

# Semantic Event Detection in SmartChainDB

CSC591 - Real Time Data Processing



## Team Members


- Darpan Dodiya (dpdodiya)
- Ravinder Singh Rajpal (rkrajpal)
- Shantanu Sharma (ssharm34)
- FNU Vivek (vvivek)

# The Project

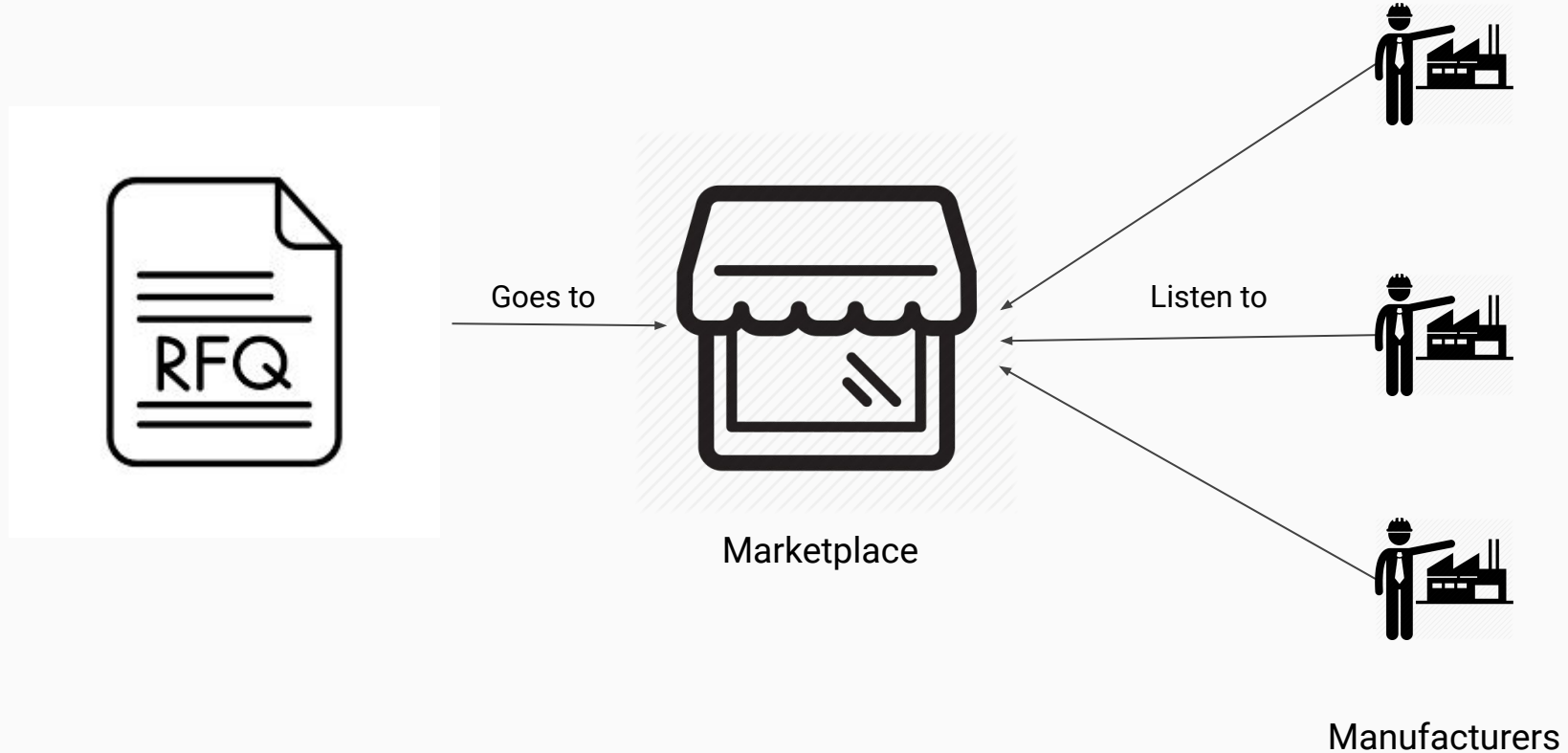
- Semantic Event Detection in SmartChainDB
- Related to product manufacturing life cycle
- Our focus is on the RFQ - request for quote
- How to map complex and vague RFQ events into an appropriate publish/subscribe event detection model?

