Being Elastic
Evolving Programming for the Cloud

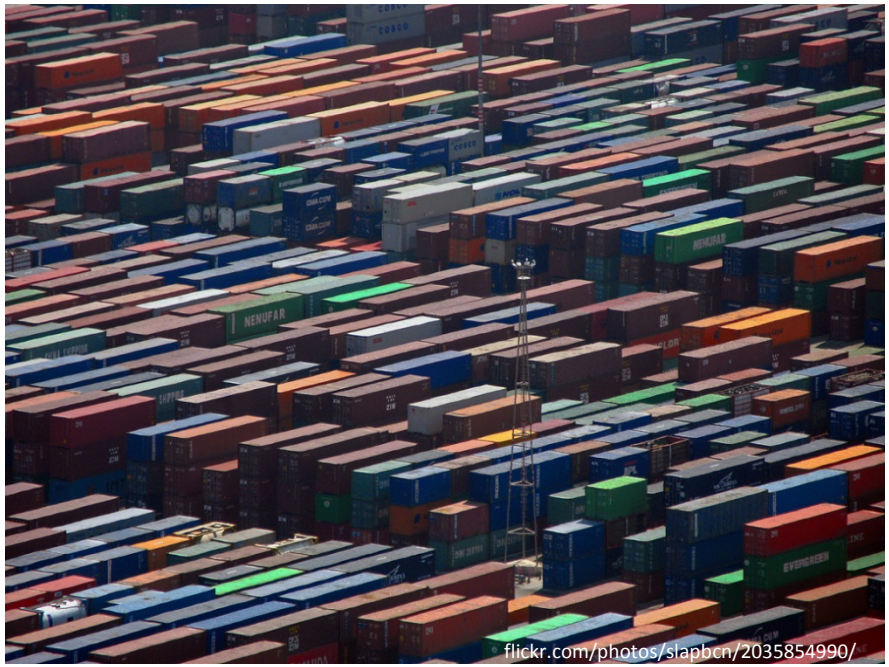
Randy Shoup
eBay Chief Engineer

QCon San Francisco
November 4, 2010

Cloud is a New Ecosystem

- Resource-Rich
- Inexpensive
- Available
- Elastic
- Managed
- Remote
- Virtual
- Variable
- Ephemeral
- Metered

The Old and the New



Developers Must Adapt

- Benefits are huge
- Constraints are real

- Adapt to the constraints to get the benefits
- Most adaptations encourage otherwise good development practices (!)

Scaling for the Cloud

- To leverage scalable infrastructure, you need
  - Scalable application
  - Scalable development practices
  - Scalable culture
- Principles and practices discussed widely (particularly at QCon!)
  - Adapting to the Cloud involves a convergence of Architecture, Agile, DevOps, etc.

Universal Scalability Law

$$C(N) = \frac{N}{1 + \alpha(N-1) + \beta N(N-1)}$$

- Formulated by Neil Gunther

  http://www.perfdynamics.com/Manifesto/USLscalability.html

- Throughput limited by two factors
  - **Contention (α)**: bottleneck on shared resource
  - **Coherency (β)**: communication and coordination among multiple nodes

Programming in the Cloud

- Parallelism
- Layering
- Services
- State management
- Data model
- Failure handling
- Testing

Parallelism

- Think parallel!
  - Simple parallel algorithm out-scales "smarter" non-parallel algorithm
  - Exploiting distributed, elastic resources requires parallelism
- Request processing
  - Parallelism through
    - Routing requests
    - Aggregating services
    - Queueing work
  - Async I/O and Futures are your friends

Parallelism

- Offline computation
  - Parallelism through
    - Workload partitioning
    - Partitioning data and processing
  - E.g., pipelines, MapReduce

