From Monoliths to Microservices

JEFF NICKOLOFF

Founder, Topple

But why migrate?

1. Release engineering for monoliths becomes challenging as groups and codebases scale.
2. Changing technological context is difficult to integrate in monolithic projects.
3. Side effects, leaky abstractions, resource isolation.
4. Security?

What do we hope to learn?

1. How and why microservice architectures and ownership end up falling along organizational lines.

2. How we can learn from monolith tooling to inform our tooling in a microservice environment.

3. How you can achieve operational excellence at scale taking a logistical approach with Docker.

Challenges

- Building, shipping, and running…
- Same challenges, but different **context**.

- How will the context change?

Distributed and Independent

- More, smaller focused components
- Components have independent lifecycles
- Network vs memory component addressing
- Distributed and localized state
- Unreliable wall clocks
- Independent component runtime state (health)
- Independent component “experiences”

Silos and Cognitive Limits

Dunbar's Number

- The number is about 150
- What is it?
- Relationships in constant flux. So is code.

Conway's Law

“Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations.”

Why should we embrace it?

Isolation in Management and Operation

- Focused development
- Focused on-call rotations
- Integration on interface reduces dev risk
- Ops specialization reduces “time to root cause”
- Change impact has “perspective”

Contract (API) Versioning

- More than “names and numbers… ”
- Versioning contracts can be extremely painful
- Validate contract changes early, and prefer backwards compatibility
- Track contract dependencies
- Anticipate long adoption cycles


Tools

Tool Forms

- Monoliths
  - Shared libraries
  - Code weaving with Aspect Oriented Programming
- Microservices
  - Middleware
  - Sidecar processes
  - Orchestrators

Trustworthy Communication

- Not simple function calls
- Inter-component:
  - Authentication
  - Authorization
  - Channel encryption
  - Maintenance

- Tool? Service Mesh; API gateways; Netflix Hystrix

Distributed Coordination

- No more in-process singletons:
  - Distributed locking
  - Leadership election

- Tools? Orchestrators; DIY with storage locks and fencing

Disjoint Event Streams

- Linearizing log/event streams without clocks
- Log aggregation and centralization

- Tools? ELK; Fluent, Syslog

Multi-Process Debugging

- Tracing the causal relationships between a set of inter-component requests
- Tools: Zipkin, Jaeger, Service Mesh

Evolution of Testability

- Monoliths
  - Tests versioned with code
  - Tests consistent snapshots of code
  - Unit, Functional, and Integration Testing
  - Loose contract between components
  - Fails before deployment

Evolution of Testability

- … do the same things, just continuously at runtime.
- Microservices
  - No runtime consistency -> snapshot testing is less useful
  - Need continuous or edge integration testing
  - Contract validation becomes critical
  - **Tests define the service contracts**

Dependency Management

- Names and versions identify contracts
  - Interfaces, structures, data. **Not just code**.
- Process vs Code Dependencies
  - Must code to interface
  - Runtime dependency modeling
- Tool? Getting into logistics…

Logistics for Scale

Artifact Management

- More artifacts
  - Release artifacts: binaries, and images
  - Runtime artifacts: processes, names, and containers
  - Configuration: config, secret material
- Securing supply chain

- Tools: Docker registry, pipelines (CI/CD)

Controlling Change

- “Release Engineering” vs. “Distribute Change Management”
- More processes
- Declarative desired state
  - Don’t describe changes
  - Describe processes, configuration, and state
- - Version control it

- Tools: Orchestrators

Monitoring and Health

- Monitoring 1 service vs N services
- How do you define health?
- How often do you test?
- How many missing or out of threshold points trigger alarms?
- What actions?