## Background - RFQ Example

```
{
  "id": "0ce27a8bbs34vd3edadb6a719..."
  "version": "2.0"
  ...
  "operation": "REQUEST_FOR_QUOTE"
  "metadata" :
  {
    "Part Name/Description" : "Phone cover",
    "Quantity" : "10",
    "Material" : "PolyCarbonate",
    "Part Volume" : "1cu in",
    "Part color" : "stock color",
    "Expected Delivery Time" : "14days",
    "Manufacturing Process" : "Additive
    Manufacturing",
    "Additional Services" : "Protected Packaging"
  }
}
```



## Background - RFQ Process



# Motivation

If a request comes in with  
either of these topics...

Polycarbonate

Additive  
Manufacturing

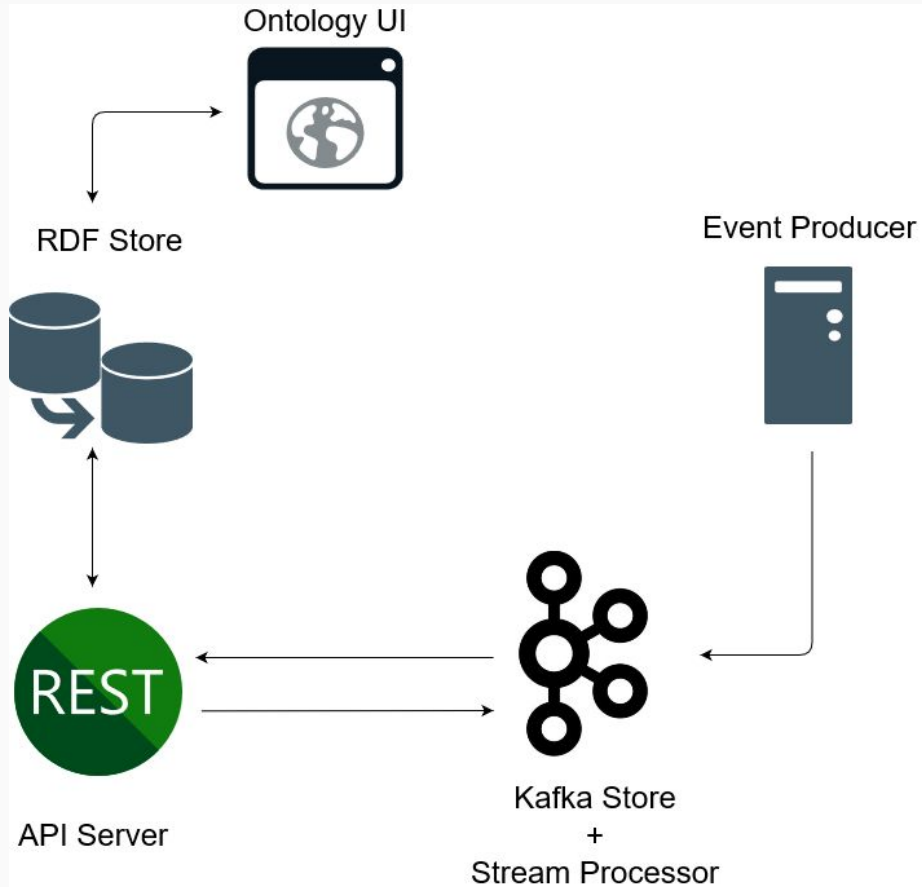
...then manufacturer  
subscribed to 3D printing  
should get notified

