



Lancope

VISION TO SECURE,

INTELLIGENCE TO PROTECT

Micro-services based architecture – a path to “Living Software”?

Jason Chambers

Director of Engineering

DEVNEXUS

Observations



```

10 CLS : PRINT "HAMURABI": PRINT ""
20 PRINT "CREATIVE COMPUTING MORRISTOWN,"
21 PRINT "NEW JERSEY"
22 PRINT "",PRINT "",PRINT ""
30 PRINT "",PRINT "",PRINT ""
30 PRINT "TRY YOUR HAND AT GOVERNING ANCIENT"
30 PRINT "SUMERIA FOR A TEN YEAR TERM OF OFFICE."
31 PRINT ""
30 PRINT ""
30 G=0:R=0
30 Z=0:P=95:S=2800:H=3000:E=H-S
31 Y=3:A=H/Y:I=5:Q=1
210 D=0
215 PRINT "",PRINT "",Z=Z+1
216 PRINT "HAMURABI, I BEG TO"
217 PRINT "IN THE YEAR",Z,"; D"
218 PRINT "STARVED, ";I" CAMP"
219 P=P+1
225 IF Q>0 THEN GOTO 230
226 P=P/2
227 PRINT "A HORRIBLE PLAGUE"
228 PRINT "HALF THE PEOPLE"
229 PRINT "POPULATION IS"
230 PRINT "THE CITY NOW"
232 PRINT "YOU HARVESTED"
235 PRINT "THE RATS ATE"
250 PRINT "YOU NOW HAVE"
260 PRINT "YOU NOW HAVE"
270 IF Z>11 THEN GOTO 80
310 C=1+RND(10):Y=C-17
312 PRINT "LAND IS TRAGEDY"
313 PRINT "PER ACRE."
314 PRINT "HOW MANY"
320 PRINT "INPUT Q:IF Q<0 THEN"
321 INPUT Q:IF Q<0 THEN"
322 IF Y*Q <= S THEN C
323 GOSUB 710
324 GOTO 320
330 IF Q=0 THEN GOTO 340
331 A=A+Q: S=S-Y*Q: C=0
334 GOTO 400
340 PRINT "HOW MANY ACRES DO YOU WISH TO SELL"
341 INPUT Q: IF Q<0 THEN GOTO 850
342 IF Q>A THEN GOTO 350
343 GOSUB 720
344 GOTO 340
350 A=A-Q: S=S+Y*Q: C=0
400 PRINT ""
410 PRINT "HOW MANY BUSHELS DO YOU WISH TO FEED"
411 PRINT "YOUR PEOPLE": INPUT Q
412 IF Q<0 THEN GOTO 850
418 REM * TRYING TO USE MORE GRAIN THAN IN SILOS
420 IF Q<=S THEN GOTO 430
421 GOSUB 710
422 GOTO 410
430 S=S-Q: C=C-1: PRINT ""
440 PRINT "HOW MANY ACRES DO YOU WISH TO PLANT"
441 PRINT "WITH SEED": INPUT D
442 IF D=0 THEN GOTO 511
443 IF D<0 THEN GOTO 450
444 REM * PLANTING MORE ACRES THAN YOU OWN
445 IF D>A THEN GOTO 450
446 GOSUB 720
447 GOTO 440
449 REM * ENOUGH GRAIN FOR SEED
450 IF (D/2)<5 THEN GOTO 455
452 GOSUB 710
453 GOTO 440
454 REM * ENOUGH PEOPLE TO TEND THE CROPS
455 IF D<=10*P THEN GOTO 510
460 PRINT "BUT YOU HAVE ONLY",P," PEOPLE TO"
461 PRINT "TO TEND THE FIELDS! NOW THEN,"
470 GOTO 440
510 S=S-D/2

```

This is similar in structure to the first BASIC programs I wrote as a kid

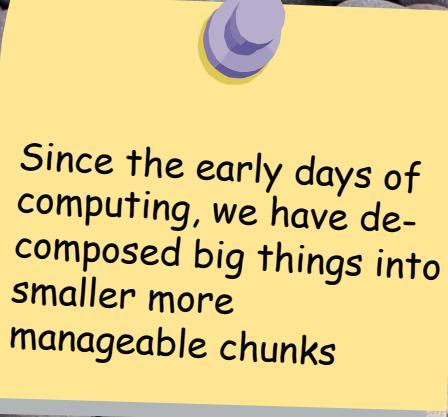
```

511 GOSUB 800
512 REM * A BOUNTIFUL HARVEST
515 Y=C: H=D*Y: E=0
521 GOSUB 800
522 IF C/2=0 THEN GOTO 530: REM ZERO REMAINDER
523 REM * RATS ARE RUNNING WILD
525 E=S/C
530 S=S-E+H
531 GOSUB 800
532 REM * LET'S HAVE SOME BABIES
533 I=C*(20*A+S)/P/100+1
539 REM * HOW MANY PEOPLE HAD FULL TUMMIES
540 C=Q/20
541 REM * HORRORS, A 15% CHANCE OF PLAGUE
542 Q=7-RND(8): REM Q LESS THAN ZERO IS PLAGUE
550 IF P<C THEN GOTO 210
551 REM * STARVE ENOUGH FOR IMPEACHMENT
552 D=P-C: IF D>P/2 THEN GOTO 560
553 G=((Z-1)*R+D*100/P)Z
555 P=C: G=G+D: GOTO 215
560 PRINT ""
561 PRINT "YOU STARVED ",D," PEOPLE IN ONE YEAR!"
565 PRINT "DUE TO THIS MISMANAGEMENT"
566 PRINT "NOT ONLY BEEN IMPEACHED"
567 PRINT "OUT OF THE OFFICE BUT"
568 PRINT "ALSO BEEN DECRALED"
569 GOTO 990
710 PRINT "HAMURABI, THINK AGAIN"
711 PRINT S;" BUSHELS OF GRAIN. NOW"
712 RETURN
720 PRINT "HAMURABI, THINK AGAIN"
721 PRINT A;" ACRES. NOW THEN,"
730 RETURN
800 C=1+RND(5)
801 RETURN
850 PRINT "": PRINT ""
851 PRINT "HAMURABI, I CANNOT"
855 PRINT "GET YOURSELF ANOTHER"
857 GOTO 990
860 PRINT "IN YOUR 10-YEAR TERM OF OFFICE,"
861 PRINT R;" PERCENT OF THE POPULATION STARVED"
862 PRINT "PER YEAR ON AVERAGE, I.E. A TOTAL OF"
865 PRINT G;" PEOPLE DIED!": L=A/P
870 PRINT "YOU STARTED WITH 10 ACRES PER PERSON"
871 PRINT "AND ENDED WITH",L;" ACRES PER"
875 PRINT "PERSON.": PRINT ""
880 IF R>10 THEN GOTO 940
885 IF L<9 THEN GOTO 940
895 IF R>3 THEN GOTO 960
896 IF L<10 THEN GOTO 960
900 PRINT "A FANTASTIC PERFORMANCE! CHARLEMAGNE"
901 PRINT "DISREALI, AND JEFFERSON COMBINED"
902 PRINT "COULD NOT HAVE DONE BETTER!"
905 GOTO 990
940 PRINT "YOUR HEAVY-HANDED PERFORMANCE SMACKS"
941 PRINT "OF NERO AND IVAN IV, THE PEOPLE"
942 PRINT "(REMAINING) FIND YOU AN UNPLEASANT"
943 PRINT "RULER, AND, FRANKLY, HATE YOUR GUTS!"
950 GOTO 990
960 PRINT "YOUR PERFORMANCE COULD HAVE BEEN"
961 PRINT "SOMETHING BETTER, BUT REALLY WASN'T"
962 PRINT "TOO BAD AT ALL."
963 PRINT 1+RND(P); " PEOPLE WOULD DEARLY LIKE"
964 PRINT "TO SEE YOU ASSASSINATED BUT WE ALL"
965 PRINT "HAVE OUR TRIVIAL PROBLEMS."
990 PRINT ""

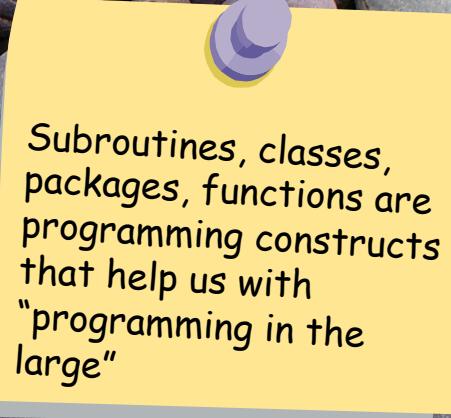
FOR NOW.": PRINT ""
G***": REM * DEBUG
PEOPLE DIED",G
POPULATION STARVED",R
EAR IS ",Z
ON IS ",P
STORAGE ",S
AS ",H
LD, PRICE PER ACRE ",Y
110 PRINT "PEOPLE CAME TO CITY ",I
1120 PRINT "CURRENT INPUT OR RND VALUE ",Q
";D
1130 PRINT "RND VALUE FROM 1 TO 10 ",C
1140 PRINT "ACRES PER PERSON ",L
1150 PRINT "RND VALUE FROM 1 TO 5 OR FULL "
1151 PRINT "TUMMIES ",C
1160 PRINT "***DEBUG***": REM END OF DEBUG
1170 RETURN

