From the Monolith to Microservices

Lessons from Google and eBay

Randy Shoup
@randyshoup
linkedin.com/in/randyshoup

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 C++ → Java / Scala → microservices

Architecture Evolution

- The Monolith
- Ecosystem of Services
- Building and Operating a Service
- Service Anti-Patterns

Architecture Evolution

- The Monolith
- Ecosystem of Services
- Building and Operating a Service
- Service Anti-Patterns

The Monolithic Architecture

2-3 monolithic tiers

- {JS, iOS, Android}
- {PHP, Ruby, Python}
- {MySQL, Mongo}

The Monolithic Application

Pros

- Simple at first
- In-process latencies
- Single codebase, deploy unit
- Resource-efficient at small scale

Cons

- Coordination overhead as team grows
- Poor enforcement of modularity
- Poor scaling (vertical only)
- All-or-nothing deploy (downtime, failures)
- Long build times

The Monolithic Database

Pros

- Simple at first
- Join queries are easy
- Single schema, deployment
- Resource-efficient at small scale

Cons

- Coupling over time
- Poor scaling and redundancy (all-or-nothing, vertical only)
- Difficult to tune properly
- All-or-nothing schema management

"If you don't end up regretting your early technology decisions, you probably over-engineered"

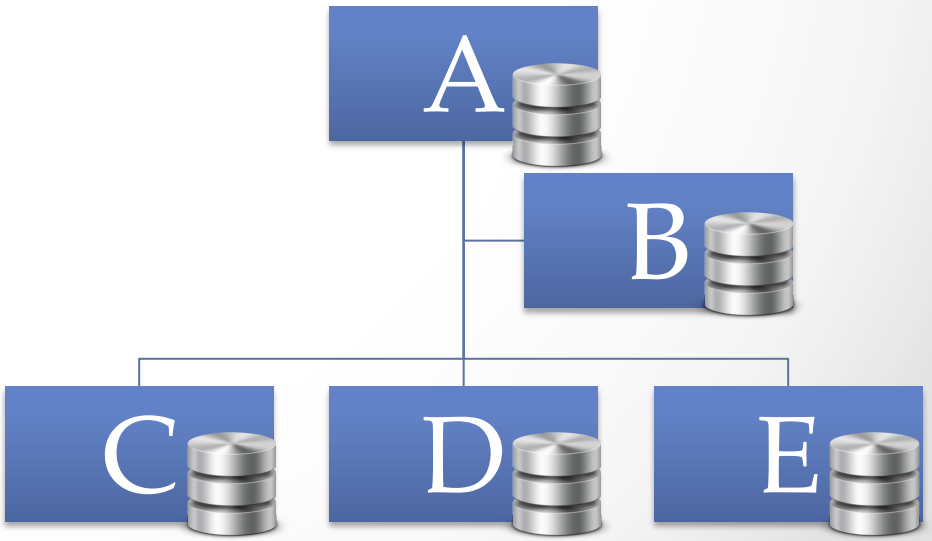
-- me

Architecture Evolution

- The Monolith
- Ecosystem of Services
- Building and Operating a Service
- Service Anti-Patterns

Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and independent
- Fullest expression of encapsulation and modularity
- Isolated persistence (!)



Microservices

Pros

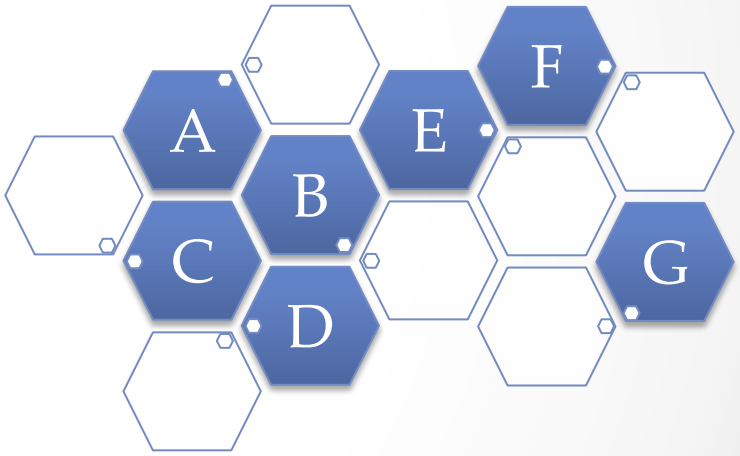
- Each unit is simple
- Independent scaling and performance
- Independent testing and deployment
- Can optimally tune performance (caching, replication, etc.)

Cons

- Many cooperating units
- Many small repos
- Requires more sophisticated tooling and dependency management
- Network latencies

Ecosystem of Services

- Hundreds to thousands of independent services
- Many layers of dependencies, no strict tiers
- Graph of relationships, not a hierarchy



Google Service Layering

- Cloud Datastore: NoSQL service
  - Highly scalable and resilient
  - Strong transactional consistency
  - SQL-like rich query capabilities
- Megastore: geo-scale structured database
  - Multi-row transactions
  - Synchronous cross-datacenter replication
- Bigtable: cluster-level structured storage
  - (row, column, timestamp) -> cell contents
- Colossus: next-generation clustered file system
  - Block distribution and replication
- Borg: cluster management infrastructure
  - Task scheduling, machine assignment

Evolution, not Intelligent Design

- No centralized, top-down design of the system
- Variation and Natural selection
  - Create / extract new services when needed to solve a problem
  - Deprecate services when no longer used
  - Services justify their existence through usage
- Appearance of clean layering is an emergent property

"Every service at Google is either deprecated or not ready yet."

