Java is a trademark of Sun Microsystems, Inc.

JavaOne℠

Best Practices for Large-Scale Websites: Lessons from eBay

Randy Shoup
eBay

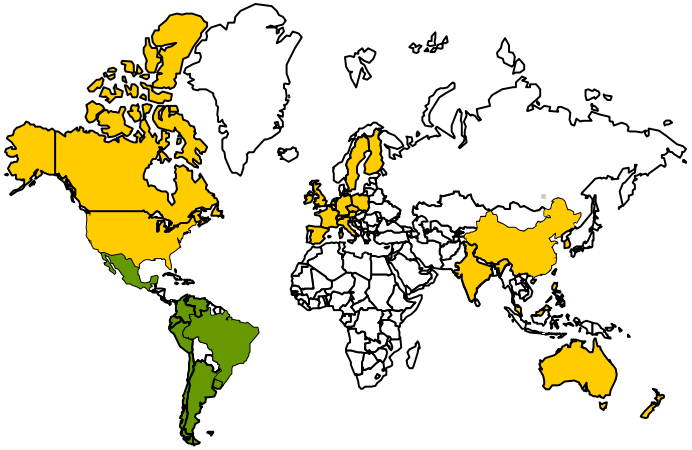
Challenges at Internet Scale

- eBay manages …
  - 88.3 million active users worldwide
  - 160 million items for sale in 50,000 categories
  - Over 2 billion page views per day

- … in a dynamic environment
  - >300 new features per quarter
  - 100,000 lines of code changed every 2 weeks

Challenges at Internet Scale

- … worldwide
  - In 39 countries and 8 languages
  - 24x7x365



- **>48 billion SQL executions / day**

Architectural Forces at Internet Scale

- Scalability
  - Resource usage should increase linearly (or better!) with load
  - Design for 10x growth in data, traffic, users, etc.
- Availability
  - Resilience to failure
  - Graceful degradation and rapid recoverability
- Latency
  - User experience latency
  - Data / execution latency

Architectural Forces at Internet Scale

- Manageability
  - Simplicity and maintainability
  - Diagnostics
- Cost
  - Development effort and complexity
  - Operational cost (TCO)

Best Practices for Internet Scale

- 1. Partition Everything
- 2. Asynchrony Everywhere
- 3. Automate Everything
- 4. Remember Everything Fails
- 5. Embrace Inconsistency

Best Practice 1: Partition Everything

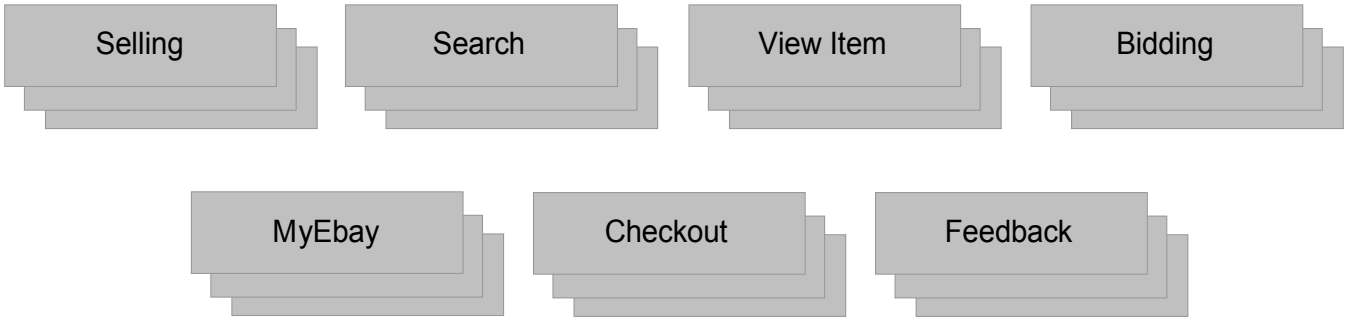
- Split every problem into manageable chunks
  - Split by data, load, and/or usage pattern
  - *"If you can't split it, you can't scale it"*

- Motivations
  - Scalability: scale horizontally and independently
  - Availability: isolate failures
  - Manageability: decouple different segments and functional areas
  - Cost: choose partition size that maximizes price-performance