3D Printing

Manufacturer is  
subscribed to



# Architecture Diagram



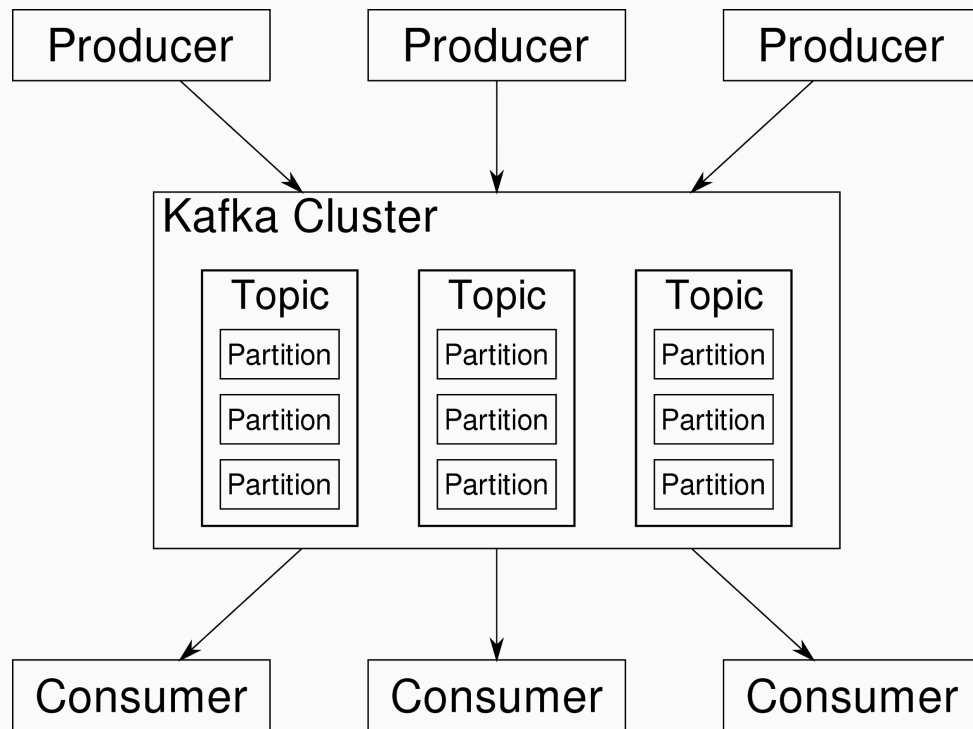
- Marketplace is based on SmartChainDB, which is based on BigChainDB
- Real-time notifications can be emitted from SmartChainDB
- The RFQ could be in form of notification
- Focus of our project was not SmartChainDB, we assumed that the RFQ events will reach to us from SmartChainDB

The logo for BigChainDB, featuring the word "BIGCHAIN" in a dark blue, bold, sans-serif font, followed by "DB" in a smaller, teal-colored, bold, sans-serif font.



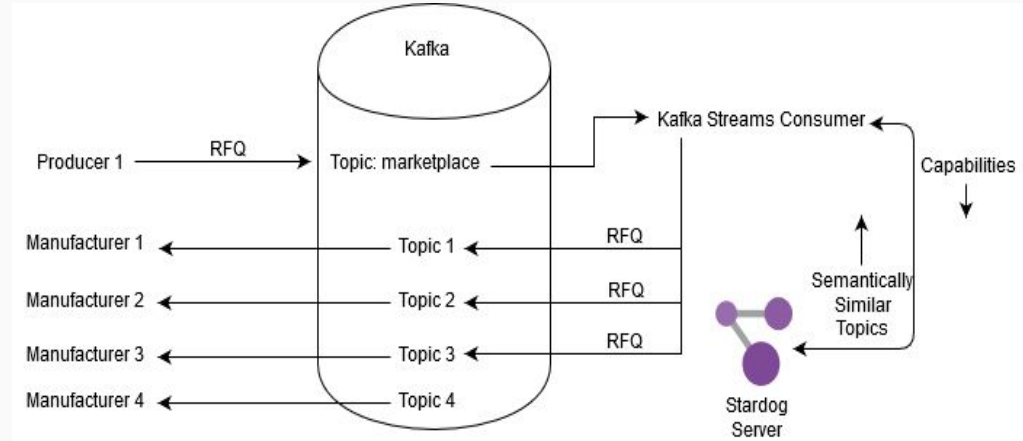
# Kafka - For Stream Processing

- Unified, high-throughput and low-latency stream processing software platform
- Used to handle real-time data feeds
- Kafka follows producer / consumer model