-- Google engineering proverb

Architecture without an Architect?

- No "Architect" title / role
- (+) No central approval for technology decisions
  - Most technology decisions made locally instead of globally
  - Better decisions in the field
- (-) eBay Architecture Review Board
  - Central approval body for large-scale projects
  - Usually far too late in the process to be valuable
  - Experienced engineers saying "no" after the fact vs. encoding knowledge in a reusable library, tool, or service

Standardization

- Standardized communication
  - Network protocols
  - Data formats
  - Interface schema / specification
- Standardized infrastructure
  - Source control
  - Configuration management
  - Cluster management
  - Monitoring, alerting, diagnosing, etc.

Standards become standards by being better than the alternatives!

Service Independence

- No standardization of service internals
  - Programming languages
  - Frameworks
  - Persistence mechanisms

In a mature ecosystem of services, we standardize the arcs of the graph, not the nodes!

Architecture Evolution

- The Monolith
- Ecosystem of Services
- Building and Operating a Service
- Service Anti-Patterns

Goals of a Service Owner

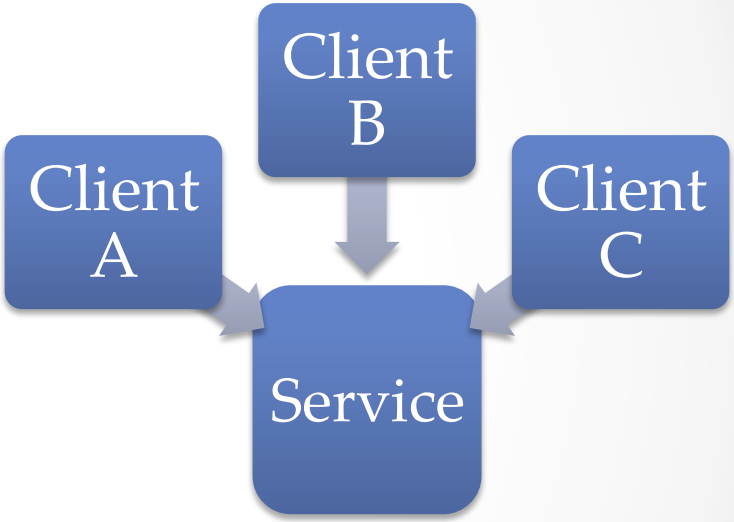
- Meet the needs of my clients …
  - Functionality
  - Quality
  - Performance
  - Stability and reliability
  - Constant improvement over time
- … at minimum cost and effort
  - Leverage common tools and infrastructure
  - Leverage other services
  - Automate building, deploying, and operating my service
  - Optimize for efficient use of resources

Responsibilities of a Service Owner

- End-to-end Ownership
  - Team owns service from design to deployment to retirement
  - No separate maintenance or sustaining engineering team
  - DevOps philosophy of "You build it, you run it"
- Autonomy and Accountability
  - Freedom to choose technology, methodology, working environment
  - Responsibility for the results of those choices

Service as Bounded Context

- Primary focus on my service
  - Clients which depend on my service
  - Services which my service depends on
  - Cognitive load is very bounded
- Very little worry about
  - The complete ecosystem
  - The underlying infrastructure
- ➔ Small, nimble service teams



Service-Service Relationships

- Vendor – Customer Relationship
  - Friendly and cooperative, but structured
  - Clear ownership and division of responsibility
  - Customer can choose to use service or not (!)
- Service-Level Agreement (SLA)
  - Promise of service levels by the provider
  - Customer needs to be able to rely on the service, like a utility

Service-Service Relationships

- Charging and Cost Allocation
  - Charge customers for *usage* of the service
  - Aligns economic incentives of customer and provider
  - Motivates both sides to optimize for efficiency
  - (+) Pre- / post-allocation at Google

Maintaining Interface Stability

- Backward / forward compatibility of interfaces
  - Can *never* break your clients' code
  - Often multiple interface versions
  - Sometimes multiple deployments
  - Majority of changes don't impact the interface in any way
- Explicit deprecation policy
  - Strong incentive to wean customers off old versions (!)

Architecture Evolution

- The Monolith
- Ecosystem of Services
- Building and Operating a Service
- Service Anti-Patterns

Service Anti-Patterns

- The "Mega-Service"
  - Overbroad area of responsibility is difficult to reason about, change
  - Leads to more upstream / downstream dependencies
- Shared persistence
  - Breaks encapsulation, encourages "backdoor" interface violations
  - Unhealthy and near-invisible coupling of services
  - (-) Initial eBay SOA efforts

Thank You!

- @randyshoup
- linkedin.com/in/randyshoup
- Slides will be at slideshare.net/randyshoup