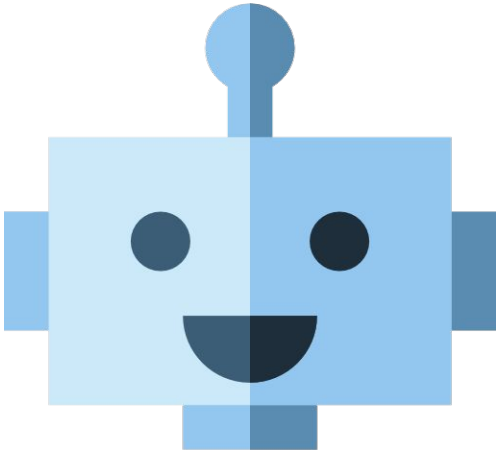
Democratizing data at Kiwi.com



Challenge

Navigate in existing data in any form and tool

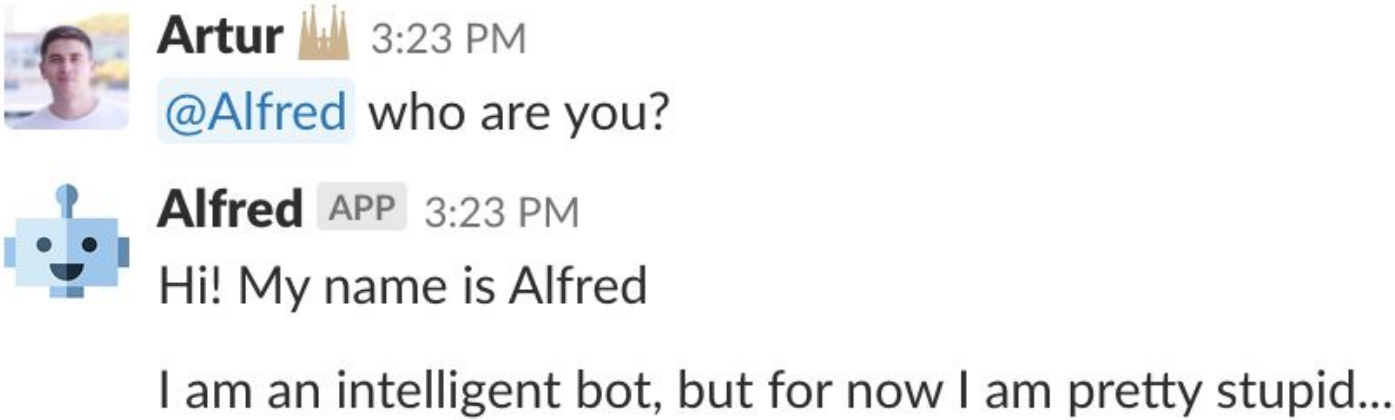
- **Gooddata**: > 2000 metrics, > 3000 reports
- **Metabase**: > 2500 questions, > 140 dashboards
- **Exponea**: > 1800 reports, > 140 dashboards
- **Slack**: > 900 questions with answers (**#plz-analytics**)
- **Confluence**: ~90 documents (only in Analytics space)
- and many more...

Problem

- People ask repetitive questions
- People want to find existing analyses/documents
- We need people’s time to answer these questions

Solution

Slack chatbot which will provide all the necessary information by human-like interaction.

