

# Operating Microservices



## Surfacing Governance: Human Registry

- You're going to have a lot of services
- One (or more) for each business capability
- Sometimes humans want to understand the service estate
  - Governance, planning, organisation alignment, process re-engineering, etc
- You'll need to surface your estate in an accessible way
  - Not JSON apis
- But if you have an automated operational view of your estate, you already have what you need...

## Mapping the Estate

- You already have excellent data about the operational characteristics of your services
- Add small amounts of useful descriptive text into your service definitions in that operational management system
- Then you have your registry
  - Even better than just having a wiki
  - And way better than some heavyweight “proper” registry
- See operating microservices for how this is done for real

# Motivating Example: Lending Club

The screenshot shows the Lending Club homepage. At the top, there's a navigation bar with links for Personal Loans, Business Loans, Patient Solutions, Investing, How It Works, and About Us. Below the navigation is a large green banner with the text "Better Rates. Together." and an illustration of four diverse people. To the right of the banner is a form for "Personal Loans up to \$40,000" with fields for "How much do you need?", "What is it for?", "How is your credit?", and a "Check Your Rate" button. A note below the button says "Won't impact your credit score". A "Privacy & security PROTECTION" badge is also present. Below the banner, there's a section titled "Financial Innovation" with a paragraph about the company's mission to connect borrowers and investors. To the right of this is a diagram showing "INVESTORS" (represented by three icons) and a "BORROWER" (represented by one icon). Arrows indicate the flow of "Invest" from the investors to the borrower, and "Monthly Payments" from the borrower back to the investors. Below this is a "Featured Borrower" section for "Rebecca Durham, NC Pay Off Credit Cards \$5,000 at 9.98%". A quote from her is included: "The application process was simple, and it was neat to watch the process as people invested in my loan."

- Lending Club is the world's largest credit marketplace
  - P2P lending
- Real money at stake for borrowers and investors