# Kafka - For Stream Processing

- Event consumers are the manufacturers that are listening out for relevant events implemented as topics
- Events are produced from SmartChainDB
- Every published RFQ request initially goes to marketplace topic
- Stream processor consumes the topic, processes and then forwards it to all relevant topics



# Scalability of Kafka

- Kafka architecture is inherently distributed and fault-tolerant
- The data can be partitioned into different "partitions" within different "topics"
- Zookeeper for scale co-ordination
- We have currently hosted Kafka on Google Cloud Platform
- Can be auto-scaled out based on the request load



# Kafka Streams vs Spark Streams

- We had two main choices for stream processing, Kafka Streams and Spark Streams
- Apache Spark Streaming is a scalable fault-tolerant streaming processing system
- Evaluated its performance and features in our use case



# Kafka Streams vs Spark Streams

Spark Streams	Kafka Streams
Data received form live input data streams is Divided into <b>Micro-batched</b> for processing.	Processes per data stream (real-time)
Separated processing cluster is required	No separated processing cluster is required
Needs re-configuration for Scaling	Scales easily by just adding java processes, No reconfiguration required
Standalone framework which requires extra integration	Seamless integration with Kafka store, can be used via library

We wanted to achieve real-time processing, so batch processing of Spark streams was deciding factor

- Keyword topic labels are not sufficient
- Need to support semantic technologies e.g. ontologies and ontological reasoning integrated with publish/subscribe models
- Needed a platform that allows to store complex structural data - some kind of graph database
- Evaluated a couple of possible ontology stores: Neo4j, Stardog



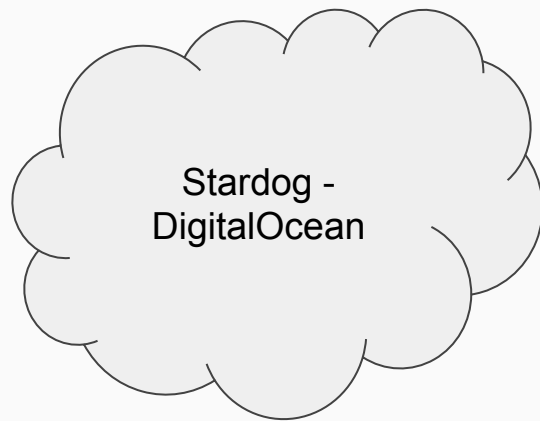
# Neo4j vs Stardog

Neo4j	Stardog
Primary database model: Graph DBMS	Primary database model: Graph DBMS <b>RDF store</b>
Needed extra plugin to support RDF/OWL (Load RDF into Neo4J nodes and load it into neo)	Excellent native support
No support for reasoning	Can query with and without <b>reasoning</b>
Based Cypher Query Language	Native support for SPARQL/GRAPHQL
Lack of documentation and tutorials - via third party blog and outdated	First party documentation and library

Native support of Stardog for ontology related operations was major deciding factor

