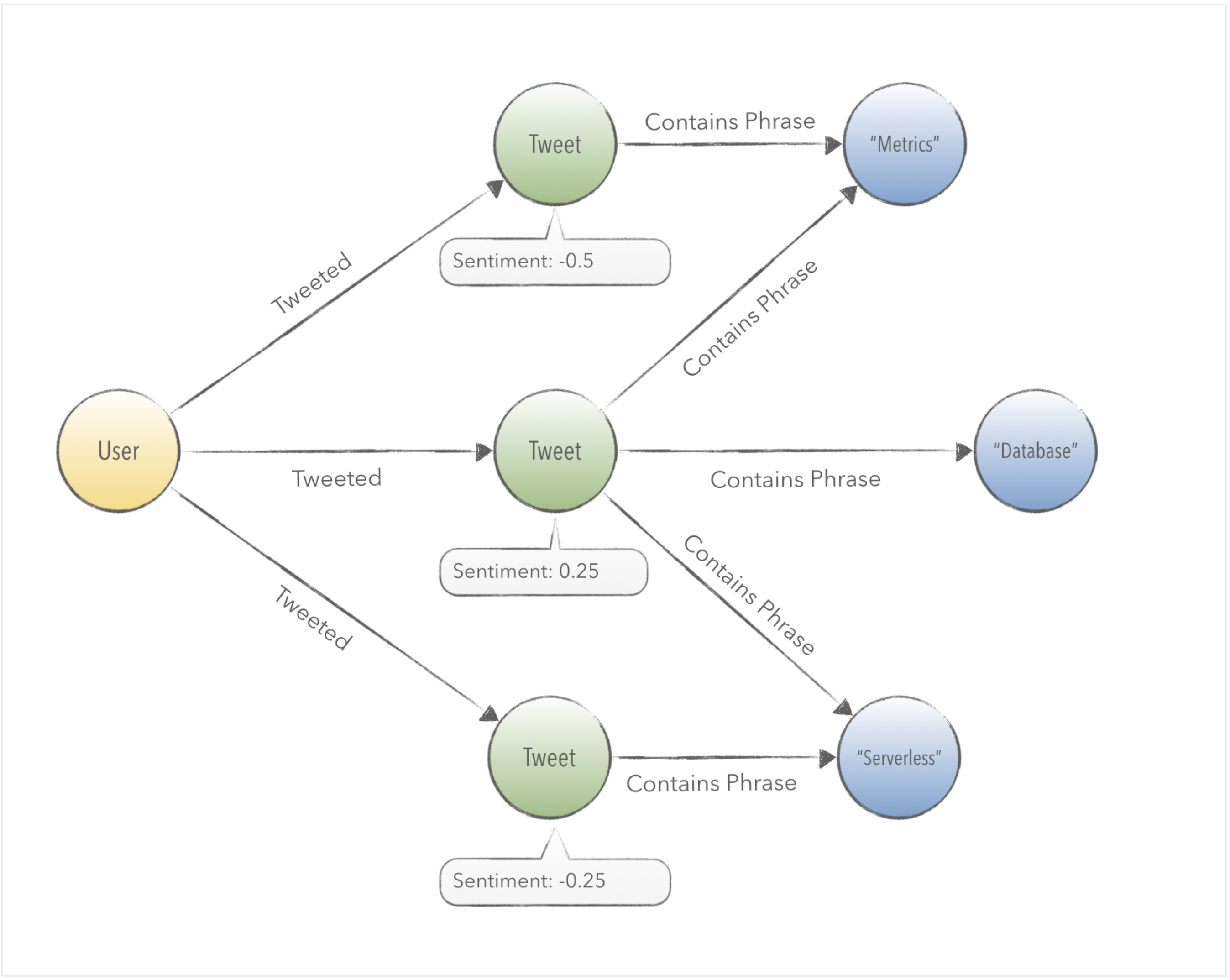


The Twitter crawling algorithm in our application iteratively imports a set of tweets from each ranked user. We'll lean on the Google Cloud's Natural Language API to help us structure the semantic relationships between users, tweets, phrases, and topics.

Sentiment analysis

The first Google Cloud Natural Language API we'll use is the *sentiment analysis* endpoint. The sentiment analysis API endpoint is described in the Google Cloud developer documentation, and is explained below.

> *Sentiment analysis* (https://cloud.google.com/natural-language/docs/analyzing-sentiment) inspects the given text and identifies the prevailing emotional opinion within the text, especially to determine a writer's attitude as positive, negative, or neutral.