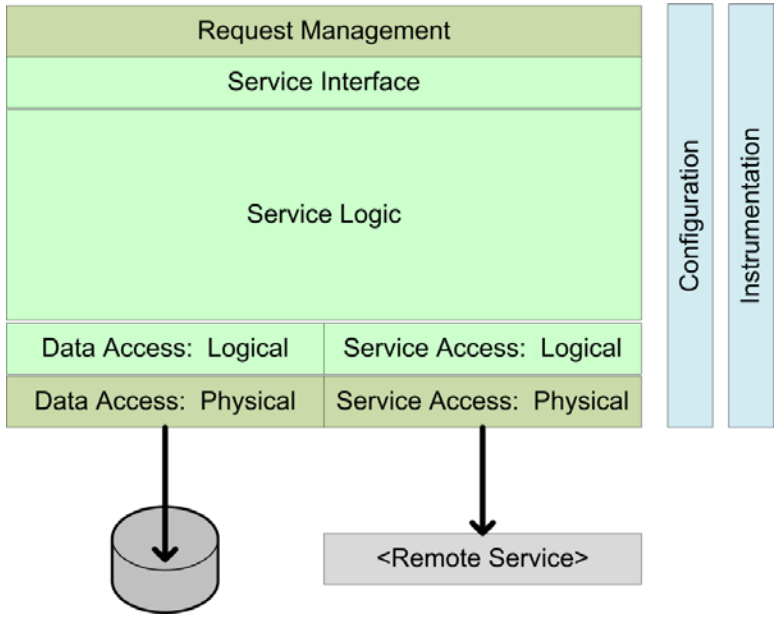
Parallelism

“There are 3 rules to follow when parallelizing large codes. Unfortunately, no one knows what these rules are.”

– W. Somerset Maugham and Gary Montry

Layering

- Strictly layered system
  - Software layers inside components
  - Between system components
  - Cloud makes clean layering particularly important
- Common code in frameworks
  - Configuration
  - Instrumentation
  - Failure management



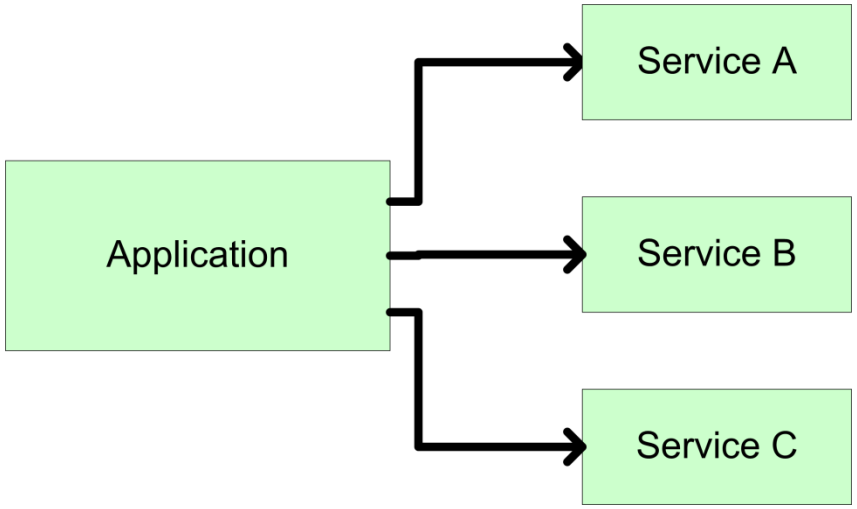
Layering

"All problems in computer science can be solved by another level of indirection … Except for the problem of too many layers of indirection."