```

I still come across shell-scripts written like this <sigh/>



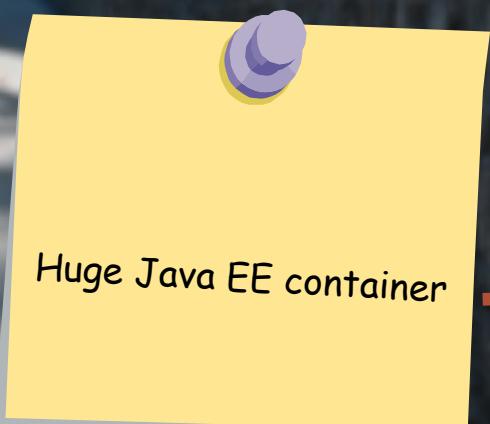
Since the early days of computing, we have decomposed big things into smaller more manageable chunks



Subroutines, classes, packages, functions are programming constructs that help us with "programming in the large"

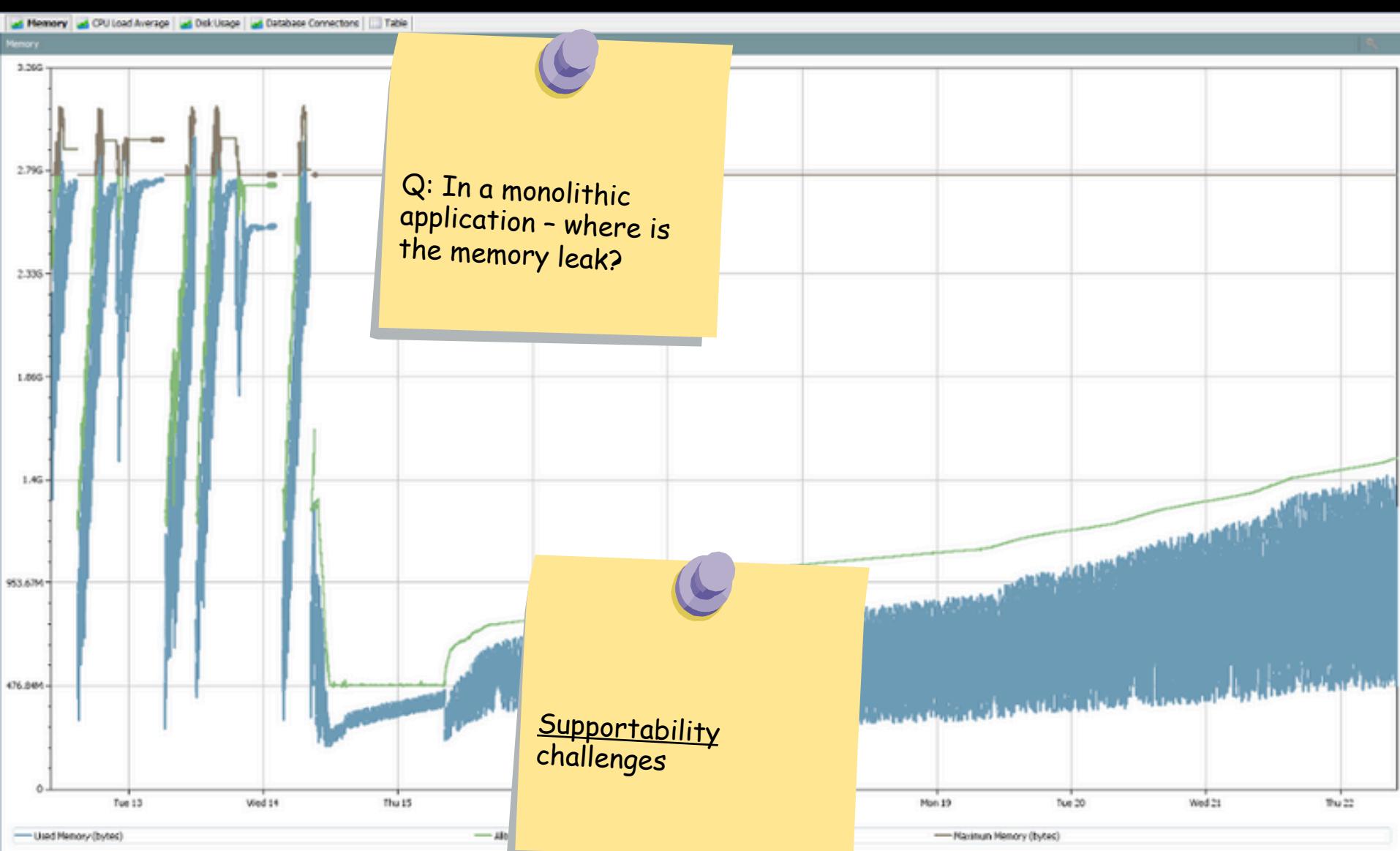


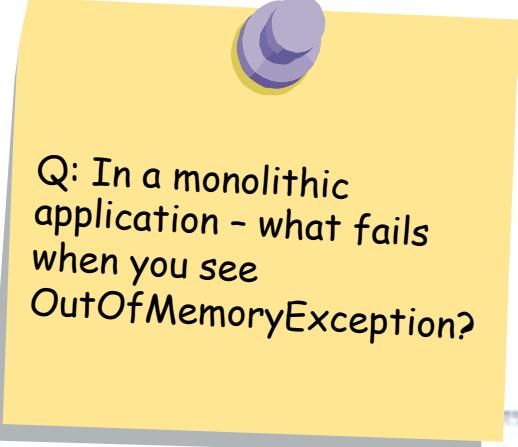
Yet we deploy as a monolith



Huge Java EE container



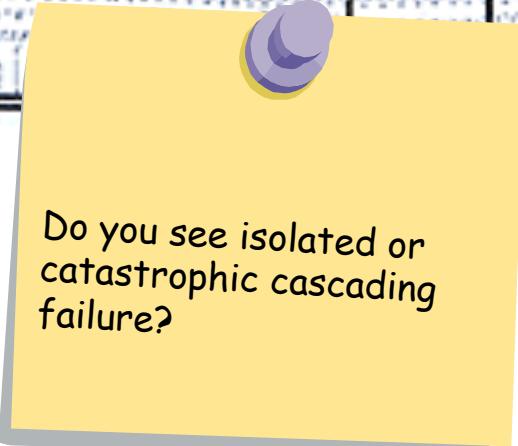




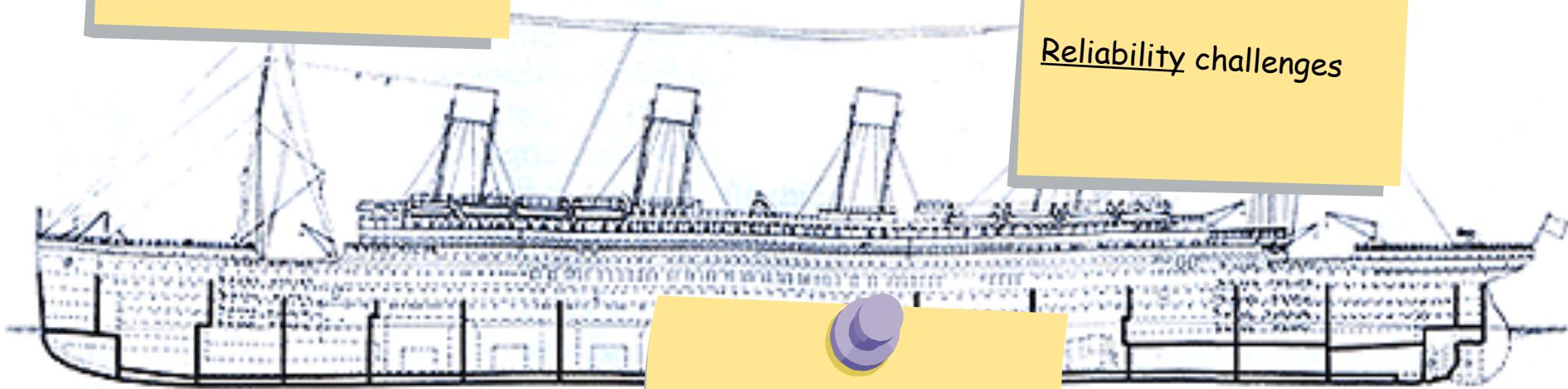
Q: In a monolithic application - what fails when you see OutOfMemoryException?

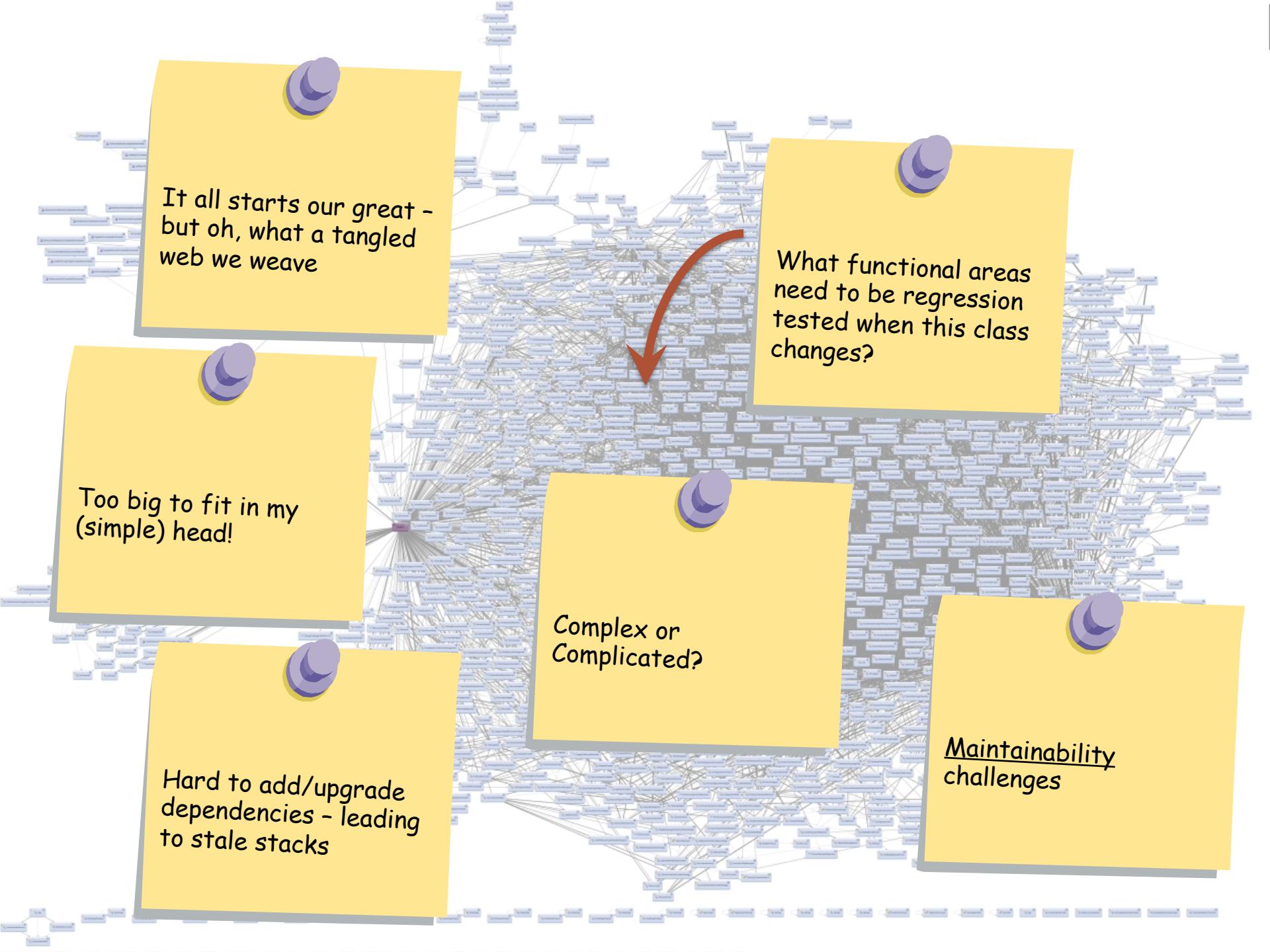


Reliability challenges

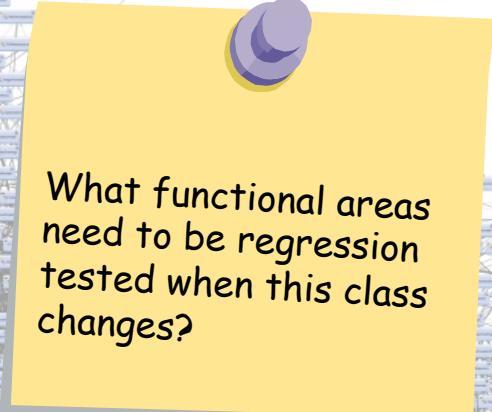


Do you see isolated or catastrophic cascading failure?





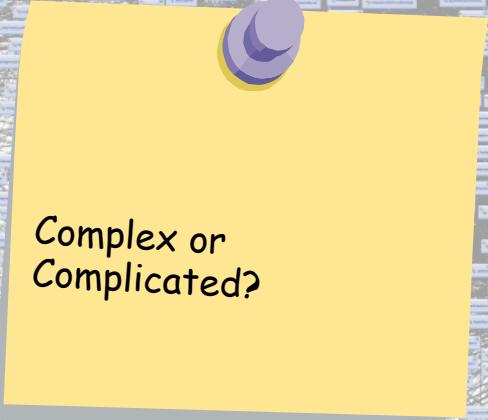
It all starts our great -
but oh, what a tangled
web we weave



What functional areas
need to be regression
tested when this class
changes?



Too big to fit in my
(simple) head!



Complex or
Complicated?



Hard to add/upgrade
dependencies - leading
to stale stacks

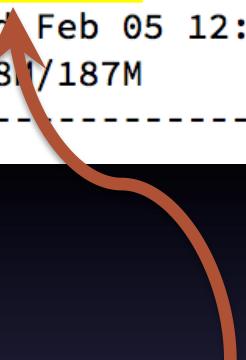


Maintainability
challenges

Do monolithic architectures impede Continuous Delivery?

“Adopting Continuous Delivery: Adjusting your Architecture”,
Rachel Laycock, Qcon,
<http://www.infoq.com/presentations/adopting-continuous-delivery>

```
[INFO] BUILD SUCCESS  
[INFO] -----  
[INFO] Total time: 19:14.974s  
[INFO] Finished at: Wed Feb 05 12:11:44 EST 2014  
[INFO] Final Memory: 781/187M  