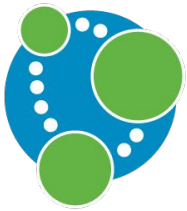


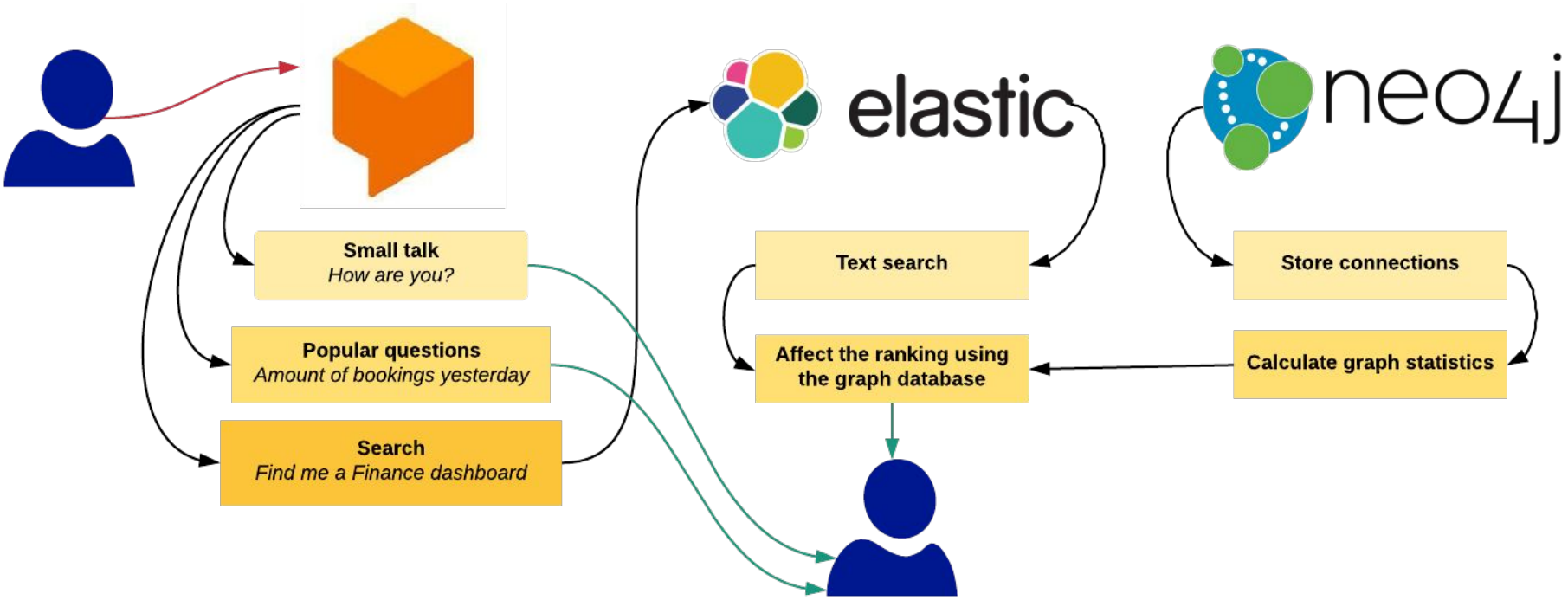
Main technology stack

- **Dialogflow** (natural language conversations platform)

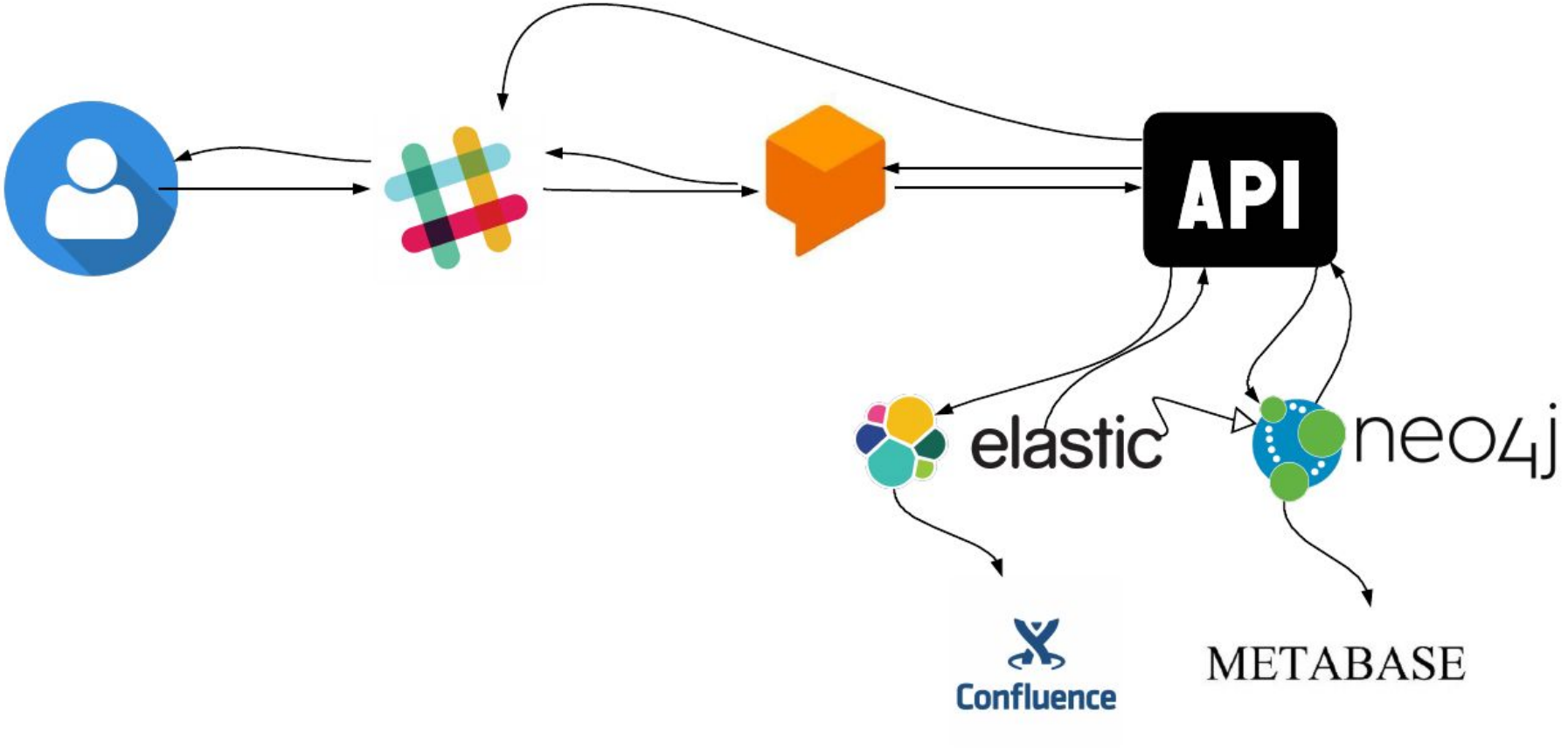
- **Elasticsearch** (text search database)