- Tools: Orchestrators, Nagios, Visibility/Monitoring platforms

Build-time vs Run-time Integration

- Monoliths integrate on memory addresses
  - At build-time with static linking (once during build)
  - At run-time with dynamic linking (once at startup)
- Microservices integrate on network addresses
  - Runtime only
  - Distributed and continuous linking

Need a distributed linker

(we already have a few)

“Distributed Linking”

- Linking is all about name to address resolution
- On a network we call that service registration and discovery (SD)
  - Bootstrap addresses (static)
  - DNS (built-in)
  - VIP (load balancing, reverse-proxies, etc.)
  - Native SD (proprietary interfaces)
  - Service Mesh (woven sidecars – configuration, not code)

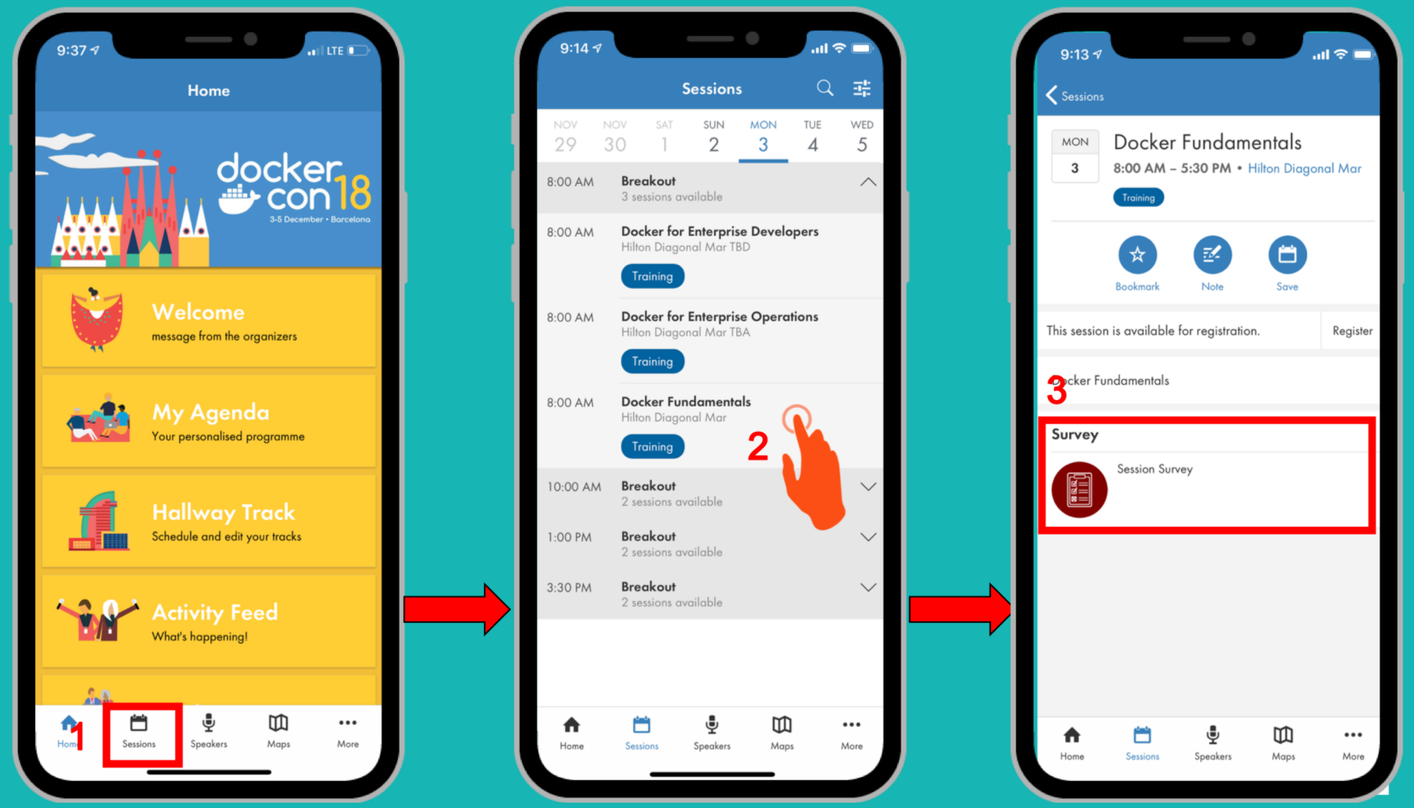
Remember

If you experience a new problem, look for equivalent tools from your monolith stack.

Take A Breakout Survey

Access your session and/or workshop surveys for the conference at any time by tapping the Sessions link on the navigation menu or block on the home screen.

Find the session/workshop you attended and tap on it to view the session details. On this page, you will find a link to the survey.



Ciao

Jeff Nickoloff - @allingeek
gotopple.com