Concurrency at Scale:
The Evolution to Micro-Services

Randy Shoup
@randyshoup
linkedin.com/in/randyshoup

Background

- CTO at KIXEYE
  - Real-time strategy games for web and mobile
- Director of Engineering for Google App Engine
  - World's largest Platform-as-a-Service
  - Part of Google Cloud Platform
- Chief Engineer at eBay
  - Multiple generations of eBay's real-time search infrastructure

Evolution in Action

- 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++ → Perl / C++ → Java / Scala → microservices

Evolution to Micro-Services

- The Monolith
- Micro-Services
- Reactive Systems
- Migrating to Micro-Services

Evolution to Micro-Services

- The Monolith
- Micro-Services
- Reactive Systems
- Migrating to Micro-Services

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

Evolution to Micro-Services

- The Monolith
- Micro-Services
- Reactive Systems
- Migrating to Micro-Services

Micro-Services

"Loosely-coupled service oriented architecture with bounded contexts"

-- Adrian Cockcroft

Micro-Services

"Loosely-coupled service oriented architecture with bounded contexts"

-- Adrian Cockcroft

Micro-Services

"Loosely-coupled service oriented architecture with bounded contexts"

-- Adrian Cockcroft

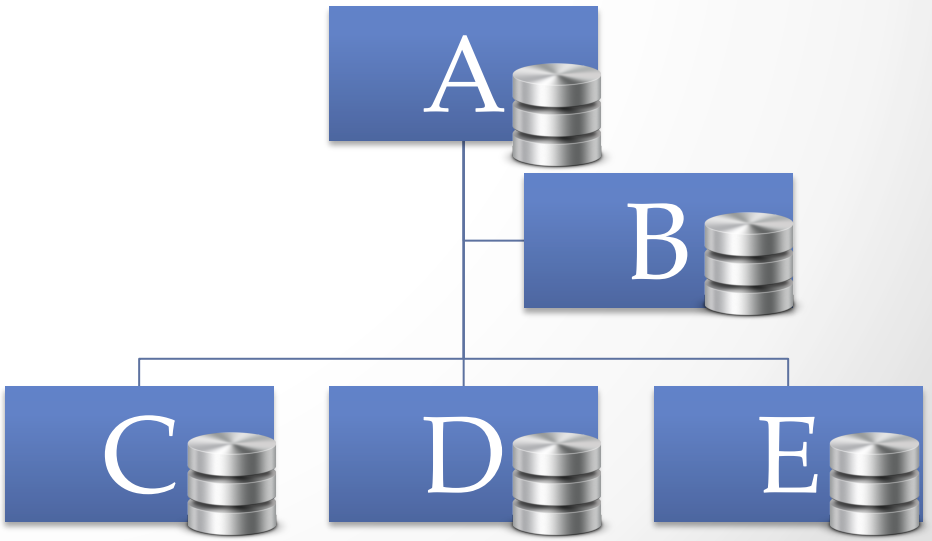
Micro-Services

"Loosely-coupled service oriented architecture with bounded contexts"

-- Adrian Cockcroft

Micro-Services

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



Micro-Services

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

Google Services

- All engineering groups organized into "services"
  - Gmail, App Engine, Bigtable, etc.
- Self-sufficient and autonomous
- Layered on one another

➔ Very small teams achieve great things

Google Cloud Datastore

- 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
- Cluster management infrastructure
  - Task scheduling, machine assignment

Evolution to Micro-Services

- The Monolith
- Micro-Services
- Reactive Systems
- Migrating to Micro-Services

Reactive Micro-Services

- Responsive
  - Predictable performance at 99%ile trumps low mean latency (!)
  - Tail latencies far more important than mean or median
  - Client protects itself with asynchronous, non-blocking calls
- Resilient
  - Redundancy for machine / cluster / data center failures
  - Load-balancing and flow control for service invocations
  - Client protects itself with standard failure management patterns: timeouts, retries, circuit breakers

