Reactive Summit 2018: Integrating Machine Learning, Reactive Microservices, and Akka with Kafka

Andy Bonham, Director Enterprise Architect, Capital One
Sairam Tadigadapa, Director Software Engineering, Capital One
Thiaga Manian, Lead Software Engineer, Capital One
Oct 23, 2018
Capital One at a Glance

- A leading diversified bank with $365.7 billion in assets, $255.4 billion in loans and $243.7 billion in deposits¹
  - 8th largest bank based on U.S. deposits²
  - 6th largest retail depository institution in metro New York³
  - Largest consumer and commercial banking institution headquartered in the Washington, DC region
  - 3rd largest credit card issuer in the U.S.⁴
  - The 3rd largest issuer of small business credit cards in the U.S.⁵
  - Largest financial institution auto loan originator⁶
  - Largest U.S. direct bank⁷
- Major operations in 15 U.S. cities, Canada, U.K.
- More than 70 million customer accounts and 49,000 associates
- A FORTUNE 500 Company - #100
- Numerous recent awards including:
  - Named to 100 Best Companies to Work For by FORTUNE Magazine
  - Best Places to Work for LGBT Equality by Human Rights Campaign
  - Received J.D. Power & Associates Call Center Certification
  - Aon Hewitt’s Top Companies for Leaders
  - Named to Working Mother’s 100 Best Companies list & Best Companies for Hourly Workers
  - Ranked #14 on Military Times’ 2017 “Best for Vets”
  - Recipient of the Secretary of Defense Employer Support Freedom Award

1) Source: Company reported data as of Q4’17
2) Source: FDIC, Domestic deposits ranking as of Q4’17
3) Source: FDIC, June 2017, deposits capped at $1B per branch
4) Source: Company-reported domestic credit card outstandings, Q4’17
5) Source: The Nilson Report, Issue #1111, June 2017
6) Note: Financial institutions includes banks & specialty finance lenders, Source: AutoCount, most recent quarter originations as of October 2017
7) Source: Regulatory filings, company reports as of June 2017
We are one of the largest banks in the U.S.

<table>
<thead>
<tr>
<th>Q4 2017 Total Loans ($B)</th>
<th>Q4 2017 U.S. Deposits ($B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Wells Fargo</td>
<td>1. Bank of America</td>
</tr>
<tr>
<td>2. Bank of America</td>
<td>2. Wells Fargo</td>
</tr>
<tr>
<td>3. JPM Chase</td>
<td>3. JPM Chase</td>
</tr>
<tr>
<td>5. U.S. Bancorp</td>
<td>5. U.S. Bancorp</td>
</tr>
<tr>
<td>6. Capital One</td>
<td>8. Capital One</td>
</tr>
<tr>
<td>7. PNC</td>
<td>9. SunTrust</td>
</tr>
<tr>
<td>8. TD Bank</td>
<td>10. BB&amp;T</td>
</tr>
<tr>
<td>10. BB&amp;T</td>
<td>12. Key</td>
</tr>
<tr>
<td>11. American Express</td>
<td>13. Fifth Third</td>
</tr>
<tr>
<td>13. Citizens</td>
<td>15. Ally</td>
</tr>
<tr>
<td>14. Fifth Third</td>
<td></td>
</tr>
<tr>
<td>15. M&amp;T</td>
<td></td>
</tr>
</tbody>
</table>

1. Wells Fargo: 976.9
2. Bank of America: 948.2
3. JPM Chase: 930.7
4. Citigroup: 688.1
5. U.S. Bancorp: 283.4
6. Capital One: 255.4
7. PNC: 223.1
8. TD Bank: 154.3
9. SunTrust: 145.5
10. BB&T: 144.8
11. American Express: 127.4
12. Ally: 123.0
13. Citizens: 111.3
14. Fifth Third: 92.5
15. M&T: 88.0
1. Bank of America: 1,227.5
2. Wells Fargo: 1,207.3
3. JPM Chase: 1,187.3
4. Citigroup: 445.6
5. U.S. Bancorp: 321.9
6. TD Bank: 264.1
7. PNC: 262.4
8. Capital One: 243.2
9. SunTrust: 160.8
10. BB&T: 157.4
11. Citizens: 115.0
12. Key: 105.3
13. Fifth Third: 103.1
14. Regions: 97.2
15. Ally: 93.2