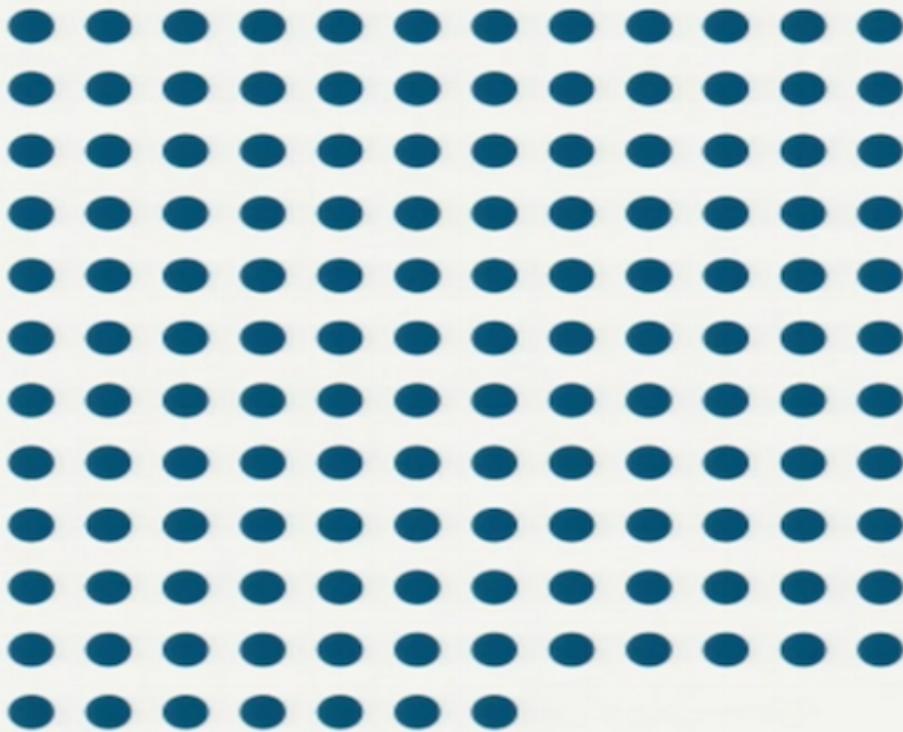


<https://neo4j.com/blog/managing-microservices-neo4j/>

## Explosive Growth in Services



**2013**



**2015**

# It's not (Just) About Containers



**Because no talk on microservices would be  
complete without a container analogy**

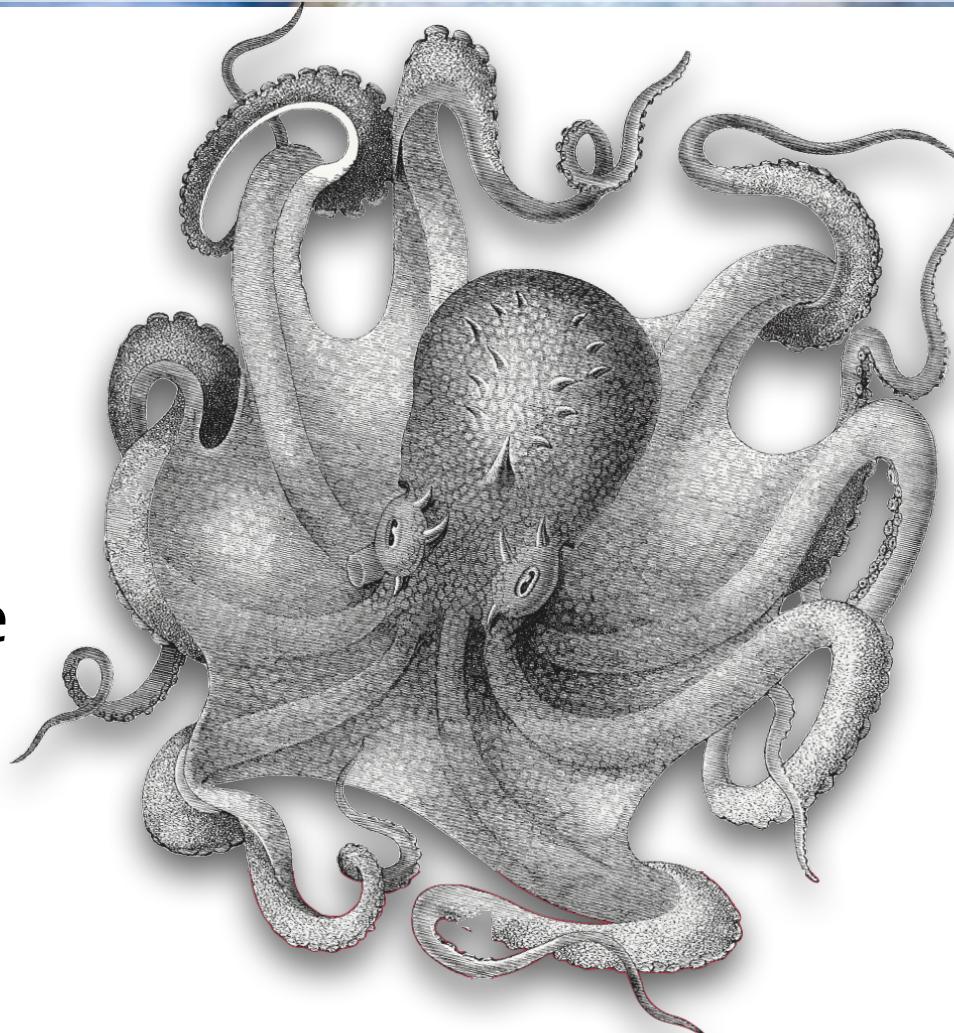


*Can you check the load balancer?*



## MacGyver has its Tendrils Everywhere

- GitHub
- AWS
- Splunk
- HipChat
- Vmware
- And plenty more



Important information is siloed

**Operational Data Is Distributed**

**but real-time access is costly**

## App Check-In and Service Discovery

- Over 1,000 servers, hard to keep track of what's out there, and what's going on,
- All servers to phone home to MacGyver every minute
  - App ID
  - Revisions
  - Environment

# App Check-In and Service Discovery



**"Show me all the instances of app X in environment Y"**

```
match (x:AppInstance)  
      {appId:'lcui', environment:'prod_nevada'})  return x;
```

## Deployment and Release Automation

- Blue-green deployments used
- One live pool and one dark pool at any given time
  - half the servers are live and half of them are dark within a service group
- Live pool takes all traffic
- During deployment, deploy new code to the dark pool, and then QA and Release testing
- Switch over