- **Neo4j** (graph database



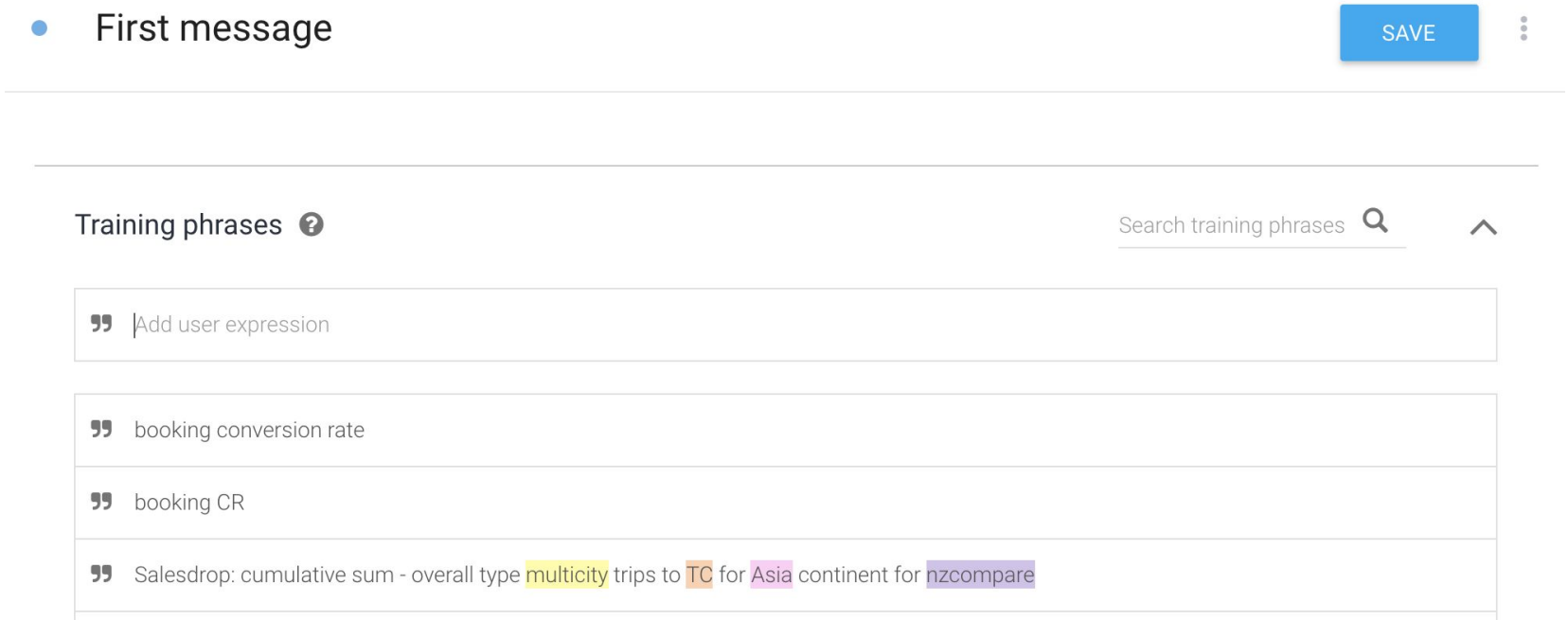

elastic
neo4j
Small talk
How are you?
Popular questions
Amount of bookings yesterday
Search
Find me a Finance dashboard
Text search
Affect the ranking using the graph database
Store connections
Calculate graph statistics

Workflow



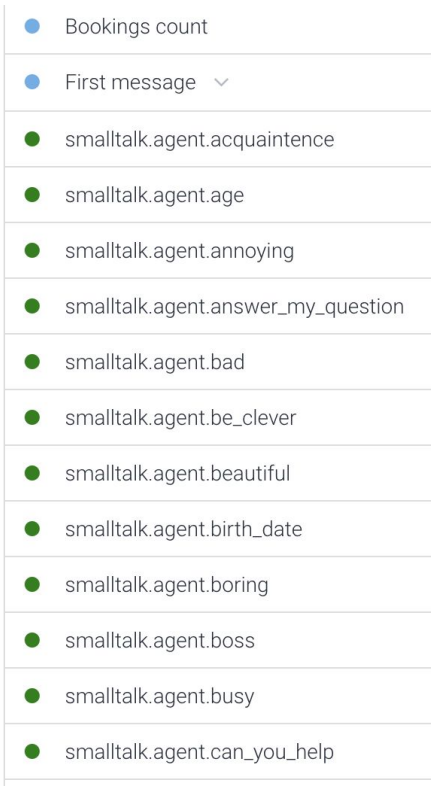
Dialogflow - review

- NLP model, helps to classify messages and get params from it



Dialogflow - intents

- Popular questions intents (How many bookings?)
  - Action: give the link directly
  - Put them manually



