Implementing an Event-Driven Microservices Architecture in a Functional Language

NIKHIL BARTHWAL

SENIOR ENGINEER, JET.COM

Background – Jet.com

- Launched in July 2015
- Dynamic pricing engine (2 m calculations/sec)
- 8 Million customers
- 25K orders daily
- Have 15 million SKU's in inventory

Acquired by Walmart for $3.3 Billion in Sept 2016!

Technology Stack

- Runs on Microsoft Azure
- Uses .Net framework
- Use a mix of Kafka, Redis, Splunk, New Relic, …

Backend in F#: Functional Language for the .NET platform!

Architecture

- Uses Domain Driven Design (DDD) to architect services
- Event-Driven Architecture
- Event Sourcing
- Microservices based Architecture

Domain Driven Design (DDD)

- Ubiquitous Language
- Bounded Contexts
- Aggregates

DDD: Bounded Contexts

DDD: CQRS Design Pattern

CustomerService:

- Customer GetCustomer(CustomerId)
- CustomerSet GetCustomersWithName(Name)
- CustomerSet GetPreferredCustomers()
- void ChangeLocale(CustomerId, NewLocale)
- void CreateCustomer(Customer)
- void EditCustomerDetails(CustomerDetails)

CustomerReadService:

- Customer GetCustomer(CustomerId)
- CustomerSet GetCustomersWithName(Name)
- CustomerSet GetPreferredCustomers()

CustomerWriteService:

- void ChangeLocale(CustomerId, NewLocale)
- void CreateCustomer(Customer)
- void EditCustomerDetails(CustomerDetails)

Command Query Responsibility Segregation (CQRS)

Source: http://codebetter.com/gregyoung/2010/02/16/cqrs-task-based-uis-event-sourcing-agh/

DDD: Architecture Layers

Domain Layer (Bulk of code):

- Value Objects are Immutable
- Modeled by F# Discriminated unions
- Attached member functions

  (Not Anemic Domain models)

Event Driven Architecture

- Event occurs when a change happens in system
- All listeners get notified of the event, some take action
- Highly distributed/loosely coupled architecture
- Often used for asynchronous flows of work/information

Event Sourcing

- Modeling state changes as sequence of events
- Storing the event that could trigger the state change
- Enables rolling back to particular time in history

Different applications can process same event but create a different view

Microservice

- Microservices are small, autonomous services that work together.
- Small and Focused on Doing One Thing Well
- Follow the Single Responsibility Principle

Microservice Architecture

- Approach to developing application as set of small services
- Each Microservice runs in its own process
- Communicates with lightweight mechanisms

We have 1000+ in production…

Advantages

- Independent Releasability: Better Scaling
- Technology Heterogeneity
- Resilience
- Ease of Migration

Microservices Deployment Pattern

Request/Response (synchronous)

Event-Driven (asynchronous)

Apache Kafka (Kafunk as client)

N-Tier Microservices

Microservice: A functional view

Mathematical representation: $y=f(x)$

Pure Service (Majority)
No side effects except Logging

Mathematical representation: $y, s_{out}=f(x, s_{inp})$

Impure service (Minority)
Side effects like I/O to DB etc.

Why did we not use OOP?: History

- 40's to 50's: Early machines based on von Neumann architecture
- 50's to 70's: Early (Impressive) Programming languages - C & Fortran
- 80's: OOP was an extension of imperative paradigm with new concepts: encapsulation and polymorphism
- 90's: Internet with problems of Portability & Security
- 90's - 2000's: Managed languages like Java & C#

OOP became Dominant paradigm!

Why did we not use OOP?

