

Natural Language Interface to Knowledge Graph



Vaticle



NLI₂KG

Enzo FENOGLIO

Emerging Technology & Incubation – Cisco France
Research Fellow in Computer Science at UCL



TypeDB

Berlin - Nov 17, 2022

The Problem



How to represent
knowledge for sparse
heterogeneous datasets ?



How to query for
knowledge discovery
w/o knowing a query
language?

The Solution

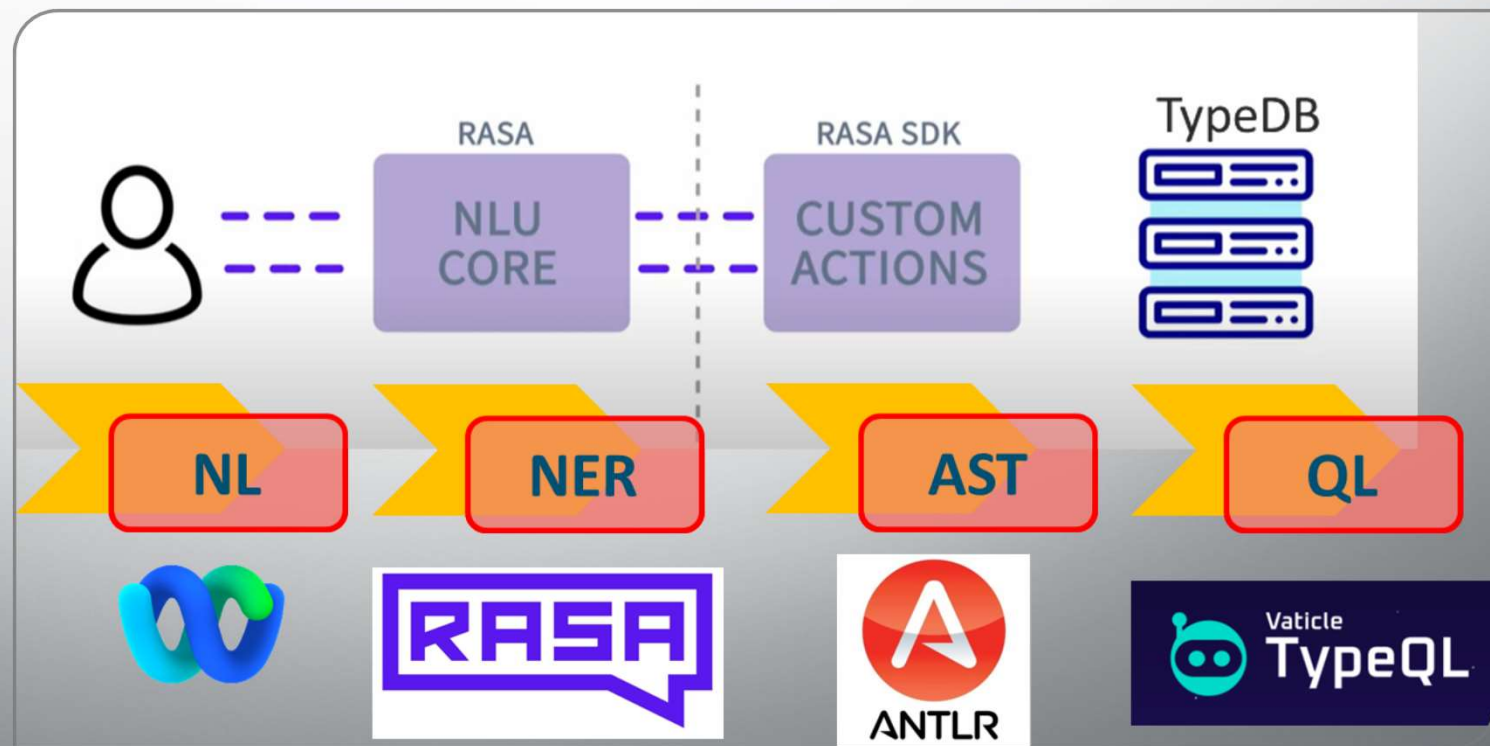


**Store data + ontology
in a Knowledge Graph**



**Use NL to query the KG
using the built-in reasoner**

The bridge from NL to KG



Raspberry Pi4 SDRAM 8Go
Ubuntu 22.04.01 LTS



Power

typeDB Server on ARM64
192.168.1.24:1729

Rasa Server
192.168.1.23:5005
Tensorflow v2.3 w/o AVX

HDMI_1

HDMI_2

USB Keyboard

Power

**Wireless
Mouse**

Intel Celeron J3455
Quad Core Micro PC
Windows 10, SDRAM 8Go

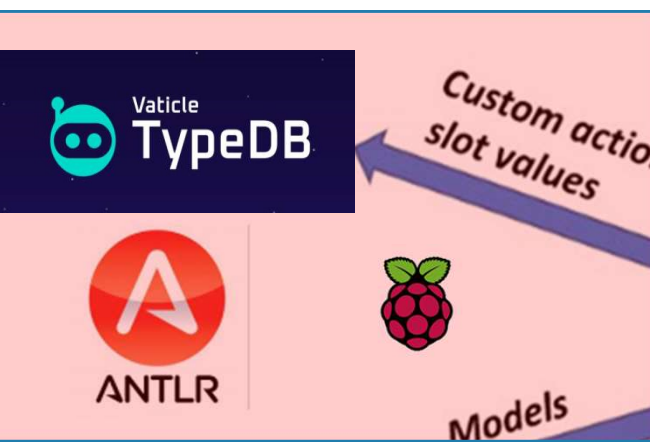


Useful references for pkg(s) installations

- <https://qengineering.eu/install-tensorflow-on-raspberry-64-os.html>
- <https://www.eggwall.com/2020/09/compiling-tensorflow-without-avx.html>
- https://en.wikipedia.org/wiki/Advanced_Vector_Extensions
- <https://repo1.maven.org/maven2/com/google/ortools/ortools-linux-aarch64/9.4.1874/ortools-linux-aarch64-9.4.1874.jar>



