Intelligent Commerce

Maxim Noah Khailo

December 20, 2016

1 Smarter, Faster, Simple

Fox Commerce can bring something innovative making the first intelligent commerce system. We can do better than the competition by bring intelligence into the core part of our platform.

To do this, our platform needs to be built with the following the three principles

1. Answer the Questions our Customers Care About.

2. Impact of Changes is Known Immediately.

3. Platform Makes Positive Changes Automatically.

1.1 Answer Questions Customers Care About

We need to enable all users of our system to make intelligent decisions.

They should not have to learn a data processing and analytics tool to answer their most important questions.

Their questions should be answered with beautiful UI at all the right times. No query builders, no filtering, no PHd necessary.

1.2 Impact of Changes Known Immediately

Our platform knows when our users make changes via activities. We should be able to tell our users the impact they to the questions they care about.

Users should not have to analyze data to understand if a change they made created an impact. The platform knows when things change and how it is used. If something has an impact, positive or negative, we can tell the user as soon as the platform knows.

1.3 Platform Makes Positive Changes Automatically

Our Customers should not have to guess which changes make a positive impact to their user's experience. They should not have to set up A/B tests and run laborious experiments. The system should automatically make and find the positive changes and let our customer know what they are.

Fox Commerce supports variations in all data via our idea of context. We can utilize this powerful concept to automatically make changes to any object in our object store and run experiments. We can use intelligent algorithms to converge on positive changes and let the user know what they are.

The user should always feel in control in making decisions permanent.

2 Overview

We need to build several foundational pieces based on the principles above.

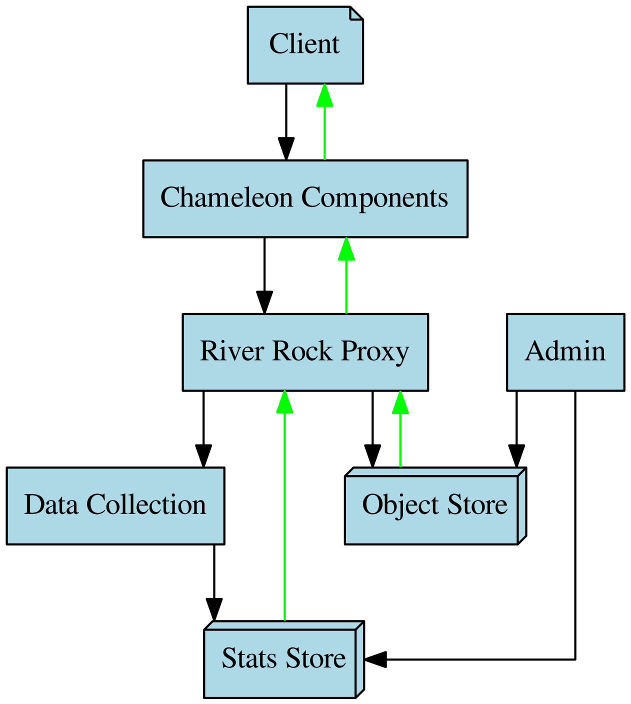
2.1 What We Have Already Built

- Object Variation via Context
- Event Sourcing
- Activities
- Search

2.2 What We Have to Build

- Data Collection Pipeline
- Stats Store
- Counting and Aggregating Consumers
- Channels
- Oracles
- CMS/Object Store
- River Rock Proxy
- Chameleon Components

2.3 High Level Architecture



2.4 Data Collection Pipeline

We need to dump everything into kafka and write consumers that process logs and activities and count events we care about. Take the counts and dump them into a time series DB. Henhouse was written for this purpose.

2.4.1 Counting and Aggregating Consumers

With all data in kafka, we can write consumers to count events and put them into the stats store.

2.4.2 Stats Store

Henhouse is a timeseries DB that can answer count and rate questions over any time period in constant time.

We can combine Henhouse and ES to create a statistics store for data collected.

2.5 Channels

Channels allow us to understand where a customer is coming from and what they see. Channels are foundational to support multiple store fronts and selling product in multiple places. We can tie a channel to a set of contexts allowing to build the bandit proxy.

2.6 Oracles

These are kafka consumers that process events and use the stats store to decide whether something important has happened. For example, predicting when a product will go out of stock, or when something is trending. Whether an AD campaign had an impact.

These answer questions customers and what impact changes have.

2.7 CMS/Object Store

We currently have an object store driving most PIM and MERCH data. However, we need to remove the constraints of certain data types and allow any kind of data to be stored in a flexible and generic way, including drive our CMS.

2.8 River Rock Proxy

This is a proxy that uses the Stats Store to decide which context to show the consumer for a particular channel. This allows our system to make and track intelligent changes. It use a particular version of Markov Decision Process called the Multi Armed Bandit algorithm. It is called the River Rock Proxy because it uses green river to polish the output to a channel like a river rock.

2.9 Chameleon Components

These are UI components that look different given a context. The River Rock Proxy will decide Context and the UI can adjust depending on what is chosen.