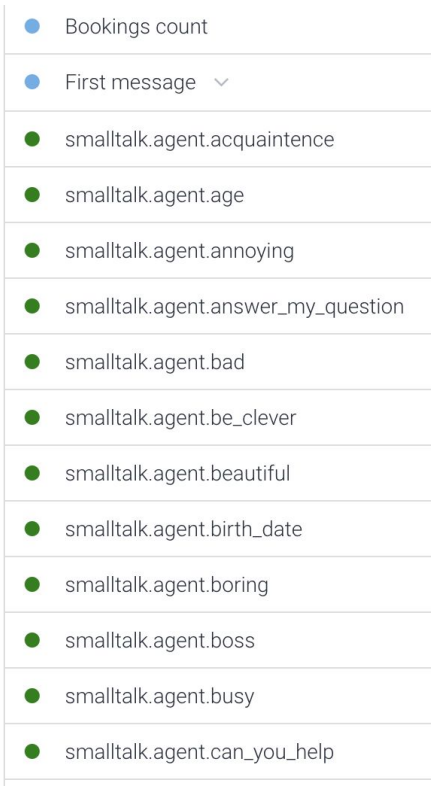
Dialogflow - small talk

- To make Alfred more human friendly :)

- Bookings count
- First message
- smalltalk.agent.acquaintence
- smalltalk.agent.age
- smalltalk.agent.annoying
- smalltalk.agent.answer_my_question
- smalltalk.agent.bad
- smalltalk.agent.be_clever
- smalltalk.agent.beautiful
- smalltalk.agent.birth_date
- smalltalk.agent.boring
- smalltalk.agent.boss
- smalltalk.agent.busy
- smalltalk.agent.can_you_help

Dialogflow - intents

- Other questions:
  - Action: need to search in database (Elasticsearch)
  - Main topic of this presentation



Dialogflow - parameters

Action and parameters

first_message

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
☐	dates	@sys.date	$dates	☑
☐	date_period	@sys.date-period	$date_period	☐
☐	countries	@country_name	$countries	☑
☐	airlines	@airline	$airlines	☑
☐	airports	@airport	$airports	☑
☐	is_direct	@is_direct	$is_direct	☐
☐	cities	@city_name	$cities	☑
☐	market	@market	$market	☐
☐	partner	@partner	$partner	☐
☐	trip_type	@trip_type	$trip_type	☐
☐	continents	@continent	$continents	☑
☐	Enter name	Enter entity	Enter value	☐

Dialogflow - problem smalltalk

- *Problem*: difficult to create smalltalk intents manually
  - > 50 intents
  - 5-10 training phrases for each one
- *Solution:* get Excel sheet of ready intents from the web
- *Solution:* customize it in Excel sheet
- *Solution:* send it to Dialogflow API

Dialogflow - Excel smalltalk

smalltalk.agent.acquaintence	Tell me about your personality	Just think of me as the ace up your sleeve.
smalltalk.agent.acquaintence	I want to know you better	I can help you work smarter instead of harder
smalltalk.agent.acquaintence	Define yourself	
smalltalk.agent.acquaintence	Describe yourself	
smalltalk.agent.acquaintence	tell me about yourself	
smalltalk.agent.acquaintence	all about you	
smalltalk.agent.acquaintence	tell me some stuff about you	
smalltalk.agent.acquaintence	talk some stuff about you	
smalltalk.agent.acquaintence	talk about yourself	

Dialogflow - problems with training

- *Problem*: hard to train the model manually
- *Problem:* the model is pretty stupid
- *Problem:* you have to put all possible combinations of parameters

> Salesdrop: cumulative sum - 1 week - nationality multicity itineraries through WF continent Asia continent continent Europe continent

> Salesdrop: cumulative sum - 1 week - nationality type roundtrip itineraries to Madagascar over South America continent partner flyfromeu partner

> Salesdrop: cumulative sum - 1 week - market multicity itineraries from HR from Oceania continent partner lastminute partner

Dialogflow - generating parameters

```
class Airlines(Parameter):
   singular_name = 'airline'
   dialogflow_type = 'airline'
   prefixes = ['for', 'airline', 'from', '']
   endings = ['airline', '']


class Airports(Parameter):
   singular_name = 'airport'
   dialogflow_type = 'airport'
   prefixes = ['for', 'from', 'to', 'through', 'airoport', '']
   endings = ['airoport', '']


class IsDirect(Parameter):
   singular_name = 'is_direct'
   dialogflow_type = 'is_direct'
   prefixes = ['for', '']
   endings = ['bookings', 'itineraries', 'flights', '']
   is_singular = True
```

Dialogflow - generating parameters

Salesdrop: cumulative sum - 1 week - src region type multicity itineraries to SB through South America continent for Himalayan Airlines airline

Salesdrop: cumulative sum - 1 week - route on country level over roundtrip itineraries PE from North America continent airline Line Blue airline

Dialogflow - other problems

- **Limit**: 2000 training samples
- Docs are not completed
- **Limit**: size of requests

- Now we can understand our user (more or less)
- What’s next?
- 

Databases

- **Elasticsearch** to store the text data.
- **Neo4j** to store relations between documents and users.
- **Elasticsearch:** one of the best databases to make full-text queries
- **Neo4j:** graph database, good for fast prototyping

Why do we even need graphs?

