Microservices: 
Service Oriented Development 
Rafael Schlomming
How do I break up my monolith?
How do I architect my app with microservices?
What infrastructure do I need in place before I can benefit from microservices?
Microservices at Datawire ...

- Building a cloud application using microservices in 2013
- Distributed systems engineers
- Multiple services
- Prototyping was really fast

... then we launched and things slowed down...
Debugging Velocity (or lack thereof)
Tooling Architecture Process!!!
Debugging our Pipeline
Velocity comes from Process, not Architecture
Service-Oriented Architecture
Service-Oriented Development
A single process is inefficient
(Forces a single Stability vs Velocity Tradeoff)
Centralized process

- Specialized teams
- Fixed policies (e.g., release criteria)
A single process doesn’t scale
How do I break up my monolith?
How do I break up my process?
Microservices lets you run multiple processes!
Microservices is a distributed development architecture workflow.
- How do I get to Continuous Deployment incrementally?
- How do I limit the scope of PCI (audit process)?
- How do I ship feature X as fast as possible?
Microservices is ...

- **Multiple workflows**
  - Including your existing workflow!
  - Workflows designed for different stability/velocity tradeoffs

- **Simultaneous workflows**
Doing things this way shifts how people operate!

- Requires both organizational and technical changes
Organizational Implementation
You gotta give in order to get

Education

● Everyone exposed to full dev cycle

Communication

● Nobody speaks the same language

Delegation

● Small teams own big important parts
But you get a lot

Education

- Specialists become generalists -> Better holistic systems
- Learning, personal growth -> Job satisfaction

Communication

- Conflict -> Collaboration

Delegation

- Massive organizational scale
Create self-sufficient, autonomous software teams.
Why self-sufficiency and autonomy?

- Self-sufficient
  - Team does not need to rely on other teams to achieve its goals
- Autonomy
  - Team is able to independently make (process) decisions on how to achieve its goals
Eliminate centralized specialist functions

Centralized architecture

Centralized infrastructure / ops* (You might need a platform team)
Think Spinoff
Technical Implementation
The Workflows

Stage
- Prototype
- Production Users & Growth
- Mission Critical

Goals
- Fast Feedback from both Tools & Users
- Add Features & Don’t Disrupt Users
- Stability
One Platform, Parallel Workflows, Seamless Transitions

<table>
<thead>
<tr>
<th>Stage</th>
<th>Goals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prototype</td>
<td>Fast Feedback from both Tools &amp; Users</td>
</tr>
<tr>
<td>Production Users &amp; Growth</td>
<td>Add Features &amp; Don’t Disrupt Users</td>
</tr>
<tr>
<td>Mission Critical</td>
<td>Stability</td>
</tr>
</tbody>
</table>
Kubernetes / Docker / Envoy give you the infra you need
How do I actually use these technologies to build my workflows?
Stage 1: Prototyping

Goal:
Fast Feedback from both Tools and Users

Org Problem:
You need buy-in for prototyping in production

Tech Problem:
You can’t run microservices locally
Strategy: Self Service Provisioning & Development Containers
Provide fast self-service provisioning

Make this fast and easy!

- Too much friction leads to accidental coupling
Problem: Coding on remote infra is slow...

VM based pipeline:
- Deploy time: maybe 45 minutes?

Docker based pipeline:
- Deploy time: maybe a few minutes?

Hacking react on my laptop with live reload:
- Maybe 1-2 seconds?

Hacking flask on my laptop with live reload:
- Instantaneous
How can we do better?
Develop inside a container

Helps with onboarding and jumping between services:

- Single source of truth for build & dependencies
- Consistent and portable dev environment

You can make a faster feedback loop:

1. Sync local files -> remote build
2. Sync local files -> local build; snapshot image; deploy in seconds
3. Sync local files -> local build; proxy into remote cluster

Shameless self promotion:

- See https://forge.sh for (2) and https://telepresence.io for (3)
Fast Deploy == Resilience
Stage 2: Production Users & Growth

Goal:
Add Features
&
Don’t Disrupt Users

Org Problems:
Recognize the Tradeoff
&
How to measure user impact

Tech Problem:
Software Bugs
Strategy: Genetic Diversity (Multiversion Deployment)
Multiple versions for software redundancy

End user

Primary version

Canary version

Dev version
Stage 3: Mission Critical

Goal: Stability

Org Problem: Avoid regressing

Tech Problem: L7 Observability
Strategy:
Service Level Objectives
&
L7 Observability
Cascade Failures
Summary

1. Start with: “How do I break up my monolithic process?”
2. Spinoff self sufficient & autonomous teams
3. Build awesome tooling for Service Oriented Development
Thank you!

- rhs@datawire.io
- If you want to learn more about these ideas, check out our hands-on tutorial here:
  - https://datawire.io/faster
- If you’re interested in any of our open source tools, check them out:
  - https://forge.sh for deployment
  - https://www.telepresence.io for real-time live coding
  - https://www.getambassador.io self-service API Gateway built on Envoy