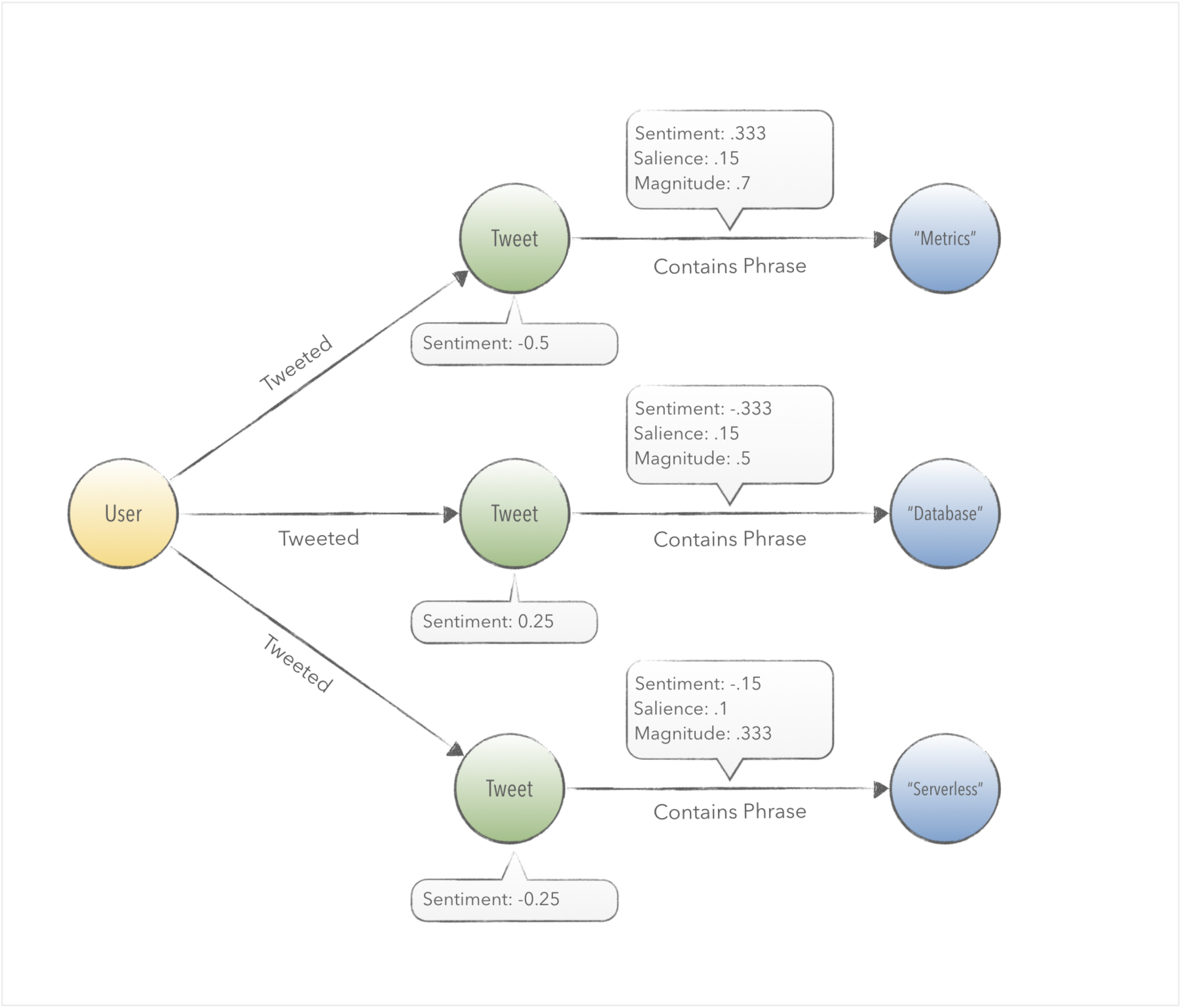


The sentiment analysis endpoint allows us to add a sentiment score on each `Tweet`. The next step is to extract phrases, shown in the diagram above, and to fetch the sentiment score in relation to a tweet's text.

Entity sentiment analysis

Entity sentiment analysis is a Cloud Natural Language endpoint that provides us with a collection of phrases and their sentiment scores in context to a tweet's text.

> *Entity sentiment analysis* (https://cloud.google.com/natural-language/docs/analyzing-entity-sentiment) inspects the given text for known entities (proper nouns and common nouns), returns information about those entities, and identifies the prevailing emotional opinion of the entity within the text, especially to determine a writer's attitude toward the entity as positive, negative, or neutral.



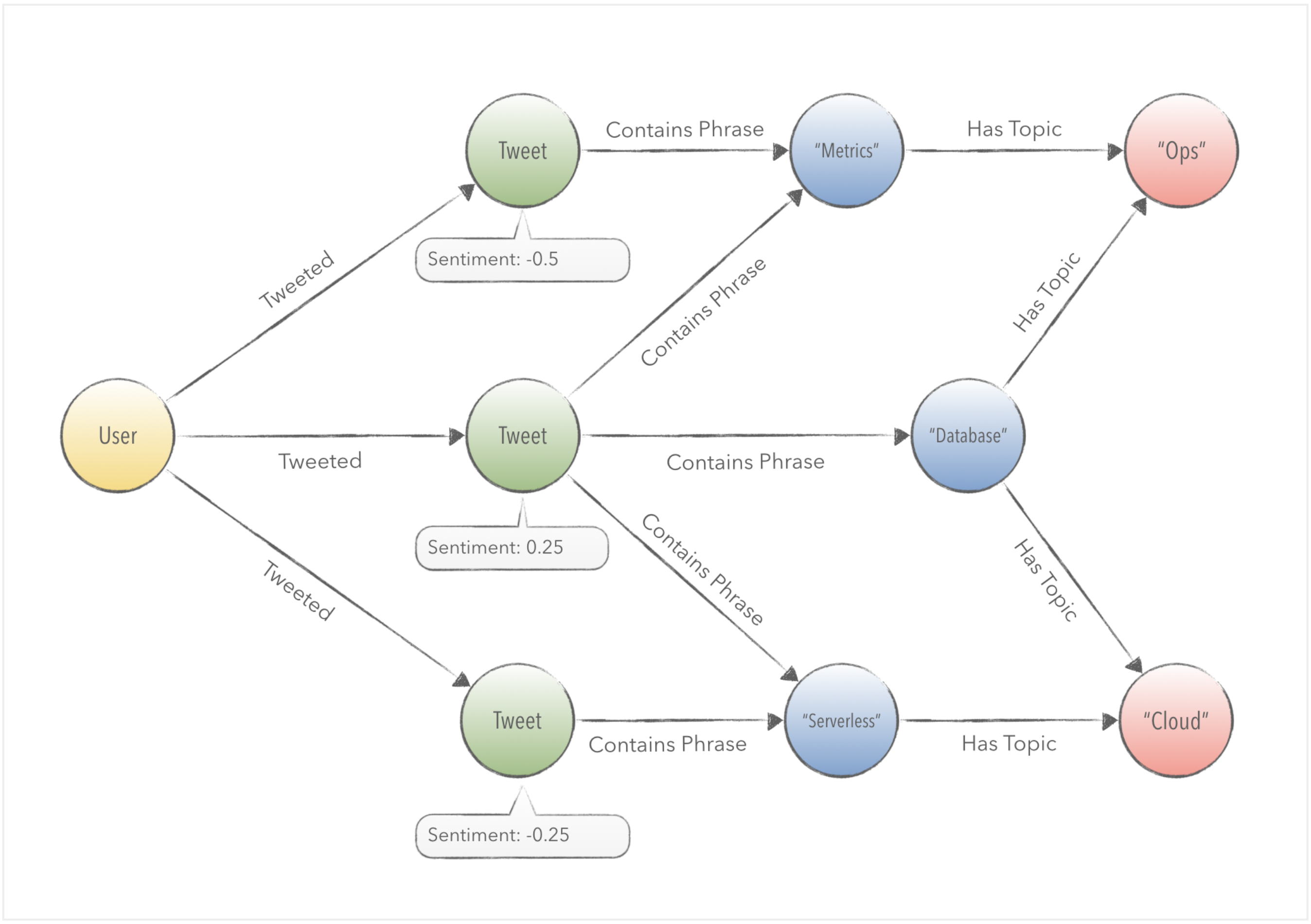
Here we can see that the entity sentiment analysis endpoint will return back a collection of phrases. The endpoint provides a sentiment score for each phrase in context to the text it was extracted from. We store this calculation in the relationship that connects a `Tweet` to a `Phrase`, as shown in the diagram above.

Content classification

Content classification is an API provided by GCP that will allow you to provide a string of text as a document, and be returned a set of categories that classifies the content.

> *Content classification* (https://cloud.google.com/natural-language/docs/classifying-text) analyzes text content and returns a content category for the content.

The diagram below shows how a `Tweet` contains certain phrases that share a topic, and can be linked back to a `User`.



Categorizing a group of tweets into a set of topics is a difficult proposition. More so, the computation required to categorically segment topics is cost prohibitive. Tweets, by themselves, do not always contain enough text to meaningfully classify their content. Meanings also vary widely in context to the audience and author of a tweet.

To generate an index of topics for groups of tweets, we can use PageRank scoring on phrases mentioned in imported tweets. By doing this, we can group together batches of tweets for the top ranked phrases. Further, this allows us to ask questions related to a user's sentiment for particular topics, in addition to phrases.