Green is Live

# Blue-Green Deployment

“Live” Pool

Server 1

Server 2

Server 3

Server 4

“Dark” Pool

Server 5

Server 6

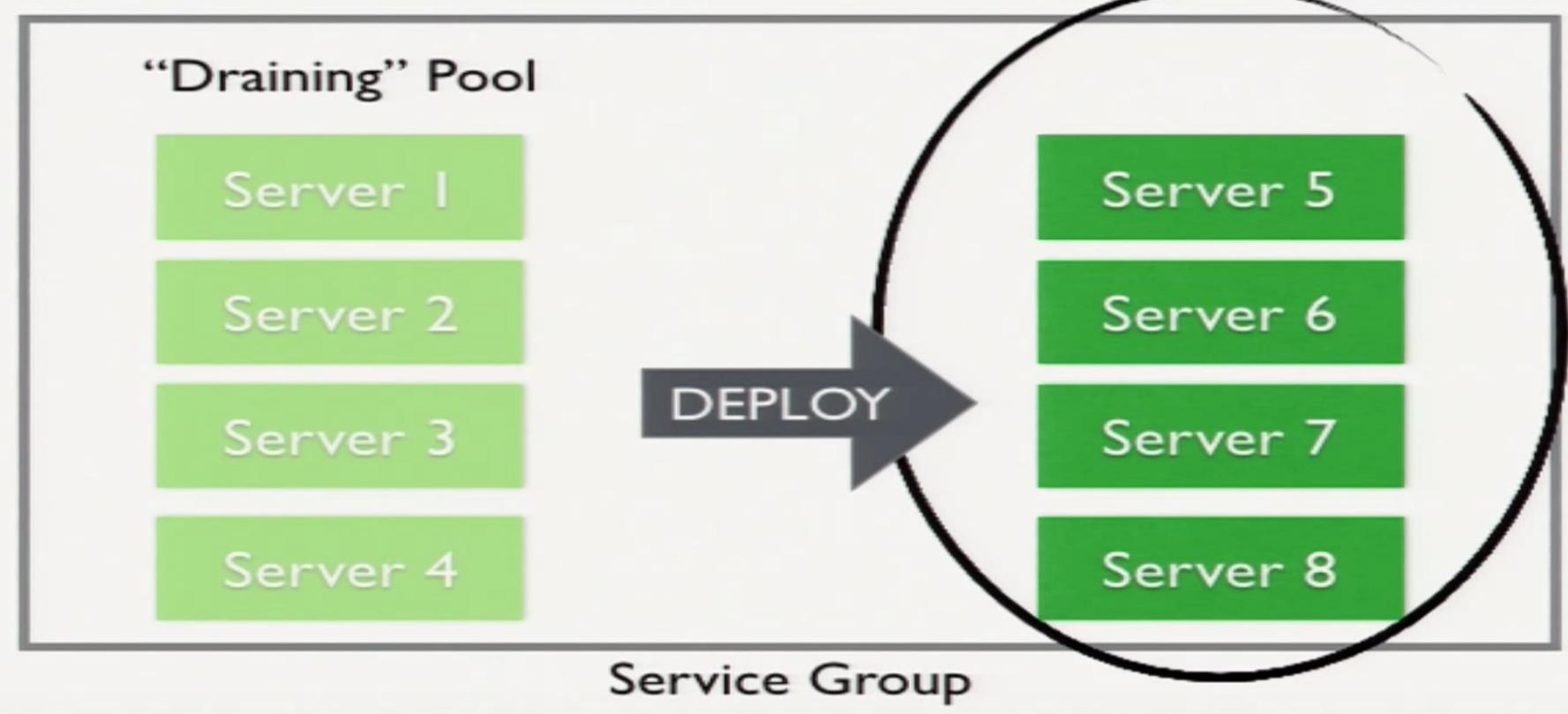
Server 7

Server 8

Service Group

## Transitioning

# Blue-Green Deployment



Old Blue is new Green and is Live

# Blue-Green Deployment

Dark Pool

Server 1

Server 2

Server 3

Server 4

Live Pool

Server 5

Server 6

Server 7

Server 8

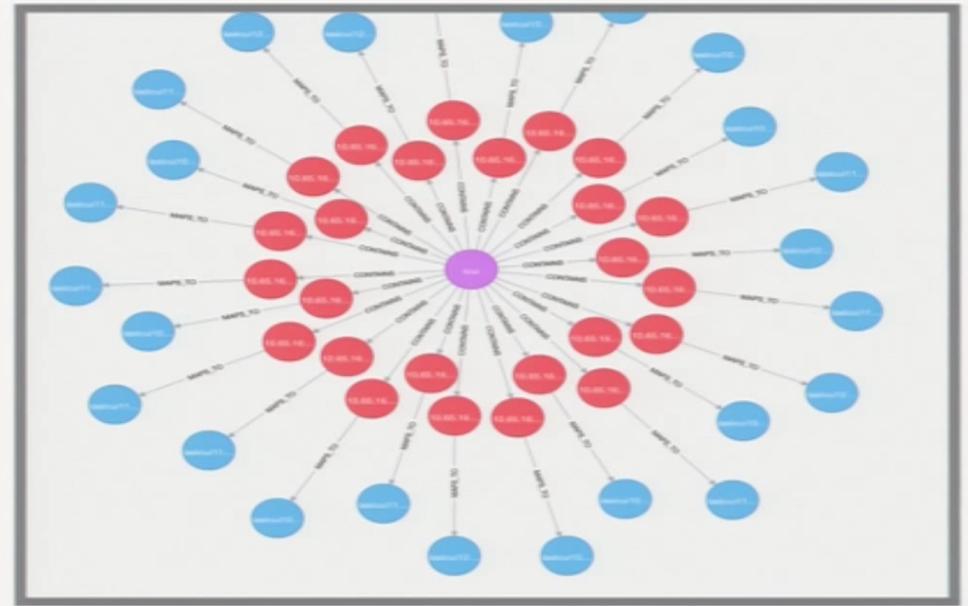
Service Group

## Problem

- Load balancer associates servers with each service group
  - But couldn't tell which servers were in each pool
- App instances reported to MacGyver knew their revision and app ID, but not their state
  - They didn't know if they were live or dark