Notes: Excludes banks with high non-loan asset concentrations: Goldman Sachs, Morgan Stanley, BONY, State Street, Charles Schwab. Gross loans and domestic deposit data as of 12/31/2017. Based upon total gross loans and total aggregated domestic deposits for bank holding company. Sources: SNL, FDIC.
We have transformed the company into a top 10 bank

2017  Acquires Notch
2016  Acquires Critical Stack and Paribus
2015  Acquires GE Capital’s Healthcare Financial Services, Level Money and Monsoon
2014  Acquires Adaptive Path, a digital design leader and AmeriCommerce, an online e-commerce company
2013  Acquires Beech Street Capital, an originator, underwriter and servicer of multifamily commercial real estate loans
2012  Acquires ING DIRECT, HSBC US Card portfolio
2010  Enters into card partnerships with Kohl's and Sony in the US and Hudson's Bay Company and Delta in Canada
2009  Acquires Chevy Chase Bank in the Washington, DC area
2006  Acquires North Fork Bank, one of the largest banks in the New York metro area
2005  Acquires Hibernia National Bank, #1 bank in Louisiana
2002  Launches its Small Business credit card
2000  Introduces slogan, “What’s in your wallet?”
1998  Enters Auto Finance Market
1996  Expands into Canada and the U.K.
1995  Spins off from Signet Bank
1994  Initial Public Offering (IPO)
Background

• Machine learning is gaining adoption in many industries

• Microservices are independently deployable services that reduce coordination

• Reactive Architecture enables asynchronous processing

• Kafka is a fast distributed streaming platform that helps decouple your services

• Akka is a powerful framework that can be used to bring all of these together
A monolithic application can be challenging ...

A monolithic application puts all its functionality into a single process…

... and scales replicating the monolith on multiple servers

Monoliths can have many dependencies

Monoliths are also hard to change

*This illustration is from http://martinfowler.com/articles/microservices.html
A microservices architecture puts each element of functionality into a separate service...

... and scales by distributing these services across servers, replicating as needed.

This illustration is from http://martinfowler.com/articles/microservices.html

Microservices to the rescue! But are they the silver bullet?

Microservices are:

- **Small**: A microservice is very targeted in functionality and scope. Its codebase is relatively small and manageable.
- **Loosely Coupled**: Passing messages between services via appropriate protocol, microservices decouple resources from the underlying technologies.
- **Continuously Deployed**: Microservices require good DevOps process, automation, acceptance and testing.
- **Disposable**: The systems microservices may be long-lived, but the microservices themselves may be short-lived.

"Organizations often want to frequently roll out updates, even multiple times a day. Consequently, it’s no longer adequate to develop simple, monolithic web applications that serve up HTML to desktop browsers."

Chris Richardson, microservices.io
Using a reactive architecture with microservices can help achieve additional benefits.
The Reactive Manifesto highlights the key principles of a reactive architecture

Responsive

Elastic

Resilient

Message Driven

http://www.reactivemanifesto.org/
A reactive architecture has both benefits and tradeoffs

**BENEFITS**

- **Better resource utilization, saving cost**
  - Can get higher efficiency out of CPUs (multi-core processors), doing more with less

- **More Agile**
  - Decoupling enables services to be updated independently

- **Faster response times as requests can run in parallel**
  - Back pressure can be used for flow control
    - Fast producers don’t overwhelm slower consumers
    - Enables a consumer to control queue bounds

- **Extensible**
  - New components can be added that listen to the event stream without re-writing the system

**TRADEOFFS**

- Async programming is a mind shift
- **Complexity**
  - The flow of the system is shifted from a central place to distributed services
Akka is built off the actor model, which originated in 1973 per Carl Hewitt)

- The unit of execution is the Actor and your microservices are built as actors.
- An actor is lightweight and there can be several million actors per GB of heap memory.
- The actor is an object that encapsulates state and behavior, and communicates exclusively by exchanging messages which are placed into the recipient’s mailbox.
Credit Card Fulfillment Process with Akka
Four categories of tools you can use to build reactive apps

