USING GRAPH THEORY & NETWORK SCIENCE TO EXPLORE YOUR MICROSERVICES ARCHITECTURE

Nicki Watt - CTO @techiewatt
OpenCredo
Please
Remember to rate this session
Thank you!
Hands-on software delivery
Consulting
provides services
with expertise in
Cloud Native / Microservices Architectures

Data Engineering Platforms

includes experience gained building
lead projects on
Co-author of
is CTO at
OpenCredo
A Trifork Company

@techiewatt
AGENDA
OVERVIEW

- INTRO & HYPOTHESIS
- A BIT OF THEORY
- MICROSERVICE EXPLORING
- SUMMARY
INTRODUCTION
Graph theory is already being used to drive efficiencies in, and produce more reliable software systems
INTRODUCTION

**Code:** Analysing software package & license dependencies
Infrastructure: Reliably managing and updating cloud and other infrastructure-as-code resources
INTRODUCTION

Ops & Monitoring: Troubleshooting, Distributed Tracing & Latency analysis

@techiewatt
Can we also use it to help inspect and drive us towards improvements in our microservices architecture?
In microservice architectures, the most important aspects are not properties of the code but properties of the system
- Adam Tornhill
HYPOTHESIS
General Hypothesis: Data Driven Architectural Improvement

You can extract metrics and KPIs from a microservices architecture using graph theory AND use these to gain insight into the structure and characteristics of your microservices architecture.
For this talk ...

Can we use these metrics to detect bad microservice architectural smells and anti-patterns like a tightly coupled architecture (distributed monolith)
A QUICK BIT OF THEORY
GRAPH THEORY 101

A graph is a way to formally represent a network, or collection of related objects, in a mathematical way.
GRAPH INSIGHTS 101

● **Graph Analytics**
  ○ *An Action Performed:* The act of analysing connected data - using any appropriate graph-based approach or tools (visualisations, queries, statistics, algorithms)

● **Graph Algorithms**
  ○ *Programmable process or set of rules:* Leverages the mathematical properties of graphs to explore, classify and interpret connected data. A subset of tooling used to do graph analytics.
GRAPH INSIGHTS 101

GRAPH THEORY

Roots date back to 1786, includes study of:

- Abstract theoretical graph forms (e.g. random graphs, trees, directed graphs)
- Graphs of any size

NETWORK SCIENCE

New academic field, circa 21st century includes study of:

- Real-world representations in order to understand the universal properties of networks (Biological, social, transport)
- Large, complex networks
“Based on the mathematics of graph theory, graph algorithms use the relationships between nodes to infer the organization and dynamics of complex systems. Network scientists use these algorithms to uncover hidden information, test hypotheses, and make predictions about behavior”.

- Graph Algorithms by M Needham, A Hodler
Show me some of this graph theory / network science stuff ...
HIGH LEVEL ALGORITHM TYPES

- Path Finding
- Centrality
- Community Detection
HIGH LEVEL ALGORITHM TYPES

- Path Finding
- Centrality
- Community Detection
DEGREE

How connected is a specific node?
How connected is a specific node?

A is more highly connected than B and C
How tightly is a group clustered, compared to how tightly it could be clustered?
CLUSTER COEFFICIENT
CLUSTER COEFFICIENT
CLUSTER COEFFICIENT
2 CLUSTER COEFFICIENT

![Diagram of a network with a cluster coefficient of 0.66]
CLUSTER COEFFICIENT

0.66
2. CLUSTER COEFFICIENT

0.66

1.0
COMMUNITY DETECTION

Used to find related communities, uncover groupings, and quantify the quality of groupings.
DIVE IN!
EXPLORING MICROSERVICES (AS A NETWORK)
#1
A “Microservice” Architecture (v1.0)
V1.0 - NETWORK STATS

<table>
<thead>
<tr>
<th>name</th>
<th>degree</th>
<th>degree_in</th>
<th>degree_out</th>
<th>triangles</th>
<th>cluster_coeff</th>
</tr>
</thead>
<tbody>
<tr>
<td>ShippingFacade</td>
<td>14.0</td>
<td>5.0</td>
<td>9.0</td>
<td>17</td>
<td>0.21794871794871795</td>
</tr>
<tr>
<td>Order</td>
<td>10.0</td>
<td>5.0</td>
<td>5.0</td>
<td>17</td>
<td>0.6071428571428571</td>
</tr>
<tr>
<td>PricingCalculator</td>
<td>8.0</td>
<td>4.0</td>
<td>4.0</td>
<td>17</td>
<td>0.8095238095238095</td>
</tr>
<tr>
<td>User</td>
<td>6.0</td>
<td>6.0</td>
<td>0.0</td>
<td>12</td>
<td>0.8</td>
</tr>
<tr>
<td>Product</td>
<td>6.0</td>
<td>6.0</td>
<td>0.0</td>
<td>12</td>
<td>0.8</td>
</tr>
<tr>
<td>PaymentFacade</td>
<td>6.0</td>
<td>1.0</td>
<td>5.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Web API</td>
<td>5.0</td>
<td>0.0</td>
<td>5.0</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td>Mob API</td>
<td>5.0</td>
<td>0.0</td>
<td>5.0</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td>B2B API</td>
<td>5.0</td>
<td>0.0</td>
<td>5.0</td>
<td>9</td>
<td>0.9</td>
</tr>
<tr>
<td>UPS</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>DHL</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Royal Mail</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Shipper X</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Shipper Y</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Shipper Z</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>MasterCard</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>PayPal</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Provider A</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Provider B</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>Provider C</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Density: 0.19  
Number of nodes: 20  
Number of edges: 36  
Average degree: 3.6000  
Average Clustering Co-eff: 0.30
V1.0 - DEGREE