## Solution

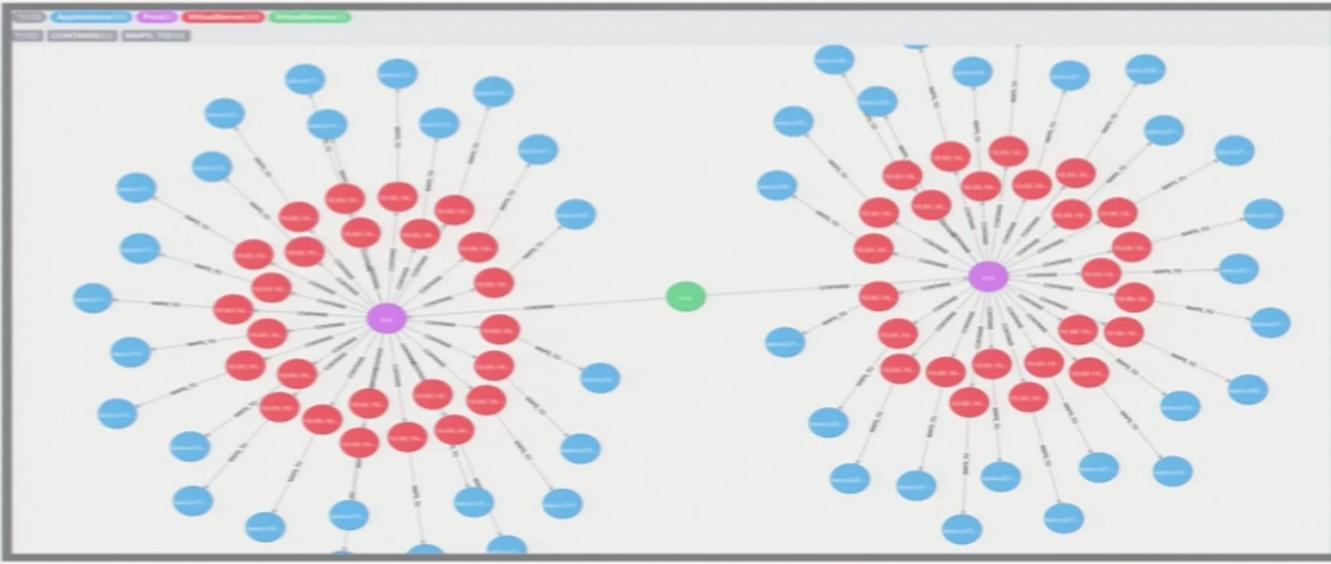
# Application Pools



- Collect virtual servers and aggregate according to their app ID and state
    - Create pools
  - Purple dots are pools
  - Each pool contains many servers
  - Each server has a 1:1 mapping to app instances

## Infrastructure Dependencies

### Virtual Service



- Each pool knows whether it's live, dark or draining
- Each service has a live pool and a dark pool
- Map pools into a virtual service (green dot)

Wherfore art thou?

*What pool should I deploy to?*

```
match (x:VirtualService {appId:'lcui',  
                         environment:'prod_nevada'}) return x.darkPool;
```

Or even more importantly...

