Beam data pipelines on microservice architectures

Pragalbh Srivastava
Wayfair
Agenda

- Wayfair & imagery
- Microservice essentials
- Digital Studio
- Domain event challenges
- Cloud Dataflow [Beam]
- Learnings
- QA

Explore building data pipelines for microservice architectures. Includes Wayfair Digital Studio domain event landscape and deriving key business metrics real-time in a decoupled scalable approach.
Wayfair & imagery

Wayfair is the world's largest online destination for all things home incl furniture, household items, appliances etc.

Unparalleled selections and high quality imagery are keys to provide a rich & unique user experience. Photo studios are expensive to operate and require significant time to produce an image. 3D modeling and custom imagery is one of the main focus areas of investment.

We don’t sell furnitures, we sell images.

3D model to Image
Imagery Template
Model swaps
Common streaming publications

Database supports **transactions**, provide consistent view, durable and battle tested but have a weakness - **scalability**

The ones which scale doesn’t provide above greatness

*What if you need to build a data pipeline where you don’t have guarantees of database but the system provides all elasticity in the most decoupled ways?*

*Let’s talk about microservices!*
Microservice essentials

Object Oriented Programming (OOP) domination with reusability, flexibility & effective problem solving
Small independently deployable services that work together, modelled around a business domain
**DDD, CQRS & Event Sourcing** are talked a lot in microservice conversations

**Domain-driven design (DDD)** is the concept that the structure and language of software code (*class names, class methods, class variables*) should match the business domain. For example, if a software processes loan applications, it might have classes such as `LoanApplication` and `Customer`, and methods such as `AcceptOffer` and `Withdraw`.

**Command Query Responsibility Segregation (CQRS)** talks about separation of commands (*write requests*) and queries (*read requests*). Read stores are optimized for handling queries.

**Event Sourcing (ES)** is persisting changes that are happening in the application as a sequence of events.

*Any fool can write code that a computer can understand. Good programmers write code that humans understand.*

— *Refactoring: Improving the Design of Existing Code, 1999*
Wayfair Digital Studio

Platform made up of web applications, services and database to create 3D assets at Wayfair
Re-developed using domain driven design architecture patterns

**Domain**: Business context on which a system is built.
Examples: Request, Job, Task

**(Domain) Events**: are described as something that happens in the domain and is very domain focused.
Example: RequestSubmitted, RequestDispatched, JobAssigned

**Services**: Gateway for external interactions
Example: Object Service, Eligibility Service
Applications are modelled on domains - Request, Job, Task, 3d, Renders etc
Supports multiple workflows - Model & Image rendering, Image modifications
Domain events triggered on state change, step completion etc
Request & Job domains

**Request** is a unit of human or an application desire for an asset creation.

**Job** is the execution plan for completing request. Manage all tasks to fulfill request.

**Task** is an atomic unit of work needed to complete a job.
Data pipeline challenges

- Need to stitch multiple events across domains to answer business KPIs (Ex: TAT of 3d model creation, % of requests blocked)
- Domain events represent an activity within a domain for domain experts
- Not suitable for external consumption
- Pure domain events must process in-memory & within the same transaction
- Fire & forget nature can cause inconsistencies if transaction fails
- Services endpoints (REST / GraphQL) designed for application interactions
  - High frequency, low data volume, low latency requests
  - Restricts payload size, rate of requests per hour, # of requests
Potential options

- **Event Sourcing** is persisting changes that are happening in the application as a sequence of events.
- This sequence can be used to reconstruct the current state.
- Banking transactions example:
  - Credit and debits occurring in an account are events.
  - All these events can be queried to derive a current balance.
  - Alternatively utilizing event sourcing concepts current balance can be pre-calculated and stored.

