Scaling Your Architecture With Services and Events

Randy Shoup
@randyshoup
linkedin.com/in/randysghoup
Background

• VP Engineering at WeWork
  o Physical space as a service

• VP Engineering at Stitch Fix
  o Software and data science in clothing retail

• Director of Engineering for Google App Engine
  o World’s largest Platform-as-a-Service

• Chief Engineer at eBay
  o Multiple generations of eBay’s infrastructure
Evolution to Microservices

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

@randyshoup  linkedin.com/in/randyshoup
No one starts with microservices

... Past a certain scale, everyone ends up with microservices
If you don’t end up regretting your early technology decisions, you probably over-engineered.
Services and Events

• Migrating to Microservices
• Challenges of Data in Microservices
• End-to-end Correctness in Event-Driven Systems
Services and Events

• Migrating to Microservices

• Challenges of Data in Microservices

• End-to-end Correctness in Event-Driven Systems
Microservices

- Single-purpose
- Simple, well-defined interface
- Modular and independent
- Isolated persistence (!)
Extracting Microservices

- Problem: Monolithic shared DB
  - stitchfix.com
  - Styling app
  - Warehouse app
  - Merch app
  - CS app
  - Logistics app
  - Payments service
  - Profile service
  - Clients
  - Shipments
  - Items
  - Styles, SKUs
  - Warehouses
  - etc.

@randyshoup
linkedin.com/in/randyshoup
Extracting Microservices

- Decouple applications / services from shared DB

- stitchfix.com
- Styling app
- Warehouse app
- Merch app
- CS app
- Logistics app
- Payments service
- Profile service

- Clients
- Shipments
- Items
- Styles, SKUs
- Warehouses
- etc.

@randyshoup
linkedin.com/in/randysoup
Extracting Microservices

• Decouple applications / services from shared DB
Extracting Microservices

• Step 1: Create a service
Extracting Microservices

• Step 2: Applications use the service

- Styling app
- Warehouse app
- client-service

- `core_client`
- `core_sku`
- `core_item`

randyshoup
linkedin.com/in[randyshoup]
Extracting Microservices

- Step 2: Applications use the service

Do NOT stop here!
① All the problems of a distributed system
② All the problems of a shared database
③ None of the benefits of microservices 😞
Extracting Microservices

- Step 3: Move data to private database

@randyshoup
linkedin.com/in/randysΟuρ
Extracting Microservices

• Step 4: Rinse and Repeat

Styling app

Warehouse app

client-service

item-service

core_sku

core_client

core_item

@randyshoup

linkedin.com/in/randyshoup
Extracting Microservices

- Step 4: Rinse and Repeat

@randyshoup
linkedin.com/in/randyshoup
Extracting Microservices

- Step 4: Rinse and Repeat

![Diagram showing relationships between microservices]

- @randyshoup LinkedIn: linkedin.com/in/randyshoup
Services and Events

• Migrating to Microservices

• Challenges of Data in Microservices
  o Two Architectural Tools
  o Shared Data
  o Joins
  o Transactions

• End-to-end Correctness in Event-Driven Systems
Services and Events

• Migrating to Microservices

• Challenges of Data in Microservices
  o Two Architectural Tools
  o Shared Data
  o Joins
  o Transactions

• End-to-end Correctness in Event-Driven Systems
System of Record

- Single System of Record
  - Every piece of data is owned by a single service
  - That service is the *canonical system of record* for that data
- Every other copy is a *read-only, non-authoritative cache*

@randyshoup

linkedin.com/in/randyshoup
Events as First-Class

• “A significant change in state”
  o Statement that some interesting thing occurred

• Traditional 3-tier system
  o Presentation ➔ interface / interaction
  o Application ➔ stateless business logic
  o Persistence ➔ database

• Fourth fundamental building block
  o State changes ➔ events
  o 0 | 1 | N consumers subscribe to the event, typically asynchronously
Microservices and Events

• Events are a first-class part of a service interface

• A service interface includes
  o Synchronous request-response (REST, gRPC, etc)
  o Events the service produces
  o Events the service consumes
  o Bulk reads and writes (ETL)

• The interface includes any mechanism for getting data in or out of the service (!)
Services and Events

- Migrating to Microservices

- Challenges of Data in Microservices
  - Two Architectural Tools
  - Shared Data
  - Joins
  - Transactions

- End-to-end Correctness in Event-Driven Systems
Shared Data

• Monolithic database makes it easy to leverage shared data

• Where does shared data go in a microservices world?

 randyshoup
 @randyshoup

randyshoup

linkedin.com/in/randyshoup
Shared Data

Option 1: Synchronous Lookup
- Customer service owns customer data
- Fulfillment service calls customer service in real time
Shared Data

Option 2: Async event + local cache

- Customer service owns customer data
- Customer service sends `address-updated` event when customer address changes
- Fulfillment service caches current customer address

@randyshoup
linkedin.com/in/randysliopulou
Shared Data

Option 3: Shared metadata library
- Read-only metadata, basically immutable
- E.g., size schemas, colors, fabrics, US States, etc.
Services and Events

• Migrating to Microservices

• Challenges of Data in Microservices
  o Two Architectural Tools
  o Shared Data
  o Joins
  o Transactions

• End-to-end Correctness in Event-Driven Systems
Joins

• Monolithic database makes it easy to join tables

SELECT FROM A INNER JOIN B ON ...

