The Start-Up Dilemma

1. You are releasing Online Storefront V 1.0
2. It could be a complete bust
3. But it could be *really* big

How do you implement data management at hacking velocity without compromising the future?
The Answer: Data Fabric Design

Data Fabric

App Tier

App Server

App Server

App Server

Data Tier

High Availability
Large Data Volumes
Run Anywhere
Cross-Site Distribution
Polyglot DBMS
Technology Upgrades
Use App Design Principles for Data

DBMS Server → Data Service

Client Interface(s)
Responsibilities
Capabilities
Compose Services via Design Patterns

- Connector
- Connector
- Connector

- Fabric Connector
- Transactional Data Service
- Fault-Tolerant Data Service
- Sharded Data Service

- Bridge
- Real-Time Data Bridge
- Multi-Site Data Service
1. Transactional Data Service

- **Responsibility:** Reliable and accessible data storage
- **Capabilities:**
  - Well-defined client interface
  - Transactions and recovery
  - Backup and restore
  - Log for replication
MySQL: A Mature Building Block

Relational modeling

Single join type

Sales Table

<table>
<thead>
<tr>
<th>id</th>
<th>cust_id</th>
<th>prod_id</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>335301</td>
<td>532</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>2378</td>
<td>6235</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Product Table

<table>
<thead>
<tr>
<th>id</th>
<th>sku</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>532</td>
<td>C00135</td>
<td>consumer</td>
</tr>
<tr>
<td>533</td>
<td>S09957</td>
<td>specialty</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Prod_ID Index

<table>
<thead>
<tr>
<th>prod_id</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>532</td>
<td>1</td>
</tr>
<tr>
<td>6235</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Fast row operations, slow scans of columns

Indexes speed query, slow updates

Transaction support

Single-threaded query

Relational modeling

Single join type

MySQL: A Mature Building Block
MySQL Implementation

SQL Interface (MySQL Dialect)

MySQL Wire Protocol

MySQL 5.6/InnoDB

Percona XtraBackup

MySQL Binlog

Backup Images
Comments and Trade-Offs

- Interface need not be SQL
- Backups are critical to restore broken replicas
- Pay close attention to DBMS configuration
- Design Key: Use ACID transactions (No MyISAM!)
2. Fault-Tolerant Data Service

- Responsibility: **Continuously available data using shared-nothing replicas**
- Capabilities:
  - Real-time replication between replicas
  - Provision/reprovision from backups
  - Failover and rolling maintenance
Replication Technology Overview

- Synchronous vs. asynchronous
- Master/slave vs. multi-master
- Logical vs. binary
- Trigger vs. log-based
Implementation using MySQL & MHA

- Master
- Oracle MySQL 5.6
- Async Native Replication
- Slave
- Oracle MySQL 5.6
- Slave
- Oracle MySQL 5.6
- MHA Manager Daemon
  (Performs failover and switch)
Comments and Trade-Offs

- **Design key:** Pick the right replication method
- Capabilities vary greatly
- Failover is hard to implement from scratch
- **Design key:** Address rolling maintenance
3. Fabric Connector

- **Responsibility:** Connect applications transparently to replicas

- **Capabilities:**
  - Encapsulation of DBMS server location
  - Transparent request proxying
  - Notification protocol for failover/switch operations
  - Load balancing for performance
Lots of Places to Intercept Requests

- Application Logic
- Access Library
- DBMS driver

- Client Wire Protocol
- Virtual IP Address

- Client proxy
- Level 7 router
- DNS name switching

- Access Library Wrapper
- Proxy Driver
- Proxy DBMS server (to other DBMS processes)
Quorum and Split Brain

App Server

Master/Online  Slave/Offline

Slave/Online
Library Wrapper + ZooKeeper

- Application Logic
- Access Library Wrapper
- Java ORM
- MySQL JDBC Driver

ZooKeeper Cluster
- zk1
- zk2
- zk3

Fault Tolerant Data Service

DBMS locations stored in ZK directory; failover through ZK event notifications
Comments and Trade-Offs

- VIPs and DNS renaming are easy but treacherous
- Wire protocol translation is hard
- Wrapper libraries are a good Goldilocks solution
- Design key: balance between availability and split brain
4. Sharded Data Service

- **Responsibility:** Partition a large dataset into manageable subsets

- **Capabilities:**
  - “Buckets” to contain groups of shards
  - Partitioning function to locate shards
  - Provisioning and shard migration
  - Intra- and cross-shard query
Easy vs. Hard Sharding Problems

Independent, small subsets

Very large subsets with dependent data
Sharding Is Easy when It’s Easy

Application Logic
Access Library Wrapper
Java ORM
MySQL JDBC Driver

ZooKeeper Cluster
zk1
zk2
zk3

DBMS locations stored in ZK directory; failover through ZK event notifications

Sharded Data Service
Comments and Trade-Offs

• Design key: factor data model to enable sharding
• Sharding scheme must be simple to work
• Moving shards is difficult
• Push cross-shard queries into data warehouse
• Some problems really need Cassandra or HBase, not a SQL DBMS
5. Multi-Site Data Service

- **Responsibility:** *Spread data across multiple geographic locations*

- **Capabilities:**
  - Fast cross-site replication
  - Primary/DR topology for business continuity
  - Master/master topology to push writes closer to users
Multi-Master vs. Primary/DR

- **Write-anywhere model**
  - Applications w/ few conflicts
  - Simpler failover model
  - Problems with constraints, large transactions

- **No writes on DR sites**
  - Any application
  - Easier to set up and manage
  - Does not affect SQL features
Tungsten Primary/DR Service

App Server + Connector

San Jose

Master

App Server + Connector

NYC

Slave
Comments and Trade-Offs

- Primary/DR is robust, easy to add later
- Multi-master is a better model if you can make it work
- Design key: Keep data model simple to enable multi-master
- SQL DBMS do not support multi-master as well as NoSQL (cf. Riak, Cassandra, HBase,...)
6. Real-Time Data Bridge

- **Responsibility:** Transfer data in real-time between different DBMS types

- **Capabilities:**
  - Heterogeneous replication
  - [Near] Real-time
  - High performance without app changes
  - Efficient conversion between storage models
Why Polyglot Persistence?

Online Storefront Application

User Session Service
- Key-value store with serialized user sessions
  - riak

Web Sales Service
- Row store with strong transactional support
  - MySQL

Sales Analytics Service
- Column store with compression + time series analysis
  - Vertica
Implementing Real-Time Data Bridge

- Serial extraction from MySQL binlog
- Parallel loading via CSV files into Vertica
- Tungsten Master Replication Services
- Tungsten Slave Replication Services
- Sharded Data Service
Comments and Trade-Offs

- Heterogeneous replication is time-consuming to deploy
- Can apply the pattern after the fact
- Avoid transformation as much as possible
- Design key: Balance complexity of data movement vs. getting data into the right silo for processing
Data Fabric Design Pattern Recap

- Fabric Connector
- Transactional Data Service
- Fault-Tolerant Data Service
- Sharded Data Service

Bridge

Real-Time Data Bridge

Multi-Site Data Service
Conclusion

- Use application architecture principles to create data services
- Use design patterns to combine services into data fabrics
- Apply patterns incrementally
- Early focus on design keys pays off big in the long run