# Stardog Scalability

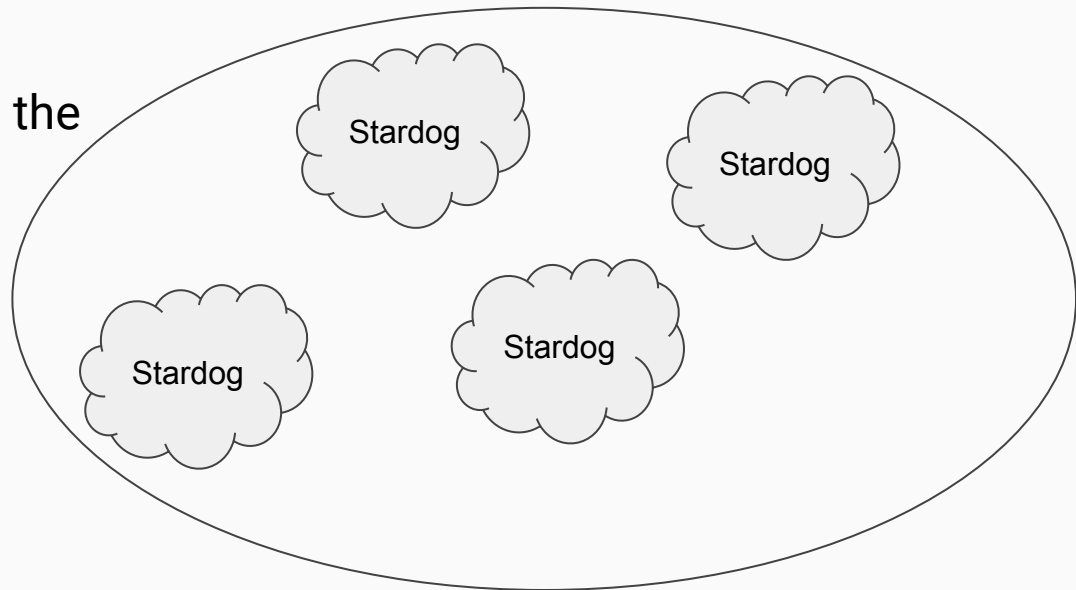
- Server, IDE, SDKs
- Started development on local Stardog Server, currently hosted on DigitalOcean cloud server (ease of access and universal database for all team members)
- Has native support for cloud providers and can scale out massively
- Stardog Cluster contains a coordinator and one or more participants. All of the Stardog nodes connect to a ZooKeeper ensemble, which consists of an odd number of servers, typically 3 or more.





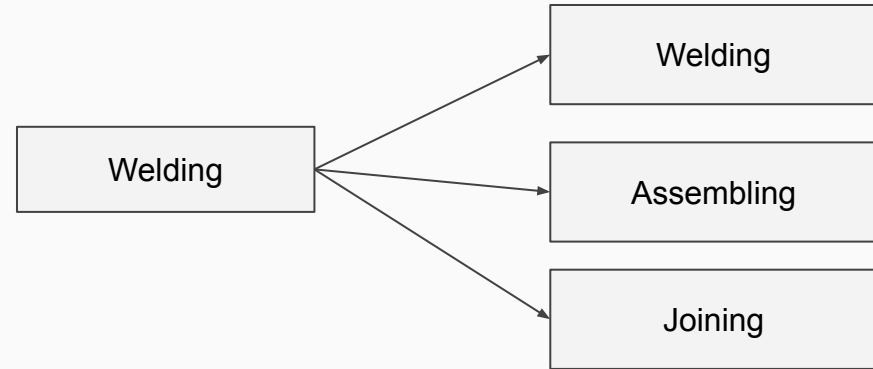
# Stardog Cluster

- Stardog Cluster guarantees that all members of the cluster are consistent
- First class support for running Stardog's HA cluster and virtual graph caching in Kubernetes (K8s)
- Server is scalable, what about the service that offers semantic mapping functionality