• Splitting the data across microservices makes joins very hard
Joins

Option 1: Join in Client Application

- Get a single customer from customer-service
- Query matching orders for that customer from order-service

@randyshoup
linkedin.com/in/randyshoup
Option 2: Service that “Materializes the View”

- Listen to events from item-service, events from order-service
- Maintain denormalized join of items and orders together in local storage
Joins

Many common systems do this
• “Materialized view” in database systems
• Most NoSQL systems
• Search engines
• Analytic systems

@randyshoup  
linkedin.com/in/randyshoup
Services and Events

• Migrating to Microservices

• Challenges of Data in Microservices
  o Two Architectural Tools
  o Shared Data
  o Joins
  o Transactions

• End-to-end Correctness in Event-Driven Systems
Transactions

• Monolithic database makes transactions across multiple entities easy

```
BEGIN; INSERT INTO A ...; UPDATE B...; COMMIT;
```

• Splitting data across services makes transactions very hard
“In general, application developers simply do not implement large scalable applications assuming distributed transactions.”

-- Pat Helland

*Life After Distributed Transactions: An Apostate’s Opinion*, 2007
“Grownups don’t use distributed transactions”

-- Pat Helland
Who has heard of database isolation levels?

Who runs at isolation level SERIALIZABLE?
Workflows and Sagas

• Transaction \(\rightarrow\) Saga
  o Model the transaction as a state machine of atomic events

• Reimplement as a workflow

• Roll back by applying compensating operations in reverse
Many real-world systems work like this

- Payment processing
- Expense approval
- Travel
- Software development process
Workflows and Sagas

• Simple event-driven processing
  o Very lightweight logic
  o Stateless
  o Triggered by an event

• Consider Function-as-a-Service ("Serverless")

@randyshoup
linkedin.com/in/randyshoup
Services and Events

- Migrating to Microservices
- Challenges of Data in Microservices
- End-to-End Correctness in Event-Driven Systems
  - Producer correctness
  - Consumer correctness
System-Level "Correctness"

- Producer will produce the event
- Transport will deliver the event
- Consumer will process the event

@randyshoup
linkedin.com/in/randyshoup
System correctness is a distributed coordination problem.
Services and Events

• Migrating to Microservices

• Challenges of Data in Microservices

• End-to-End Correctness in Event-Driven Systems
  o Producer correctness
  o Consumer correctness
Producer “Correctness”

• Problem: Write state change AND send event
  o Write to the database
  o Send event to the event bus

• The producer must guarantee it will send the event if it changes the state
Producer “Correctness”

Approach 1: Change Data Capture

- Write state change to database
- *(Database writes change to its transaction log)*
- “Connector” tails transaction log, sends event
Approach 2: Change Data Capture (in reverse)

- Send event to event bus
- Consumer writes state change to database
Approach 3: Atomic write + send to single system

- State changes and events are stored in the same system
- E.g., state and events live in database tables
Approach 3: Atomic write + send to single system

• State changes and events are stored in the same system
• E.g., state and events live in the event bus

Producer “Correctness”
Producer “Correctness”

**Approach 4: Persistent local queue**
- Queue local tasks to write state change, send event
- Continue trying to write and send until done
Services and Events

• Migrating to Microservices

• Challenges of Data in Microservices

• End-to-End Correctness in Event-Driven Systems
  o Producer correctness
  o Consumer correctness
Consumer “Correctness”

• Problem: Receive event and process action
  o Receive event from the event bus
  o Do some processing

• The consumer must guarantee it will process the event eventually
Consumer “Correctness”

Receive: Multiple Consumer Clusters

- Cluster per consumer type, each with its own queue

@randyshoup linkedin.com/in/randyshoup
Consumer “Correctness”

Receive: Multiple Consumer Clusters, with Multiple Consumers
• Each cluster has multiple consumers
Consumer “Correctness”

Process: Lease
- Lease the event from the bus for X period
- Process the event
- Terminate the lease (*commit the offset, etc.*)
• This stuff is hard
• There are decades of experience in this area
• Resist the temptation to roll your own from first principles 😞
Services and Events

- Migrating to Microservices
- Challenges of Data in Microservices
- End-to-End Correctness in Event-Driven Systems
Ačiū!

@randyshoup

linkedin.com/in/randyshoup

medium.com/@randyshoup
Resources

• Caitie McCaffrey, “Distributed Sagas: A Protocol for Coordinating Microservices” (J On the Beach 2017)

• Chris Richardson, “Data Consistency in Microservices with Sagas” (QCon San Francisco 2017)

• Gregor Hohpe, Enterprise Integration Patterns, 2003

• Martin Kleppmann, Designing Data-Intensive Applications, 2017