*Are live pool revisions in sync in different environments?*

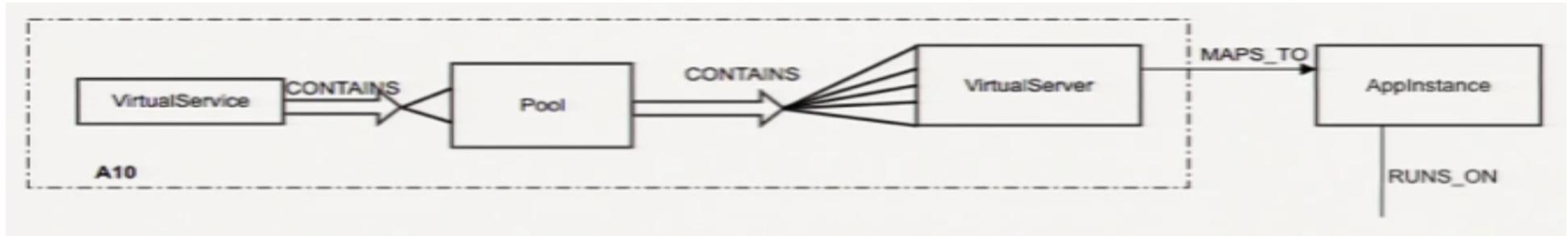
```
match (p:Pool {appId:'lcui', state:'live'})  
      return p.revision, p.environment;
```

And from a governance perspective...

*Do multiple revisions  
exist within a single pool?*

```
match (p:Pool)-[:CONTAINS]-(s:VirtualServer)  
      return distinct s.revision;
```

# Infrastructure Mapping



- Problem: low visibility into what pools were live, what pools were dark, what servers were in a virtual service or even what app instances were out there
- Problem extended to entire infrastructure

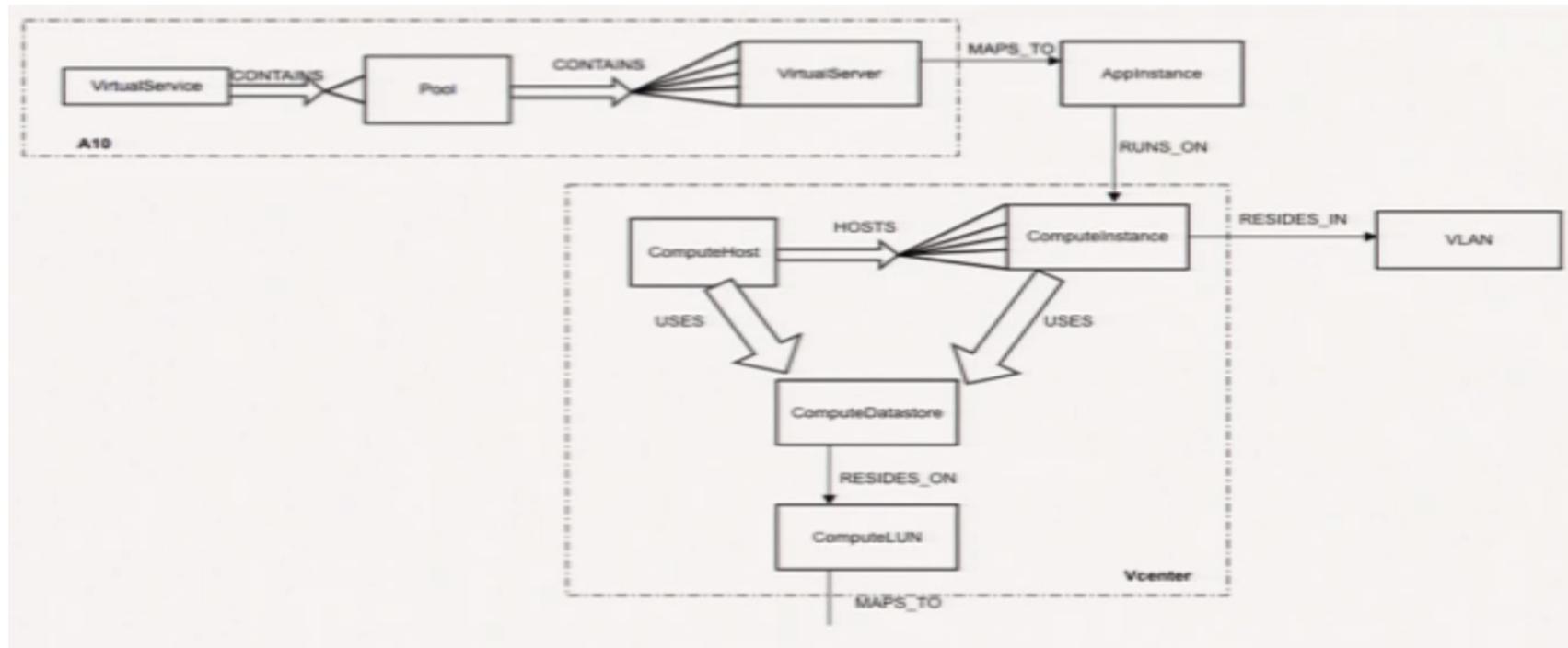
## Infrastructure Mapping

*Are any servers in a live pool degraded?*

```
match (x:Pool {state:'live'})--(y:VirtualServer)  
      where NOT y.state=1 and NOT y.priority=16 return y;
```

- Every virtual service contains two pools, which contain a number of virtual servers, which maps to an app instance.
- Exposing this data allowed us to ask, “Are any servers in the live pool degraded?”