– David Wheeler

Services

- Decompose system functionality into services
  - Simple
  - Single-purpose
  - Modular
  - Stateless
  - Multi-instance
- Compose complex application behavior from simple elements



Services

The service should be the fundamental unit of …

- Composition
  - Combine simple services into complex systems
- Dependency
  - Depend on a service interface, not implementation
- Addressing
  - Talk to logical endpoint (URI), not IP:port
- Persistence
  - Abstract and isolate persistence behind a service
- Deployment

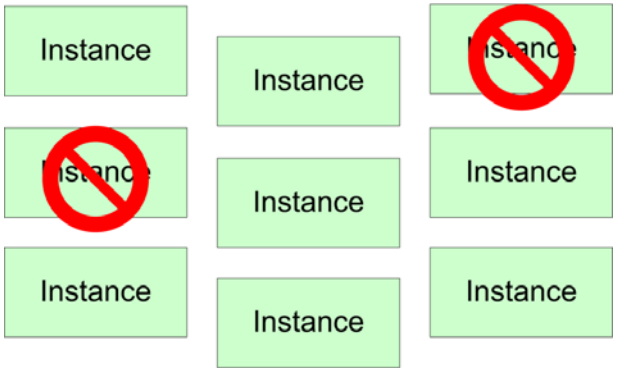
Services

“Make everything as simple as possible, but not simpler.”

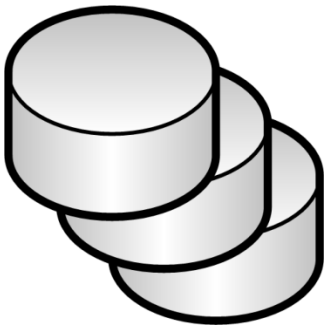
– Albert Einstein

State Management

- Stateless instances
  - Instances are ephemeral
  - Local memory / storage is fast, but transient and inconsistent
  - Equivalent to a cache



- Durable state in persistent storage
  - (Many implementations)



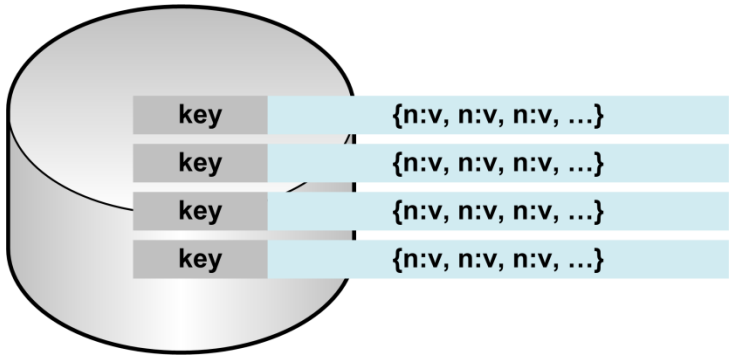
State Management

“Here today, gone tomorrow.”

– American proverb

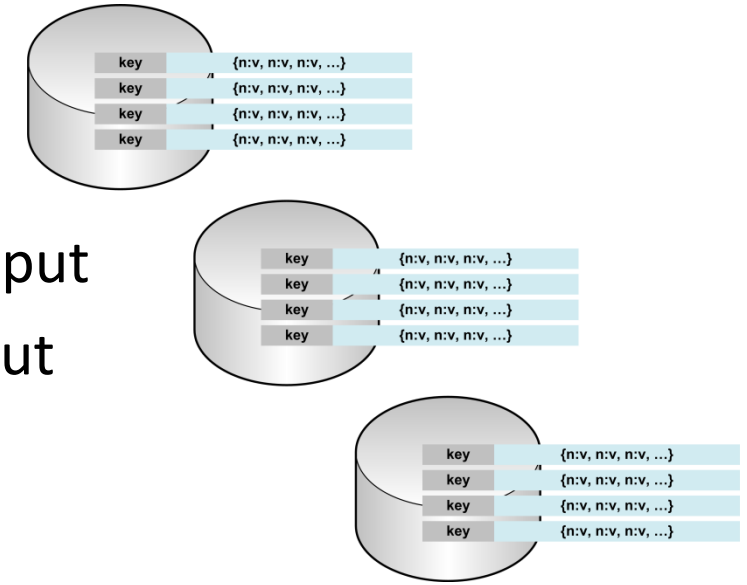
Key-Value Data Model

- Distributed key-value stores
  - Simple
  - Horizontally scalable
  - (Many implementations)
- Constrained by design
  - Cannot express complex relationships
  - Limited or no fixed schema
  - Predictable, bounded performance
- Greater burden on application
  - Joins, aggregations, sorts, etc.
  - Integrity and schema validation