Ngrok: serve localhost as public URL



Custom actions & slot values

Models

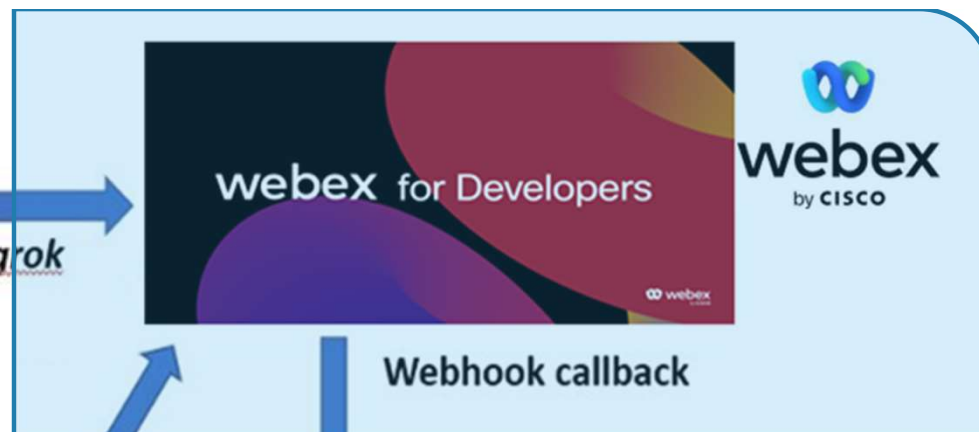


NiPOGi

8GB Intel Celeron J3455
TensorFlow 2.3 w/o AVX

Public URL from ngrok

Token from
credentials.yml

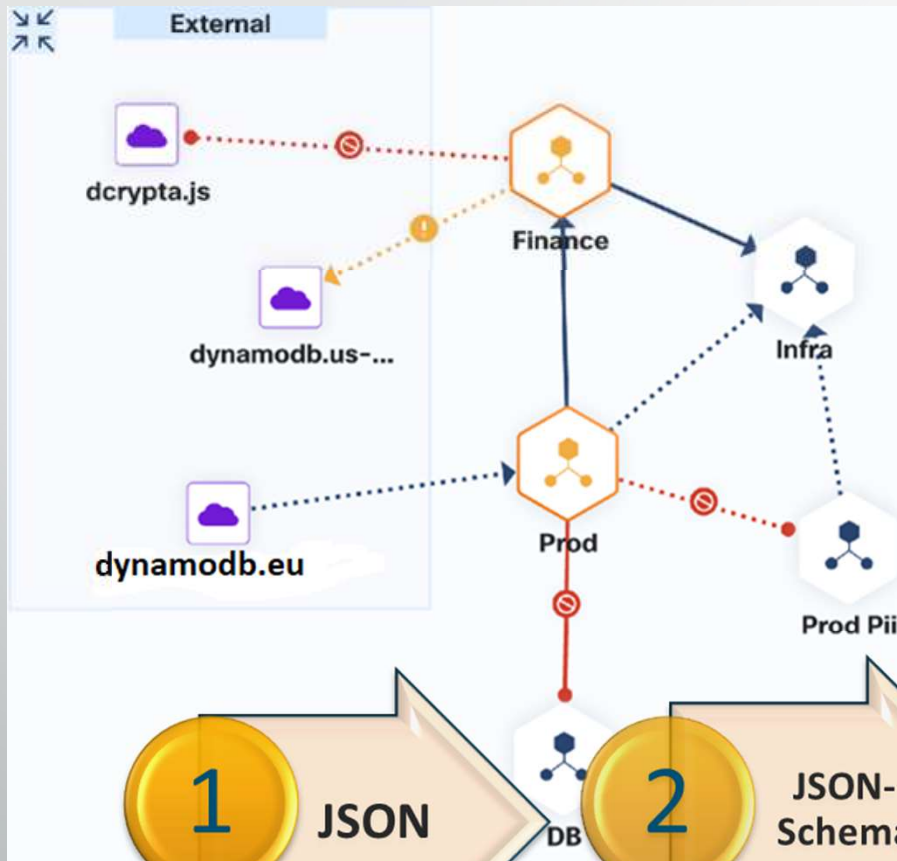


Link between RASA Server
and Webex Teams

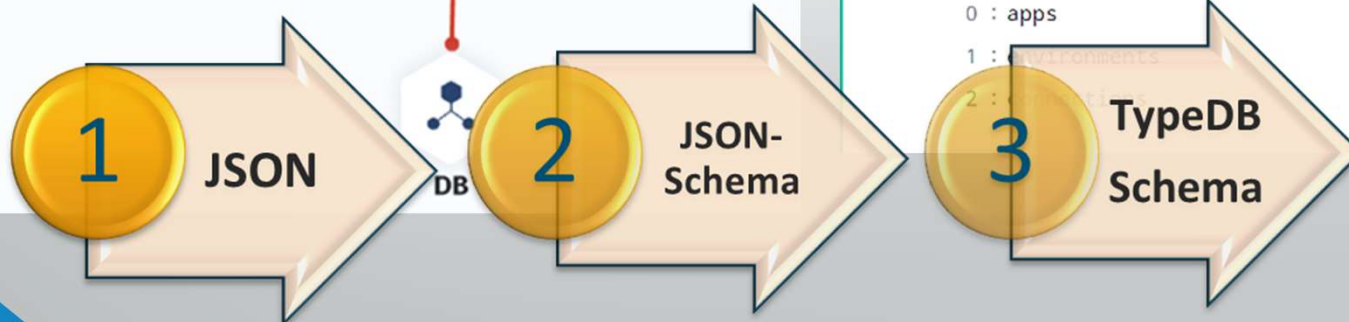


Webex Teams Front End

Data and Ontology



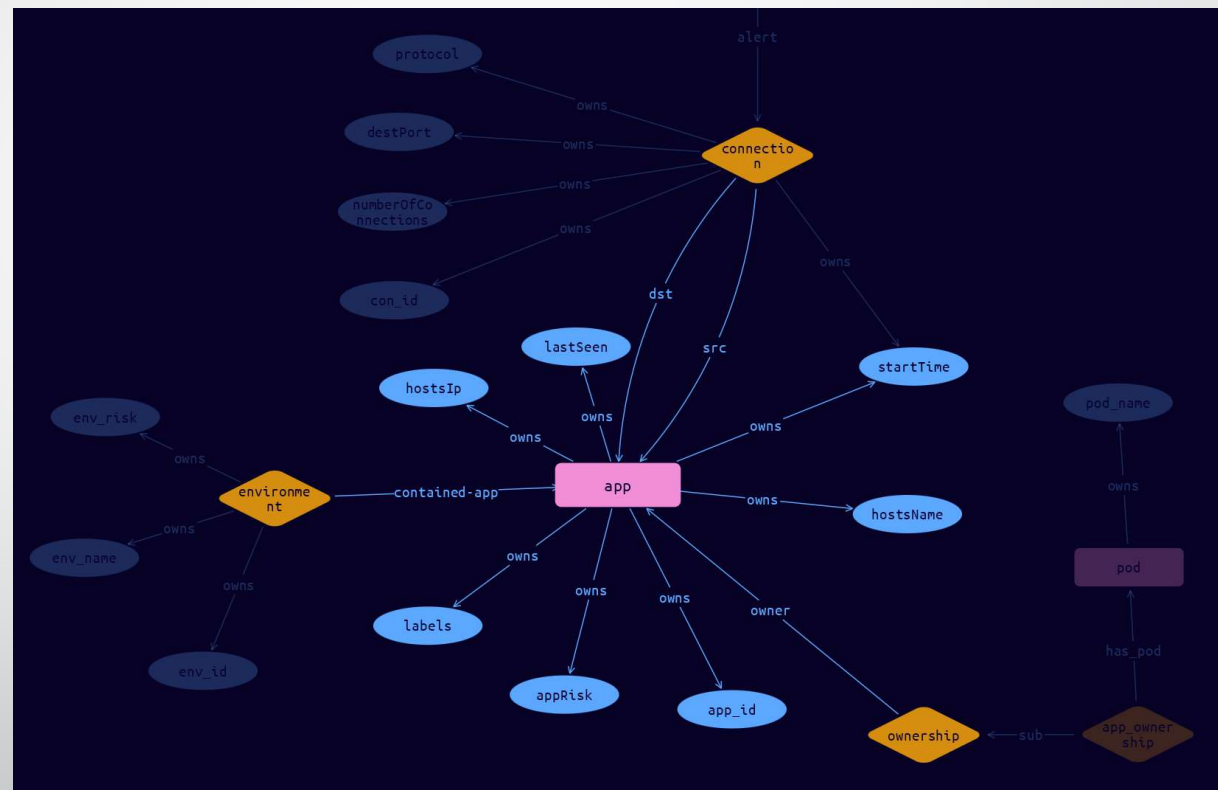
```
object ▶ required ▶  
▼ object {4}  
  $schema : http://json-schema.org/draft-04/schema#  
  type : object  
  ▼ properties {3}  
    ▼ apps {2}  
      type : array  
      ▶ items [1]  
    ▼ environments {2}  
      type : array  
      ▶ items [1]  
    ▼ connections {2}  
      type : array  
      ▶ items [1]  
  ▼ required [3]  
    0 : apps  
    1 : environments  
    2 : connections
```