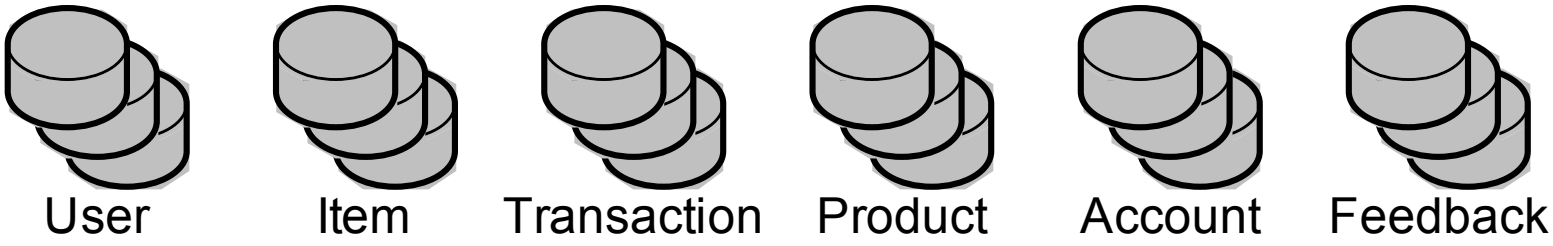
Best Practice 1: Partition Everything

Pattern: Functional Segmentation

- Segment processing into pools, services, and stages



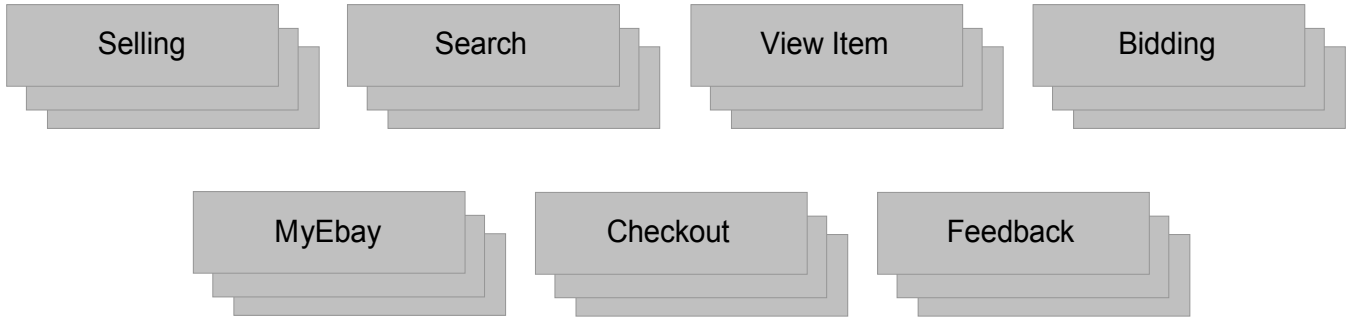
- Segment data by entity and usage pattern