Arguments	Counter-Arguments
OOP models application as objects with states	Microservices mostly don't have states
OOP extends imperative with encapsulation and polymorphism	
Imperative was designed for Van-Neumann style hardware	Microservices hosted on the cloud don't interact with hardware directly
Being dominant paradigm, OOP Languages like Java/C# have very good ecosystem	Languages like F#/Scala can inter-operate with C#/Java seamlessly

Modelling a Service using FP

Services world	FP Construct
Service	Functions
Events Modelling	Algebraic Data Type
Events are immutable	Immutability
(Mostly) Stateless service	(Predominantly) Pure code

FP captures the behavior of service architecture very naturally!

F#: A Functional Language for .NET

Declarative Style

Immutability

Functions as building blocks

A statically typed Managed Functional Language for the .NET platform!

Implementation: Example in F#

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

Define Input type

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

Define Output type

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

Write a function to convert input to output…

Asynchronous workflows

```
let RecieveOrder order =
    async {
        do! ShipOrder order
        return [UpdateInventory order ;
            ReturnAcknowledgment order]
    }
```

Stock service sends async request to ship order,
And returns acknowledgment synchronously

Static Analysis of F# code

Procedure

- Define a set of Anti-Patterns
- Static Analyzer parses F# code into AST
- Analyzer detects Anti-Patterns in AST
- Bundle into SonarQube plugin

Features in work …

- Provides "pre-commit" testing
- Dashboard to tract violations
- Integration into CI/CD Pipeline

(Mostly) Pure functions

↓

Easy to analyze and reason about

↓

Static Analysis is effective way to detect issues in code

Security for F# code

Fact: F# is not a mainstream language

Problem: Lack of 3rd party security tools

Solution: Build your own!

Extended the Static Analyzer to:

- Identifying security vulnerabilities
- Weakness in applications source code

Security initiatives:

- Integrate security related tooling into the CI/CD pipeline
- Work on "pre-commit" testing of the code within the IDE
- Build and Maintain tools for security analysis

Benefits of using F#

- Scalability
- Productivity
- Code Correctness

Benefits of using F#: Scalability

- Infinitely scalable in terms of massive parallelism out-of-the box
- Immutability → Not to worry about mutex/lock/race condition
- Allows you to scale from 30K to 2.5M easily

Benefits of using F#: Productivity

- F# code is much more concise than say C#
- Conciseness & clarity result in lower maintenance costs
- Simpler code means reduced ramp-up for new devs

Productivity in F#: Example

	C# project	F# project
Useful Lines	4610 (50%)	1436 (84%)
Null Checks	195 (2%)	4 (0.003%)
Blank Lines	1386 (15%)	94 (6%)
Brace Lines	2686 (29%)	11 (1%)
Comment Lines	290 (3%)	164 (10%)
Total	9167	1709

F# code is 5x shorter than C#

Source: http://fpbridge.co.uk/why-fsharp.html

Benefits of using F#: Code Correctness

- Strict typing makes writing correct code easy
- Immutability makes code more predictable
- Exhaustive pattern matching traps errors at compile time.
- Type system makes illegal states unpresentable

Testing Microservices (Request/Response)

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)!

Testing Microservices (Event Driven)

Original Architecture

Test Setup

Tracability: X-Ray platform

SLA Monitoring: Pulse

Rules to detect SLA Violations are computed in the query to pull data

Chaos Engineering: Definition

"Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production"

Source: http://principlesofchaos.org

Chaos Engineering: In Practice …

- Start by defining 'steady state' that indicates normal behavior
- Introduce "chaos", events that model disruptions
- Monitor the effects on system & its recovery

Automate Experiments to run Continuously in Production

Distributed Scheduler: Nomad

- Flexible, Scalable, Multi-Region Aware scheduler
- Tool for managing a cluster of machines & running applications on them
- Declarative style, abstracts away machines & locations

Conclusion

- Very few startups scaled to Jet's size in same time
- Scalability & Productivity due to F#
- Sound architecture, Unchanged since start

Questions?

Nikhil Barthwal

Senior Engineer, Jet.com

@nikhilbarthwal

nikhilbarthwal@yahoo.com

www.nikhilbarthwal.com