The Knowledge Graph Schema

```
app sub entity,  
  owns app_id,  
  owns lastSeen,  
  owns startTime,  
  owns appRisk,  
  owns hostsIp, owns hostsName,  
  owns labels,  
  plays connection:src,  
  plays connection:dst,  
  plays environment:contained-app,  
  plays app_ownership:owner;
```

```
env_risk sub attribute, value string;  
env_id sub attribute, value string;  
environment sub relation,  
  owns env_name,  
  owns env_risk,  
  owns env_id,  
  relates contained-app;
```



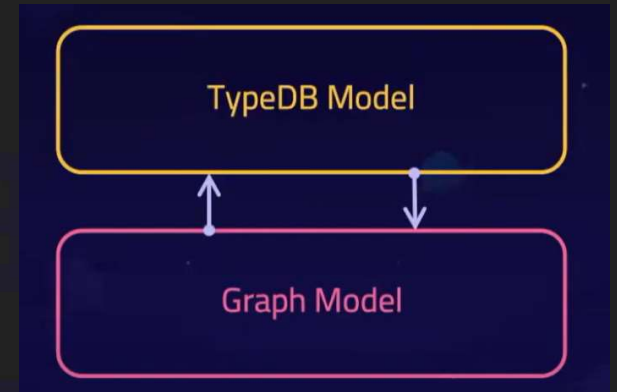
Why TypeDB ?

Open Source: Great documentation & Community support

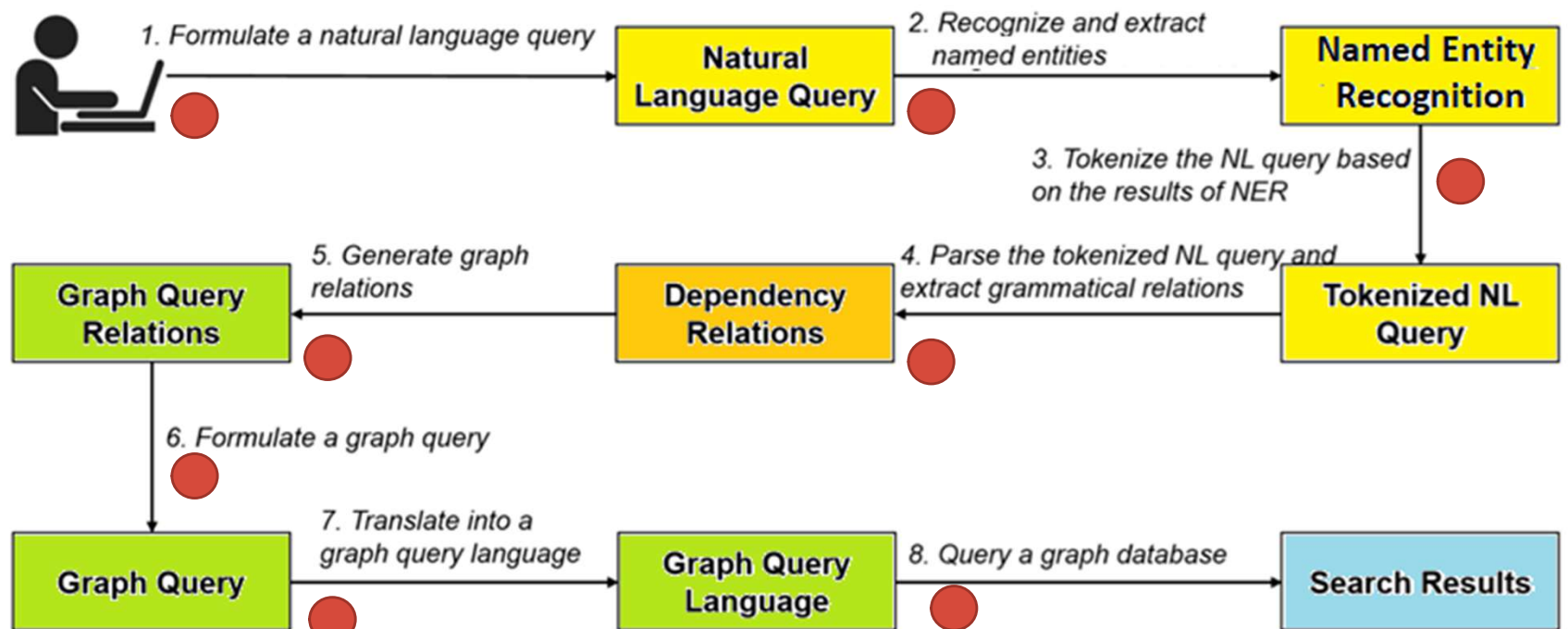
Implements a concept level entity-relationship model

Implements hypergraphs data model with schema/logical verification

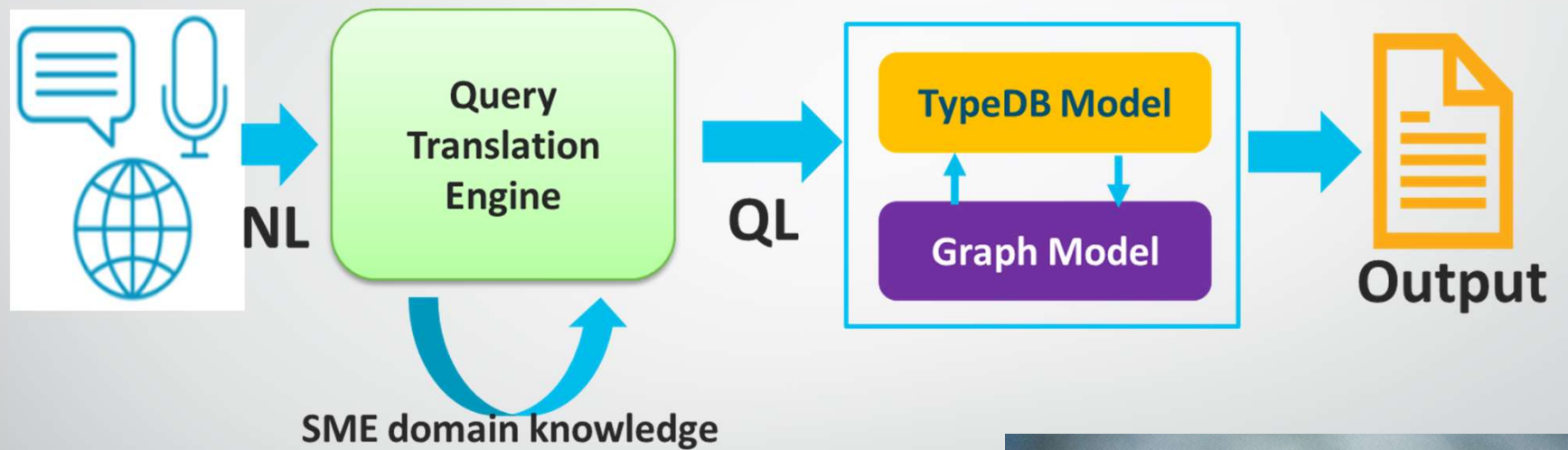
Provides a reasoning engine for reasoning at query time



