From CRUD to Event Sourcing
an Investible Stock Universe
Marc Siegel
Team Lead (@ms_yd)

Brian Roberts
Senior Developer / Team Lead (@flicken)
What's in it for you?

● Answer the Unanswerable
  ○ both now and future
Vocabulary

1. Domain-Driven Design (DDD)
   - Event
   - Command
   - Aggregate

2. Event Sourcing (ES)
   - Projection
   - Read Model
How does Event Sourcing Work?
<table>
<thead>
<tr>
<th>Datum</th>
<th>Konto</th>
<th>Debit</th>
<th>Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1876</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>January</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>February</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>March</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>April</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>May</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>June</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>July</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>August</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>September</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>October</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>November</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>December</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1877</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>January</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>February</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>March</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>April</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>May</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>June</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>July</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>August</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>September</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>October</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>November</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>December</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
How does Event Sourcing Work?

(Quotes from Greg Young)

"State transitions are an important part of our problem space and should be modeled within our domain"
How does Event Sourcing Work?

(Quotes from Greg Young)

"State transitions are an important part of our problem space and should be modeled within our domain"

"Event Sourcing says all state is transient and you only store facts."
Vocabulary

1. Domain-Driven Design (DDD)
   - Event: something that happened in the past; a fact; a state transition
   - Command
   - Aggregate

2. Event Sourcing (ES)
   - Projection
   - Read Model
Q: How to represent the meaningful state transitions of the domain as events?
Listing Lifecycle Events

Events:

- TickerListed
ticker: MSFT
date: 1986
- TickerListed
ticker: IPET
date: 2000
- TickerListed
ticker: IPET
date: 2000
- TickerListed
ticker: FB
date: 2012

MSFT (Microsoft)
IPET (Pets.com)
FB (Facebook)
Q: How do we determine the state as of a given year? Or as of today?
Q: How do we determine the state as of a given year? Or as of today?

A: "When we talk about Event Sourcing, current state is a left-fold of previous behaviors"
-- Greg Young
Current State is a Left Fold of Events

- FP: Left Fold aggregates a collection via a function and an initial value
  - Ruby: \([1, 2, 3].\text{inject}(\emptyset, :+)\) == 6 # symbol fn name
  - Scala: \(\text{List}(1, 2, 3).\text{foldLeft}(\emptyset)(\_ + \_\_)\) == 6 // anon function

- Provide an initial state \(s_0\) and a function \(f : (S, E) \rightarrow S\)

- Current State after event \(e_3\) is:
  - \(= \text{leftFold}\( [e_1, e_2, e_3], s_0, f \)\)
  - \(= f(f(f(s_0, e_1), e_2), e_3)\)
State in 1998 = `[ ] .apply(TickerListed, ticker: MSFT, date: 1986)`
Replay to Earlier State: 1998

State in 1998 = [MSFT Listed]
Replay to Earlier State: 2000

State in 2000 = 

- MSFT (Microsoft)
- IPET (Pets.com)
- FB (Facebook)

\[ 	ext{State in 2000} = \left[ \begin{array}{c} \text{MSFT Listed} \\ \text{IPET Listed} \\ \text{FB Listed} \end{array} \right]. \text{apply(} \begin{array}{c} \text{TickerListed} \\ \text{date: 2000} \end{array} \text{)} \]
Replay to Earlier State: 2000

State in 2000 = [MSFT Listed, IPET Listed]
Replay to Earlier State: 2001

State in 2001 = 
\[
\begin{bmatrix}
\text{MSFT Listed} \\
\text{IPET Listed}
\end{bmatrix}
\].apply(
  \text{TickerDelisted}
  \text{ticker: IPET}
  \text{date: 2000})
Replay to Earlier State: 2001

State in 2001 =

- **MSFT (Microsoft)** Listed
- **IPET (Pets.com)** Delisted

- **FB (Facebook)**
Replay to Earlier State: 2012

State in 2012 =

```
[ MSFT Listed
  IPET Delisted ]
```

apply(
  TickerListed
  ticker: FB
  date: 2012
)
Replay to Earlier State: 2012

State in 2012 = [MSFT Listed, IPET Delisted, FB Listed]
Review - Only the Events are Stored

Events:

- TickerListed
ticker: MSFT
date: 1986

- TickerListed
ticker: IPET
date: 2000

- TickerDelisted
ticker: IPET
date: 2000

- TickerListed
ticker: FB
date: 2012

Timeline:
- MSFT (Microsoft)
- IPET (Pets.com)
- FB (Facebook)
Potential Benefits