Reactive Micro-Services

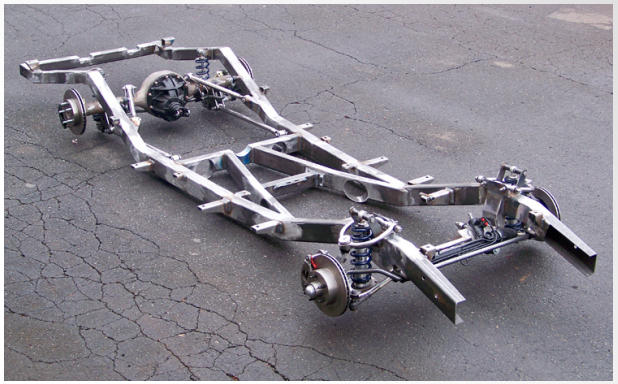
- Elastic
  - Scale up and down service instances according to load
  - Gracefully handle spiky workloads
  - Predictive and reactive scaling
- Message-Driven
  - Asynchronous message-passing over synchronous request-response
  - Often custom protocol over TCP / UDP or WebSockets over HTTP

KIXEYE Game Services

- Minimize request latency
  - Respond as rapidly as possible to client

- Functional Reactive + Actor model
  - Highly asynchronous, never block (!)
  - Queue events / messages for complex work
  - Heavy use of Scala / Akka and RxJava at KIXEYE
- Highly Scalable and Productive
  - (-) eBay uses threaded synchronous model
  - (-) Google uses complicated callback-based asynchronous model

KIXEYE Service Chassis

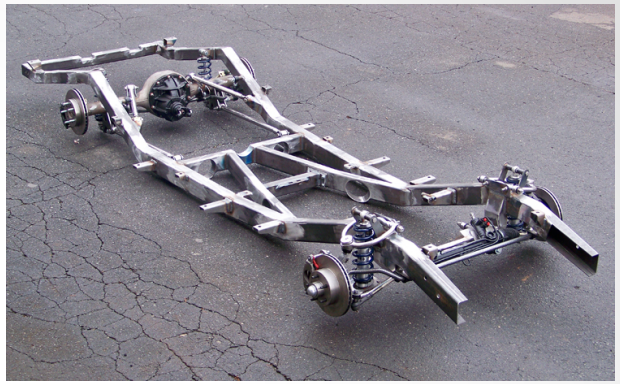


- Goal: Make it easy to build and deploy micro-services

- Chassis core
  - Configuration integration
  - Registration and Discovery
  - Monitoring and Metrics
  - Load-balancing for downstream services
  - Failure management for downstream services

- Development / Deployment Pipeline
  - Transport layer over REST / JSON or WebSockets
  - Service template in Maven
  - Build pipeline through Puppet -> Packer -> AMI
  - Red-black deployment via Asgard

KIXEYE Service Chassis



- Heavy use of NetflixOSS
  - Asgard
  - Hystrix
  - Ribbon + WebSockets ➔ Janus
  - Eureka

➔ Results
  - 15 minutes from no code to running service in AWS (!)
  - Open-sourced at https://github.com/Kixeye

Evolution to Micro-Services

- The Monolith
- Micro-Services
- Reactive Systems
- Migrating to Micro-Services

Migrating Incrementally

- Find your worst scaling bottleneck
- Wall it off behind an interface
- Replace it
- ➔ Rinse and Repeat

Building Micro-Services

- Common Chassis / Framework
  - Make it trivially easy to build and maintain a service
- Define Service Interface (Formally!)
  - Propose
  - Discuss with client(s)
  - Agree
- Prototype Implementation
  - Simplest thing that could possibly work
  - Client can integrate with prototype
  - Implementor can learn what works and what does not

Building Micro-Services

- Real Implementation
  - Throw away the prototype (!)
- ➔ Rinse and Repeat

"So you are really serious about this …"

- Distributed tracing
  - Trace a request chain through multiple service invocations
- Network visualization
  - "Weighted" communication paths between microservices / instances
  - Latency, error rates, connection failures
- Dashboard metrics
  - Quickly scan operational health of many services
  - Median, 99%ile, 99.9%ile, etc.
  - Netflix Hystrix / Turbine

Micro-Service Organization

- Small, focused teams
  - Single service or set of related services
  - Minimal, well-defined "interface"
- Autonomy
  - Freedom to choose technology, methodology, working environment
  - Responsibility for the results of those choices
- Accountability
  - Give a team a goal, not a solution
  - Let team own the best way to achieve the goal

Micro-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
- Charging and Cost Allocation
  - Charge customers for *usage* of the service
  - Aligns economic incentives of customer and provider
  - Motivates both sides to optimize

Recap: Evolution to Micro-Services

- The Monolith
- Micro-Services
- Reactive Systems
- Migrating to Micro-Services

Thank You!

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