- **Event aggregation** are set of handler (continuous listener) to maintain an effective read model.
- **Observer pattern** to avoid losing decoupling in domain architecture.
- Example: Create an aggregate when a Job is assigned to a Modeler after the request is submitted.
- Aggregate persistence options: SQL, NoSQL, Files, Kafka.
- Aggregation of DDD is equivalent to projections of CQRS.
Data architecture

Event collector stream application
- Perform schema validation and envelop the Kafka message
- Publish Kafka messages to Cloud Pubsub

Apache Beam / Dataflow job
- Perform real-time enrichment
- Dynamic routing
- Event consolidation
- Implement observer pattern
- Outbound PubSub topic

Data Analytics
- Data processing in BigQuery
- Foundation & Curated data layers
- Data Studio dashboard
Event observer

- One BigQuery table per domain event is not efficient
- Requires joining of multiple tables and apply business logic
- Needs to happen for all the data pipelines
- Introduced an event listener to Pubsub events
- Performs filtering, consolidation, routing in real time
- Utilize BigTable (NoSQL) for short term storage
- Outbound Pubsub event triggered once a milestone is achieved
- Example: Create an aggregate when a Job is assigned to a Modeler after the request is submitted
Apache Beam: Cloud Dataflow

- **Fully managed service** for batch & stream
- Apache Beam framework
  - Unified programming model
  - Runner independent
  - Functionally biased MapReduce
- **Serverless**, auto provision of resources
- No infrastructure woes
- **Dynamic scaling**
  - Key for unbounded source
  - Predicting future data not needed
- Google provided **templates** for common use-case
- Not ideal for SQL data pipelines
  - More lines of code and complexities
Wayfair’s Dataflow usage growth

GCP cloud migration started in 2020

Exponential growth in the last 2 years

Usage across multiple teams and orgs

Cost metrics, can’t be shared publicly
Imagery Ops Dashboards

E2E Turnaround Time

Definitions
- Image Completion Date: Date an Image arrives, or would have arrived, to be processed.
- Image Start: Date an Image arrives, or would have arrived, to be processed.
- Image Completion Time: End date of Image processing.
- Image Start: Date an Image arrives, or would have arrived, to be processed.
- Image Completion Time: End date of Image processing.
- Model Start: Model Approved
- Model End: Model In Progress

General Filters: Applies to below dashboard
- Factory_Name
- request_type
- Demand_Channel_Name
- Pasted_Flag
- Line_Swap_Name
- Mile_Pix_Flag
- Jan 1, 2022 - Jul 18, 2022

TaT Request to Image Completion

90th Percentile TaT Request to Image Completion by Demand Channel

Avg. TaT by Production Stage

Volume of Requests Made by Request Month

Stage 1: Request to First Model Assignment

FILTERS
- Asset_Fk
- Demand_Channel
- Jan 1, 2022 - Jul 18, 2022

How many days does it take from request to first model assignment? (Logging Indicator)

How does the average request to first model assignment time break down? (Logging Indicator)
Learning & Recommendations

- Focused more on technical solutions like stream ingestion and processing
- Lack of understanding of Microservice architecture (DDD, CQRS, ES) in the beginning
- Treated domain events as another Kafka / Pubsub topics
- Deep dive on architecture patterns only when data stopped making sense
- Observer pattern reduced the noise, simplified the data pipeline
- Apache Beam event windows - multiple options, complex, dropped records

*Don’t jump onto technical solution with just business knowledge, try to understand the underlying design constructs*
Helpful resources

The Blue book - Domain driven design by Eric Evans. Introduced DDD as an established concept to the world in 2004

Martin Fowler
https://martinfowler.com/tags/domain%20driven%20design.html

Implementing domain driven design
https://medium.com/design-and-tech-co/implementing-domain-driven-design-for-microservice-architecture-26eb0333d72e

Aggregates in domain driven design
https://khalilstemmler.com/articles/typescript-domain-driven-design/aggregate-design-persistence/

Dataflow docs (Google official)
https://cloud.google.com/dataflow/docs

Apache beam framework (Apache official)
https://beam.apache.org/documentation/
Questions?