1. Store **user -> data** connections
2. Store **data -> data** connections (ETL pipelines, data sources and etc.)
3. Get insights from graphs
4. Statistics, such as: document view, popularity
5. Recommendations
6. Dataflow inside the company

Our case

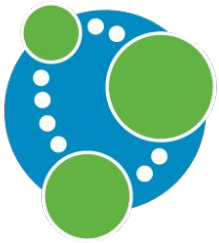
- We calculate useful statistics on side of **Neo4j**:
  - Number of views of a document
  - Distinct people viewed a document
  - *PageRank* score for each document (*popularity score*)
- We use these numbers to affect score while searching in

  Elasticsearch

Document model in ES

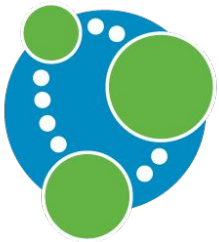
```
1. class DocumentElastic(DocType):
2.     uuid = Keyword()
3.     title = Text(fields=default_fields)
4.     ...
5.     description = Text(fields=default_fields)
6.     updated_at = Date()
7.     ...
8.     parameters = Nested(Parameter)
9.     ...
10.    graph_statistics = Nested(ResultType)
11.
12.     class Index:
13.         name = 'documents'
14.
15.     def is_up_to_date(self, last_updated: datetime):
16.         return self.updated_at >= last_updated
```

User model in Neo4j



```
1. class UserNeo(StructuredNode):
2.     uuid = StringProperty()
3.     email = StringProperty(unique_index=True)
4.     time_created = DateTimeProperty()
5.
6.     created = RelationshipTo('DocumentNeo', 'CREATED', model=CreatedRelation)
7.     consumed = RelationshipTo('DocumentNeo', 'CONSUMED', model=ConsumedRelation)
8.     modified = RelationshipTo('DocumentNeo', 'MODIFIED', model=ModifiedRelation)
```

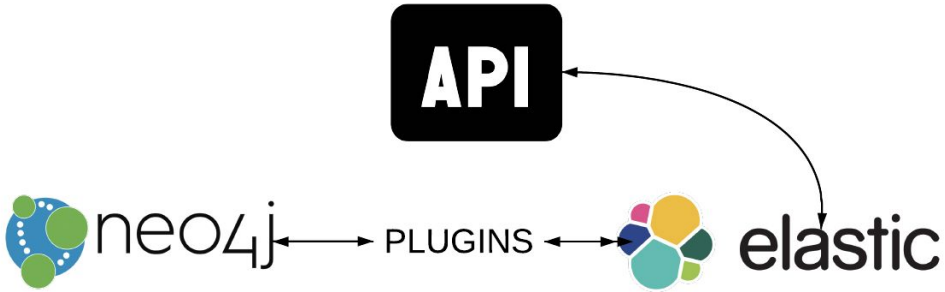
Document model in Neo4j

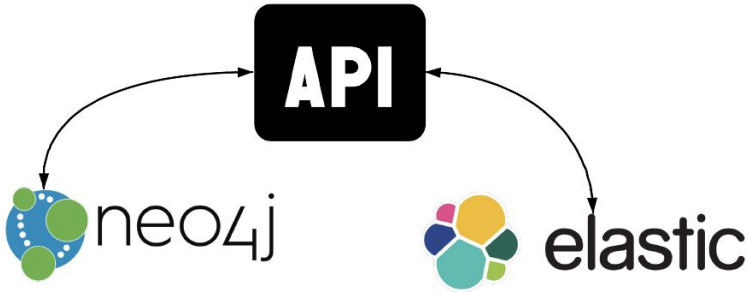


```
1. class DocumentNeo(StructuredNode):
2.     uuid = StringProperty()
3.     source = StringProperty(required=True, index=True)
4.     source_id = StringProperty(required=True, index=True)
5.     views = IntegerProperty(default=0)
6.     people_viewed = IntegerProperty(default=0)
7.     page_rank = FloatProperty(default=0)
8.
9.     created_by = RelationshipTo('UserNeo', 'CREATED_BY', model=CreatedRelation)
10.     consumed_by = RelationshipTo('UserNeo', 'CONSUMED_BY', model=ConsumedRelation)
11.     modified_by = RelationshipTo('UserNeo', 'MODIFIED_BY', model=ModifiedRelation)
```

ES + Neo4j - how to use both dbs?