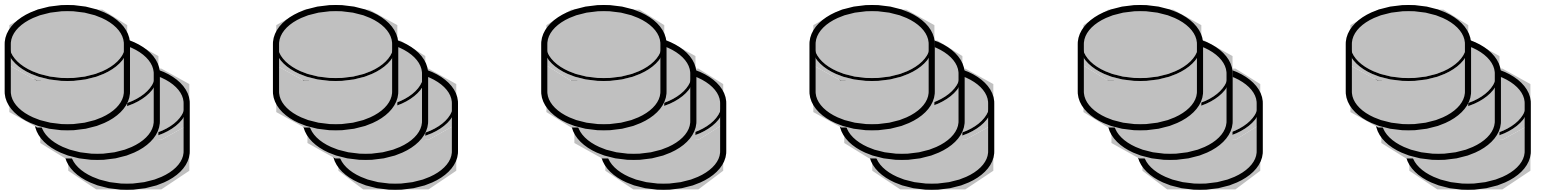
Best Practice 1: Partition Everything

Pattern: Horizontal Split

- Load-balance processing
  - All servers in a pool are created equal



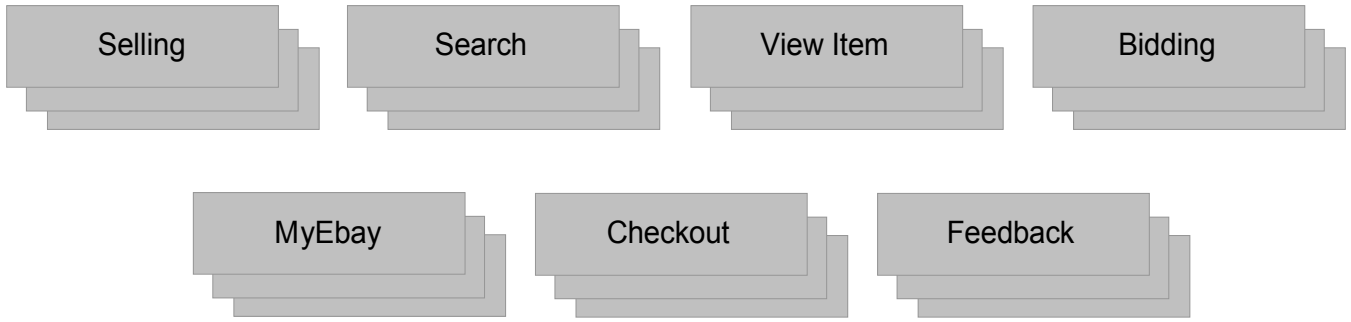
- Split (or "*shard*") data along primary access path
  - Modulo of a key, range, lookup table, etc.



Best Practice 1: Partition Everything

Corollary: No Session State

- User session flows through multiple pools



- Absolutely no session state or business object caching in application tier
  - No *HttpSession*, no EJBs
- Session state maintained in cookie, URL, or database

Best Practice 2: Async Everywhere

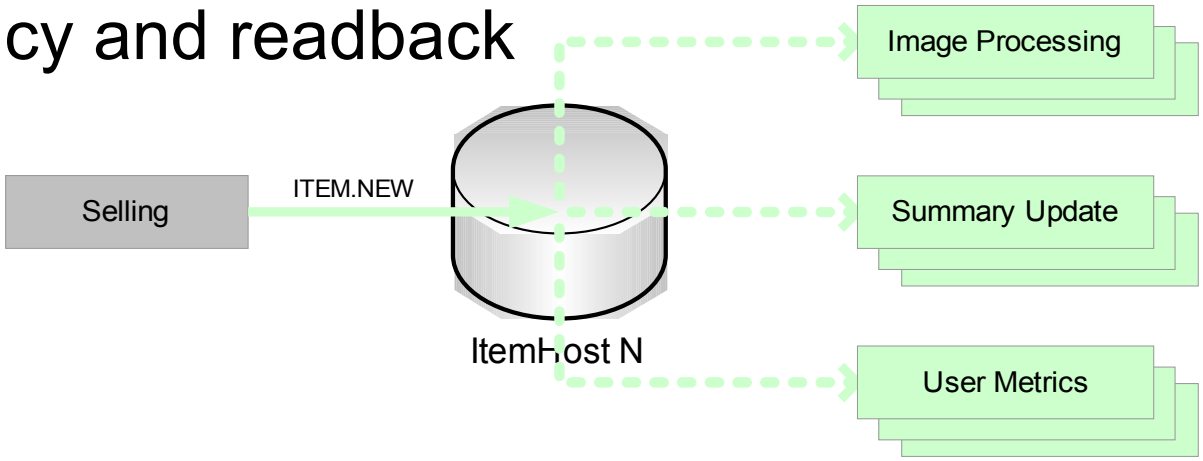
- Prefer Asynchronous Processing
  - Move most processing to asynchronous flows
  - Integrate components asynchronously

- Motivations
  - Scalability: scale components independently
  - Availability: decouple availability state, retry
  - Latency: trade execution latency for response latency
  - Cost: spread peak load over time