[INFO] -----
```



Monolithic applications
can take forever to
build

Productivity killer

1. ssh

INFO: Server startup in 100676 ms





Too many cooks spoil
the broth



Impedes ability to scale
the team



The challenges of monolithic applications

Productivity

Supportability

Serviceability

Availability

Maintainability

Team Growth

Reliability

CITY OF Atlanta

Department of Watershed Management

Home

Documents and Reports

Offices

History

Clean Water Atlanta

Better Building Challenge

Floodplain Maps

Customer Service

Newsroom

Inside

Home » Inside DWM » History

History



DWM was formed in September 2002 to manage the City of Atlanta's essential utility operations: drinking water, wastewater and stormwater systems. As part of this operation, DWM manages one of the largest water capital improvement programs in the country at an estimated cost of approximately \$4 billion. In addition, the wastewater component of the capital program is largely controlled by two Federal Consent Decrees that have perhaps the most stringent and demanding schedule and performance requirements in the country. These Consent Decrees were ordered by the Federal Court in 1998 and 1999 to compel the City of Atlanta to address the conditions of the wastewater system that had been significantly underfunded and seriously under-maintained for decades. Watershed Management completed all construction for the first Consent Decree in 2008 in what was termed by the Federal judge a

This is just infrastructure stuff - why should we care?