For example, the crawling algorithm will schedule an analysis on the phrases `metrics`, `database`, and `serverless`. For each of these terms, we'll select each `Tweet` and join together the text into a single document. The resulting document contains much more text that will be useful for content classification on Google Cloud.

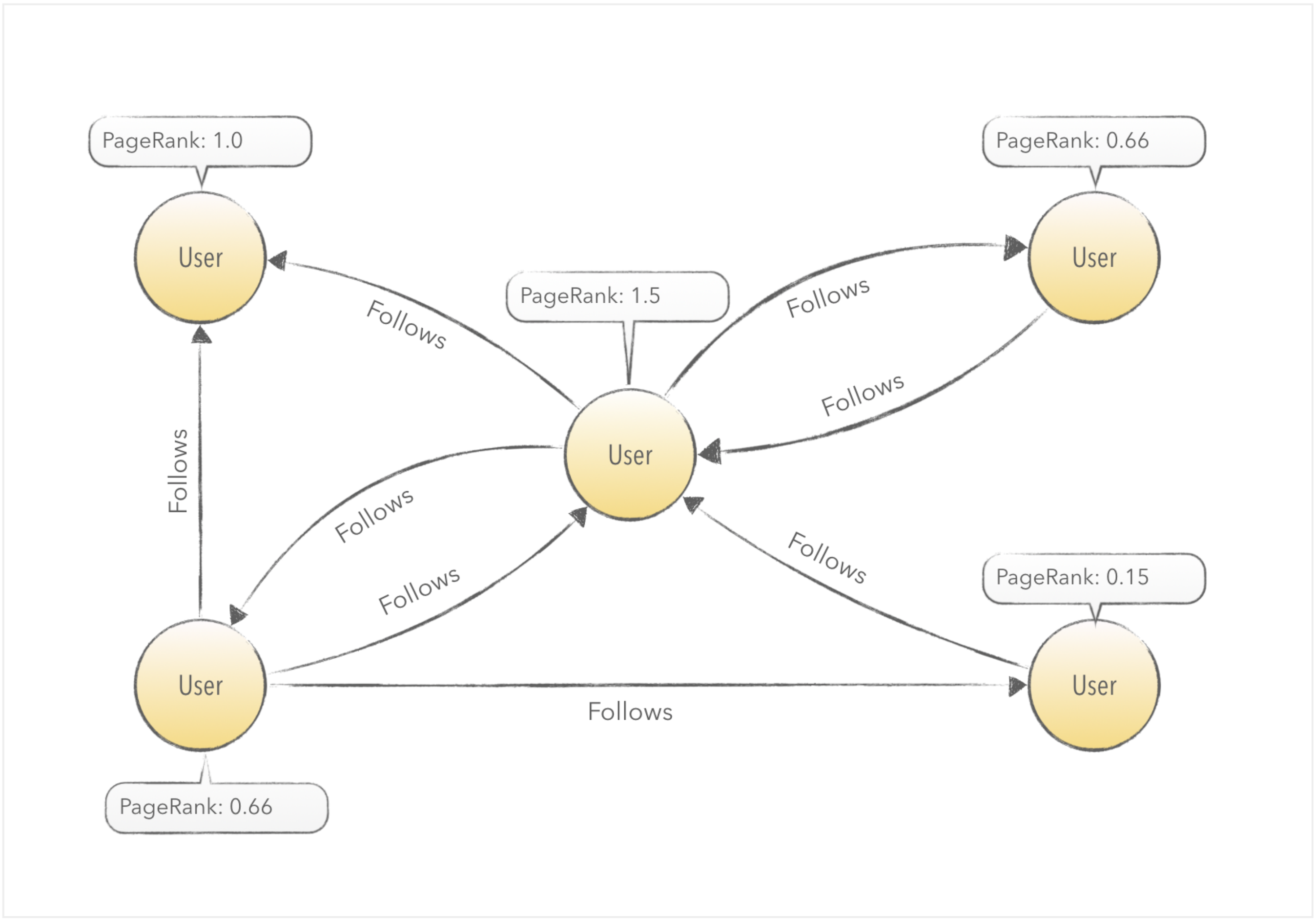
In the next section I'll describe how PageRank is used to optimize the crawling algorithm's import and analysis of Twitter data.

Graph Processing

How do you find the most influential users in your Twitter network without having access to all of the data? To do this, we need to iteratively rank and then discover profiles using the PageRank algorithm.

User Rank

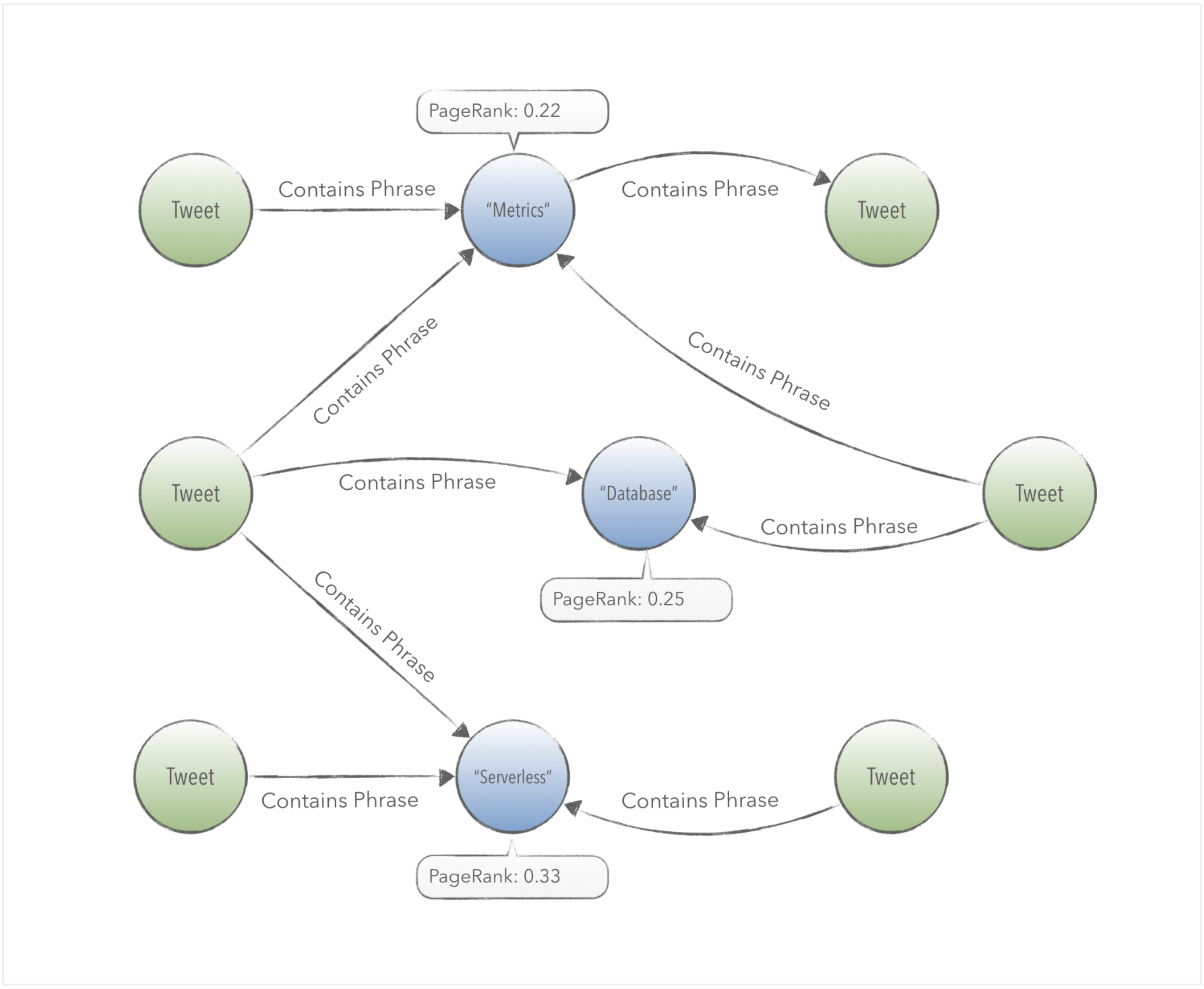
The diagram below describes the follower graph of relationships between `Users`.