Query Translation Pipeline



The Translation Pipeline from NL to QL



The translation from NL to QL is **not a classification problem**



The method: working details



STEP 1 (NER extraction)

- NER semantic parsing is done with a [RASA pipeline](#) and a simplified scheme (ontology) structure for semantic extraction in the form of a **JSON** variable:

```
[{"object_type":<value>,"slot":<value>,"role":<value>,"value":<value>},  
{ "object_type":<value>,"attribute":<value>,"role":<value>}, { . . . }]
```

- "object_type" : a distinctive concept (nodes) in the graph base
- "slot" : a qualified attribute assigned to a concept
- "attribute" : a generic attribute assigned to a concept
- "role" : a relationship modifier such as ["from", "to", "max", "min", ...]
- "value" : a qualified instance of a slot


```

language: en
pipeline:
  - name: "WhitespaceTokenizer"      -> tokenizer using whitespaces
  - name: "RegexFeaturizer"         -> creates features for NER and intent classification
  - name: "LexicalSyntacticFeaturizer" -> Creates lexical/syntactic features for NER
  - name: "CountVectorsFeaturizer"   -> counts whole words
  - name: "CountVectorsFeaturizer"   -> counts sub-sequence of n-gram
  - analyzer: "char_wb"
  - min_ngram: 1
  - max_ngram: 5
  - name: "DIETClassifier"           -> Dual Intent Entity Transformer
  - entity_recognition: true
  - epochs: 200
  - constrain_similarities: True
  - name: "EntitySynonymMapper"     -> Maps synonymous entity values

```

<https://www antlr.org/>

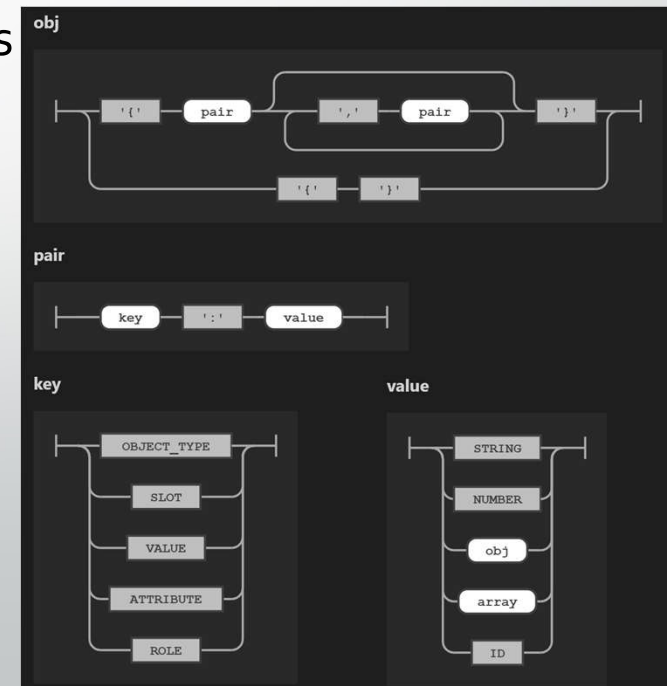


STEP 2 (AST generation)

- A parse generator (**ANTLR**) is used to build an intermediate abstract syntax tree (**AST**)
- **ANTLR: ANother Tool for Language Recognition**, is a Java-based tree parser generator that allows users to define grammar and **AST**-parsers for [specific languages](#).

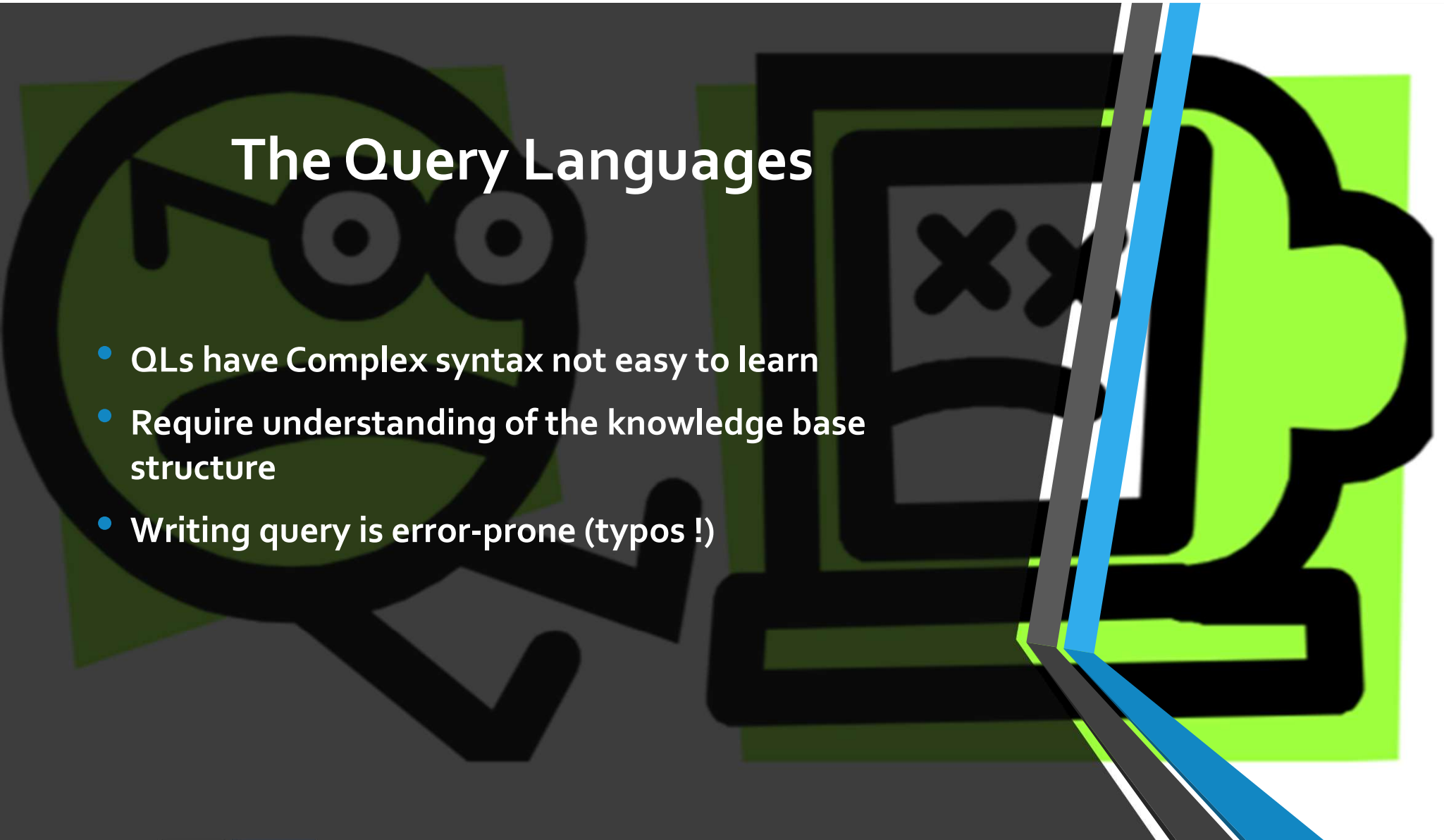
STEP 3 (Query Generation)

- The lexer and parser generated from the grammar are used to translate the **JSON** array derived from the **NER** extractor into a corresponding **TypeQL** query.



The Query Languages

- QLs have Complex syntax not easy to learn
- Require understanding of the knowledge base structure
- Writing query is error-prone (typos !)



Querying a Knowledge Graph is difficult ...