# SPOF Spotting



- Import a bunch of data from vCenter
  - All the app instances map to compute instances which are hosted by compute hosts

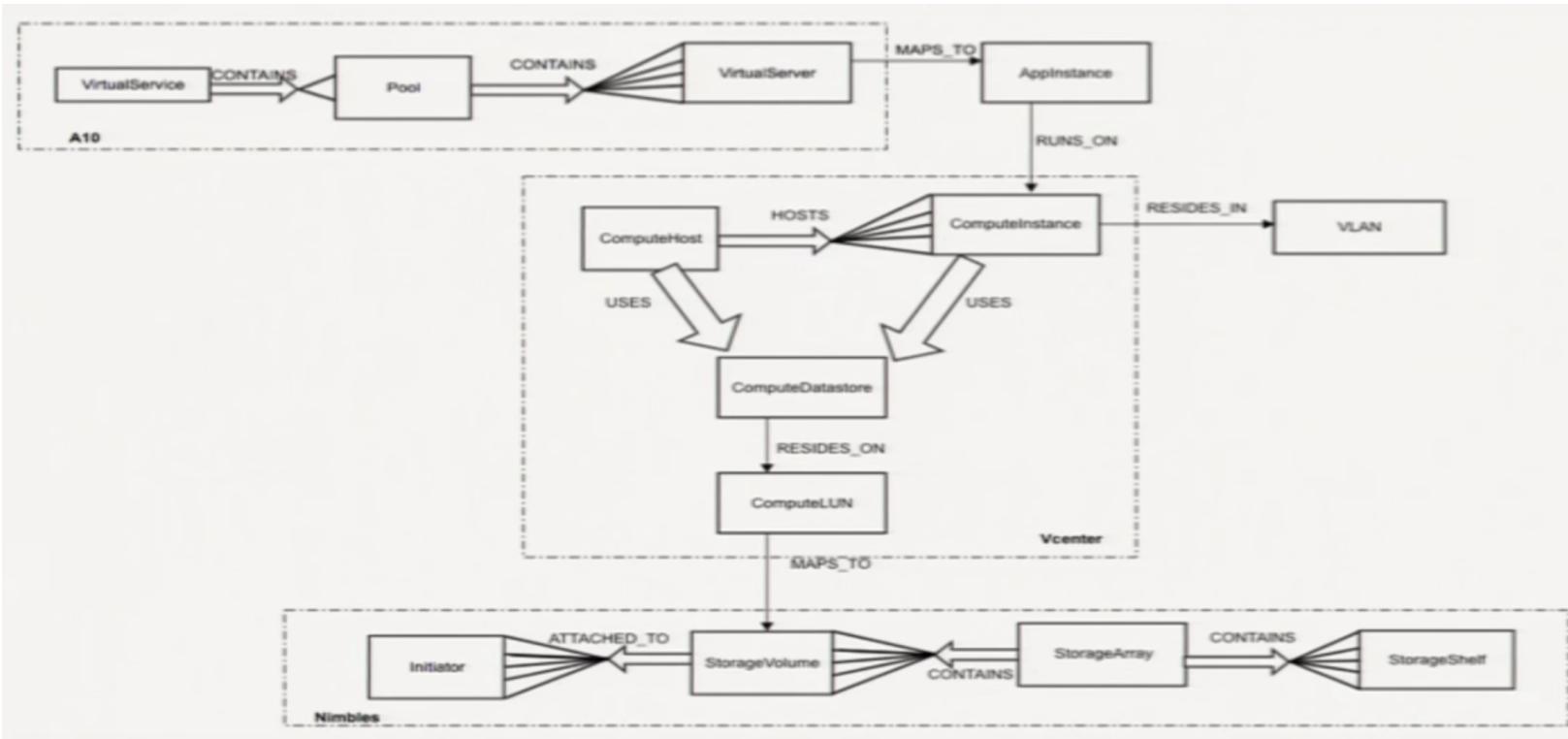
## SPOF Spotting

*Do we have a single point of failure among any of our services?*

```
match (p:Pool {appId:'lcui', environment:'prod_nevada'})-[:CONTAINS]-  
  (v:VirtualServer)-[:MAPS_TO]-(a:AppInstance)-[:RUNS_ON]-(i:ComputeInstance)<--  
  [:HOSTS]-(h:ComputeHost) return distinct h.name;
```