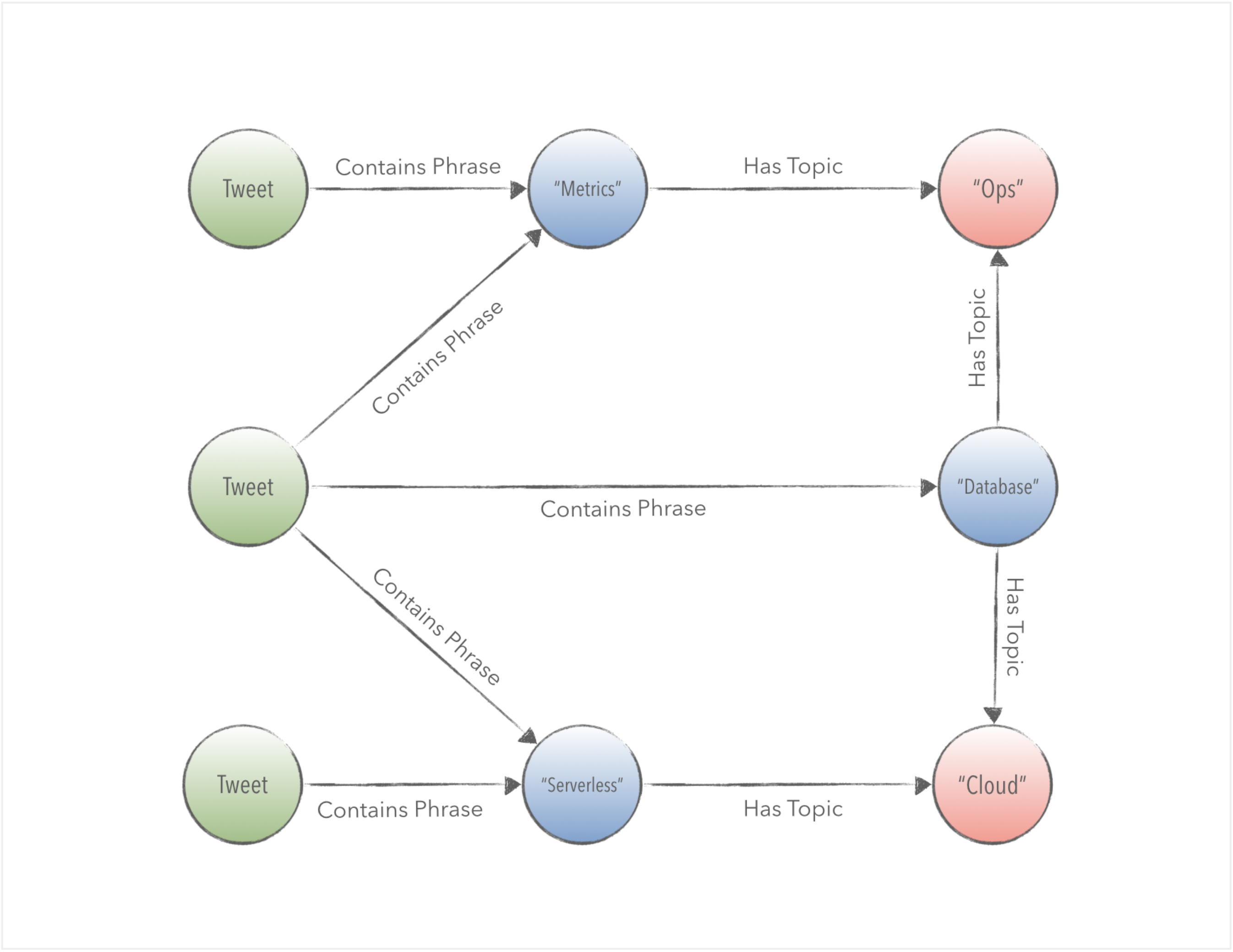
On a scheduled interval, PageRank will be run on this subgraph. The result will be used to select the next highest ranked user that has not yet been imported. By doing this, we can walk towards more influential sources of content. The added benefit is that we only focus on importing follower and friend relationships from the most influential users. This allows us to hop towards the influential center of gravity of tweets in a network without importing retweets or favorites.

Phrase Rank

The diagram below describes the semantic relationships between `Tweets` and `Phrases`.



On a scheduled interval, PageRank is calculated on the phrases of this subgraph. The results are used to classify groups of `Tweets`, as shown in the example below.