- Telcos generate high volumes of service requests,
- BUT it is extremely difficult to find relevant content that can help Technical Assistance Center (TAC) engineers to solve customers' problems quickly.

Q1: Which is the application with more connections?

```
> match $x1 isa app; (src:$x1, dst:$anyone) isa  
connection, has numberOfConnections $x3; max $x3;
```

```
3.4030913E7
```

```
>match $x1 isa app, has app_id $x2; (src:$x1,  
dst:$anyone) isa connection, has numberOfConnections  
34030913; (contained-app:$x1) isa environment, has  
env_name $x4; get $x2, $x4;
```

```
{ $x2 "787a49fb-a5d4-552e-a7f3" isa app_id; $x4 "Prod" isa env_name; }
```

Q1: Which is the application with more connections?

```
> match $x1 isa app; (src:$x1, dst:$anyone) isa  
connection, has numberOfConnections $x3; max $x3;
```

3.4030913E7

```
> match $x1 isa app, has app_id $x2; (src:$x1,  
dst:$anyone) isa connection, has numberOfConnections  
34030913; (contained-app:$x1) isa environment, has  
env_name $x4; get $x2, $x4;
```

```
{ $x2 "787a49fb-a5d4-552e-a7f3" isa app_id; $x4 "Prod" isa env_name; }
```


Why not use a **Conversational AI** assistant to access the **Knowledge Graph** using **Natural Language** queries ?

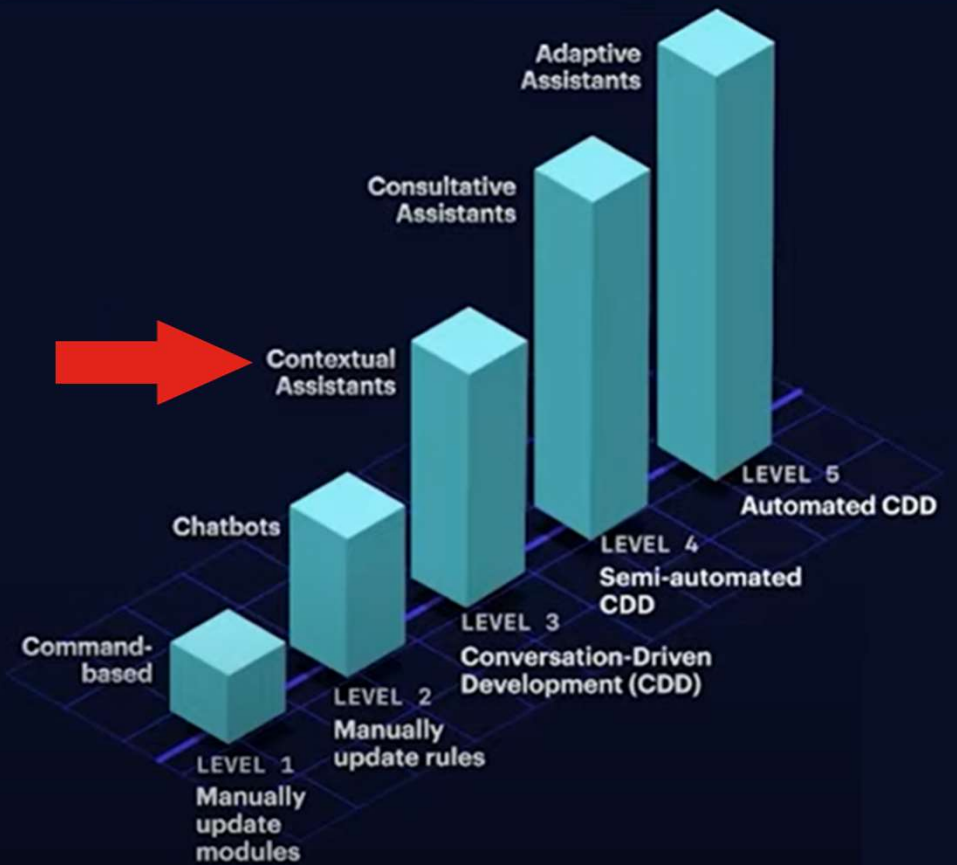




Motivations for a Natural Language Interface (NLI)

- It is impossible to anticipate all the things users could say...
... **BUT** users tell in their own words exactly what they want.
- NLI can match natural language questions with formal queries for easy access to information stored in a knowledge base
- A KG organizes and integrates data according to an ontology and applies a reasoner to derive new knowledge
- Knowledge reasoning over KGs aims to identify errors and infer new conclusions from existing data

Conversational AI Levels



Advantages of NLI



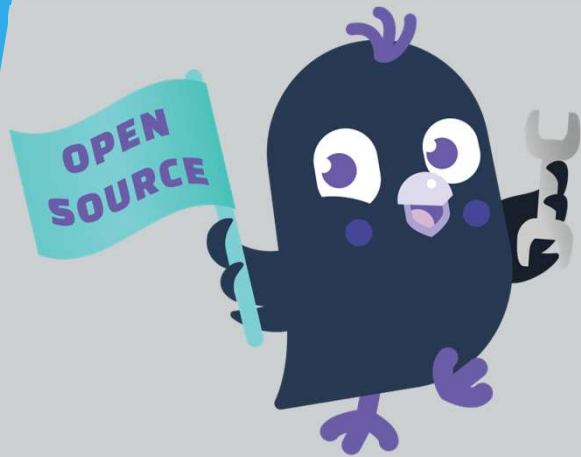
NL queries make KG more accessible to the average user



benefit for analysts with mixed backgrounds



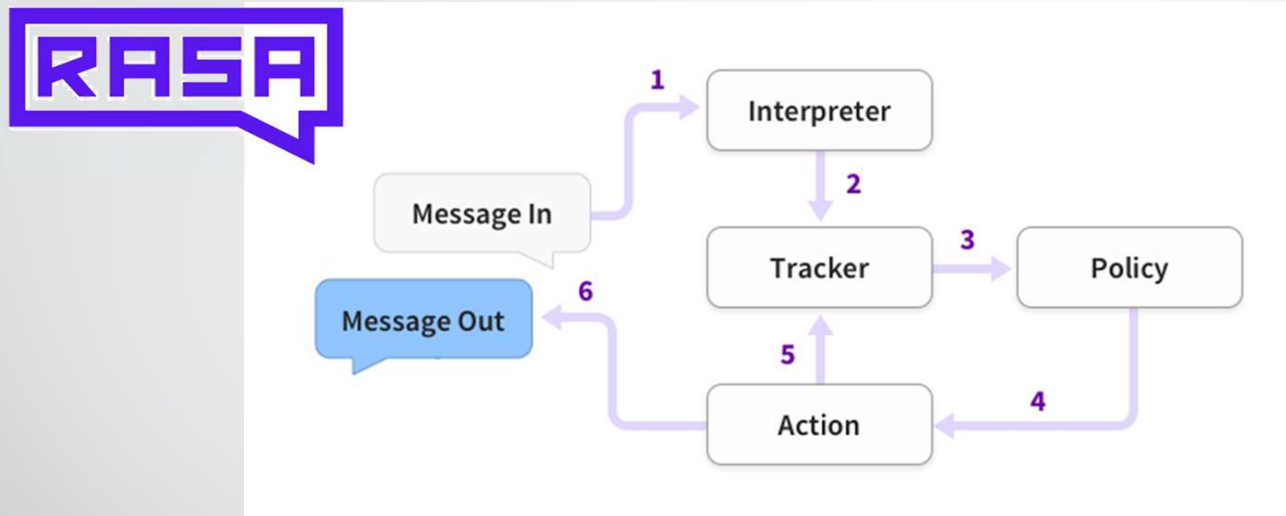
allow end-users to focus on other tasks and access KGs as they want