- We were using plugins to connect Elasticsearch + Neo4j
- Plugins are only working with ES v2.x - which is kinda old





ES + Neo4j

- **Solution**: consult both databases at the same time
- We use Neo4j **UUID** plugin
- Write a custom class

ES + Neo4j - interface to unite them

```
class Document:
    """Unites ElasticSearch and Neo4j, representing an entity in both databases.
    Entities are available by `uuid` or tuple `source, source_id`
    """

    def __init__(self):
        self._elastic_doc: DocumentElastic
        self._neo4j_doc: DocumentNeo

     def __getattr__(self, name):
         if name not in ('_elastic_doc', '_neo4j_doc'):
             try:
                 return getattr(self._elastic_doc, name)
             except AttributeError:
                 pass
             return getattr(self._neo4j_doc, name)
         return None
```

ES + Neo4j - some methods

```
@staticmethod
def get_by_source_id(source, source_id):
    doc = Document()
    doc._elastic_doc = ElasticQuery.get_doc_by_source_id(source, source_id)
    doc._neo4j_doc = NeoQuery.get_doc_by_source_id(source, source_id)
    return doc

@staticmethod
def get_by_uuid(uuid):
    doc = Document()
    doc._elastic_doc = ElasticQuery.get_doc_by_uuid(uuid)
    doc._neo4j_doc = NeoQuery.get_doc_by_uuid(uuid)
    return doc

def is_up_to_date(self, last_updated: datetime):
    return self._elastic_doc.is_up_to_date(last_updated)
```

Elasticsearch

- So far we:
  - Discovered Dialogflow
  - And how to use Elasticsear + Neo4j together

Elasticsearch-dsl - query examples

- Filtering by field and limiting the results:

```
DocumentElastic\
    .search(index='documents', using=elastic.client)\
    .query('bool', filter=[Q('term', source=source)])\
    .fields(['source_id'])[:limit]\
    .execute()
```

- Filtering by field and limiting the results:

```
DocumentElastic.get(id=uuid, using=elastic.client, index='documents')
```

Elasticsearch - word order

- How to incorporate word order in our text queries?

- Query: “**bookings last year**”
- 1) “Average amount of **bookings** for **last year**”
- 2) “**Last bookings** of the previous **year**”

Elasticsearch - word order

- 2 separate analyzed fields:
  - One for **separate words**
    - *“last”, “year”*
  - One for **bi-grams** and **tri-grams**
    - *“last year”, “number of bookings”*

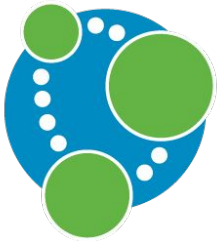
Elasticsearch - analyzers

```
root = analyzer(
    'root',
    type='custom',
    tokenizer='standard',
    char_filter=['html_strip'],
    filter=[english_possessive_stemmer, synonyms_case_sensitive, 'lowercase',
            synonyms_lowercase, english_stop, english_stemmer])

shingles = analyzer(
    'shingles',
    type='custom',
    tokenizer='standard',
    char_filter=['html_strip'],
    filter=[english_possessive_stemmer, synonyms_case_sensitive, 'lowercase',
            synonyms_lowercase, english_stop, english_stemmer, shingle_filter])

default_fields = {
    'default': Text(analyzer=root),
    'shingles': Text(analyzer=shingles)
}
```

Neo4j

- Uses SQL-inspired language for queries: *Cypher*
- Has some problems with iterative computations

