CONQUERING THE MONOLITH

ARCHITECTING FOR DEVOPS AND RELEASE ON DEMAND

MARC RIX
Continuous Delivery Pipeline

Continuous Exploration  Continuous Integration  Continuous Deployment  Release on Demand

© Scaled Agile, Inc.

A CALL TO ACTION
AVG SERVER AGE: 24 DAYS

1B TRACKED METRICS

NETFLIX

SELF-SERVICE DEPLLOYS

ZERO DOWNTIME
WE GET IT, BUT WE’RE NOT UNICORNS
IT BUDGET: $1B+ PER YEAR
PERFECTED THE DEVOPS DOJO

TARGET

80+ DEPL OYS PER WEEK
17B+ API CALLS PER MONTH
20K+ TESTS PER COMMIT

600 DEVELOPERS

DEPLOY EVERY CLEAN BUILD

ONE CODE BRANCH
IT DOESN’T MATTER IF YOUR APPS ARE GREENFIELD, BROWNFIELD OR LEGACY… CONTINUOUS DELIVERY CAN BE APPLIED TO ANY SYSTEM.

DON’T FOCUS ON THE TYPE OF SYSTEM YOU HAVE; FOCUS ON RE-ARCHITECTING FOR TESTABILITY AND DEPLOYABILITY.
You’re bought in to SAFe, DevOps and their benefits
You’re bought in to SAFe, DevOps and their benefits

Your architecture isn’t
ARCHITECTURE IS THE FUNDAMENTAL ORGANIZATION OF A SYSTEM, EMBODIED IN ITS COMPONENTS, THEIR RELATIONSHIPS TO EACH OTHER AND THE ENVIRONMENT

ANSI/IEEE Std 1471-2000
ARCHITECTURE IS ABOUT THE IMPORTANT STUFF. WHATEVER THAT IS.

Ralph Johnson, IEEE Standards Reviewer, XP Thought Leader
(NOT Martin Fowler)
WE NEED TO GET TO THESE RESULTS

- Happier, more motivated employees
- 20% - 50% increase in productivity
- 30% - 75% faster time to market
- 50%+ defect reduction

- Faster lead time: 440x
- Faster recovery: 24x, 2x
- Fewer defects: 96x
- Improved employee Net Promoter Score: 29%, 46%
- Less time spent fixing security issues: 2x
- More frequent deployments: 2x
- More likely to exceed profitability goals: 2.2x

Source: https://puppet.com/resources/whitepaper/state-of-devops-report
TEAMS ARE LOOKING MORE LIKE THIS
BUT DEPENDENCIES ARE STILL A BIG CHALLENGE
HMMM...KINDA RESEMBLES OUR ARCHITECTURE
ARTISTIC, YES?
AND THERE IT IS
I wonder what’s clogging my pipeline?
HOW TO CONQUER THE MONOLITH: LIQUIFY
HOW TO CONQUER THE MONOLITH:

DECOUPLE.
LOOSELY COUPLED ARCHITECTURES AND TEAMS ARE THE STRONGEST PREDICTOR OF CONTINUOUS DELIVERY
OUR STARTING POINT
(WHAT A CLUSTER!)
SOLUTION:

DETANGLE!
STEP 1: APPLY DDD TO IDENTIFY LOGICAL DOMAINS

- **Customer**
- **Order**
- **Search**
STEP 2: GROUP RELATED MODULES & COMPONENTS INTO SUB-SYSTEMS (BOUNDED CONTEXT)
STEP 3: WALL OFF DOMAINS ("SHARE NOTHING")
OK, SHARE VIA APIs
WHAT ABOUT ALL THIS OTHER STUFF?
STEP 4: CREATE SHARED LIBRARIES
BETTER ORGANIZED
FEWER DEPENDENCIES
MORE CHANGEABLE
ALIGNED TO BUSINESS
FASTER TESTING
MORE PORTABLE

BUT STILL MONOLITHIC
OUR SOFTWARE ARCHITECTURE SHOULD ENABLE SMALL TEAMS TO BE INDEPENDENTLY PRODUCTIVE
STEP 5: EVOlVE TO SERVICE-BASED ARCHITECTURE
Customer System

API Management

Order System

Search System

Separately deployable systems
Shared DB per system
Decoupled Biz systems
Fewer defects / breaks
More Dev options
More Infra options
Agile Team-aligned

Still could be better...
STEP 6: EVOLVE TO MICROSERVICES ARCHITECTURE
ALSO CONSIDER:

- Reverse Conway's Law
- Strangler pattern
- Be careful with "canonical" & "master" data formats
- Test within (not across) domains
- Not all monoliths are evil
- Plan to re-architect again in 5 years
### YEAH, THEY HAD TO CONQUER MONOLITHS TOO

<table>
<thead>
<tr>
<th></th>
<th>Google</th>
<th>Amazon</th>
<th>LinkedIn</th>
<th>Netflix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic in</td>
<td>2005</td>
<td>2004</td>
<td>2011</td>
<td>2011</td>
</tr>
<tr>
<td>Core System</td>
<td>Google.com</td>
<td>OBIDOS</td>
<td>Leo</td>
<td>Content Server</td>
</tr>
<tr>
<td></td>
<td>(GWS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tech</td>
<td>C++</td>
<td>Rails</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>Time to fix</td>
<td>ongoing</td>
<td>4 years</td>
<td>2 months*</td>
<td>6 years</td>
</tr>
</tbody>
</table>
Thank You!

Marc Rix
SAFe Fellow, SPC
DevOps Practice Lead, Architect

linkedin.com/in/marcrix
@marcrix