Can micro-services help to address the challenges of monolithic applications?



The UNIX philosophy

Write programs that do
one thing.. and do it well

Write programs that
work together

```
#!/bin/sh
cat test.txt | tr ' ' '\n' | sort | uniq -c | sort -rn
```

Skip the usual
checkout routine.

Learn More

MasterPass[®]
An idea from MasterCard

ENTERPRISE

software

software | developers

The Second Coming of Java: A Relic Returns to Rule Web

BY CADE METZ 09.25.13 6:30 AM

Follow @cademetz

Just in Time for Twitter

On August 3, Twitter set a new record for tweets in a single second. As thousands of people in Japan jumped onto the service to discuss the television airing of the animated film *Castle in the Sky*, it hit a one-second peak of 143,199 tweets. That's a massive spike over the norm — about 5,700-tweets-per-second — and the site stayed up. "Our users didn't experience a blip," Krikorian recently wrote.

The moment was a far cry from the day Dmitry Medvedev visited Twitter HQ, and for Krikorian, it proves the worth of the company's new architecture.

Originally, Twitter was one, monolithic application built with Ruby on Rails. But now, it's divided into about two hundred self-contained services that talk to each other. Each runs atop the JVM, with most written in Scala and some in Java and Clojure. One service handles the Twitter homepage. Another handles the Twitter mobile site. A third handles the application programming interfaces, or APIs, that feed other operations across the net. And so on.

The setup helps Twitter deal with traffic spikes. Because the JVM is so efficient, it can handle much larger amounts of traffic with fewer machines. But the new operation is also more nimble. All these services are designed to communicate with each other, but if one goes down, it doesn't take the others down with it. The day we visited Krikorian at Twitter's offices this month, the Twitter homepage went dark for many people across the globe, but other services, including the company's mobile feed, kept on

The Micro-services architecture is an approach – the choice of platform and programming language is merely an implementation detail

On August 3, Twitter hit a one-second peak of 143,199 tweets. That's a massive spike over the norm — about 5,700 tweets per second.

Hmm – looks like Netflix also seem to be moving away from monolithic application structures

The screenshot shows a web browser window with the URL techblog.netflix.com/2012/06/netflix-operations-part-i-going.html. The page title is "The Netflix Tech Blog". The main content is a post titled "Netflix Operations: Part I, Going Distributed" dated Friday, June 15, 2012. The post discusses the challenges of moving from a monolithic Java application running inside a Tomcat container to a distributed cloud environment. It highlights the need for a deployment train, the challenges of managing code check-ins, and the cultural shift required for operations and development teams. The sidebar includes links to other Netflix blogs and an archive of blog posts.

NETFLIX

Friday, June 15, 2012

Netflix Operations: Part I, Going Distributed

Running the Netflix Cloud

Moving to the cloud presented new challenges for us^[1] and forced us to develop new design patterns for running a reliable and resilient distributed system^[2]. We've focused many of our past posts on the technical hurdles we overcame to run successfully in the cloud. However, we had to make operational and organizational transformations as well. We want to share the way we think about operations at Netflix to help others going through a similar journey. In putting this post together, we realized there's so much to share that we decided to make this a first in a series of posts on operations at Netflix.

The old guard

When we were running out of our data center, Netflix was a large monolithic Java application running inside of a tomcat container. Every two weeks, the deployment train left at exactly the same time and anyone wanting to deploy a production change needed to have their code checked-in and tested before departure time. This also meant that anyone could check-in bad code and bring the entire train to a halt while the issue was diagnosed and resolved. Deployments were heavy and risk-laden and, because of all the moving parts going into each deployment, it was handled by a centralized team that was part of ITOps.

Production support was similarly centralized within ITOps. We had a traditional NOC that monitored charts and graphs and was called when a service interruption occurred. They were organizationally separate from the development team. More importantly, there was a large cultural divide between the operations and development teams because of the mismatched goals of site uptime versus features and velocity of innovation.

Built for scale in the cloud

In moving to the cloud, we saw an opportunity to recast the mold for how we build and deploy our software. We used the cloud migration as an opportunity to re-architect our system into a service oriented architecture with hundreds of individual services. Each service could be revved on its own deployment schedule, often weekly, empowering each team to deliver innovation at its desired pace. We unwound the centralized deployment team and distributed the function of owning each service.

Links

- Netflix US & Canada Blog
- Netflix America Latina Blog
- Netflix Brasil Blog
- Netflix UK & Ireland Blog
- Open positions at Netflix
- Netflix Website
- Facebook Netflix Page
- RSS Feed

About the Netflix Tech Blog

This is a Netflix blog focused on technology and technology issues. We'll share our perspectives, decisions and challenges regarding the software we build and use to create the Netflix service.

Blog Archive

- 2014 (1)
- 2013 (52)
- ▼ 2012 (37)
 - December (6)
 - November (3)
 - October (2)
 - September (2)
 - July (6)
 - ▼ June (5)
 - Scalable Logging and Tracking

Gilt too!

The screenshot shows a web browser window displaying a presentation on InfoQ. The title of the presentation is "Enabling Microservice Architectures with Scala" by Kevin Scaldeferri, posted on July 22, 2013. The slide content features the text "GETTING (THE MICROSERVICES) RELIGION". On the left, there is a video player showing a man speaking, with a play button overlay. Below the video player are links for "View Presentation", "Download MP3", and "Slides". A summary text below the video describes the content of the presentation. At the top of the page, there are navigation links for various topics like Development, Architecture & Design, Process & Practices, Operations & Infrastructure, and Enterprise Architecture. A QCon banner is visible at the top right.

InfoQ

Enabling Microservice Architectures with Scala

by Kevin Scaldeferri on Jul 22, 2013 | Discuss

Summary

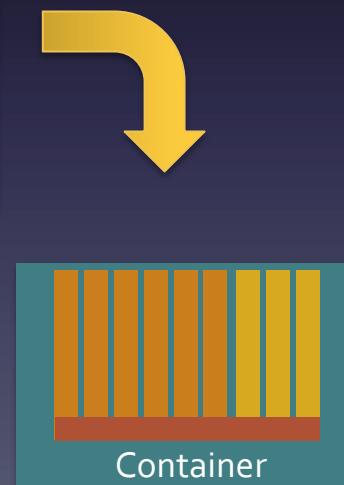
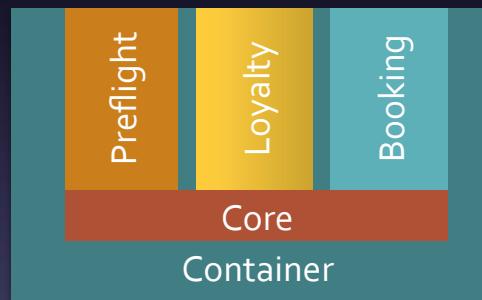
Kevin Scaldeferri reports on using Scala for an SBT plug-in, for unit and functional testing, type-safe shared configuration using Zookeeper, and live inventory with WebSocket and Akka Actors.

GETTING
(THE MICROSERVICES)
RELIGION

Bio

Kevin Scaldeferri works at Gilt Groupe, where he has worked on the core e-commerce and order processing components, as well as development tools.

Large airline website case study



Although we stopped short of micro-services (this was early 2000s) – the result of carving up into smaller chunks delivered significant value

Small (enough)

Replaceable

Independent

Runs in its own
O/S process
(daemon)

Easily
Consumable
(HTTP+JSON)

Fast/Easy to
startup

Upgrade-able

Free of
Temporal
coupling

Encapsulation

Forever young

Valuable

Testable

Single
Responsibility
Principle

What is a micro-service?