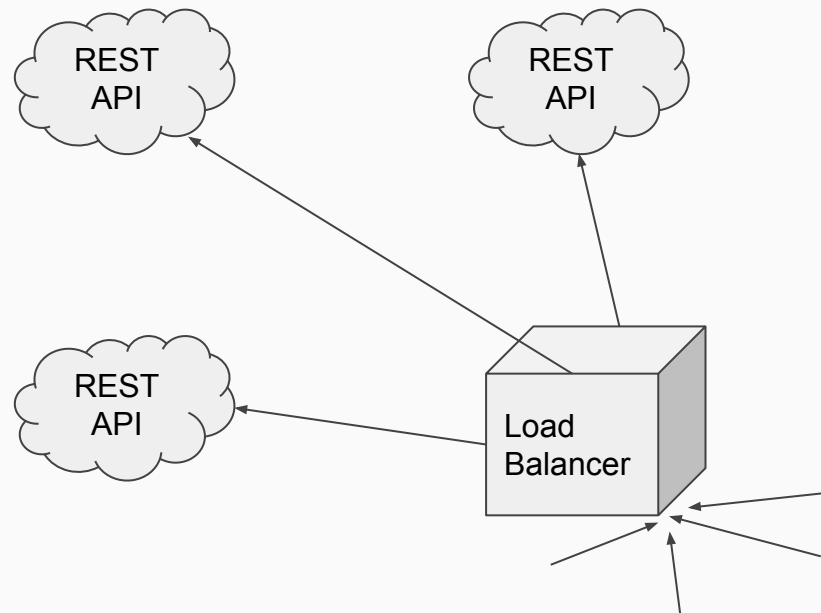
# Semantic Mapping REST API

- REST API that interacts with Stardog RDF store server
- Given a search keyword, based on the ontology stored on RDF store, returns semantically equivalent topics
- Independent service, loosely coupled with rest of architecture



# Semantic Mapping REST API

- Also hosted on cloud provider
- Written in Python, used Stardog Pystardog library
- Independent of other architectural components
- API can be first scaled out vertically and then with more scale, it can be scaled out horizontally



# REST API Example

GET https://rtdm-flask-client.herokuapp.com/ + ... No Environment 👁 ⚙

Untitled Request

GET https://rtdm-flask-client.herokuapp.com/topics?term=Bagging Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Cookies Code Comments (0)

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	term	Bagging			
	Key	Value	Description		

Body Cookies Headers (6) Test Results Status: 200 OK Time: 63ms Size: 212 B Save Response

Pretty Raw Preview JSON ≡

```
1 [
2   "Packaging",
3   "Bagging",
4   "Finalization"
5 ]
```

# REST API Example

GET https://rtdm-flask-client.herokuapp.com/topics?term=Welding

No Environment

Untitled Request

GET

https://rtdm-flask-client.herokuapp.com/topics?term=Welding

Send

Save

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Cookies

Code

Comments (0)

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	term	Welding			
	Key	Value	Description		

Body

Cookies

Headers (6)

Test Results

Status: 200 OK

Time: 265ms

Size: 208 B

Save Response

Pretty

Raw

Preview

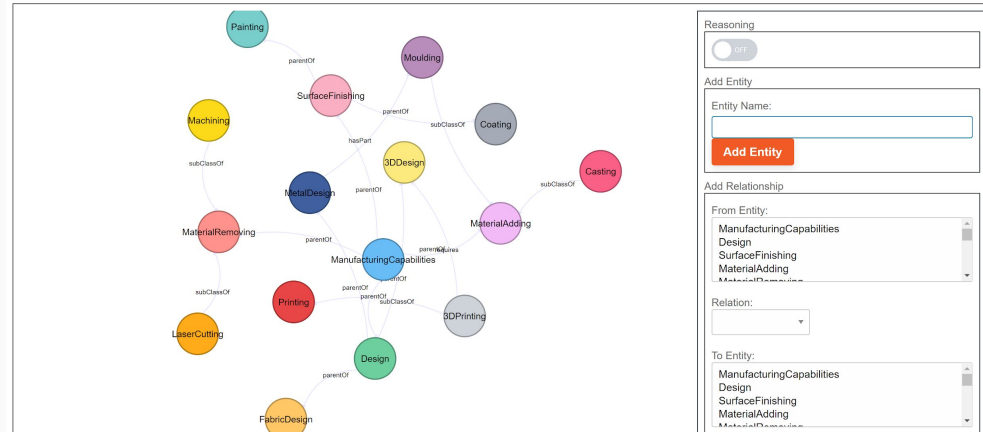
JSON

```
1  [
2    "Joining",
3    "Welding",
4    "Assembling"
5  ]
```