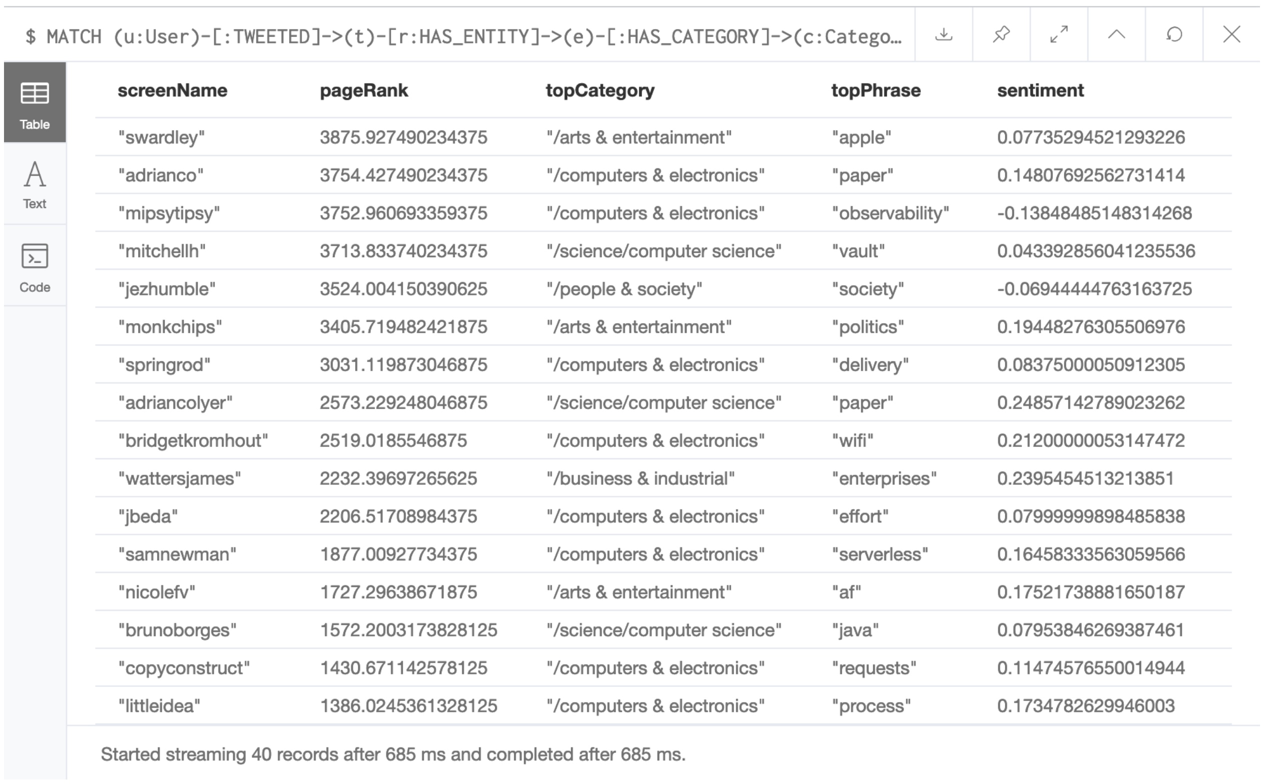
Answering Questions with Neo4j

In this first blog post we're going to keep things relatively short by focusing on answering questions using the graph data model. In later blog posts, I will focus more on operations and application development with Spring Boot (https://spring.io/projects/spring-boot).

In the next sections I'll summarize some of the queries that I formulated to answer questions related to emotional contagion (https://en.wikipedia.org/wiki/Emotional_contagion), influence (https://en.wikipedia.org/wiki/Social_influence), and memes (https://en.wikipedia.org/wiki/Meme).

Ranking users by topic and influence

One of the queries that I wanted to create was for a ranking dashboard based on topic and sentiment. It took me numerous iterations to come up with a Neo4j Cypher query that accurately extracts the most relevant category and phrase for each ranked user.



```
$ MATCH (u:User)-[:TWEETED]->(t)-[r:HAS_ENTITY]->(e)-[:HAS_CATEGORY]->(c:Catego…
```

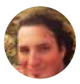
screenName	pageRank	topCategory	topPhrase	sentiment
"swardley"	3875.927490234375	"/arts & entertainment"	"apple"	0.07735294521293226
"adrianco"	3754.427490234375	"/computers & electronics"	"paper"	0.14807692562731414
"mipsytipsy"	3752.960693359375	"/computers & electronics"	"observability"	-0.13848485148314268
"mitchellh"	3713.833740234375	"/science/computer science"	"vault"	0.043392856041235536
"jezhumble"	3524.004150390625	"/people & society"	"society"	-0.06944444763163725
"monkchips"	3405.719482421875	"/arts & entertainment"	"politics"	0.19448276305506976
"springrod"	3031.119873046875	"/computers & electronics"	"delivery"	0.08375000050912305
"adriancolyer"	2573.229248046875	"/science/computer science"	"paper"	0.24857142789023262
"bridgetkromhout"	2519.0185546875	"/computers & electronics"	"wifi"	0.21200000053147472
"wattersjames"	2232.39697265625	"/business & industrial"	"enterprises"	0.2395454513213851
"jbeda"	2206.51708984375	"/computers & electronics"	"effort"	0.07999999898485838
"samnewman"	1877.00927734375	"/computers & electronics"	"serverless"	0.16458333563059566
"nicolefv"	1727.29638671875	"/arts & entertainment"	"af"	0.17521738881650187
"brunoborges"	1572.2003173828125	"/science/computer science"	"java"	0.07953846269387461
"copyconstruct"	1430.671142578125	"/computers & electronics"	"requests"	0.11474576550014944
"littleidea"	1386.0245361328125	"/computers & electronics"	"process"	0.1734782629946003

Started streaming 40 records after 685 ms and completed after 685 ms.

In the screenshot above, I run a Cypher query in the Neo4j browser to infer the top phrase and category for a user's tweets. Since I follow all of these users, I can honestly say that the query is fairly accurate. I will dive deeper into this subject in a later blog post.

Diving into the meme pool

Next, I'll explain some of my recent tweets, which I posted over the course of building this application.



Kenny Bastani
@kennybastani

I built a sentiment analysis tool that can learn stuff by reading tweets. It figured out what a container is.

Well, kind of. 😁

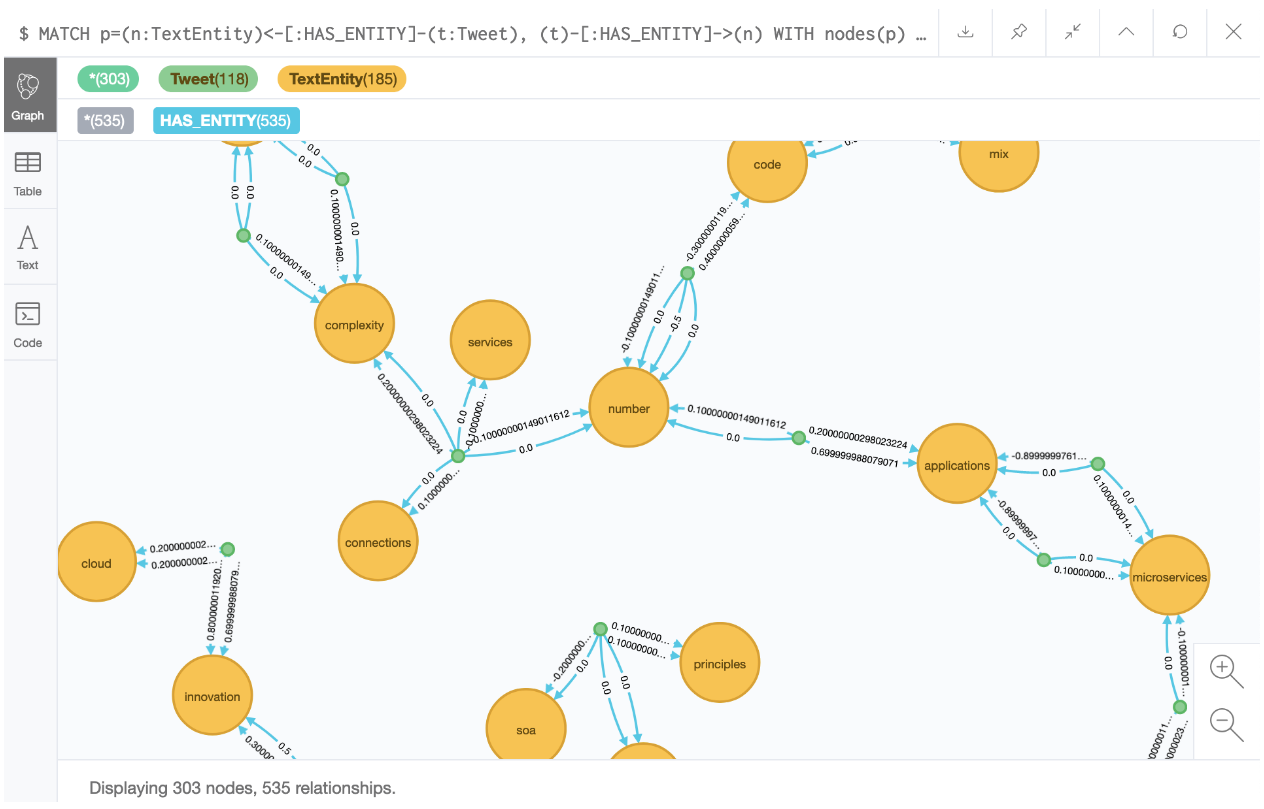
```
TCH p=(n:TextEntity)<-[:HAS_ENTITY]-(t:Tweet), (t)-[:HAS_ENTITY]->(n) WITH nodes(p) as paths UNWIND filter(x IN paths WHERE NOT x.name =~ '@.*') …
```