- Answer the unanswerable (via history replay)
- Debugging of historical states deterministically (via history replay)
- Never Lose Information (write-only store)
- Edit the Past (via new events effective at older times)
- Optimize reads (purpose-built read models)
- Enhanced Analytics (analyze all history as it occurred)
**Potential Drawbacks**

- Eventual Consistency
- No built-in querying of domain models (SELECT name WHERE …)
- Risks of using a new architectural pattern
- Lack of agreement on tools and infrastructure
- Increased storage requirements
What is different from CRUD?
CRUD Micro-Service

Data Store

ORM Model
ORM Model
ORM Model

Controllers

REST API

Background Process

Create / Update

External Market Data APIs

Client #1

Read #1

Client #2

Read #2

Client #3

Update / Delete
Event-sourced Micro-Service

Event Store

Command Processor

Projection #1

Projection #2

Domain Models (pure)

Read Model #1
(files in S3)

Read Model #2
(specific database)

Read #1

Read #2

Client #1

Client #2

Client #3

Create / Update Commands

Update / Delete Commands

External Market Data APIs

Background Process

Write Events

Event Store

Read Events

Read Events

Client #1

Client #2

Client #3

Read #1

Read #2
Vocabulary

1. Domain-Driven Design (DDD)
   ○ Event: *something that happened in the past; a fact; a state transition*
   ○ **Command:** *a request for a state transition in the domain*
   ○ Aggregate

2. Event Sourcing (ES)
   ○ Projection
   ○ Read Model
Aside: Domain Model is Pure?

- FP: A "pure" function doesn't cause any side effects
  - No reads or writes that modify the world
  - No altering a mutable data structure
  - Substitute \( f(x) \) for its result without changing meaning of program

- An event-sourced domain can be two pure functions
  - \( \text{process(currentState, command)} \rightarrow e_1, e_2, e_3 \)
  - \( \text{apply (currentState, event)} \rightarrow nextState \)

- Separates the logic of the model from interactions with any changing state in the world
What is different from CRUD?

(Quotes from Greg Young)

"The model that a client needs for the data in a distributed system is screen-based and different than the domain model."
Trade-offs

• ACID vs. Eventual Consistency
  ○ CRUD w/ ACID database
    ■ Once a row is written, subsequent reads reflect it
    ■ But: no help with domain-level consistency!
  ○ Event Sourced
    ■ Once event is written, subsequent reads reflect it
    ■ Projections eventually consistent

• Up-front costs
  ○ Domain modeling is hard!
CRUD Models of Market Data
CRUD Models of Market Data

**Universe**
- name
- ...

**Org**
- name
- ...

**Listing**
- ticker
- exchange
- trading status
- ....

**Price**
- date
- value
- currency

**Fundamentals**
- earnings per share
- market cap
- ...

**Other Listings**
- merged with
- spin-off from
Q: Was AOL in **Investible Universe** in 1995?
2001 - Merger of AOL/Time Warner

Org
name: Time Warner

Listing
ticker: TWX
trading status: Listed
....

Org
name: America Online

Listing
ticker: AOL
trading status: Listed
....

Orgs
  .findByName("Time Warner")
  .mergeWith("America Online")

Orgs
  .findByName("America Online")
  .setName("AOL Time Warner")
  .addListing("TWX")

Listings
  .findByTicker("TWX")
  .setTradingStatus("Delisted")
2001 - Merger of AOL/Time Warner

Listings
.findByTicker("AOL")
.setTicker("TWX")

Listing
ticker: AOL
trading status: Listed
....

Listing
ticker: TWX
trading status: Delisted
...

Orgs
.findByName("Time Warner")
.mergeWith("America Online")

Orgs
.findByName("America Online")
.setName("AOL Time Warner")
.addListing("TWX")

Listings
.findByTicker("TWX")
.setTradingStatus("Delisted")
2003 - Name / Ticker Change

Listing ticker: TWX (old)
trading status: Delisted
...

Org name: Time Warner
Org name: AOL Time Warner

merged with

Listing ticker: AOL
trading status: Listed
....