<table>
<thead>
<tr>
<th>Java script libraries:</th>
<th>Languages that support reactive models natively:</th>
<th>Reactive Layers that run on top of the JDK &amp; implement the Reactive-Streams spec:</th>
<th>Reactive Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AngularJS</td>
<td>Scala</td>
<td>PROJECT REACTOR</td>
<td>ReactiveX</td>
</tr>
<tr>
<td>React</td>
<td>Clojure</td>
<td>Ratpack</td>
<td>Java: RxJava</td>
</tr>
<tr>
<td>Vue.js</td>
<td>Java</td>
<td>akka</td>
<td>JavaScript: RxJS</td>
</tr>
<tr>
<td>node</td>
<td>gotlang</td>
<td>VERT.X</td>
<td>C#: Rx.NET</td>
</tr>
<tr>
<td>Ractive.js</td>
<td>spring</td>
<td></td>
<td>C#(Unity): UniRx</td>
</tr>
<tr>
<td></td>
<td>Framework 5.0</td>
<td></td>
<td>Scala: RxScala</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Clojure: RxClojure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C++: RxCpp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Lua: RxLua</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ruby: Rx.rb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ReactiveX for platforms and frameworks</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RxNetty</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RxAndroid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RxCocoa</td>
</tr>
</tbody>
</table>

Languages that support reactive models natively:

- Java
- JavaScript
- Go
- Groovy
- Go
- ReactiveX for platforms and frameworks
Kafka is a distributed streaming platform

- Used in conjunction with Zookeeper
- Runs as a cluster
- Records are stored in categories called topics
- Provides 4 core APIs: Producer, Consumer, Streams and Connector
- Supports both publish-subscribe and queuing through a consumer group concept
- Very Fast and has very high throughput – many use it for backpressure
- Can be used for message replay as the messages do not have destructive reads like traditional messaging technologies
- Guarantees order of messages within a partition, but not across partitions
- Very easy to get up and running

http://kafka.apache.org/documentation.html
Machine Learning is a type of Artificial Intelligence

Artificial Intelligence
- Scripted Chatbots
- Medical Diagnosis Expert Systems

Machine Learning
- Product Recommendation Engines (Netflix, Amazon, etc)
- Spam Filtering
- Adaptive Pricing Systems

Deep Learning
- Image Recognition
- Self-Driving Cars
- Siri / Alexa / Google Home / Cortana

Artificial Intelligence: Systems able to perform tasks that normally require humans. E.g., If/then logic
Machine Learning: A subcategory of AI that provides ability to automatically train the system. E.g., Regression
Deep Learning: MLs with multiple layers that mimic layers of neurons in the brain. E.g., Deep Neural Networks
There are two major classes of Machine Learning

**Supervised**
A human labels the data used
Labor intensive, but more interpretable

**Unsupervised**
An algorithm discovers relationships
Easier, but less interpretable
H2O Overview

- Open source (Apache license) in-memory big data machine learning platform
- Created in 2011 and the core code is written in Java
- Can be used with Python, R, H2O Flow, Scala, Tableau, Spotfire
- Uses parallelized and distributed algorithms like GLM, Random Forest, GBM, PCA, deep learning neural networks
- Also supports supervised and unsupervised learning
- Deploy model as a Java POJO or MOJO (Model ObJect, Optimized)
Reactive Machine Learning Integration Patterns with Kafka

### Integration of Multiple Actor Systems

- Build some of the microservices as Akka actors and leverage Kafka to integrate the non-Akka microservices.
- Leverages a hybrid of internal Akka messaging and external via Kafka.

### Integration of Single Actor System

- Build the microservices as Akka actors in a single Actor System and publish to Kafka for other Akka Systems to consume.
- Leverages Akka’s built-in messaging mechanism for communication.
- Leverages Kafka to publish to other Actor Systems.
Combining all of these technologies & patterns together can create a powerful solution

- Saga Akka Actor for coordinating a reactive workflow
- Leverage Kafka as the messaging mechanism between the microservices
- Leverage Lagom to bootstrap Kafka
- H20 for machine learning
- Leverage Java for non Akka microservices
Demo of use case and Machine Learning components
Fraudulent Transaction Use Case

Kafka Producer -> Akka Saga Actor: Calc Features

Akka Saga Actor -> Kafka card.transaction topic: Calc Features

Kafka card.transaction topic -> Java CalcFeaturesMS: Features Calculated

Java CalcFeaturesMS -> Akka ProcessAppActor: Features Calculated

Transaction OK or Fraudulent Transaction

Calc Features

Features Calculated
For the machine learning components, we are using a Credit Card Fraud Detection dataset from kaggle.com

- Contains 284807 total rows, 492 which are fraud
- V1-V28 are unidentifiable numeric features along with Time
- The time column contains the seconds elapsed between each transaction and the first transaction in the dataset.
- Amount is the transaction amount
- Class column is 1 for fraud, 0 for not fraud
To build the model we used H2O Flow