56 4:43 PM - Sep 8, 2019

See Kenny Bastani's other Tweets

In the tweet above, I was excited to discover a graph visualization that demonstrated how Linux containers (https://en.wikipedia.org/wiki/LXC) were related to different phrases. I call this Neo4j Cypher query: a *meme graph*.

Memes are patterns or templates in natural language text that evolve and change over time. Twitter users will post variations of a meme, which will contain variable and static parts. The variable parts of a meme are limited to a subset of possible terms. To discover a meme in the Twitter graph, I can query for `phrases` that have multiple connections to `tweets` . By traversing tweets and their extracted phrases, I discovered potential memes by matching cycles and loops in entity relationships.



```
$ MATCH p=(n:TextEntity)<-[:HAS_ENTITY]-(t:Tweet), (t)-[:HAS_ENTITY]->(n) WITH nodes(p) …
```

Displaying 303 nodes, 535 relationships.

In the screenshot above, you'll see a flow of looped connections between tweets and phrases. I've set a criteria on the results so that only phrases mentioned twice in the same tweet are displayed. This proved to be a really clever way to determine the most relevant phrases in the network. As it turns out, people do not often use the same phrase more than once in the same tweet. Which means, that for the users who do, they are conveying something of topical importance.

Memes seem to tell a story about a network of users on Twitter.

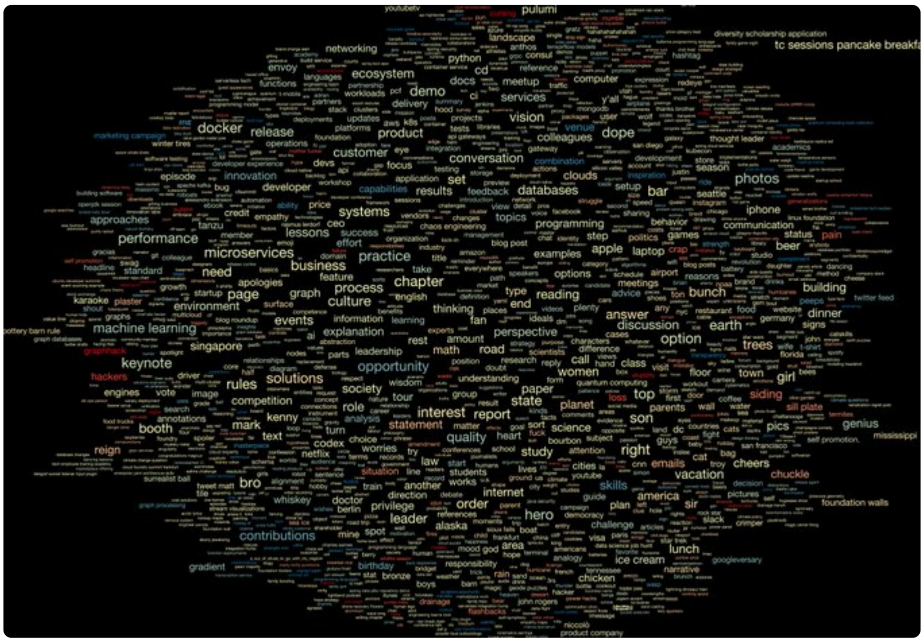
Visualizing the emotion of words

Next, I've decided to utilize the sentiment scoring from Google Cloud Natural Language to generate a visualization of popular phrases in their emotional context.

Kenny Bastani
@kennybastani

I used graph algorithms and sentiment analysis to infer and predict the "emotional feels 🤷‍♂️" for 20,000+ phrases & mentions in my Twitter network.

Blue/green phrases: 🙂
Yellow phrases: 😶
Orange/red phrases: 🙁



67 12:17 AM - Sep 12, 2019

16 people are talking about this

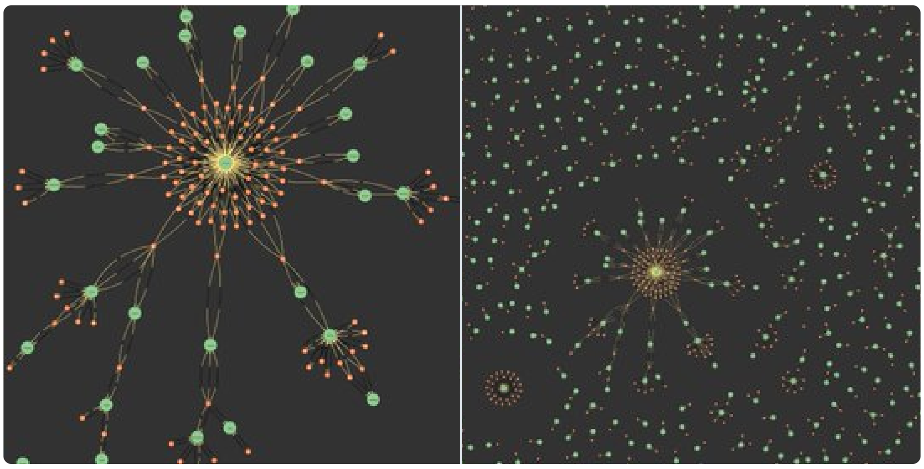
The tweet referenced above is a ranked extract of phrases that are colored and sized depending on their emotional context. Simply, I exported the graph of phrases and tweets into a visualization tool named Gephi (https://gephi.org/). Gephi has a set of features that you can use to rank and visualize graph datasets. For me, this was a good proof of concept for understanding whether or not sentiment analysis could be used to infer the larger emotional context of important phrases in my Twitter network.

