Adam Bellemare – Building Event-Driven Microservices

2. Event-Driven Microservice Fundamentals

Building Topologies

- **Microservice Topology**
  - the event-driven topology internal to a single microser vice

  Microservice topology
- **Business Topology**
  - the set of microservices, event streams, and APIs that fulfill complex business functions
  - composed of Business Communication Structures (detailed in Chapter 1)

The Contents of an Event

- An event can be ANYTHING that has happened within the scope of the Business Communication Structure
  - receiving an invoice
  - booking a meeting room
  - requesting a cup of coffee
  - hiring a new employee
  - successfully completing arbitrary code
- An event is a RECORDING of what happened
  - the single source of truth

The Structure of an Event

- **Unkeyed Event**

Key	Value
N/A	ISBN: 372779, Timestamp: 1538913600

  - used to describe an event as a singular statement of fact
- **Entity Event**

Key	Value
ISBN: 372779	Author: Adam Bellemare

  - particularly important in event-driven architectures
  - They provide a continual history of the state of an entity and can be used to materialize state
  - ⚠️ Only the latest entity event is needed to determine the current state of an entity
- **Keyed Event**

Key	Value
ISBN: 372779	UserId: A537FE
ISBN: 372779	UserId: B80012

  - contains a key but does not represent an entity
  - usually used for partitioning the stream of events to guarantee data locality within a single partition of an event stream
    - An example could be a stream of events, keyed on ISBN, indicating which user has interacted with the book
    - Can be aggregated into single Entity Event

Materializing State from Entity Events

- **"table-stream duality"**
  - 1. Materializing an event stream into a table

  - 2. Generating an event stream from the changes applied to a table

  - used for communicating state between event-driven microservices
- ⚠️ The deletion of a keyed event is handled by producing a tombstone
  - a keyed event with its value set to null
- **Compaction**

  - reduces both disk usage and the number of events

Event Data Definitions and Schemas

- The consumer needs a shema to interpret the events w/o consulting with the producer
- **Good schematizations examples**
  - 👍 Apache Avro
  - 👍 Google Protobuf
  - 1. Evolution is supported – w/o requiring downstream consumers to change the code
  - 2. Typed classes

Microservice Single Writer Principle

- Each event stream has one and only one producing microservice (owner of each event)

Powering Microservices with the Event Broker

- Multiple, distributed event brokers work together in a cluster are suitable for large-scale enterprises and provide great features
  - Scalability
  - Durability
  - High availability
  - High-performance
- **Event Storage and Serving min requirements**
  - Partitioning
  - Strict ordering
  - Immutability
  - Indexing — to specify which offset to begin reading from
  - Infinite retention
  - Replayability
- **Additional Factors to Consider**
  - Support tooling
  - Hosted services
  - Client libraries and processing frameworks
  - Community support
  - Long-term and tiered storage

Event Brokers vs Message Brokers

- 👍 Event brokers can be used in place of a message broker
- 👎 but a message broker cannot fulfill all the functions of an event broker
  - only one consumer will read event from the queue
  - event is deleted from the queue and cannot be replayed
- **Consuming from the Immutable Log**
  - usually event broker is an append-only immutable log
  - **Consuming as an event stream**

    - **Consumer group**
      - allows for multiple consumers to be viewed as the same logical entity
  - **Consuming as a queue**
    - each event is consumed by one and only one microservice instance
    - ⚠️ Event order is NOT maintained when processing from a queue
      - Parallel consumers consume and process events out of order
      - Single consumer may fail to process an event, return it to process later, and move on to the next event
    - 👍 Apache Pulsar currently supports queues
    - 👎 Kafka does not support queues

- **Providing a Single Source of Truth**
  - durable and immutable log ensures the "Single Source of Truth"
  - ⚠️ This requires a culture shift in the organization
    - 👍 Must also publish the monolith's data to the event broker!
    - 👍 Consumers now should not deal with the monolith but consume data from the event stream!

Managing Microservices at Scale

- **Putting Microservices into Containers**
  - 👎 A vulnerability in the kernel can put all the containers that host at risk
    - shared tenancy models in cloud computing make it a significant consideration
- **Putting Microservices into Virtual Machines**
  - 👎 higher overhead costs compared to containers
  - 👍 Keep an eye on new technologies – they should make the usage of VMs more competitive
    - Google's gVisor, Amazon's Firecracker, and Kata Containers
- **Managing Containers and Virtual Machines**
  - Container management systems (CMSes)
    - Kubernetes, Docker Engine, Mesos Marathon, Amazon ECS, and Nomad

Paying the Microservice Tax

- the sum of costs, including financial, manpower, and opportunity, associated with implementing the tools and components of a microservice architecture
- **Microservice Tax**
  - Small organizations would likely do best to stick to a modular monolith
  - Larger organizations need to account for the total costs of both the implementation and maintenance of a microservice platform, considering the long-term roadmap
  - The microservice tax is being steadily reduced with new tools (CMSes, event brokers, etc)