- H2O Flow is an open-source user interface for building H2O models
- Very easy to get up and running. Basically download the zip file and run the jar file
  
  unzip h2o-3.16.0.2.zip
  
  cd h2o-3.16.0.2
  
  java -jar h2o.jar
- Navigate to http://localhost:54321
H20 Flow enables you to do multiple trial runs with various algorithms

- Important that the training and validation frame are not the same dataset
- Response_column is important as this is the data element you are trying to predict
Distributed Random Forest performed better than Gradient Boosting Machine (GBM)

GBM:

Distributed Random Forest:

- Both Random Forest and GBM are machine learning techniques for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.
- Random Forest uses deep decision trees, while GBM uses shallower decision trees.
H20 identified which of the features were most important and revised the model until it was accurate.
We deployed the H20 model as a POJO

- Install H20, java (versions 7 or 8)
- Download the kaggle dataset, creditcard.csv
- Build the model, selecting the response_column
- Download the java generated code for the model and the packed library from the H20 instance
  curl -o h2o-genmodel.jar http://192.168.1.6:54321/3/h2o-genmodel.jar
- Compile the java generated code
  javac -verbose -cp h2o-genmodel.jar -J-Xmx2g -J-XX:MaxPermSize=128m
gbm_68aa4e_9808_48dc_b08d_383c94323392.java
- Create a main.java to invoke the model with features
  javac --cp h2o-genmodel.jar -J-Xmx2g -J-XX:MaxPermSize=128m
gbm_68aa4e_9808_48dc_b08d_383c94323392.java main.java
- Execute the model
  java -cp ;h2o-genmodel.jar main
- For our demo, we integrated main.java into the RunModelMS Kafka producer/consumer
Live Demo!
Capital One Tech Blogs on Medium.com

• Microservices: When to react vs. orchestrate
  https://medium.com/capital-one-developers/microservices-when-to-react-vs-orchestrate-c6b18308a14c

• A Reactive Framework Comparison
  https://medium.com/capital-one-developers/building-microservices-a-reactive-framework-comparison-fb49d8f3c8f4

• Comparing and Contrasting Open Source BPM Products

• Using Machine Learning and Open Source BPM in a Reactive Microservices Architecture
  https://github.com/andy9876/MachineLearningReactiveBPM
Questions
Lessons learned of this approach

• Need a Unique ID (correlation ID) that goes across all microservices

• For the machine learning dataset, use big data and unbiased data

• Leverage Chaos testing to validate resiliency

• Apply this pattern where:
  – there are synchronous blocks of asynchronous processing
  – there is a need to decouple as much as possible to eliminate dependencies
Human Workflow Machine Learning Integration Patterns with Kafka

**Human Workflow:**

- Plug into the event stream as another reactive microservice
- Integrate directly into BPM Suite as a workitem handler
Correspondence Use Case with Akka and Cassandra

Capabilities
1. Service Orchestration
2. Event History
3. Automatic API Call Retry
4. Real Time Status Check
5. Reporting
6. Restart Service Call From Last Failure
7. CQRS Pattern
8. Reporting Capability
## Key technology components leveraged

<table>
<thead>
<tr>
<th>Lagom</th>
<th>Akka</th>
</tr>
</thead>
</table>
| • Exposes Akka Actor as a REST API, accelerating development | • Making async calls  
• Using a reactive hybrid approach – Saga Actor handles fanning & merging |

<table>
<thead>
<tr>
<th>Java Future Interface</th>
<th>Akka Clustering Sharding</th>
</tr>
</thead>
</table>
| • Enables REST API calls to be async and not wait for the response | • One node per AZ, 3 AZs total  
• Used in combination with ELB (round robin original request to a node)  
• Location transparency – actors can exist in any of the AZs  
  - Leverage cluster sharding - actors are referenced by an ID instead of an ActorRef  
• Enables actors to scale and get automatically restarted if they die  
• Cluster sharding is using Akka Distributed Data  
  - Shares data between Akka Cluster Nodes using a key-value store |

### Constraints

**Throttling** - Each Actor can only call an API at a certain rate  
**Batch to real-time** - Batch file triggers the process, which has ~50k records in it daily
Advanced features using Lagom

- Maximized development speed.
- Created couple of POC with Lagom within short time span
- REST end point developed using Lagom
- With Lagom developed CQRS & Event Sourcing pattern
Lessons Learned

• Learning curve

• Unit testing and Performance Test metrics capturing was difficult

• Using throttling was limiting Akka performance

• Tight timeline caused few features to be dropped
  – Akka persistence - Recovery of objects during failure
  – Akka clustering - Providing resiliency and load balancing