Org
  .findByName("AOL Time Warner")
  .setName("Time Warner")

Listings
  .findByTicker("AOL")
  .setTicker("TWX")
2003 - Name / Ticker Change

Time Warner
America Online
AOL Time Warner
Time Warner
AOL

Org
name: Time Warner
merged with
Org
name: Time Warner

Listing
ticker: TWX (old)
trading status: Delisted

Listings
.findByTicker("AOL")
.setTicker("TWX")

.beginning of timeline:
1995
2000
2005
2010
2015

.end of timeline:
1995
2000
2005
2010
2015
2009 - AOL Spinoff

Org
name: Time Warner
merged with

Listing
ticker: TWX (old)
trading status: Delisted

Org
name: Time Warner

Listing
ticker: TWX
old ticker: AOL
trading status: Listed

Org
.findBy("Time Warner")
.spinOff("AOL")
.addListing("AOL")

Listings
.newListing("AOL")
2009 - AOL Spinoff

- Org name: Time Warner
  - merged with
  - Org name: Time Warner
    - Listing
ticker: TWX (old)
trading status: Delisted

- Org name: AOL
  - spin-off from
  - Org name: Time Warner
    - Listing
ticker: TWX
old ticker: AOL
trading status: Listed

- Org
  - findByName("Time Warner")
  - spinOff("AOL")
  - addListing("AOL")

- Listings
  - newListing("AOL")
Answerable with CRUD Models?

Q: Was AOL in Investible Universe in 1995?

A: No and Yes
  ○ No  current AOL org (didn't exist at time)
  ○ Yes  former America Online (now Time Warner)

Complexity in query, requires previous states
  ○ Query against version columns with date ranges?
  ○ Query against previous versions tables?
CRUD Models - How to Update?

- Update price of TWX on 1995-01-03
  - Original: $56.22  Correct: $52.62

- Complexity in query
  - Wrong: Listings.findByTicker("TWX") // this is AOL!
  - Right: Complex historical query...

- Complexity in update
  - Org Primary Listing - any change?
  - Org Universe membership - any changes?
  - Support Two-Dimensional Time aka as-of query?
Problems
Main Problem

Unanswerable questions

- Time travel intractable
- Past not always reproducible
More Problems

● Correctness
  ○ Divergent interpretations of data

● Availability
  ○ How often can data be unavailable to clients?

● Performance
  ○ How fast must operations complete?

● Determinism
  ○ Reproducing prior states for reporting, debugging, etc.

● Auditability
  ○ Who changed what when and why?
Problems - Correctness

- Need a new definition of e.g. adjusted price
  - Old: unadjusted * splits * spin-offs
  - New: unadjusted * splits * spin-offs * dividends

- But...
  - Some client systems still need the old definition
  - CRUD data store didn't store the individual factors

- Common Solutions
  - Add past to relational model? Reprocess?
Problems - Availability

● How long can data be unavailable?
  ○ Not long - End-user client
  ○ Hours to Days - Reporting

● But...
  ○ Most stringent of client requirements applies to all
  ○ Cascading failures: unavailability propagates

● Common solutions
  ○ bulk-heading, circuit-breakers, more servers
  ○ more complex than necessary?
Problems - Performance

- How fast must operations complete?
  - Writes need to keep up with input
  - Reads have varying requirements

- But..
  - Due to contention on Shared Mutable State, badly performing Reads can impact everything else

- Common Solutions
  - Caching, sharding, more server resources
  - Trade-off with ACID consistency
Problems - Determinism

- Reproducing prior states
  - Reporting Consistently on a Past period
    - Apply adjustments only to end of the period
  - Debugging
    - Reproduce state of data in past
- But..
  - Not easy with Shared Mutable State!
- Common Solutions
  - Versioned rows, audit tables, database snapshots
Problems - Auditability

● Why did data change?
  ○ Attribution (source of data)
  ○ Security (who did it)
  ○ History (what and when was previous value)

● But..
  ○ Not easy with Shared Mutable State!

● Common Solutions
  ○ Versioned rows, audit tables
Event Sourced Models of Market Data
Event Sourced Listings

Listing Reference Data Processor

- Listing Aggregate
- Our Listing Id

Reference Data
- Vendor Listing Id
- Ticker
- Exch
- ISIN
- Org Name
- Vendor Org Id
- Other Reference Data...

