Service Architectures at Scale
Lessons from Google and eBay

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

• Consulting CTO at Randy Shoup Consulting
  o Helping companies scale their organizations and technology
  o “CTO as a service”

• CTO at KIXEYE
  o Real-time strategy games for web and mobile

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

• Chief Engineer at eBay
  o Multiple generations of eBay’s infrastructure
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
Service Architectures at Scale

- Ecosystem of Services
- Designing a Service
- Building and Operating a Service
Service Architectures at Scale

• Ecosystem of Services
• Designing a Service
• Building and Operating a Service
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
  o Create / extract new services when needed to solve a problem
  o Services justify their continued existence through usage
  o Deprecate services when no longer used
“Every service at Google is either deprecated or not ready yet.”

-- Google engineering proverb
Architecture without an Architect?

- No “Architect” title / role

- Appearance of clean layering is an emergent property

- (+) No central approval for technology decisions
  - Most technology decisions made locally instead of globally

- (-) 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.
In a healthy service ecosystem, standards become standards by being better than the alternatives.
“Enforcing” Standardization

- Encouraged via
  - Libraries
  - Support in underlying services
  - Code reviews
  - Searchable code
The easiest way to “enforce” a standard practice is with working code.
Service Independence

• No standardization of service internals
  o Programming languages
  o Frameworks
  o Persistence mechanisms
In a mature ecosystem of services, we standardize the arcs of the graph, not the nodes.
Service Architectures at Scale

- Ecosystem of Services
- Designing a Service
- Building and Operating a Service
Characteristics of an Effective Service

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

• The “Mega-Service”
  o Overbroad area of responsibility is difficult to reason about, change
  o Leads to more upstream / downstream dependencies

• “Leaky Abstraction” Service
  o Interface reflects provider’s model of the interaction, not the consumer’s model
  o Consumer’s model is typically more aligned with the domain, simpler, more abstract
  o Leaking provider’s model in the interface constrains evolution of the implementation
Service Anti-Patterns

• Shared persistence
  o Breaks encapsulation, encourages “backdoor” interface violations
  o Unhealthy and near-invisible coupling of services
  o (-) Initial eBay SOA efforts
Service Persistence

• Option 1: Operate your own data store
  o Store to your own instance(s) of MySQL, etc., owned and operated by the service

• Option 2: Use a persistence service
  o Store to your own partition(s) of Dynamo, Bigtable, etc., operated as a service by another team

• ➔ Only external access to data store is through published service interface
Maintaining Interface Stability

• Backward / forward compatibility of interfaces
  o Can *never* break your clients’ code
  o Often multiple interface versions
  o Sometimes multiple deployments
  o Majority of changes don’t impact the interface in any way

• Explicit deprecation policy
  o Strong incentive to wean customers off old versions (!)
Service Architectures at Scale

- Ecosystem of Services
- Designing a Service
- Building and Operating a Service
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
  o Team owns service from design to deployment to retirement
  o No separate maintenance or sustaining engineering team
  o DevOps philosophy of “You build it, you run it”

• Autonomy and Accountability
  o Freedom to choose technology, methodology, working environment
  o 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
  o Friendly and cooperative, but structured
  o Clear ownership and division of responsibility

• Customer can choose to use service or not (!)
  o Must be strictly better than the alternatives of build, buy, borrow

• Service-Level Agreement (SLA)
  o Promise of service levels by the provider
  o Customer needs to be able to rely on the service, like a utility
Charging for Usage

• Charge customers for *usage* of the service
  o Aligns economic incentives of customer and provider
  o Motivates both sides to optimize efficiency

• Free usage leads to waste
  o No incentive to control usage or find more efficient alternatives

• E.g., App Engine usage at Google
  o Charging particularly egregious internal customer led to 10x reduction in usage
Maintaining Service Quality

- Small incremental changes
  - Easy to reason about and understand
  - Risk of code change is nonlinear in the size of the change
  - (-) Initial memcache service submission

- Solid Development Practices
  - Code reviews before submission
  - Automated tests for everything

- Google build and test system
  - Uses production cluster manager
  - Runs millions of tests per day in parallel
  - All acceptance tests run before code is accepted into source control
Service
Reliability

• Systems at scale highly exposed to failure
  o Software, hardware, service failures
  o Sharks and backhoes
  o Operator “oops”

• Resilience in depth
  o Redundancy for machine / cluster / data center failures
  o Load-balancing and flow control for service invocations
  o Rapid rollback for “oops”
Service Deployment

• Incremental Deployment
  o Canary systems
  o Staged rollouts
  o Rapid rollback

• eBay “Feature Flags”
  o Decouple code deployment from feature deployment
  o Rapidly turn on / off features without redeploying code
  o Typically deploy with feature turned off, then turn on as a separate step
Service Architectures at Scale

- Ecosystem of Services
- Designing a Service
- Building and Operating a Service