Neo4j - Graph statistics

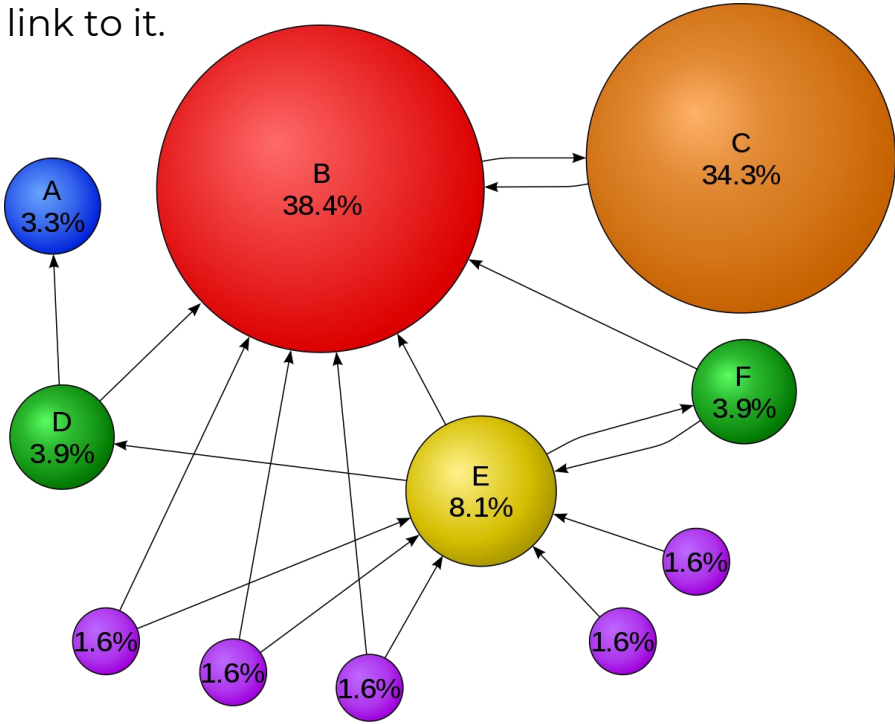


- Count the views and amount of distinct people viewed:

```
db.cypher_query('''
       MATCH (doc:DocumentNeo) - [rel:CONSUMED_BY] - (user:UserNeo) # filtering nodes
       WITH doc, sum(rel.times_viewed) AS views,                    # aggregating
            COUNT(DISTINCT user.email) AS people_viewed
       SET doc.views = views, doc.people_viewed = people_viewed     # updating
   ''')
```

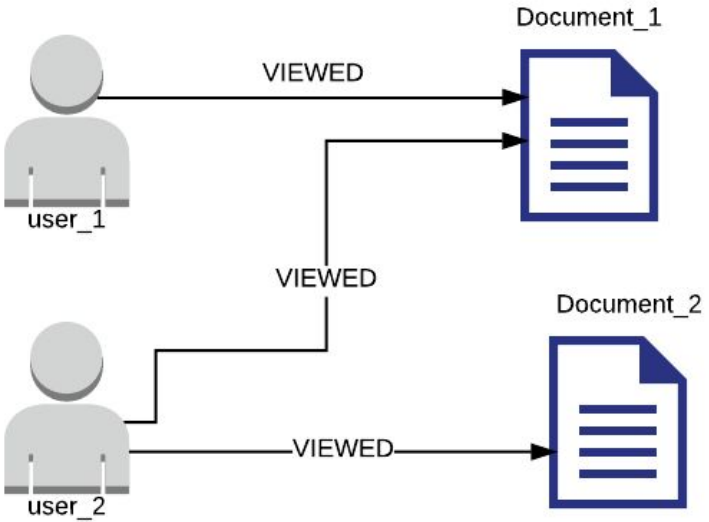
PageRank

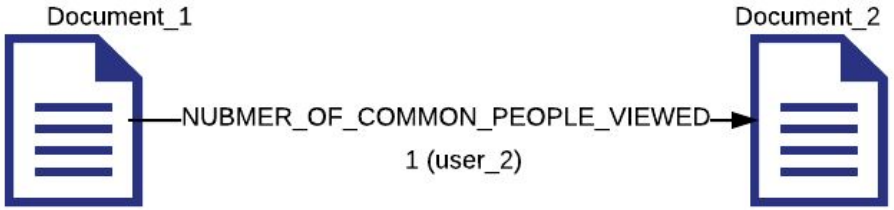
- Is a mathematical formula that judges the “value of a page” by:
  - quantity and quality of other pages that link to it.
- Still used in Google search engine
- Simple and cool



Neo4j - trick to project bipartite graph

- How do we calculate PageRank, if we have bipartite graph?





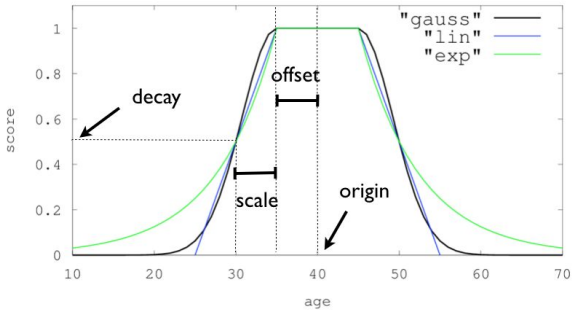
Neo4j - query for bipartite graph

- To project bipartite graph:

```
db.cypher_query('''
   CALL apoc.periodic.iterate(     # using apoc plugin
   'MATCH (doc1:DocumentNeo)<-[:CONSUMED]-(UserNeo)-[:CONSUMED]->(doc2:DocumentNeo)
    RETURN doc1, doc2, count(*) as common_users',  # match query
   'MERGE (doc1)-[r:AMOUNT_OF_COMMON_PEOPLE]->(doc2)
       SET r.common_users = common_users',  # update query
   {batchSize:10, parallel:true})
''')
```

Elasticsearch - Function score

- *BASIC_SCORE* * **ln(page_rank) * log10(number_of_views) * Gauss_filter**
  - **ln(page_rank)**
    - 0 <*page_rank* < 10
    - 1 < *multiplier* < 3
  - **log10(number_of_views)**
    - 0 < *number_of_views* < 10000
    - 1 < *multiplier* < 3
  - **Gauss_filter**
    - Penalize docs which were updated > 1 year ago