Best Practice 2: Async Everywhere

Pattern: Event Queue

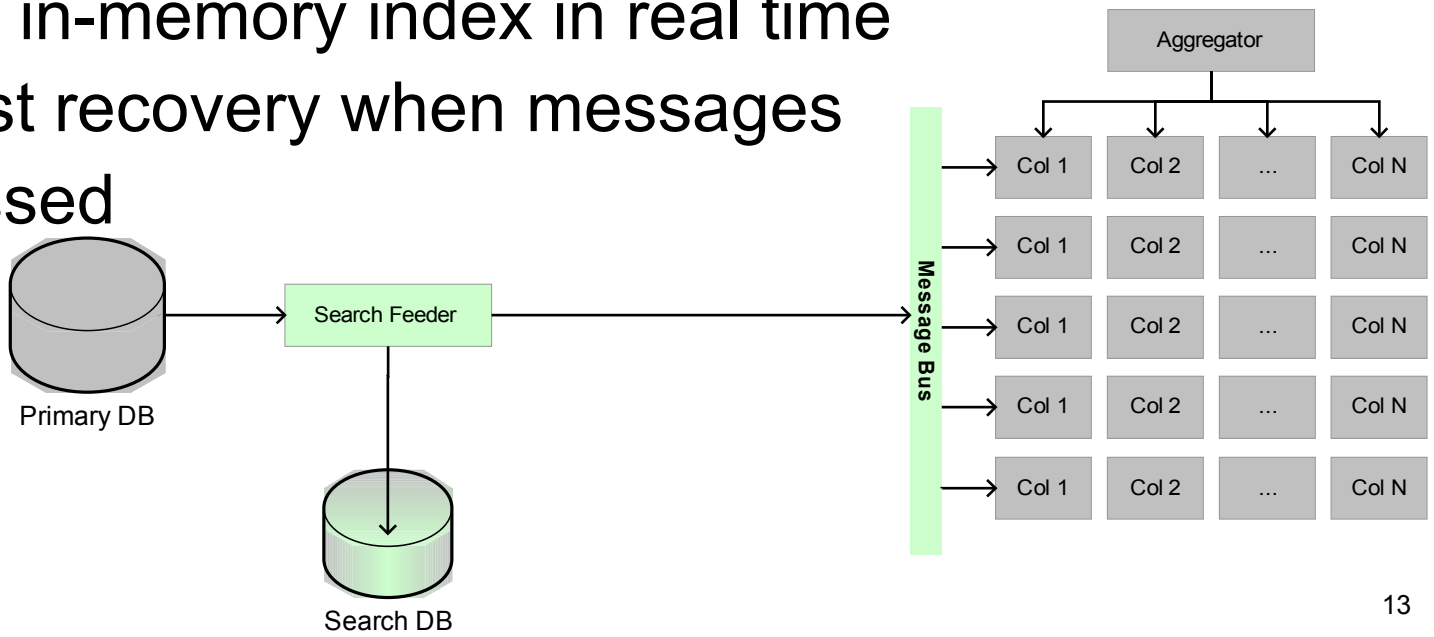
- Primary application writes data and queues event
  - Create event transactionally with primary insert / update (e.g., *ITEM.NEW, ITEM.BID, ITEM.SOLD*)
- Consumers subscribe to event
  - At least once delivery, rather than exactly once
  - No guaranteed order, rather than in-order
  - Idempotency and readback



Best Practice 2: Async Everywhere

Pattern: Message Multicast

- Search Feeder publishes item updates
  - Reads item updates from primary database
  - Publishes sequenced updates via multicast to grid
- Search engines listen to assigned messages
  - Update in-memory index in real time
  - Request recovery when messages are missed



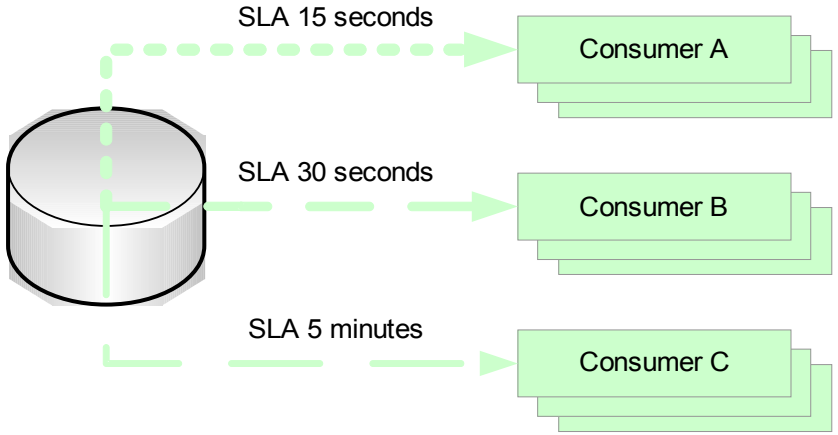
Best Practice 3: Automate Everything

- Prefer Adaptive / Automated Systems
  - Design systems which adapt to their environment
  - Engineer feedback loops to learn over time
- Motivations
  - Scalability: scale with machines, not humans
  - Availability / Latency: adapt to changing environment more rapidly
  - Cost: machines are less expensive than humans