Complexity
displacement

Many processes
to manage

Increase in
overall memory
consumption

What are the costs of micro- services?

More moving
parts

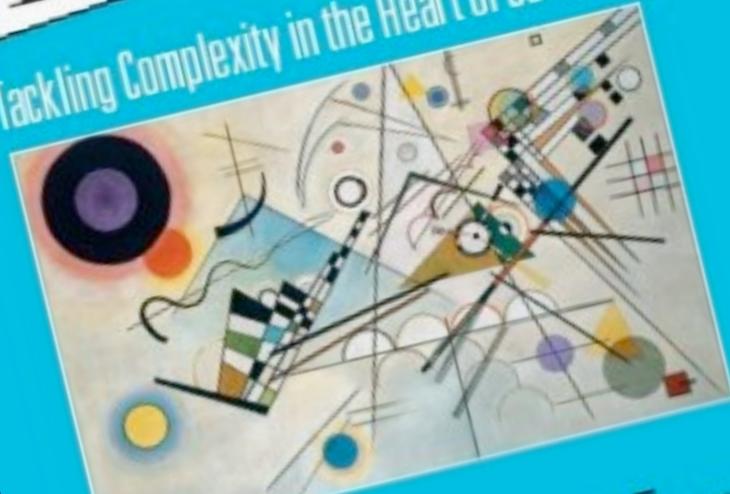
Operational
complexity

Design considerations

*“Good OO design principles
are equally applicable for the
design of micro-services”*

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

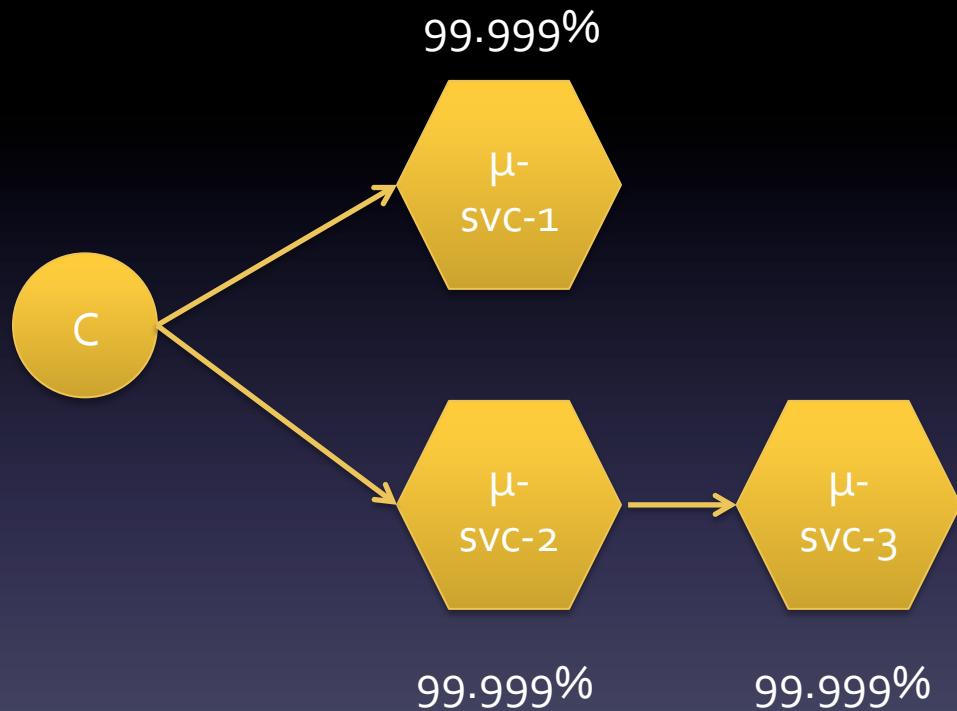


Eric Evans
Foreword by Martin Fowler

Domain-Driven Design
(DDD) is complementary to
a Micro-services based
architecture

*“The hardest part is going to
be finding the seams in a
monolithic application to form
the boundaries of the micro-
services”*

Availability



$$\text{Availability} = \frac{\text{Uptime}}{\text{Uptime} + \text{Downtime}} \times 100\%$$

$$99.999\% = 5.26 \text{ minutes downtime/year}$$

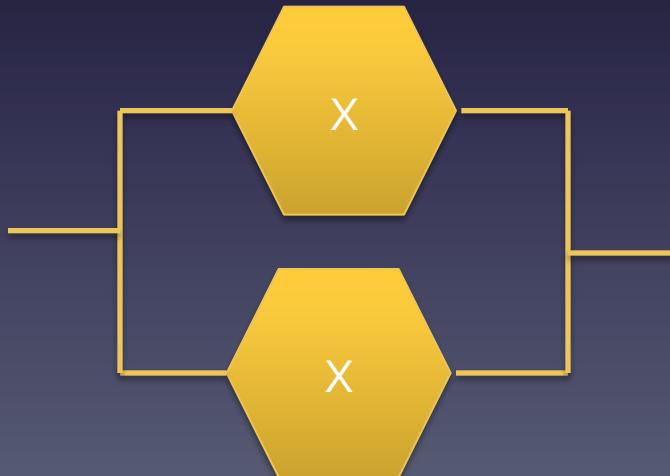
Q. What is the overall availability?
= $A(\mu\text{-svc-1}) \times A(\mu\text{-svc-2}) \times A(\mu\text{-svc-3})$
= 99.999% \times 99.999% \times 99.999%
= 99.997% = 15.77 minutes of downtime/year

Availability

$$A = A_x A_y$$



	X	Y	X & Y combined
	0.99	0.9999	0.989901
minutes/year	525600	525600	525600
Uptime mins/year	520344	525547.44	520291.9656
Downtime mins/year	5256	52.56	5308.0344
Downtime days/year	3.65	0.0365	3.686135