- If at any given time within a pool all those servers are hosted on one host in vCenter, if that host goes down, then we definitely have a problem
- Because we are mapping this, we are able to expose this data in a way that we weren't able to before
- Fix SPOFs at design time

# Deeper and Deeper...



- Can bring in shared (hardware) services like storage

## Deeper and Deeper...

*If storage volume #3 goes down,  
what services will be impacted?*

```
match (x:StorageVolume {name:'lasappesi-lasnmb01-prod'})--[*7]
      -(y:VirtualService) return distinct(y.appId);
```

- Can bring in shared (hardware) services like storage

## Powerful and Incremental

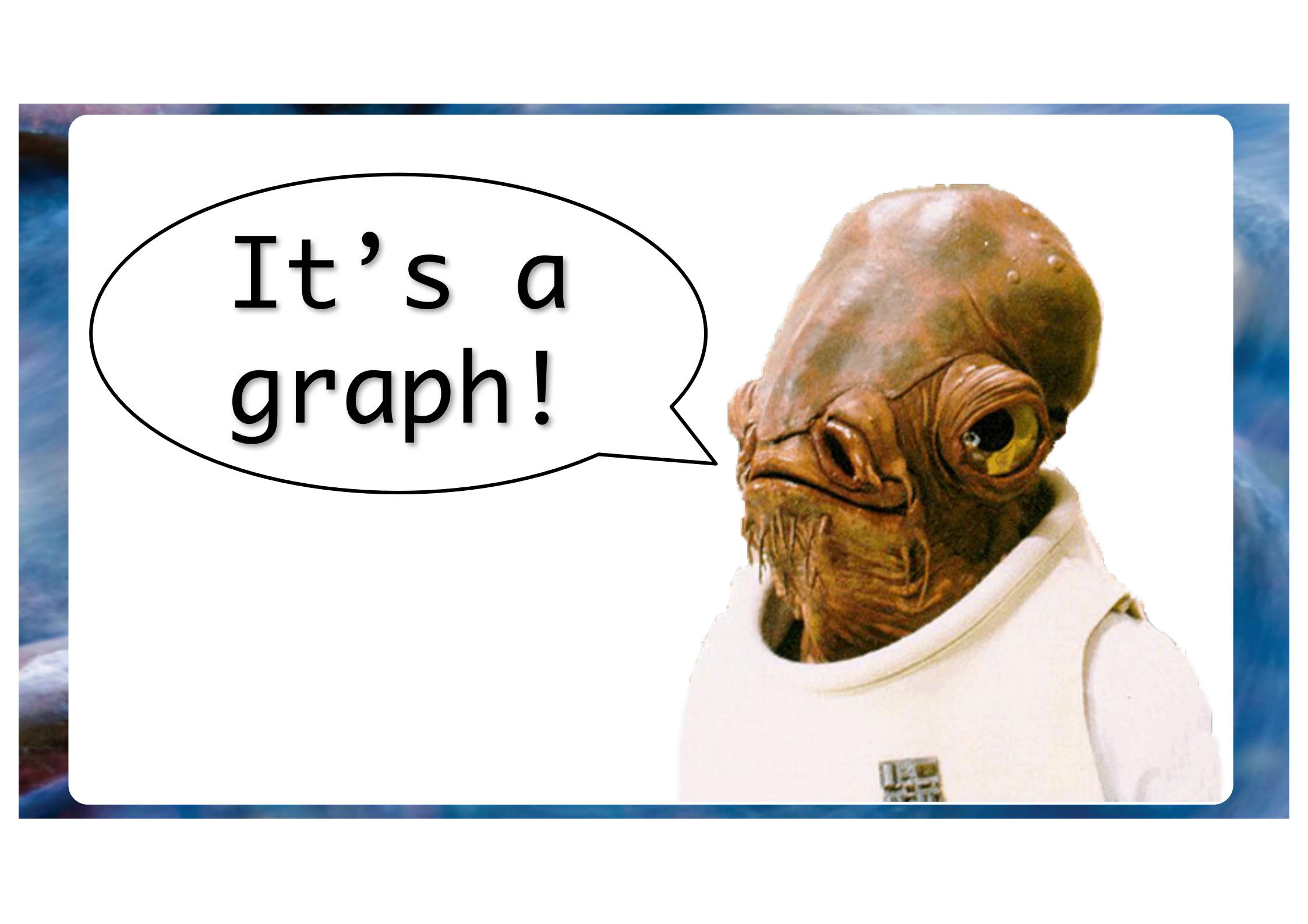
- Plan for operations that grows with your services, tames them
- Keeps them honest all the way up the stack
  - Storage to API