Visualizing the virulence of words

While putting together this blog post, I wanted to focus on determining how viral text could spread emotions. It turns out, there are academic papers (https://www.pnas.org/content/pnas/111/24/8788.full.pdf) that prove that emotional contagion can spread in social networks. This will be the topic of future blog posts, but I wanted to end this blog post with a quote from the father of memes, Richard Dawkins (https://en.wikipedia.org/wiki/Richard_Dawkins).

Kenny Bastani
@kennybastani

The shape of viral text on Twitter. 🦠🦠🦠



45 2:09 PM - Sep 15, 2019

15 people are talking about this

The most interesting thing I discovered from the data so far was related to memes. It turns out, that people construct and use memes to easily deliver meaningful information on Twitter without knowing. Memes serve as a template where static and variable parts of text provide a familiar backbone for understanding many different aspects of the intended meaning of a tweet. There are the memes we, of course, know and intend to use. There are also memes that we often use that convey meaning but are not intentional, and are not obvious.

The shape of the connected data appears to also show that biological patterns (https://en.wikipedia.org/wiki/Pattern_formation) evolve from the variations of memes that connect tweets and phrases together. This was first predicted by Dawkins, in his book The Selfish Gene (https://en.wikipedia.org/wiki/The_Selfish_Gene), where he coins the term *meme*.

> I believe that, given the right conditions, replicators automatically band together to create systems, or machines, that carry them around and work to favour their continued replication.

— Richard Dawkins
The Selfish Gene (1976)

There is an interesting body of work behind this idea, thanks to Dawkins. But we should ask, why would there be biologically mimetic patterns in the graph structure that I queried?

Because memes use natural selection to reproduce and evolve in the same way that biological organisms do, using genes. Both mechanisms are based on the translation and expression of information in the form of graphs, which Dawkins first wrote about over 40 years ago. Information is at the core of gene expression into molecular proteins, with organisms as the driver. Information is also at the core of language's expression into meaningful behavior, with emotion as the driver.

> With only a little imagination we can see the gene as sitting at the centre of a radiating web of extended phenotypic power. And an object in the world is the centre of a converging web of influences from many genes sitting in many organisms. The long reach of the gene knows no obvious boundaries. The whole world is criss-crossed with causal arrows joining genes to phenotypic effects, far and near.

— Richard Dawkins
The Selfish Gene (1976)

These are the final words of Dawkin's book. No world-wide web as we know it today had yet existed in 1976. No Twitter existed. I did not exist. And today, we find ourselves diving into the meme pool and doing so without a full understanding how our brains work. So, what do you think? What is influencing your behavior on Twitter?

Summary

In this blog post I've introduced you to a graph data model for analyzing the influences and sentiments of users on Twitter. To better understand the code behind the architecture, I recommend reading a previously posted tutorial that describes the predecessor to this blog post.

In an upcoming blog post we will focus on building a news feed that balances the negative and positive content of popular tweets, using Spring Boot. We'll also focus on a feature for understanding how emotional content balances a user's behavior over time. I'll also focus more on explaining the operational implications of running and scaling the Twitter crawling algorithm and Neo4j.

Running the example

The code for this blog post is free and openly available, but is still in active flux—as I design towards a more meaningful community project. I highly recommend exploring the code (https://github.com/kbastani/sentiment-analysis-twitter-microservices-example/blob/master/twitter-rank-crawler/src/main/java/org/kbastani/processor/RankProcessor.java) that is commented in the *Twitter Rank Crawler* service (https://github.com/kbastani/sentiment-analysis-twitter-microservices-example/tree/master/twitter-rank-crawler).

Feel free to post questions here or on the GitHub issue tracker. I've included the directions on running the sample application in the GitHub repository (https://github.com/kbastani/sentiment-analysis-twitter-microservices-example) for this example.

Posted by Kenny Bastani (https://www.blogger.com/profile/09075390099600561724) at 9:36:00 AM (2019-09-19T09: 36: 00-07: 00) (https://www.kennybastani.com/2019/09/sentiment-analysis-on-twitter-data.html) (https://www.blogger.com/email-post.g?blogID=6300367579216018061&postID=363020128279112356)

Labels: design (https://www.kennybastani.com/search/label/design) , google cloud (https://www.kennybastani.com/search/label/google%20cloud) , google cloud natural language (https://www.kennybastani.com/search/label/google%20cloud%20natural%20language) , graph algorithms (https://www.kennybastani.com/search/label/graph%20algorithms) , graph data modeling (https://www.kennybastani.com/search/label/graph%20data%20modeling) , graph processing (https://www.kennybastani.com/search/label/graph%20processing) , neo4j (https://www.kennybastani.com/search/label/neo4j) , PageRank (https://www.kennybastani.com/search/label/PageRank) , twitter (https://www.kennybastani.com/search/label/twitter)

No comments :

Post a Comment

Be curious, I dare you.

(https://www.blogger.com/comment-iframe.g?blogID=6300367579216018061&postID=363020128279112356&blogspotRpcToken=6114207)

Enter your comment...

Comment as: javimartinalon: ▼ Sign out

Publish Preview ☐ Notify me

Links to this post

Create a Link (https://www.blogger.com/blog-this.g)

w.kennybastani.com/2017/07/mic Home (https://www.kennybastani.com/) d-function-aws-lambda.html)

Subscribe to: Post Comments (Atom) (https://www.kennybastani.com/feeds/363020128279112356/comments/default)

Blog Archive

▼ 2019 (https://www.kennybastani.com/2019/) (1)

September (https://www.kennybastani.com/2019/09/) (1)

► 2017 (https://www.kennybastani.com/2017/) (2)

► 2016 (https://www.kennybastani.com/2016/) (3)

► 2015 (https://www.kennybastani.com/2015/) (5)

► 2014 (https://www.kennybastani.com/2014/) (10)

► 2013 (https://www.kennybastani.com/2013/) (3)