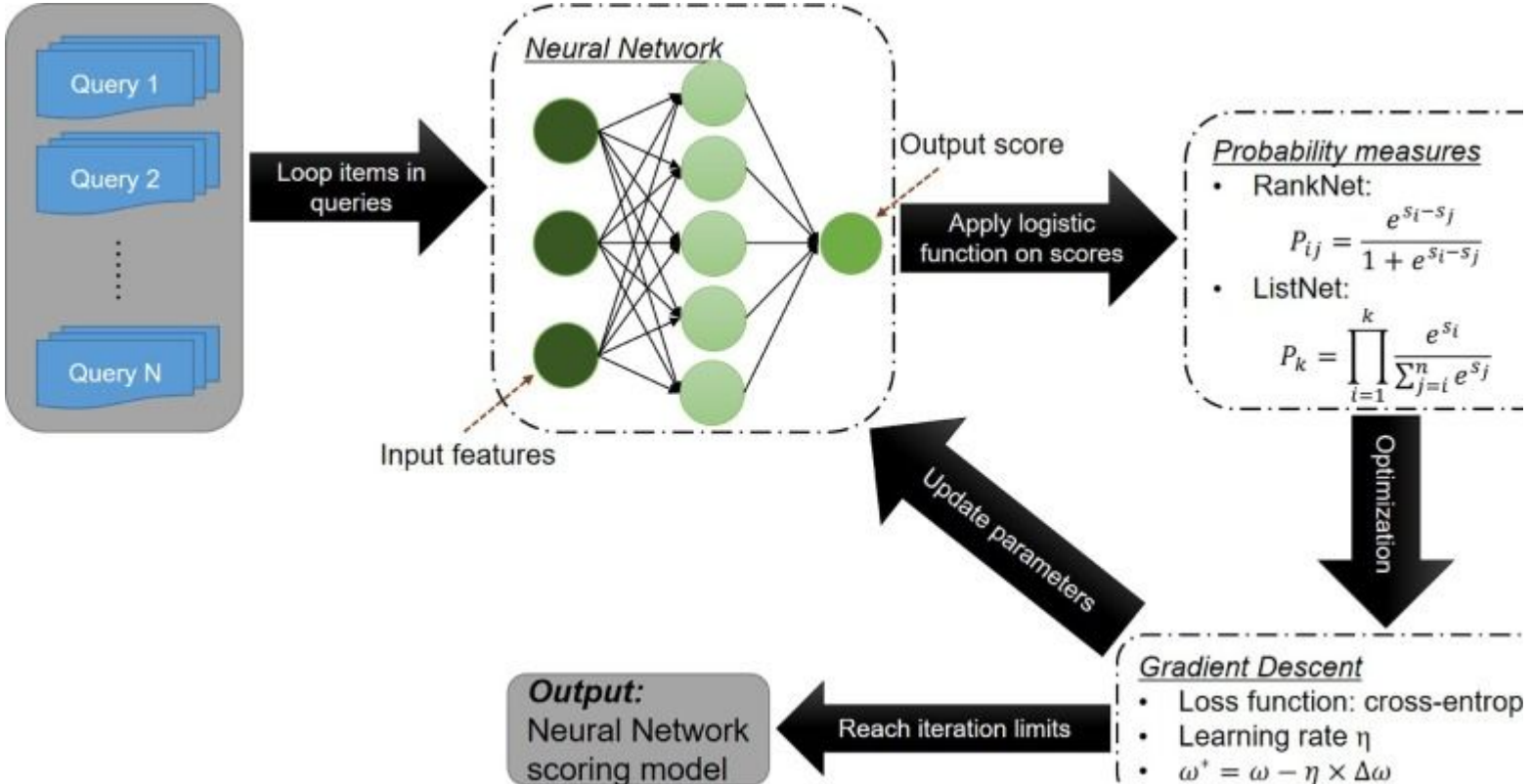
Elasticsearch-dsl - Function score

```
query = FunctionScore(
    query=query,
    functions=[
        dict(   # Gauss multiplier
            gauss={
                'updated_at': {
                    'origin': datetime.datetime.utcnow().strftime('%Y-%m-%dT%H:%M:%S'),
                    'offset': '365d',
                    'scale': '700d'
                }
            }
        ),
        dict(   # Multipliers from graph features
            script_score=dict(script=dict(
                source=score_script,
                params=dict(
                    pg_offset=1,
                    pg_multiplier=1,
                    vw_offset=1,
                    vw_multiplier=0.2
                ),
            ))])
```

Are the results good?

- Depends on how you use Alfred

Future plans - add *Ranking model*



Future plans - Dataflow graph + alerts

Future plans

- Gather feedback and statistics from the users (using Chatbase)
- Change graph database from Neo4j
- Implement own NLP model instead of Dialogflow

Questions?

Artur 10:38 AM

you are fired!

Alfred APP 10:38 AM

Oh no! My best work is yet to come.