Background Process

External Market Data APIs
- ListingInitialized
- ListingChanged

Our Listing Id

ListingListed
ListingDelisted
ListingDeleted
ListingUndeleted
ListingIsinChanged
ListingTickerChanged
ListingSourceOrgIdChanged
Event Sourced Orgs

Org Lifecycle Data Processor

Org Aggregate

Our Org Id

Listing

1..n

Reference Data

Vendor Org Id

Org Name

Listing Listed

Listing Delisted

Listing Deleted

Listing Undeleted

Listing Isin Changed

Listing Ric Changed

Org Listed

Org Delisted

Org Deleted

Org Undeleted

Org Listing Added

Org Listing Removed

Org Primary Listing Changed

Org Entered Universe

Org Left Universe

Listing

Reference

Data

Processor

(state...)

Listing Reference Data Processor
Vocabulary

1. Domain-Driven Design (DDD)
   - Event: *something that happened in the past; a fact; a state transition*
   - Command: *a request for a state transition in the domain*
   - **Aggregate:** *domain objects in a transactionally-consistent unit*

2. Event Sourcing (ES)
   - Projection
   - Read Model
Resulting Read Models
Stock Universes - Read Model

- Generate as CSV files in S3, clients retrieve via REST API

  GET /universes/1995-01-03

  Org Id, Known Universe?, Active Universe?, Investible Universe?, Primary Listing Id, ... Ticker
  "49498", true, true, true, "0x00100b000b569402", "US8873173038", ... "AOL"

- Problems?
  - **Correctness:** Interpretation only for this use case
  - **Availability:** No impact on other use cases
  - **Performance:** No read-side calculations
  - **Determinism:** Can re-generate from event source data
  - **Auditability:** Available in event source data

- Consistency is at domain level -- entire history of universes in this case
  - Generate entire history to S3 bucket, API switches buckets atomically
Answerable via Event Sourcing?

Q: Was AOL in **Investible Universe** in 1995?

```
GET /universes/1995-01-03
  Org Id, Known Universe?, Active Universe?, Investible Universe?, Primary Listing Id, ..., Ticker
  "49498", true, true, true, "0x00100b000b569402", "US8873173038", ..., "AOL"
```

A: **Yes!** former America Online (now Time Warner)

Trivial query against purpose-built Read Model
Org History - Read Model

- Build directly from indexed event stream, clients retrieve via REST API

```text
GET /orgs?ticker=TRW

[ { eventType: "ListingListed", listing_id: "1", ticker: "AOL", processedAt: "...", effectiveAt: "...", ...

{ eventType: "OrgListed", ...

{ eventType: "ListingAdded", listing_id: "1", ...

{ eventType: "ListingTickerChanged", listing_id: "1", old_ticker: "AOL", ticker: "TWX" ...

]```

- Problems?
  - Correctness: Interpretation only for this use case
  - Availability: No impact on other use cases
  - Performance: No read-side calculations
  - Determinism: Reads directly from event source data
  - Auditability: Available in event source data

- Consistency is at domain level -- entire history of single org in this case
  - Generate entire history on-the-fly directly from source (indexed!)
1. Domain-Driven Design (DDD)
   - Event: *something that happened in the past; a fact; a state transition*
   - Command: *a request for a state transition in the domain*
   - Aggregate: *domain objects in a transactionally-consistent unit*

2. Event Sourcing (ES)
   - Projection: *to derive current state from the stream of events*
   - Read Model: *a model of current state designed to answer a query*
Read Model vs Cache

A **cache** is a query intermediary. It holds a previous response until invalidated, then queries again.

A **read model** does not query. Applying new events to it changes the answer it returns.

Example: count of people in room

- both may return 10 when the answer is now 11 or 9 - staleness
- cache counts the people - slow and may require locking the doors
- read model applies the entrance/exit events - like a click counter - no impact on people nor doors
A few centuries from now, all the English of the past 400 years will sound equally old-timey and interchangeable.
Conclusions

- Answer the "unanswerable"
- Avoid impacts on other use cases
- Keep facts that may answer future questions
“So long, and thanks for all the fish.”
Events vs Audit Tables or Versioned Rows

An audit table or row version column use shared mutable state as the source of truth, and additionally store some history. Inconsistencies are hard to fix, edits of the past are challenging, new use cases can be challenging.

An event store is an append-only list of immutable facts. What has occurred is recorded, and can be replayed to interpret according to a future use case.

Example: count of people in room

- audit table is a logbook in the room - query may be complex, facts may not be consistent, depends on keeping it up to date
- versioned rows is a logbook carried by each person - same issues as audit table
- event store is "just the facts", only interpretation changes