# UI to Update Ontology

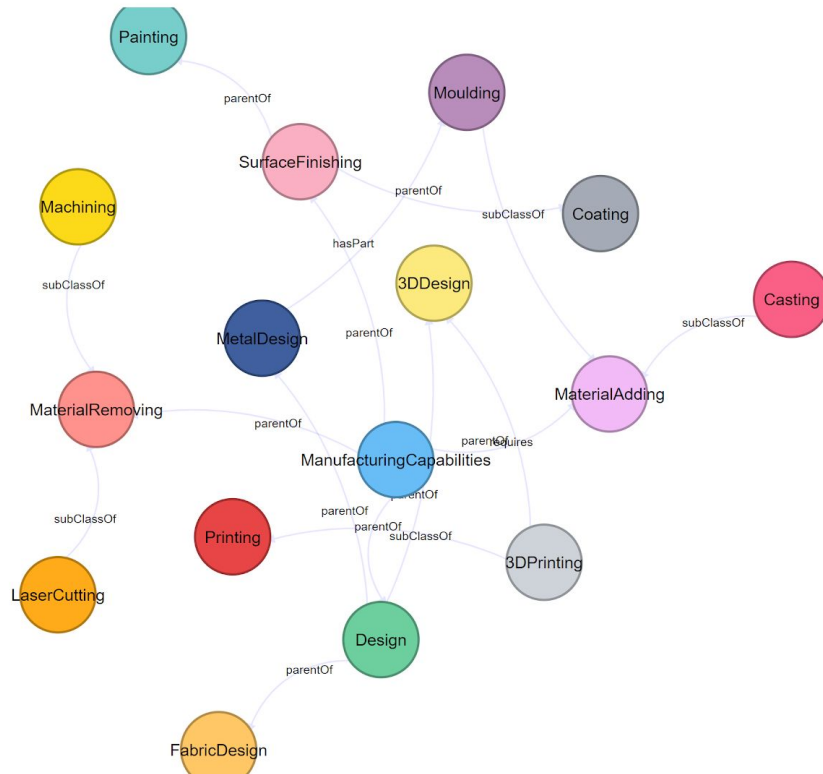
- Ontology can be ever changing
- User interface to facilitate this change / new mappings
- Written in JavaScript
- Use D3.js as visualization library and stardog.js to interact with Stardog server
- Front-end app with no backend, thus can be scaled out without issues

RTDM RFQ Demo



## UI Example

## RTDM RFQ Demo



Reasoning

OFF

Add Entity

Entity Name:

Add Entity

Add Relationship

From Entity:

ManufacturingCapabilities  
Design  
SurfaceFinishing  
MaterialAdding  
MaterialRemoving

Relation:

To Entity:

ManufacturingCapabilities  
Design  
SurfaceFinishing  
MaterialAdding  
MaterialRemoving

## Limitations

- KafkaStream implementation duplicates the messages to send it to subscribed topics (limited by KafkaStream current versions).
- Fault-tolerance issues if the Stardog API does not respond. Need to tackle that by a cache mechanism.
- No implementation for addition of custom relationships.
- Manufacturing capability in RFQ is syntactically correct and matches resource in ontology.



# Conclusion

- Reasonably achieved goal of semantic event detection
- All architecture components work well with each other
  - Kafka, Streams, Stardog server, REST API, UI
- Have a working demo that is efficiently scalable and can achieve real time processing requirements
- Thanks Dr Kemafor Ogan for giving the opportunity to work in a new domain and learning that have resulted to

## Primary References

- The development of an ontology for describing the capabilities of manufacturing resources  
[Used ontologies related to manufacturing from here]
- Towards Detecting Semantic Events on Blockchains  
Abhisha Bhattacharyya, Rahul Iyer, and Kemafor Anyanwu
- Semantic Event Detection: Semantic Complex Event Detection System of Express Delivery Business with Data Support from Multidimensional Space", Applied Mechanics and Materials
- Kafka documentation
- Stardog documentation

- More optimized implementation for RFQ transfer over topics in KafkaStream
- Provide an interface
  - Updating rules for existing resources in ontology
  - Adding new relationships

# Thank you

Questions?