v1 - degree

- Provider C
- Provider B
- Provider A
- PayPal
- MasterCard
- Shipper Z
- Shipper Y
- Shipper X
- Royal Mail
- DHL
- UPS
- B2B API
- Mob API
- Web API
- PaymentFacade
- Product
- User
- PricingCalculator
- Order
- ShippingFacade
V1.0 - CLUSTER COEFFICIENT
We’ve been directed to look a bit closer at a few potential problem services ... 

Where do they fit in architecturally?
V1.0 - ARCHITECTURE DIAGRAM
V1.0 ANALYSIS (DEGREE)
V1.0 ANALYSIS (CLUSTER COEFF)
V1.0 ANALYSIS (AVERAGES)

Average degree: 3.6
Average Clustering Coefficient: 0.3
- Entity Services Anti-Pattern
- Distributed Monolith
V1.0 INVESTIGATION

- Entity Services Anti-Pattern
- Distributed Monolith
- Entity Services Anti-Pattern
- Distributed Monolith
- Help detect tightly coupled areas
- Not always DDD aligned
COMMUNITIES NOT ALWAYS DDD ALIGNED

Conway’s Law?

Credit: http://bonkersworld.net/organizational-charts
COMMUNITIES NOT ALWAYS DDD ALIGNED

Erich Eichinger Blog: “Heuristics for Identifying Service Boundaries”
https://opencredo.com/blogs/identify-service-boundary-heuristics/
#2
The Revised Microservice Arch
User Domain Decoupling (v2.0)
Recap ... V1.0
V1.0 —> V2.0

Diagram showing relationships and connections between Front End, Back End, External Integration, and Database components with a central focus on User.
V1.0 —> V2.0
V1.0 → V2.0
V2.0 - NETWORK STATS

Density: 0.11  
Number of nodes: 29  
Number of edges: 46  
Average degree: 3.1724  
Average Clustering Co-eff: 0.20
V2.0 - COMBINED STATS
V2.0 - COMBINED STATS (CLUSTER COEFF)
V2.0 - COMMUNITY DETECTION
COMPARING V1 and V2
COMPARING

Combined Measurements (V1 & V2)

Degree

Cluster Coefficient

V1 User Service

@techiewatt
COMPARING

Combined Measurements (V1 & V2)

V1 User Service

New V2 Refactored User Services

@techiewatt
COMPARING

Combined Measurements (V1 & V2)

V1 User Service

V2 Mono User Service
(allow distmon to continue functioning)

New V2 Refactored User Services

Degree

Cluster Coefficient

Combined Measurements (V1 & V2)

v1 degree v2 degree v1 cluster coefficient v2 cluster coefficient

@techiewatt
COMPARING

Combined Measurements (V1 & V2)

Degree

Cluster Coefficient

V1 (5)

V2 (4)
FURTHER OPTIONS
OTHER MEASURES & ALGORITHMS

- **Strongly Connected Components (SCC)**

- Help detect circular dependencies
OTHER MEASURES & ALGORITHMS

- Path Finding Algorithms
  Detect who is calling deprecated services
CONCLUSION
Early days ...
Can’t be applied blindly ...

Common sense and understanding of architectural approaches and patterns still required!!
General Hypothesis: Data Driven Architectural Improvement

You can extract metrics and KPIs from a microservices architecture using graph theory AND use these to gain insight into the structure and characteristics of your microservices architecture.
General Hypothesis:
Data Driven Architectural Improvement

You can extract metrics and KPIs from a microservices architecture using graph theory
AND
use these to gain insight into the structure and characteristics of your microservices architecture
Specifically

Can we use these metrics to detect bad microservice architectural smells and anti-patterns like a tightly coupled architecture (distributed monolith)?
CONCLUSION

Demonstrated using degree and cluster coefficient measures to detect tightly coupled (distributed monolith) architectures.

Demonstrated using community detection algorithms to uncover related groupings (boundaries) of microservices.
RECOMMENDED LEARNING RESOURCES

Graph Algorithms
Practical Examples in Apache Spark & Neo4j
Mark Needham & Amy E. Hodler

@techiewatt
OpenCredo
A TRIFORK COMPANY

Questions?

Want to find out more?

Want Some Help?

Come chat with me!

@techiewatt
Please Remember to rate this session

Thank you!

Follow us @gotocph