## Blatant Neo4j Plug



- Neo4j has all of the characteristics of a database that we need:
  - Really fast queries
  - Really great join and traversal capabilities
  - Really easy to use
  - Really flexible
  - Really scalable
  - Really great ad hoc queries through Cypher

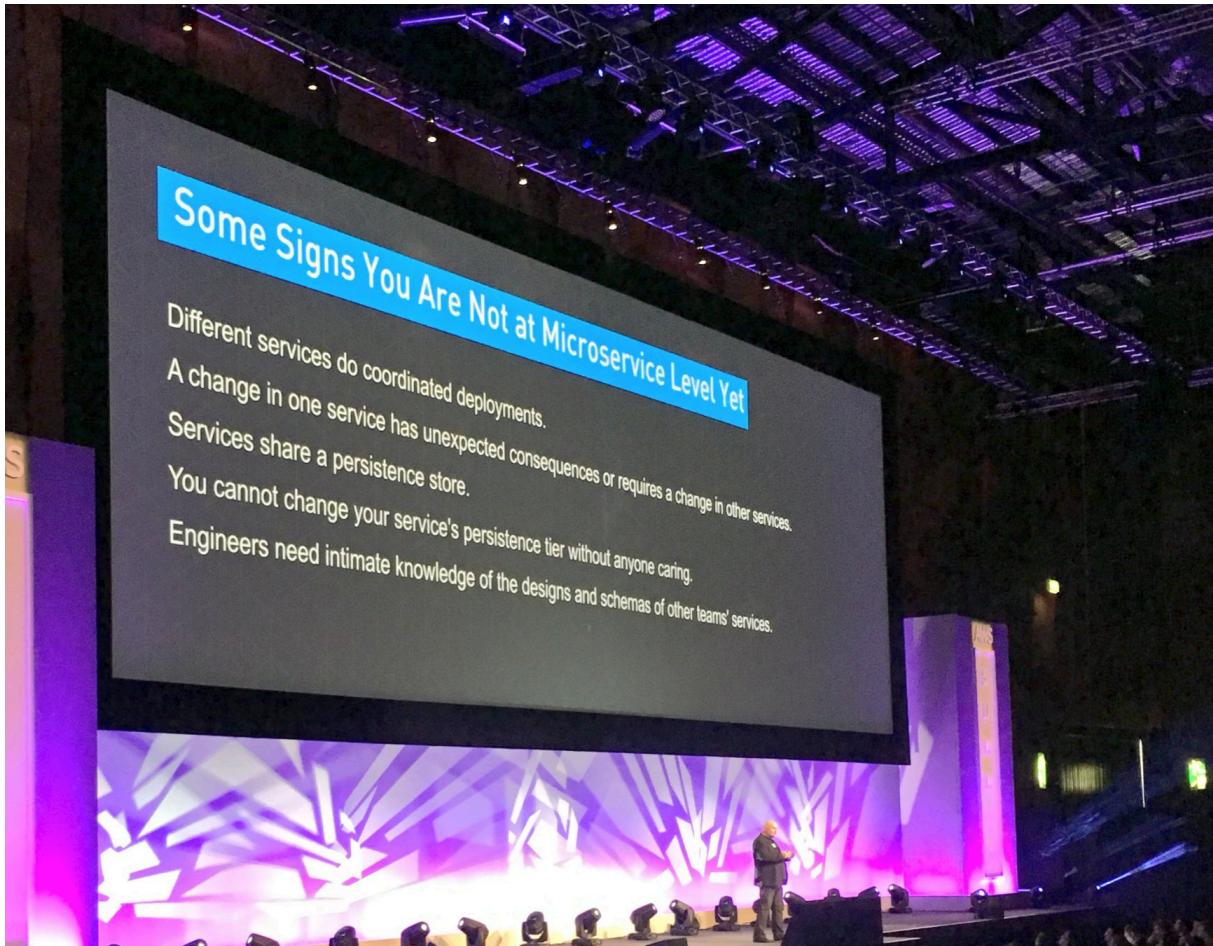


It's a  
graph!

## And then the obvious stuff...

- Once you have a map of your system-of-services, then:
  - Global charts and alerts:
    - KPIs of each service
    - Shared services (e.g. storage)
  - Localised charts and alerts:
    - Service SLA
    - Logs
    - etc

# Spotting Failures



Dr. Werner Vogels  
CTO, Amazon  
AWS Summit London 2016

## Summary

- Microservices will grow in number, as will third-party dependencies
- Think about how you deploy your service and embrace automation
- A map or registry is really helpful, but...
- But tooling needed to bring it all together
- Great tooling provides insight and the ability to reason about your system-of-services