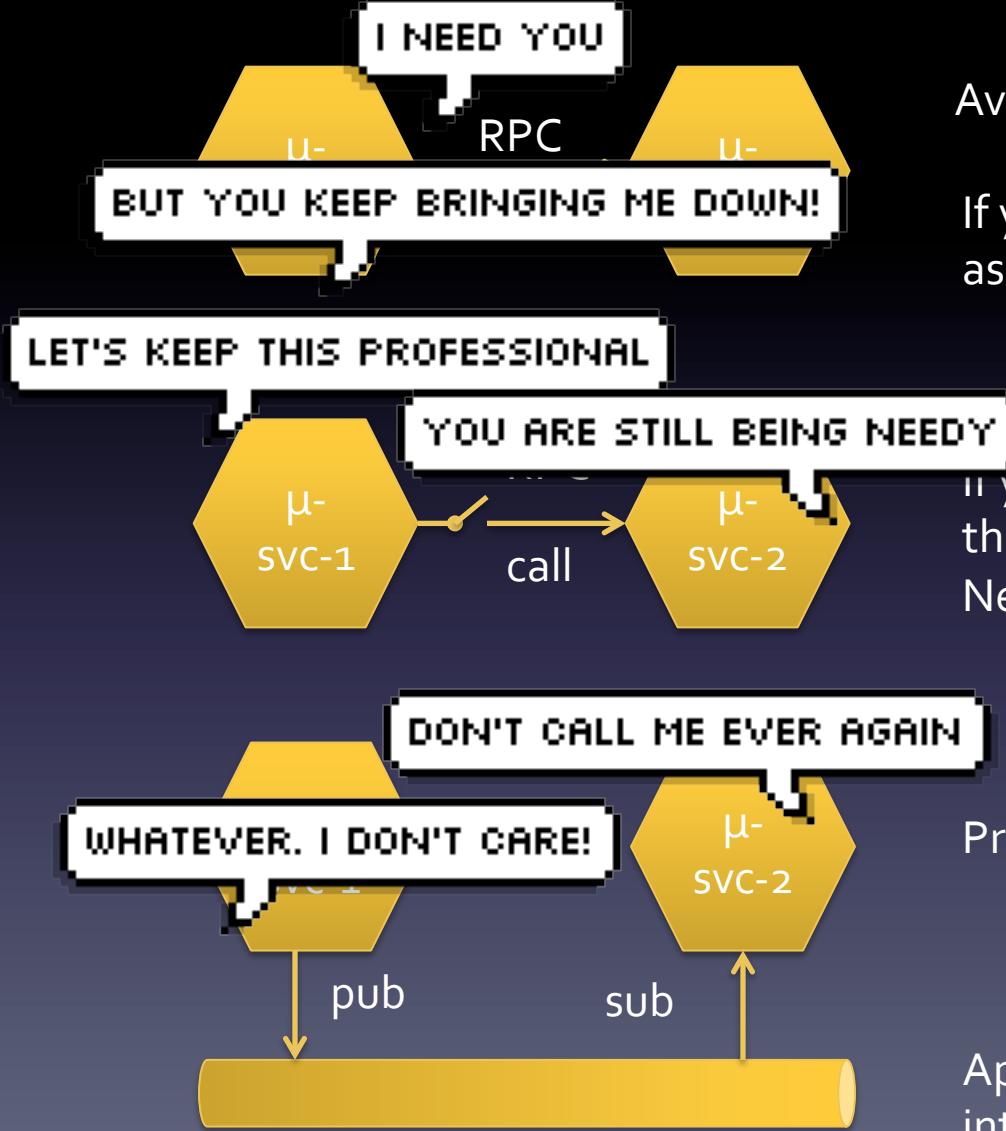


$$A = 1 - (1 - A_x)^2$$

	X	2x X	3x X
	0.99	0.9999	0.999999
minutes/year	525600	525600	525600
Uptime mins/year	520344	525547.44	525599.4744
Downtime seconds/year	315360	3153.6	31.536
Downtime mins/year	5256	52.56	0.5256
Downtime days/year	3.65	0.0365	0.000365

“Overall availability is reduced with each dependency. With micro-services the number of dependencies may increase.”

Micro-services working together



Avoid synchronous RPC calls – why?

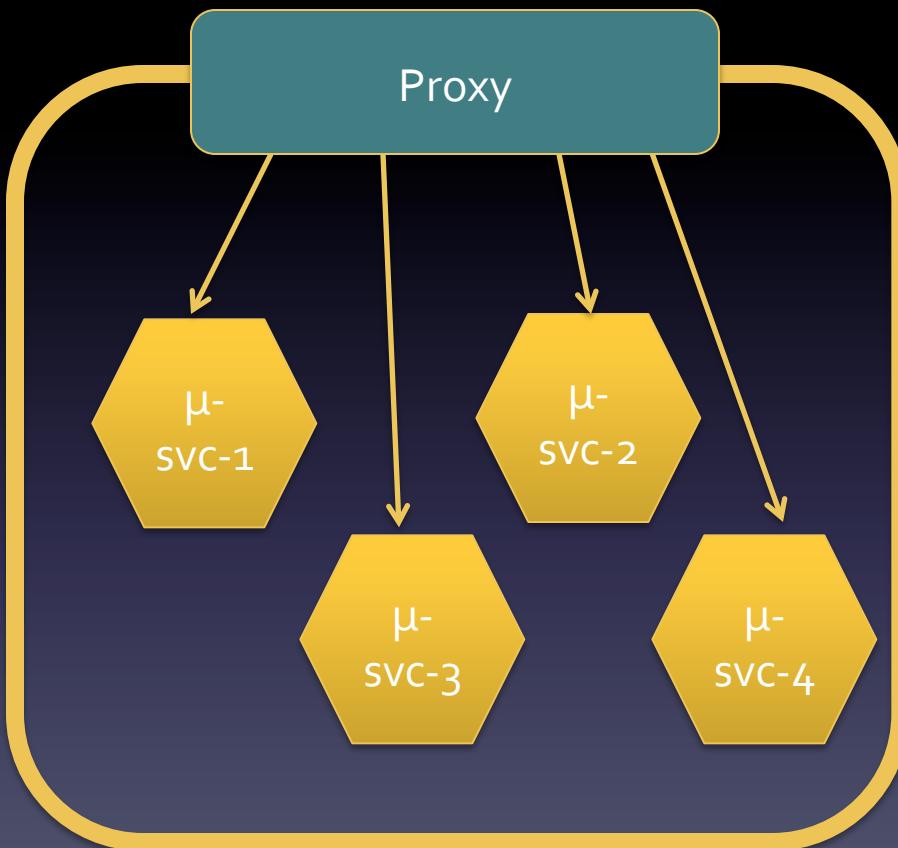
If you feel you need to do this, first ask – did I get the boundaries right?

If you still feel you need to do this, then consider a circuit breaker (e.g. Netflix Hystrix)

Prefer async pub/sub – why?

Apache Kafka may be worth looking into as a messaging backbone

Securing micro-services

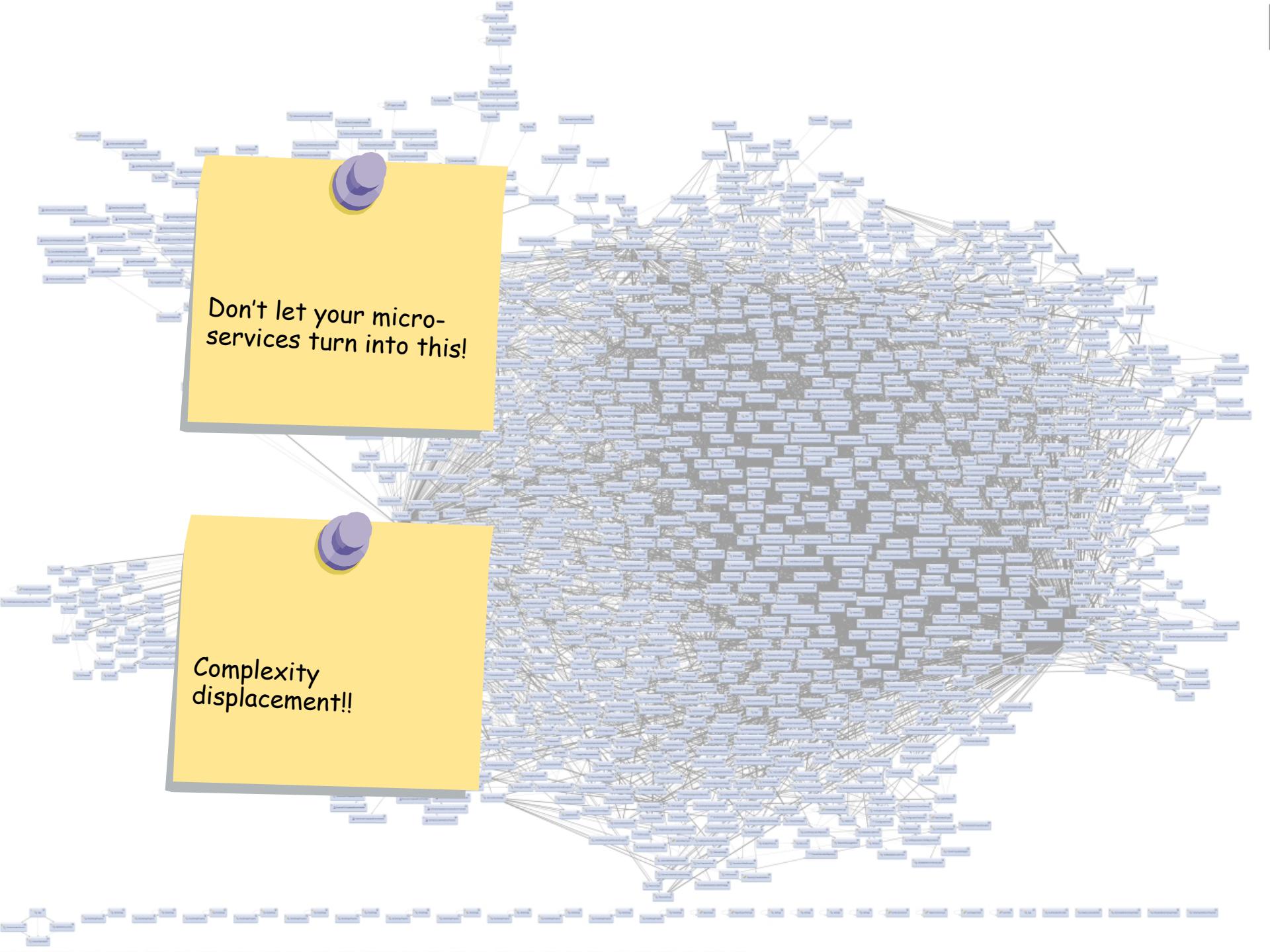


Consider fronting the micro-services with a proxy (with a circuit breaker)

