Monoliths, Migrations, and Microservices

Randy Shoup
@randyshoup
linkedin.com/in/randysouh
Background

• VP Engineering at WeWork
  o Physical space as a service

• VP Engineering at Stitch Fix
  o Using technology and data science to revolutionize clothing retail

• Director of Engineering for Google App Engine
  o World’s largest Platform-as-a-Service

• Chief Engineer at eBay
  o Evolving multiple generations of eBay’s infrastructure
“Tell us how you did things at Google and eBay.”

“Sure, I’ll tell you, but you have to promise not to do them! […] yet]”
Architecture Evolution

• eBay
  • 5th generation today
  • Monolithic Perl → Monolithic C++ → Java → microservices

• Twitter
  • 3rd generation today
  • Monolithic Rails → JS / Rails / Scala → microservices

• Amazon
  • Nth generation today
  • Monolithic Perl / C++ → Java / Scala → microservices
No one starts with microservices

... Past a certain scale, everyone ends up with microservices
If you don’t end up regretting your early technology decisions, you probably over-engineered.
What problem are you trying to solve?
Idea Phase: “Prototype” Architecture

- Goal: Explore the space as rapidly and cheaply as possible
  - Find business model
  - Find product market fit
  - Acquire first customers

- ➔ Rapid Iteration
  - *Everything* is a prototype
  - You *(will)* throw it all away
Idea Phase: “Prototype” Architecture

- Ideally No Technology At All
  - Paper prototypes
  - Google ads
  - Excel spreadsheet
  - Wordpress blog

- If you *really* need to build something ...
  - Familiar technology
  - Cobble it together
“Do things that don’t scale”

-- Paul Graham
One Team
3-6 month horizon

https://ittybiz.com/s-curve/
Two-Pizza Team

“A team should be no larger than can be fed by two large pizzas.”

-- Jeff Bezos, Amazon
Starting Phase: “Just Enough” Architecture

- Goal: Meet near-term, evolving customer needs as cheaply as possible
  - Delight first customers
  - Acquire more

- Rapid learning and improvement

- Team productivity

- NOT about scaling
“The best code you can write now is code you’ll discard in a couple of years time”

-- Martin Fowler

http://martinfowler.com/bliki/SacrificialArchitecture.html
Starting Phase: “Just Enough” Architecture

- Simple, Familiar Technology
  - Ease of Use
  - Expressive Power
  - Rapid prototyping frameworks (Ruby / Rails, PHP, etc.)

- Monolithic Architecture
  - Single application
  - Single database

- Minimal Infrastructure
  - Ideally serverless
  - PaaS or Lambda instead of IaaS
The Monolithic Architecture

2-3 monolithic tiers

Presentation

Application

Database
Monolithic Architecture

**Pros**
- Simple at first
- In-process latencies
- Single build and deployment unit
- Resource-efficient at small scale

**Cons**
- Coordination overhead as team grows
- Poor enforcement of modularity
- No horizontal scaling
- Single point of failure, single performance bottleneck
Starting Phase: Preparing to Scale

- Modularity Discipline
  - Use “shared libraries” within the monolith
  - Easy to modify or replace

- Detailed Logging
  - Understanding user behavior
  - Instrumenting for diagnosis and recovery

- Continuous Delivery
  - Deploy many times per day
When to Rearchitect?

• **Velocity**
  - Time to market is constrained by coupling and lack of isolation in the monolith
  - Teams step on each others’ toes, and can no longer develop independently
  - Difficult for new engineers to be productive

• **Scaling**
  - Vertical scaling of the monolith no longer works
  - Parts of the system need to scale independently of others
When to Rearchitect?

- Deployment
  - Parts of the system need to deploy independently of others
  - Monolithic release is too slow, too complicated, too risky
More and More Teams

6-24 month horizon

https://ittybiz.com/s-curve/
Scaling Phase: Scalable Architecture

- Goal: Stay ahead of rapidly growing business. Keep the site up (!)

- Scaling the Teams

- Scaling the Technology

- Repeatable Processes
Many Two-Pizza Teams
Scaling Phase: Scalable Architecture

- Technology that Scales
  - Common migrations to {Python, Go, JVM languages}
  - Concurrency
  - Asynchrony

- Independent systems
  - Fit-for-purpose systems: analytics, search
  - Separated services: payments, etc.
  - Layered services: caching, etc.