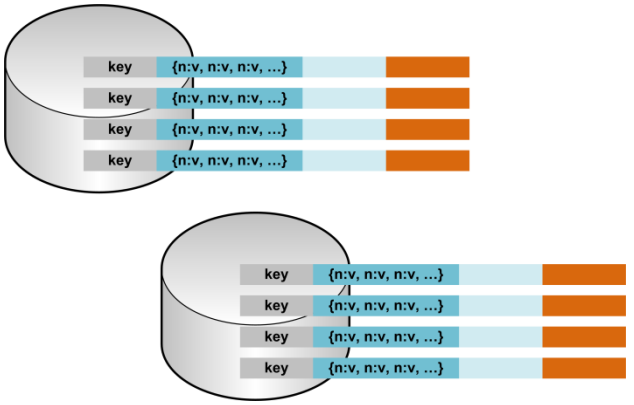


Key-Value Data Model

- Plan to Shard
  - Partition by key
  - Parallelize writes for write throughput
  - Parallelize reads for read throughput



- Plan to Denormalize
  - Optimize for reads
  - Precalculate joins, aggregations, sorts
  - Use async queue to update
  - Learn to tolerate transient inconsistency



Failure Handling

- Expect failures and handle them
  - `e.printStackTrace()` does not count!
- Failure handling means
  - Graceful degradation
  - Timeouts and retries
  - Throttling and back-off
- Abstract through frameworks and policies

flickr.com/photos/davidwatts1978/3199405401/

Failure Handling

“Hope is not a strategy.”

– Various

Testing

- Test early and often
  - Test-driven and Test-first approaches work particularly well in the cloud
  - Automated testing is essential
  - Incremental development and deployment
- Test end-to-end
  - Much easier to test at load
  - More challenging to simulate all combinations and failure modes (!)
  - Mocking and fault injection

Testing

“In the data center, robust code is best practice. In the cloud, it is essential.”

– Adrian Cockcroft

Operating in the Cloud

- DevOps Mindset
- Configuration Injection
- Instrumentation
- Monitoring
- Metering

DevOps Mindset

- In the Cloud, Developers ⇔ Ops
  - Everyone will use tools for deployment, monitoring, etc.
  - Everyone will share responsibility for system
- Manageability as an investment
  - Management interfaces need as much engineering discipline as functional interfaces
- Automate, automate, automate
  - Repeatable builds and deployments
  - Provisioning, scaling, failure management, etc.
  - If you do it more than twice, script it

Configuration Injection

- Boot instance from bare minimum package
  - Externalize all variability
  - Separate code and configuration
- Configuration should drive
  - Instance role
  - Service and resource dependencies
  - Behavior / capabilities
- Inject
  - At boot time
  - At runtime

flickr.com/photos/andresrueda/2983149263/

Configuration Injection

“I do my work at the same time each day – the last minute.”

– Unknown

Instrumentation

- Fully instrument all components
  - Cannot attach debugger / profiler
  - Remotely diagnose, profile, debug
  - Avoid “monitoring fatigue”
- Logging is insufficient
  - Needs to be automatically interpretable and actionable
- Use a framework
  - Make part of your infrastructure

Monitoring

- Monitor requests end-to-end
  - Trace all requests as they flow from component to component
- Monitor component activity and performance
  - Metrics, metrics, metrics
- Understand your typical system behavior
  - Spiky, flat, sinusoidal?
  - Know the baseline so you know what is abnormal

Metering

- Fixed cost -> variable cost
  - Processing
  - Storage
  - Network
- Efficiency matters
  - Any savings goes directly to the bottom line
  - Any inefficiency is multiplied by many instances (!)

Metering

*"*We *should* forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%.*"*

– *Donald Knuth*

Evolving for the Cloud

“It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change.”

– Charles Darwin