Best Practice 3: Automate Everything

Pattern: Adaptive Configuration

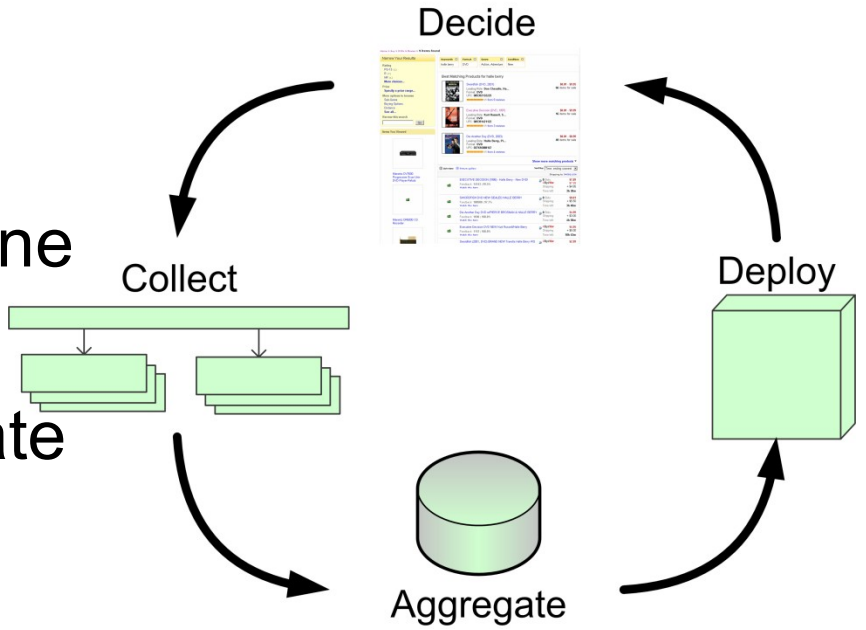
- Do not manually configure event consumers
  - Polling size and frequency, number of threads, etc.
- Define SLA for a given consumer
  - E.g., process 99% of events within 15 seconds
  - Each consumer dynamically adjusts to meet SLA
- Consumers automatically adapt to changes in
  - Load
  - Event processing time
  - Number of consumers



Best Practice 3: Automate Everything

Pattern: Machine Learning

- Dynamically adapt search experience
  - Determine best inventory and assemble optimal page for that user and context
- Feedback loop enables system to learn and improve over time
  - Collect user behavior
  - Aggregate and analyze offline
  - Deploy updated metadata
  - Decide and serve appropriate experience



Best Practice 4: Everything Fails

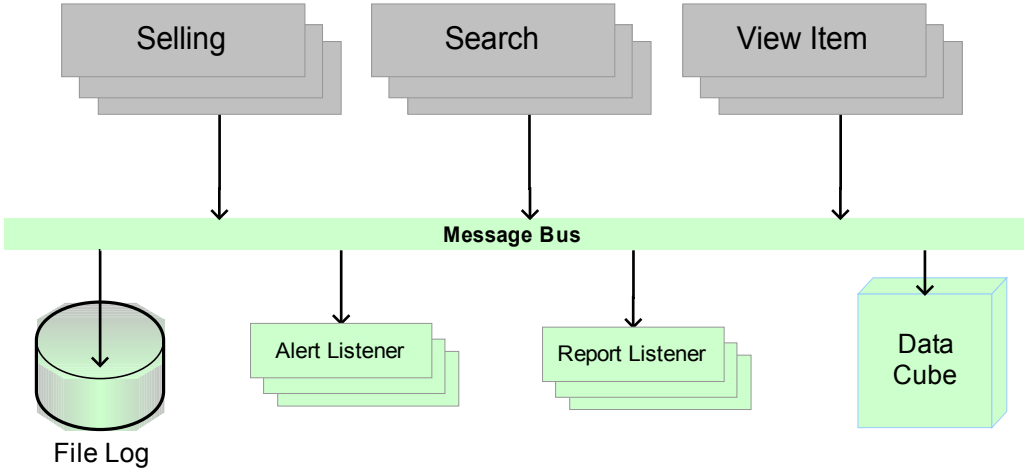
- Build all systems to be tolerant of failure
  - Assume every operation will fail and every resource will be unavailable
  - Detect failure as rapidly as possible
  - Recover from failure as rapidly as possible
  - Do as much as possible during failure