- **RASA** is a conversational AI framework for building *contextual assistants*.
- It provides E2E tools to build advanced NLI interacting with human users.
- We use RASA to train an NLU model for multi-intent classification and **Named Entity Recognition (NER)**
- **NER** is the task of identifying the entities that appear in a text in multiple contexts

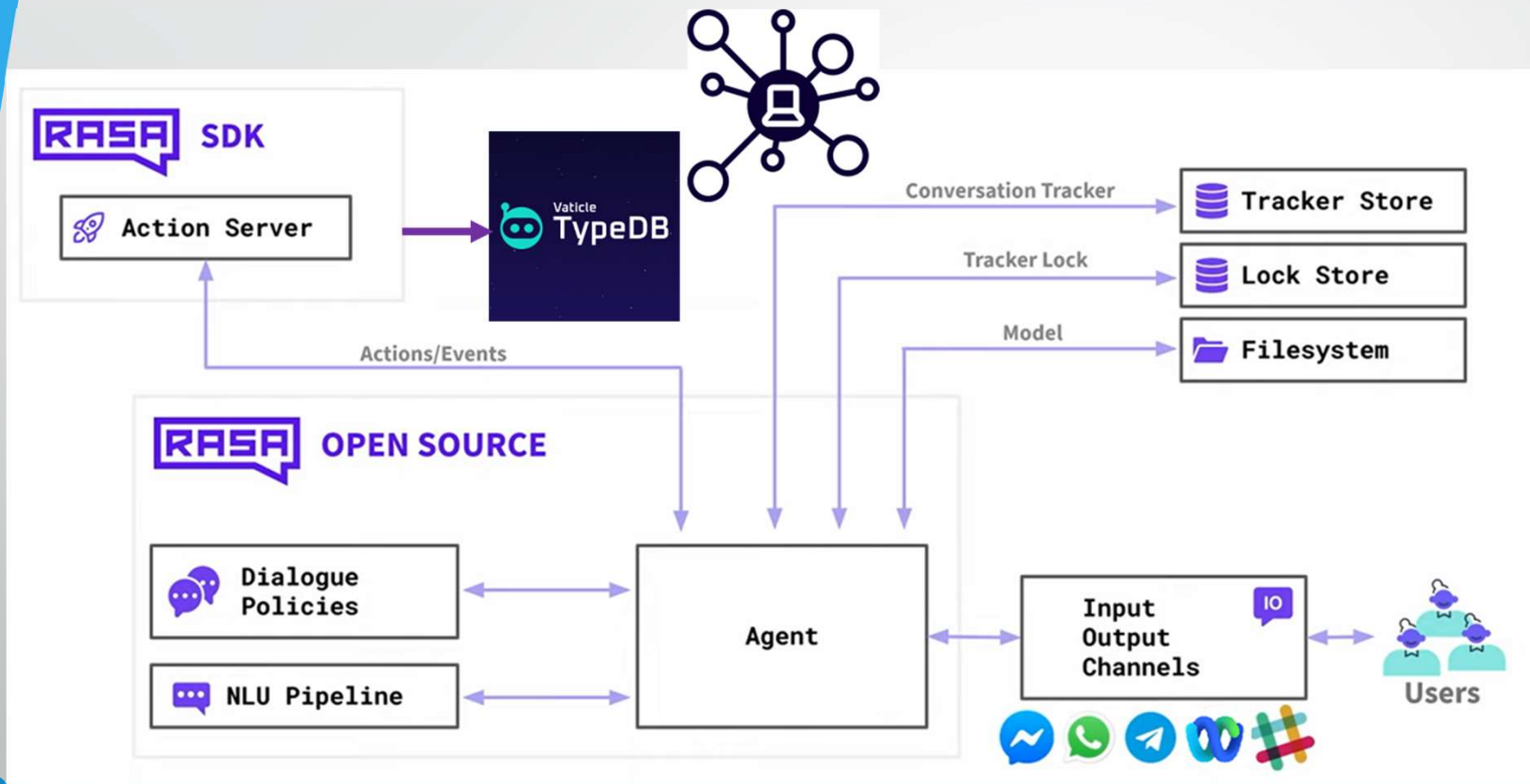
<https://rasa.com>

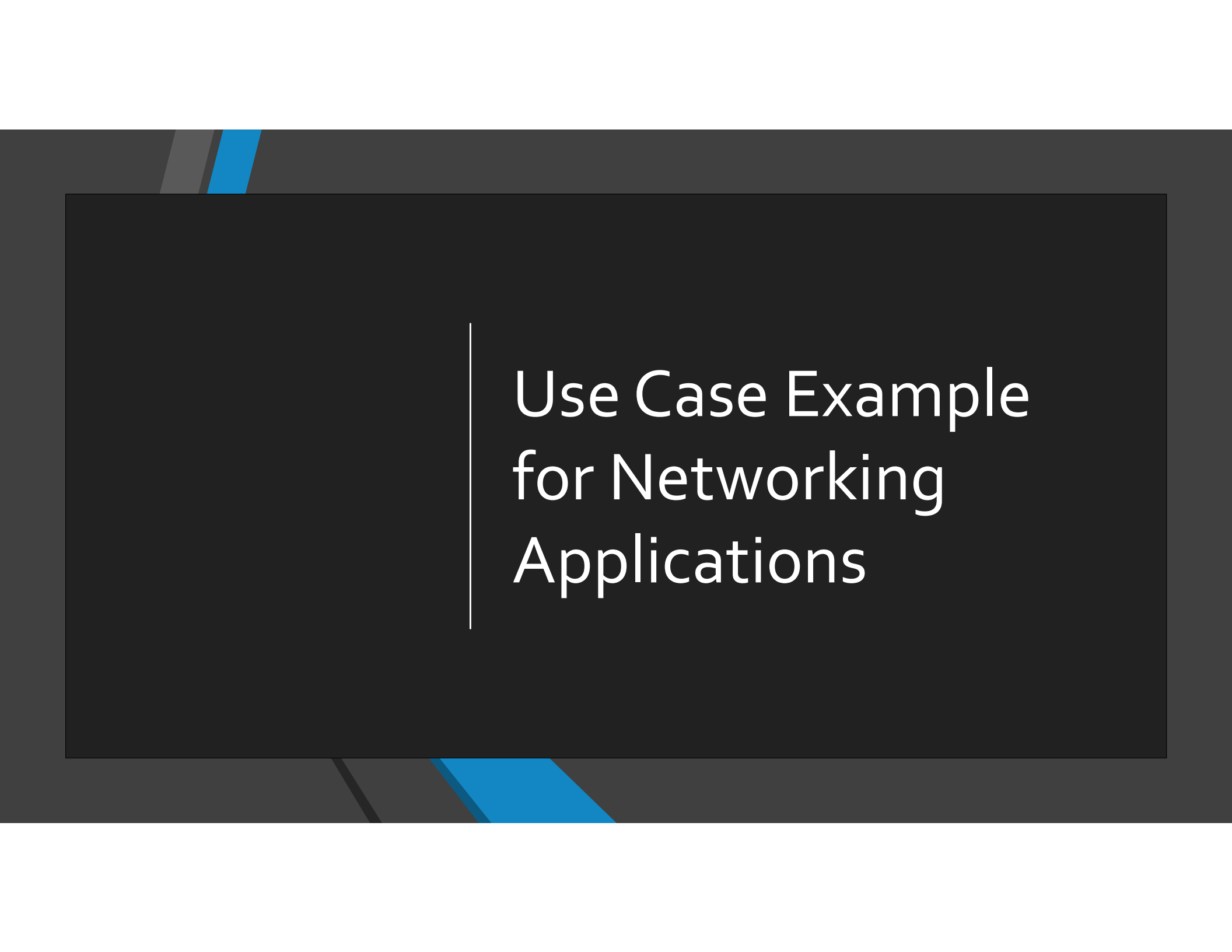
RASA Architecture



```
- rule: query knowledge base
  steps:
  - intent: query_knowledge_base
  - action: action_check_typos
  - action: action_query_knowledge_base
```