- Event queue
  - Use events to communicate between applications and services
Scaling Phase: Scalable Architecture

- **Scalable persistence**
  - Break up the monolithic database
  - Functional partitioning
  - Sharding
Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and independent
Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and independent
- Isolated persistence (!)

•
Microservice Architecture

Pros

• Each unit is simple
• Independent scaling and performance
• Independent testing and deployment
• “Optimal” technology stack
• Security boundary

Cons

• Multiple cooperating units
• Exchange in-process for network latencies
• More sophisticated deployment and monitoring tools
• Overall system complexity
Migrating to Services

- Incremental Migration
- Migrating a Monolithic Application
- Migrating a Monolithic Database
Migrating to Services

- Incremental Migration
- Migrating a Monolithic Application
- Migrating a Monolithic Database
“The only thing a Big Bang migration guarantees is a big *Bang*.”

-- Martin Fowler
Incremental Migration

• Step 0: Pilot Implementation
  o Choose initial end-to-end vertical experience to migrate / create
  o (+) Opportunity to learn and adjust
  o (+) Demonstrate feasibility and gain confidence
  o (+) Bound investment and risk
  o (+) Provide real customer value

• Initial step is the hardest
  o Learning how to do things in the new way
  o Building out basic supporting capabilities
Incremental Migration

• Steps 1-N: Incremental Migration
  o Prioritize business value -- highest ROI areas first
  o Focus on areas with greatest rate of change
  o (+) Maximize near-term payoff from investment
  o (+) Confront and solve hard problems sooner rather than later

• New feature development in parallel
  o Typically cannot pause all feature work in all areas to migrate 😊
  o Within a particular area, try to separate feature work from migration work in distinct steps
Incremental Migration

• Residual monolith may remain indefinitely
  o Lowest business value
  o Most stable and least changing
  o Can migrate – or not – opportunistically
Migrating to Services

- Incremental Migration
- Migrating a Monolithic Application
- Migrating a Monolithic Database
Carving up the Monolith

• Look for (or create) a “seam” in the monolith
  o This is often the hardest part (!)

• Wall it off behind an interface

• Write automated tests around the interface

• Replace implementation with an independent component

• ➔ Rinse and Repeat
Migrating to Services

- Incremental Migration
- Migrating a Monolithic Application
- Migrating a Monolithic Database
• Problem: Monolithic shared DB

- stitchfix.com
- Styling app
- Warehouse app
- Merch app
- CS app
- Logistics app
- Payments service
- Profile service

- Clients
- Shipments
- Items
- Styles, SKUs
- Warehouses
- etc.
Extracting Microservices

- Decouple applications / services from shared DB

- stitchfix.com
- Styling app
- Warehouse app
- Merch app
- CS app
- Logistics app
- Payments service
- Profile service

- Clients
- Shipments
- Items
- Styles, SKUs
- Warehouses
- etc.
Extracting Microservices

- Decouple applications / services from shared DB
Extracting Microservices

- Step 1: Create a service
Extracting Microservices

- Step 2: Applications use the service
Extracting Microservices

- Step 3: Move data to private database
Extracting Microservices

• Step 4: Rinse and Repeat
Extracting Microservices

- Step 4: Rinse and Repeat
Extracting Microservices

- Step 4: Rinse and Repeat

[Diagram showing relationships between Styling app, Warehouse app, client-service, item-service, style-service, core_client, core_item, core_sku]
Getting to rearchitect a system is a sign of success, not failure.
Service as System of Record

• Single System of Record
  o Every piece of data is owned by a single service
  o That service is the **canonical system of record** for that data

• Every other copy is a **read-only, non-authoritative cache**
Microservices and Events

• Events are a **first-class part** of a service interface

• A service interface includes
  o Synchronous request-response (REST, gRPC, etc)
  o Events the service produces
  o Events the service consumes
  o Bulk reads and writes (ETL)

• The interface includes **any mechanism for getting data in or out of the service (!)**
STARTING

SCALING

OPTIMIZING

Fewer Teams

2-5 year horizon

https://ittybiz.com/s-curve/
STARTING

SCALING

OPTIMIZING

https://ittybiz.com/s-curve/
Merci!

@randyshoup

linkedin.com/in/randyshoup

medium.com/@randyshoup