- Motivation
  - Availability

Best Practice 4: Everything Fails

Pattern: Failure Detection

- Application servers log all requests
  - Log all application activity, database and service calls on multicast message bus



  - ***Over 2TB of log messages per day***
- Listeners automate failure detection and notification

Best Practice 4: Everything Fails

Pattern: Rollback

- *Absolutely no changes to the site which cannot be undone (!)*

- Every feature has on / off state
  - Can immediately turn feature off for operational or business reasons
  - Can deploy “wired-off” to unroll dependencies
    - Decouple code deployment from feature deployment
  - For an application, feature “availability” is just like resource availability

Best Practice 4: Everything Fails

Pattern: Graceful Degradation

- Application “marks down” a resource if it is unavailable or distressed
- Remove or ignore non-critical functionality
- Retry or defer critical functionality
  - Failover to alternate resource
  - Defer processing to guaranteed async event
- Explicit “markup”
  - Restore and bring resource online in a controlled way

Best Practice 5: Embrace Inconsistency

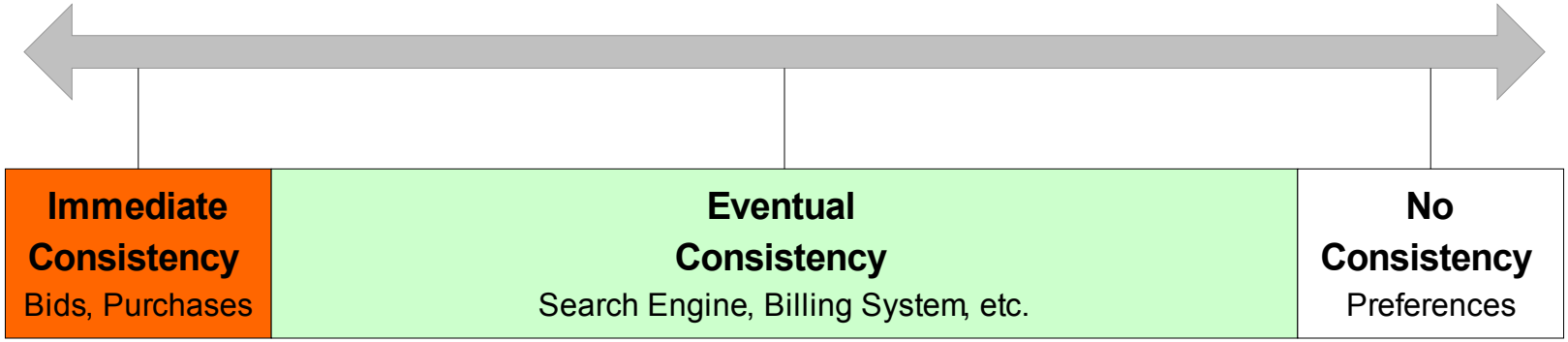
- Brewer's CAP Theorem
  - Any shared-data system can have at most two of the following properties:
    - ***C**onsistency: All clients see the same data, even in the presence of updates*
    - ***A**vailability: All clients will get a response, even in the presence of failures*
    - ***P**artition-tolerance: The system properties hold even when the network is partitioned*
- This trade-off is fundamental to all distributed systems

Best Practice 5: Embrace Inconsistency

Choose Appropriate Consistency Guarantees

- Typically eBay trades off immediate consistency for availability and partition-tolerance
- Most real-world systems do not require immediate consistency (even financial systems!)

- Consistency is a spectrum



Best Practice 5: Embrace Inconsistency

Avoid Distributed Transactions

- eBay does absolutely no distributed transactions
  - No JDBC client transactions, no 2PC, etc.
- Minimize inconsistency through state machines and careful ordering of database operations
- Reach eventual consistency through asynchronous event or reconciliation batch

Recap: Best Practices for Internet Scale

- 1. Partition Everything
- 2. Asynchrony Everywhere
- 3. Automate Everything
- 4. Remember Everything Fails
- 5. Embrace Inconsistency

Thank You

Randy Shoup
rshoup@ebay.com