Implementing an Event-Driven Microservices Architecture: A case study of Jet.com

NIKHIL BARTHWAL
SENIOR ENGINEER, JET.COM
Background

- Launched in July 2015
- 26 Million visitors a month, 25K orders daily
- 8 Million customers
- Have 15 million SKU’s in inventory

Acquired by Walmart for $3.3 Billion in Sept 2016!
Microservices based (Over 700+ in production)
Event-Driven Architecture
Event Sourcing
Technology Stack

- Runs on Microsoft Azure
- Uses .Net framework
- Use a mix of Kafka, Redis, Splunk, Event Store, ...

Bulk of backend implemented in F#!
A view of Microservice

Mathematical representation: \( y = f(x) \)

**Pure Service (Majority)**
No side effects except Logging

**Impure service (Minority)**
Side effects like I/O to DB etc.
### Why did we not use OOP?

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Counter-Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOP models application as objects with state</td>
<td>Microservices mostly don’t have states</td>
</tr>
<tr>
<td>OOP extends imperative with encapsulation and</td>
<td></td>
</tr>
<tr>
<td>polymorphism</td>
<td></td>
</tr>
<tr>
<td>Imperative was designed for Van-Neumann style</td>
<td>Microservices hosted on the cloud don’t interact</td>
</tr>
<tr>
<td>hardware</td>
<td>with hardware directly</td>
</tr>
<tr>
<td>Being dominant paradigm, OOP Languages like</td>
<td>Languages like F#/Scala can inter-operate with</td>
</tr>
<tr>
<td>Java/C# have very good ecosystem</td>
<td>C# /Java seamlessly</td>
</tr>
</tbody>
</table>
## Modelling a Service using FP

<table>
<thead>
<tr>
<th>FP Construct</th>
<th>Mapping to the Service world</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algebraic Data Type</td>
<td>Events Modelling</td>
</tr>
<tr>
<td>Functions</td>
<td>Services</td>
</tr>
<tr>
<td>Immutability</td>
<td>Events are immutable</td>
</tr>
</tbody>
</table>
Implementation: Example in F#

Define Input type

```fsharp
type input =
    | ItemName of string
    | ItemSKU of int
```

Define Output type

```fsharp
type output =
    | ItemInInventory of int
    | ItemNotInInventory of (System.DateTime option)
    | ItemNotSold
```

Write a function to convert input to output...

```fsharp
let CatalogSearch (query:input) : output =
    // the code ...
```
Testing Microservices

**Pure Services:**
- Being pure, behavior is very predictable
- Can be tested exhaustively!

**Impure Services:**
- Unpredictable, due to external state
- Certain services can’t be tested exhaustively (e.g. Payment gateway)!

Diagram:

- WebServer
  - ServiceA
    - ServiceC
    - ServiceD
  - ServiceB
    - DB
  - ServiceE
    - DB
Benefits of using F#

- Scalability
- Productivity
- Code Correctness
Conclusion

- Very few startups scaled to Jet’s size in same time
- Using F# was the most forward looking decision
- Scalability, parallelism & productivity
Questions?

NIKHIL BARTHWAL
SENIOR ENGINEER, JET.COM