Labels

neo4j (https://www.kennybastani.com/search/label/neo4j) graph database (https://www.kennybastani.com/search/label/graph%20database) spring boot (https://www.kennybastani.com/search/label/spring%20boot) microservices (https://www.kennybastani.com/search/label/microservices) spring cloud (https://www.kennybastani.com/search/label/spring%20cloud) apache spark (https://www.kennybastani.com/search/label/apache%20spark) docker (https://www.kennybastani.com/search/label/docker) PageRank (https://www.kennybastani.com/search/label/PageRank) cloud native java (https://www.kennybastani.com/search/label/cloud%20native%20java) data science (https://www.kennybastani.com/search/label/data%20science) docker compose (https://www.kennybastani.com/search/label/docker%20compose) Mazerunner (https://www.kennybastani.com/search/label/Mazerunner) big data (https://www.kennybastani.com/search/label/big%20data) graph analytics (https://www.kennybastani.com/search/label/graph%20analytics) graphx (https://www.kennybastani.com/search/label/graphx) open source software (https://www.kennybastani.com/search/label/open%20source%20software) analytics (https://www.kennybastani.com/search/label/analytics) event sourcing (https://www.kennybastani.com/search/label/event%20sourcing) graph processing (https://www.kennybastani.com/search/label/graph%20processing) pattern recognition (https://www.kennybastani.com/search/label/pattern%20recognition) cqrs (https://www.kennybastani.com/search/label/cqrs) cypher (https://www.kennybastani.com/search/label/cypher) event-driven microservices (https://www.kennybastani.com/search/label/event-driven%20microservices) github (https://www.kennybastani.com/search/label/github) graph data modeling (https://www.kennybastani.com/search/label/graph%20data%20modeling) graphs (https://www.kennybastani.com/search/label/graphs) information theory (https://www.kennybastani.com/search/label/information%20theory) meetup (https://www.kennybastani.com/search/label/meetup) natural language processing (https://www.kennybastani.com/search/label/natural%20language%20processing) node.js (https://www.kennybastani.com/search/label/node.js) open source (https://www.kennybastani.com/search/label/open%20source) reporting (https://www.kennybastani.com/search/label/reporting) serverless (https://www.kennybastani.com/search/label/serverless) text classification (https://www.kennybastani.com/search/label/text%20classification) twitter (https://www.kennybastani.com/search/label/twitter) PCF dev (https://www.kennybastani.com/search/label/PCF%20dev) algorithms (https://www.kennybastani.com/search/label/algorithms) apache hadoop (https://www.kennybastani.com/search/label/apache%20hadoop) api gateway (https://www.kennybastani.com/search/label/api%20gateway) architecture (https://www.kennybastani.com/search/label/architecture) artificial intelligence (https://www.kennybastani.com/search/label/artificial%20intelligence) c# (https://www.kennybastani.com/search/label/c%23) cloud foundry (https://www.kennybastani.com/search/label/cloud%20foundry) coding rocks (https://www.kennybastani.com/search/label/coding%20rocks) computable numbers (https://www.kennybastani.com/search/label/computable%20numbers) data (https://www.kennybastani.com/search/label/data) data import (https://www.kennybastani.com/search/label/data%20import) declarative query language (https://www.kennybastani.com/search/label/declarative%20query%20language) deep learning (https://www.kennybastani.com/search/label/deep%20learning) delete duplicate records (https://www.kennybastani.com/search/label/delete%20duplicate%20records) design (https://www.kennybastani.com/search/label/design) discovery service (https://www.kennybastani.com/search/label/discovery%20service) docker swarm (https://www.kennybastani.com/search/label/docker%20swarm) document classification (https://www.kennybastani.com/search/label/document%20classification) eventual consistency (https://www.kennybastani.com/search/label/eventual%20consistency) execution planning (https://www.kennybastani.com/search/label/execution%20planning) feature learning (https://www.kennybastani.com/search/label/feature%20learning) gephi (https://www.kennybastani.com/search/label/gephi) google cloud (https://www.kennybastani.com/search/label/google%20cloud) google cloud natural language (https://www.kennybastani.com/search/label/google%20cloud%20natural%20language) graph algorithms (https://www.kennybastani.com/search/label/graph%20algorithms) graph analysis (https://www.kennybastani.com/search/label/graph%20analysis) graph theory (https://www.kennybastani.com/search/label/graph%20theory) graphify (https://www.kennybastani.com/search/label/graphify) heroku scheduler (https://www.kennybastani.com/search/label/heroku%20scheduler) james gleick (https://www.kennybastani.com/search/label/james%20gleick) jeff hawkins (https://www.kennybastani.com/search/label/jeff%20hawkins) legacy modernization (https://www.kennybastani.com/search/label/legacy%20modernization) legacy systems (https://www.kennybastani.com/search/label/legacy%20systems) machine learning (https://www.kennybastani.com/search/label/machine%20learning) monoliths (https://www.kennybastani.com/search/label/monoliths) mysql (https://www.kennybastani.com/search/label/mysql) nodes (https://www.kennybastani.com/search/label/nodes) polyglot persistence (https://www.kennybastani.com/search/label/polyglot%20persistence) rabbitmq (https://www.kennybastani.com/search/label/rabbitmq) ray kurzweil (https://www.kennybastani.com/search/label/ray%20kurzweil) reactive streaming (https://www.kennybastani.com/search/label/reactive%20streaming) reactor (https://www.kennybastani.com/search/label/reactor) recommendation engine (https://www.kennybastani.com/search/label/recommendation%20engine) sentiment analysis (https://www.kennybastani.com/search/label/sentiment%20analysis) service block architecture (https://www.kennybastani.com/search/label/service%20block%20architecture) service blocks (https://www.kennybastani.com/search/label/service%20blocks) soa (https://www.kennybastani.com/search/label/soa) spark neo4j (https://www.kennybastani.com/search/label/spark%20neo4j) spring cloud function (https://www.kennybastani.com/search/label/spring%20cloud%20function) spring data (https://www.kennybastani.com/search/label/spring%20data) time scales (https://www.kennybastani.com/search/label/time%20scales) triangle count (https://www.kennybastani.com/search/label/triangle%20count) tutorial (https://www.kennybastani.com/search/label/tutorial) ubigraph (https://www.kennybastani.com/search/label/ubigraph) universal turing machine (https://www.kennybastani.com/search/label/universal%20turing%20machine) visualization (https://www.kennybastani.com/search/label/visualization) wikipedia (https://www.kennybastani.com/search/label/wikipedia)

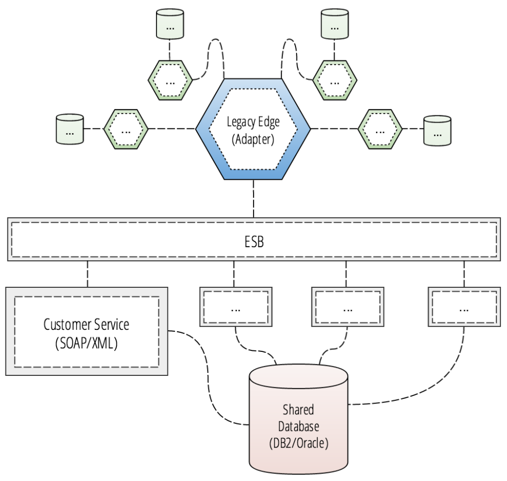
Follow by Email

Email address... Submit

Featured

Building Spring Cloud Microservices That Strangle Legacy Systems (https://www.kennybastani.com/2016/08/strangling-legacy-microservices-spring-cloud.html)

It's safe to say that any company who was writing software ten years ago—and is building microservices today—will need to integrat...



Popular Posts

Building Microservices with Spring Cloud and Docker (https://www.kennybastani.com/2015/07/spring-cloud-docker-microservices.html)

This blog series will introduce you to some of the foundational concepts of building a microservice-based platform using Spring Cloud a...

Event Sourcing in Microservices Using Spring Cloud and Reactor (https://www.kennybastani.com/2016/04/event-sourcing-microservices-spring-cloud.html)
When building applications in a microservice architecture, managing state becomes a distributed systems problem. Instead of being ab...

Building Event-driven Microservices Using CQRS and Serverless (https://www.kennybastani.com/2017/01/building-event-driven-microservices.html)
Esta serie de blogs le presentará la creación de microservicios basados en eventos como aplicaciones nativas de la nube. En el ...