Maybe you want to control access

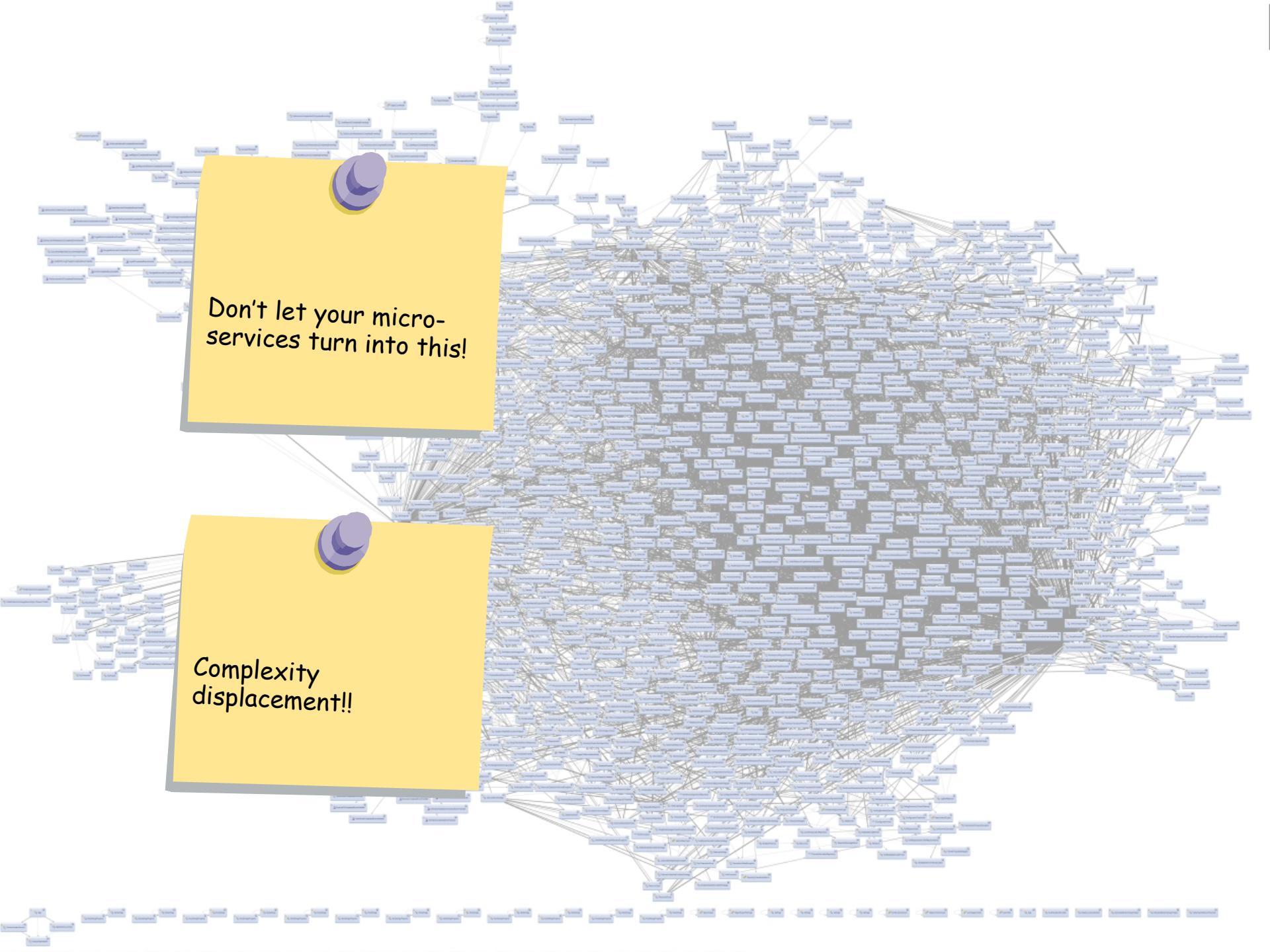
Maybe you need to invoke the services from a web client

Maybe you want to avoid having to open more holes in the firewall



A large, dense network graph composed of numerous small, light-blue rectangular nodes connected by a complex web of thin grey lines, representing a highly interconnected system of microservices.

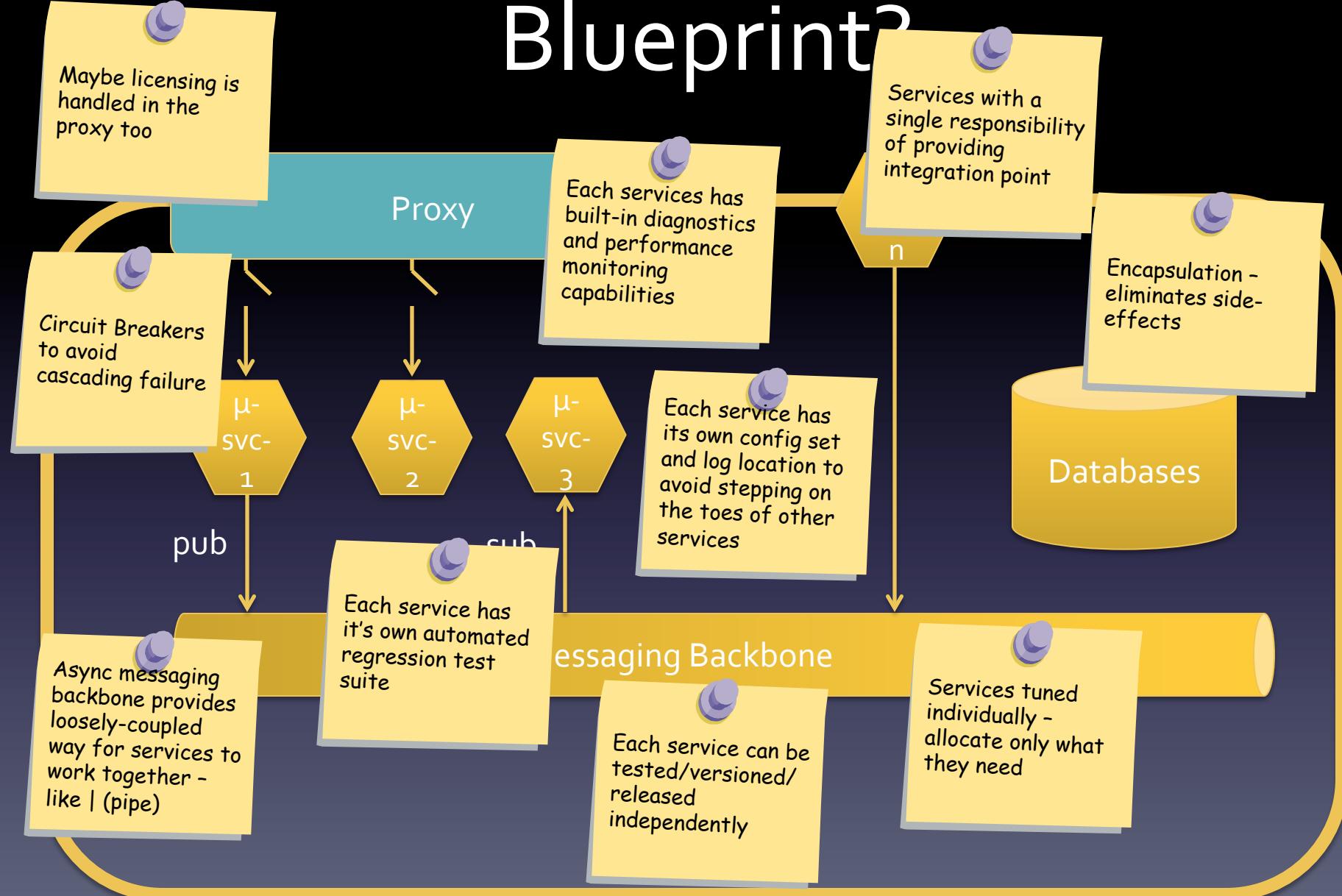
Don't let your micro-services turn into this!