RASA meets TypeDB





Use Case Example for Networking Applications

**UC#1 : FAQs
and
predefined
Q/As**

- Show the attributes of the environment
- Display the knowledge base schema

**UC#2 : Query
an object and
its attributes**

- Show the environments with names
- Show applications with app risk and app ID

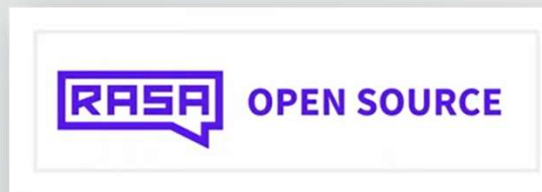
UC#3 :
Query multiple
objects and
attributes

- Show applications in Finance with pod names
- Show links from Prod to Finance

UC#4 :
Computation

- What application has more connections ?
- What application in Finance has higher risk ?

RASA / Cisco Webex BOT API solution



- access token
- room ID
- webhook

```
$ typedb server
```

-to start the typedb server

```
$ ngrok http --region=eu 5005
```

-to redirect local traffic to port 5005

```
$ python getWebhook.py
```

-to create the Webhook for RASA endpoints

```
$ rasa run --enable-api
```

-to start RASA server on port 5055 and enable APIs

```
$ rasa run actions
```

-to allow communication from RASA Custom Action server to **TypeDB**



Quorra the Bot

<https://disney.fandom.com/wiki/Quorra>



Vaticle
TypeDB



Vaticle
TypeQL



Annotated Example

❏ Input Query (NL) :

```
show links from proad to fananze with pod names
```

NER extraction

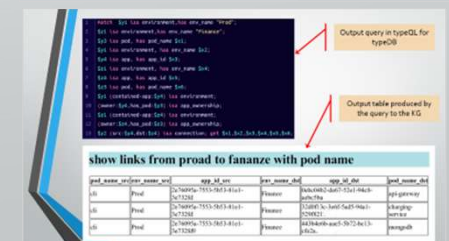
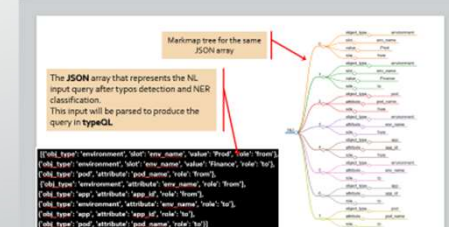
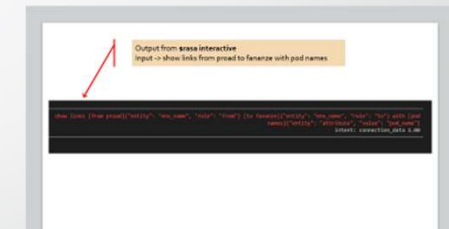
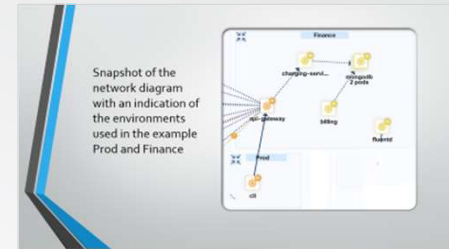
❑ Typos correction

We use [Levenshtein Distance](#) to calculate the differences between sequences based on the [fuzzywuzzy](#) python package

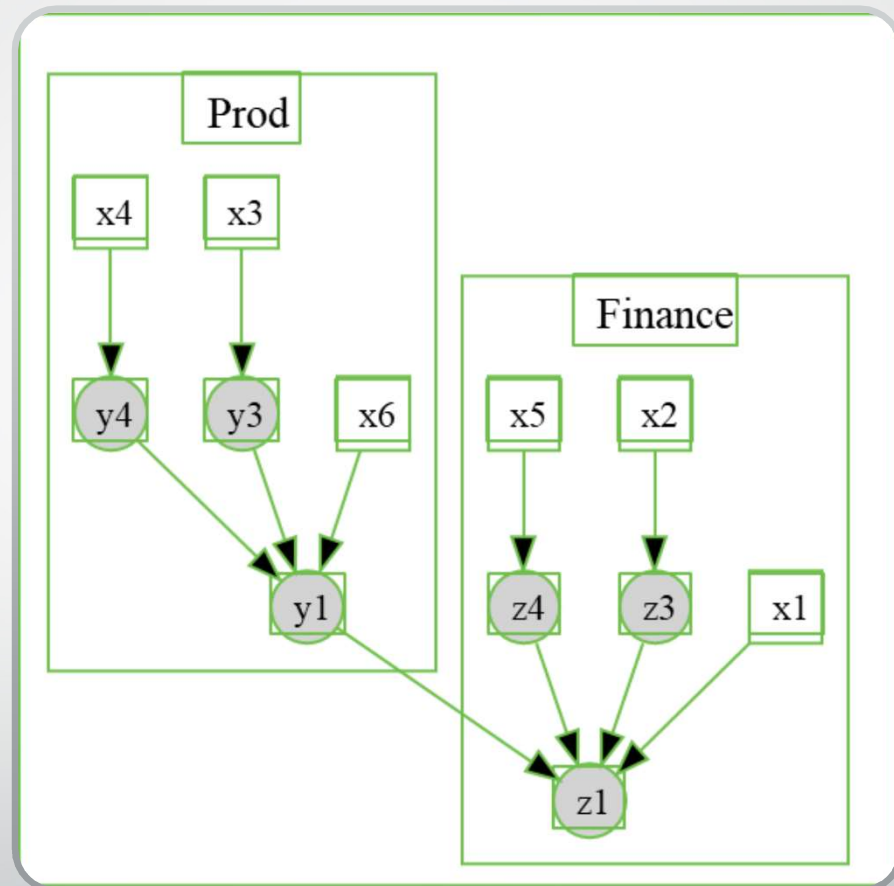
NER integration with implicit references