A large, dense network graph composed of numerous small, light-blue rectangular nodes connected by a complex web of thin grey lines, representing a highly interconnected system of microservices.

Complexity displacement!!

Blueprint?



Execution considerations

Use whatever language/
platform you are
comfortable with today

For existing projects,
carve off micro-
services incrementally -
avoid big-bang

.. as long as it meets
"your" definition of
what micro-services are

Polyglot (be careful of
Polyglut)

Something like Dropwizard can help you get started quickly with a Java based implementation of micro-services



Dropwizard is a Java framework for developing ops-friendly, high-performance, RESTful web services.

Developed by [Yammer](#) to power their JVM-based backend services, Dropwizard pulls together **stable, mature** libraries from the Java ecosystem into a **simple, light-weight** package that lets you focus on *getting things done*.

Dropwizard has *out-of-the-box* support for sophisticated **configuration, application metrics, logging, operational tools**, and much more, allowing you and your team to ship a *production-quality* HTTP+JSON web service in the shortest time possible.

[Getting Started »](#)

[User Manual »](#)

[About Dropwizard »](#)



Something like this can help
with performance
monitoring



Metrics is a Java library which provides unparalleled insight into what does in production.

Metrics provides a powerful toolkit of ways to measure the behavior of critical components in your production environment.

With modules for common libraries like **Jetty**, **Logback**, **Log4j**, **Apache HttpClient**, **Ehcache**, **JDBI**, **Jersey** and reporting backends like **Ganglia** and **Graphite**, Metrics provides you with full-stack visibility.

[Getting Started »](#)

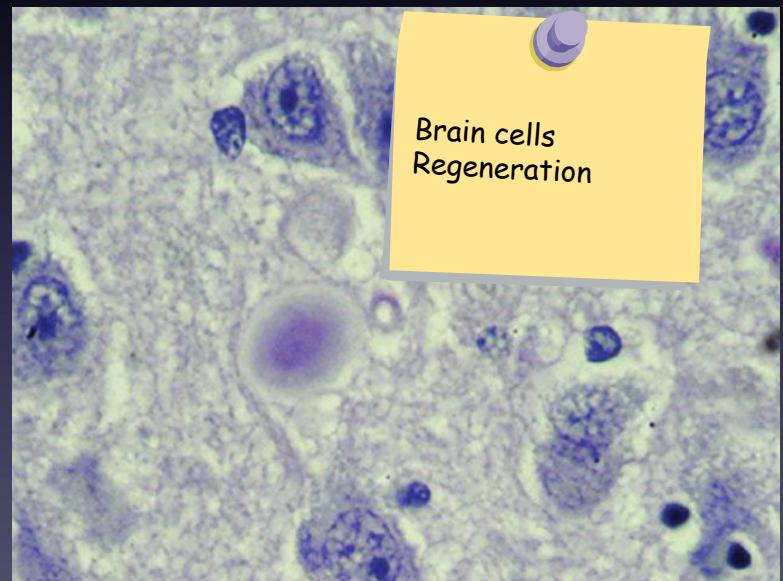
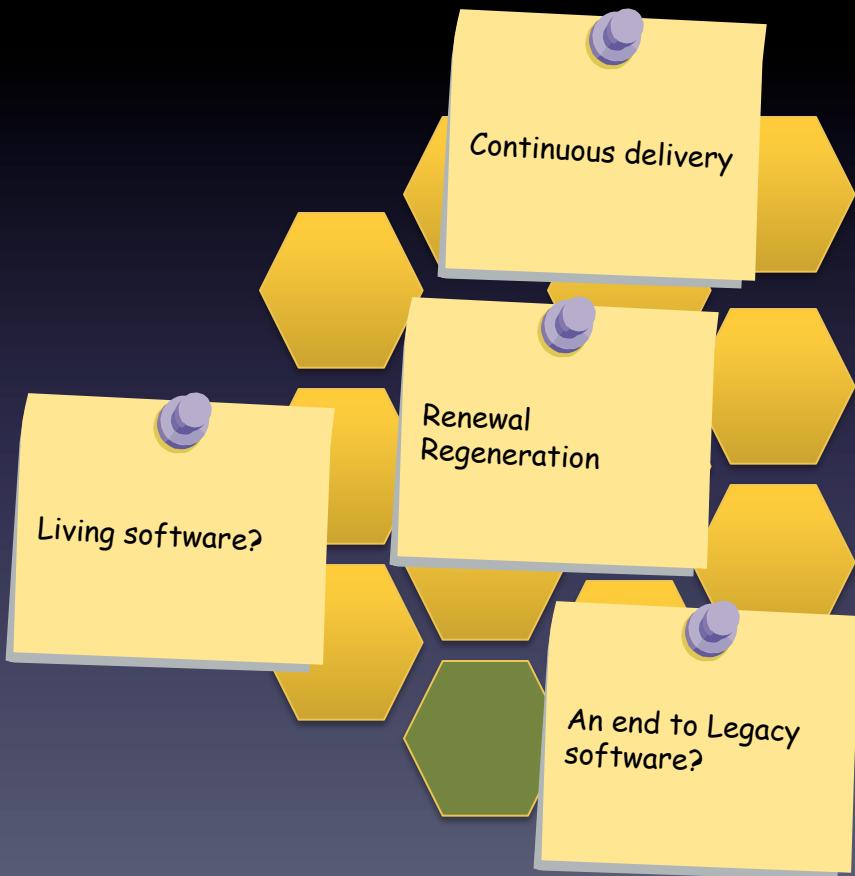
[User Manual »](#)

[About Metrics »](#)

close()

Micro-services

Microbiology



<http://www.genome.gov/dmd/img.cfm?node=Photos/Technology/Cells%20and%2obiological%2opathways&id=79194>

*“If you look after the
pennies, the pounds will
look after themselves.”*



Further Reading

- "The Second Coming of Java: A Relic Returns to Rule Web",
Wired, Cade Metz, September 2013,
<http://www.wired.com/wiredenterprise/2013/09/the-second-coming-of-java/2/>
- "Micro Services: Java, the Unix Way",
QCon San Francisco 2012, James Lewis
<http://www.infoq.com/presentations/Micro-Services>
- "GOTO Berlin: Microservices as an Alternative to Monoliths",
InfoQ, Jan Stenberg, October 2013
<http://www.infoq.com/news/2013/10/goto-ber-microservices>
- "Domain-Driven Design: Tackling Complexity in the Heart of Software",
Addison-Wesley, Eric Evans, August 2003
<http://www.amazon.com/Domain-Driven-Design-Tackling-Complexity-Software/dp/0321125215>
- "Baruco 2012: Micro-Service Architecture",
Fred George, November 2012
<http://www.youtube.com/watch?v=2rKEveL55TY>
- "Enabling Microservice Architectures with Scala ",
Kevin Scaldeferri (Gilt), Jul 22 2013
<http://www.infoq.com/presentations/Microservice-Architectures-Scala>
- "Adopting Continuous Delivery: Adjusting your Architecture",
Rachel Laycock, Qcon,
<http://www.infoq.com/presentations/adopting-continuous-delivery>
- <http://yobriefca.se/blog/2013/04/29/micro-service-architecture/>
- Dropwizard, <http://dropwizard.codahale.com/>
- Metrics, <http://metrics.codahale.com/>
- Netflix Hystrix, <https://github.com/Netflix/Hystrix>
- Apache Kafka, <https://kafka.apache.org/>
- System Reliability and Availability
http://www.eventhelix.com/realtimemantra/faulthandler/system_reliability_availability.htm#.UtASOGRDuAR
- City of Atlanta Department of Watershed Management history <http://www.atlantawatershed.org/inside-dwm/history/>



Lancope

VISION TO SECURE,
INTELLIGENCE TO PROTECT

THANK
YOU

Jason Chambers
Director of Engineering

DEVNEXUS