❑ TypeQL query generation

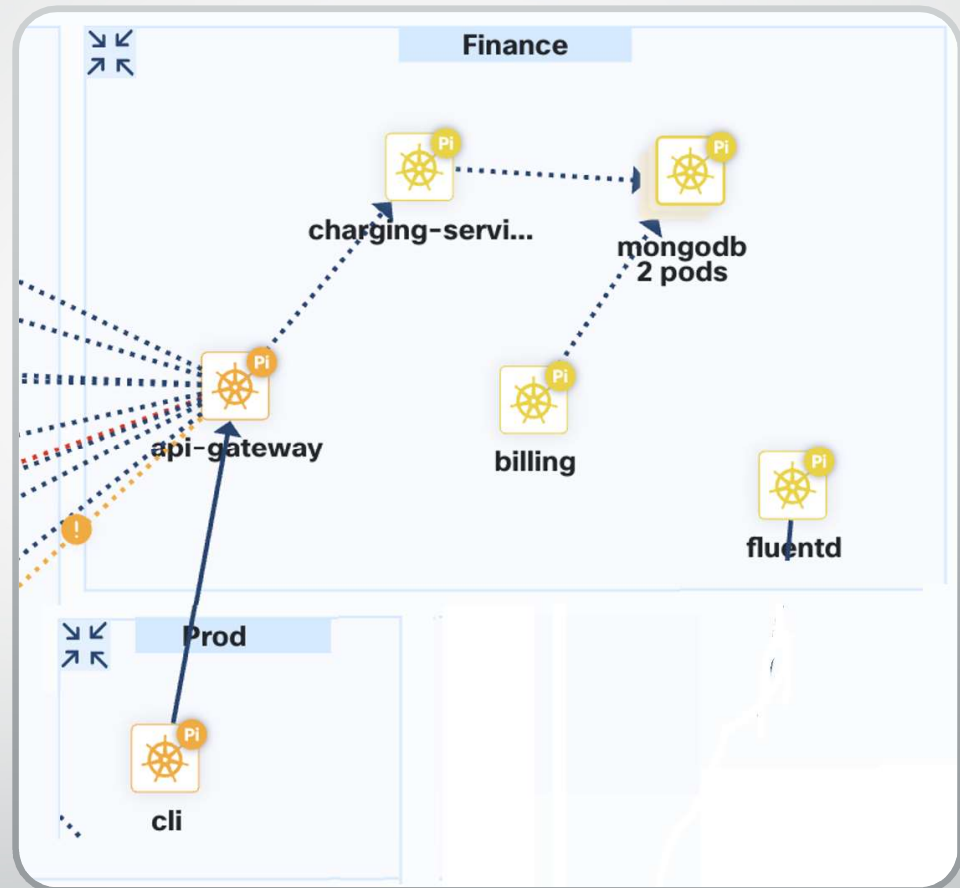
- Execute inferences



Graph representation
of the output query



Snapshot of the network diagram with an indication of the environments used in the example Prod and Finance





Output from **\$rasa interactive**

Input -> show links from proad to fananze with pod names

```
show links [from proad>{"entity": "env_name", "role": "from"} [to fananze>{"entity": "env_name", "role": "to"} with [pod
names>{"entity": "attribute", "value": "pod_name"}
intent: connection_data 1.00
```



```

1 match $y1 isa environment, has env_name "Prod";
2 $z1 isa environment, has env_name "Finance";
3 $y3 isa pod, has pod_name $x1;
4 $y1 isa environment, has env_name $x2;
5 $y4 isa app, has app_id $x3;
6 $z1 isa environment, has env_name $x4;
7 $z4 isa app, has app_id $x5;
8 $z3 isa pod, has pod_name $x6;
9 $y1 (contained-app:$y4) isa environment;
10 (owner:$y4, has_pod:$y3) isa app_ownership;
11 $z1 (contained-app:$z4) isa environment;
12 (owner:$z4, has_pod:$z3) isa app_ownership;
13 $y2 (src:$y4, dst:$z4) isa connection; get $x1, $x2, $x3, $x4, $x5, $x6;

```

Output query in typeQL for typeDB

Output table produced by the query to the KG

show links from proad to fananze with pod name

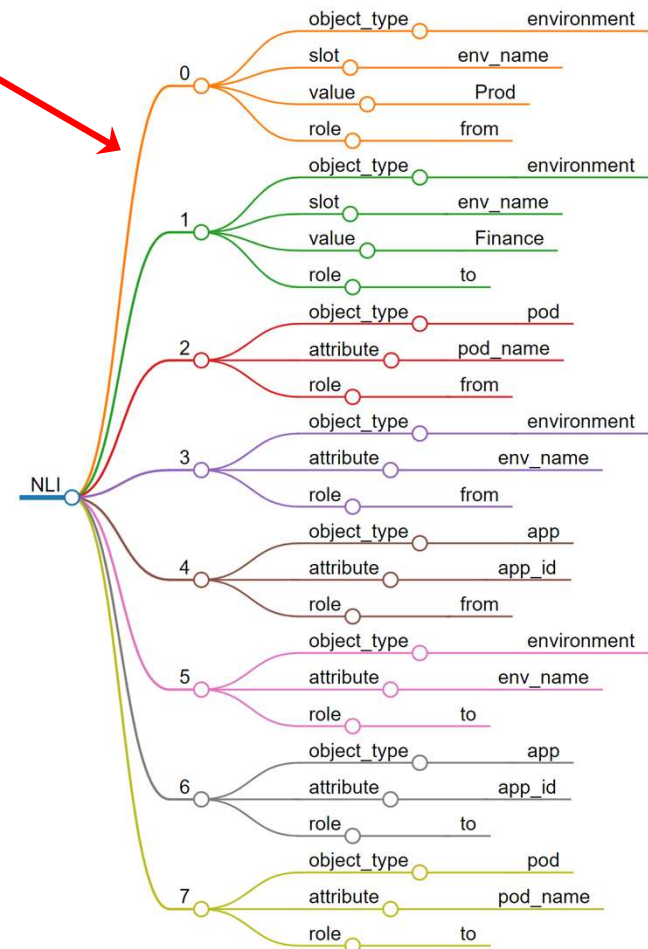
pod_name_src	env_name_src	app_id_src	env_name_dst	app_id_dst	pod_name_dst
cli	Prod	2e76095a-7553-5b53-81e1-3e732fd.	Finance	0ebc04b2-de67-52e1-94c8-aebc5ba	api-gateway
cli	Prod	2e76095a-7553-5b53-81e1-3e732fd.	Finance	32d0f13c-3e6f-5ad5-94a1-529f821.	charging-service
cli	Prod	2e76095a-7553-5b53-81e1-3e732fd9	Finance	443b4e6b-aae5-5b72-bc13-cfe2a.	mongodb

Markmap tree for the same
JSON array

The **JSON** array that represents the NL
input query after typos detection and NER
classification.

This input will be parsed to produce the
query in **typeQL**

```
[{'obj_type': 'environment', 'slot': 'env_name', 'value': 'Prod', 'role': 'from'},  
{ 'obj_type': 'environment', 'slot': 'env_name', 'value': 'Finance', 'role': 'to'},  
{ 'obj_type': 'pod', 'attribute': 'pod_name', 'role': 'from'},  
{ 'obj_type': 'environment', 'attribute': 'env_name', 'role': 'from'},  
{ 'obj_type': 'app', 'attribute': 'app_id', 'role': 'from'},  
{ 'obj_type': 'environment', 'attribute': 'env_name', 'role': 'to'},  
{ 'obj_type': 'app', 'attribute': 'app_id', 'role': 'to'},  
{ 'obj_type': 'pod', 'attribute': 'pod_name', 'role': 'to'}]
```



```
TypeDBOptions.core()  
opts.infer = True
```

Inference shall be enabled
programmatically

The transitive rule
implemented in the typeDB
schema

```
rule transitive-connections:  
when {  
    (src:$x, dst:$y) isa connection;  
    (src:$y, dst:$z) isa connection;  
} then {  
    (src:$x, dst:$z) isa connection;  
};
```

Takeaways

- allow non-technical users to query a KG with NL. All the advantages of KG reasoner w/o the complexities of a query language.
- Track information collected from users during the dialogue for dynamic dashboard presentations
- Useful for intelligent troubleshooting systems, e.g. RCA for TAC engineers
- **TypeDB** and **RASA** allow easy integration with WebexTeams REST APIs and other clients (Facebook, Telegram, Twilio, etc.)

