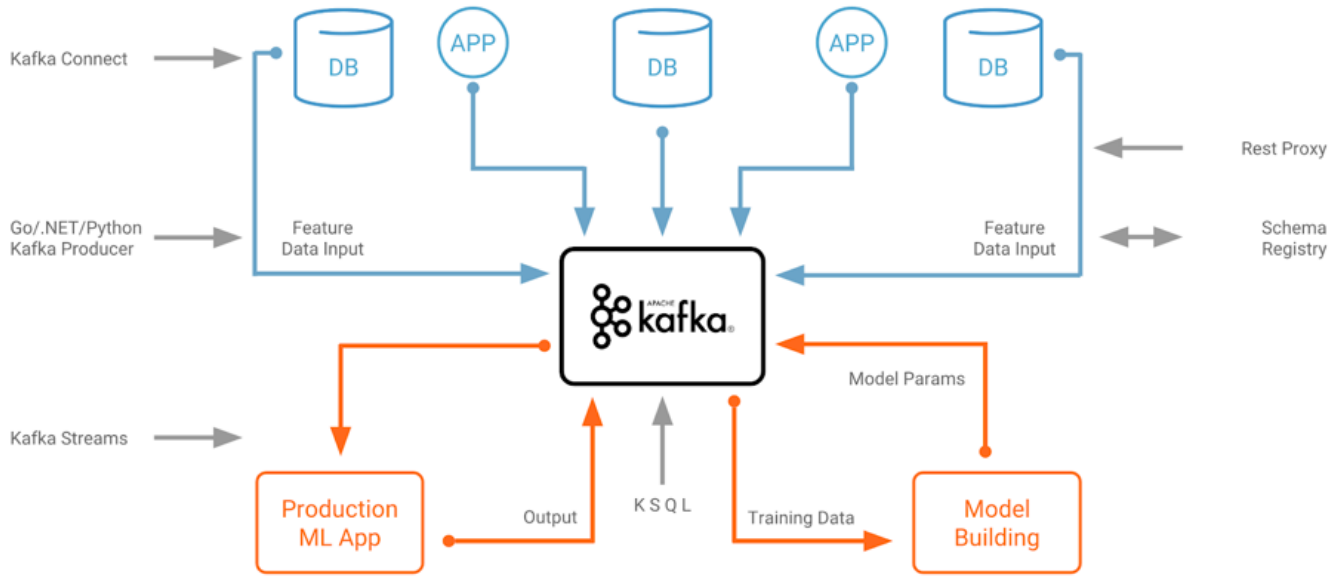
confluent

STREAM PROCESSING

Using Apache Kafka to Drive Cutting-Edge Machine Learning

Author:Kai Waehner
November 19, 2018

Machine learning and the Apache Kafka® ecosystem are a great combination for training and deploying analytic models at scale. I had previously discussed potential use cases and architectures for machine learning in mission-critical, real-time applications that leverage the Apache Kafka ecosystem as a scalable and reliable central nervous system for your data. Let's recap the architecture:



We'll expand on that point of view in this blog post by discussing new trends and innovations around Apache Kafka and machine learning, and how they are related. Let's take a look!

Overview

- Embedded model deployment
    - Model deployment in Kafka applications

- Hybrid cloud and on-prem architectures
    - Building hybrid cloud architectures with Kafka
- Real-time streaming analytics
    - KSQL and machine learning for preprocessing and model deployment
- Simplify building analytic models with AutoML
    - AutoML and the Apache Kafka ecosystem
- Scalable and flexible machine learning platforms
    - Apache Kafka ecosystem for machine learning platforms
- Apache Kafka as a key to flexible and future-ready machine learning

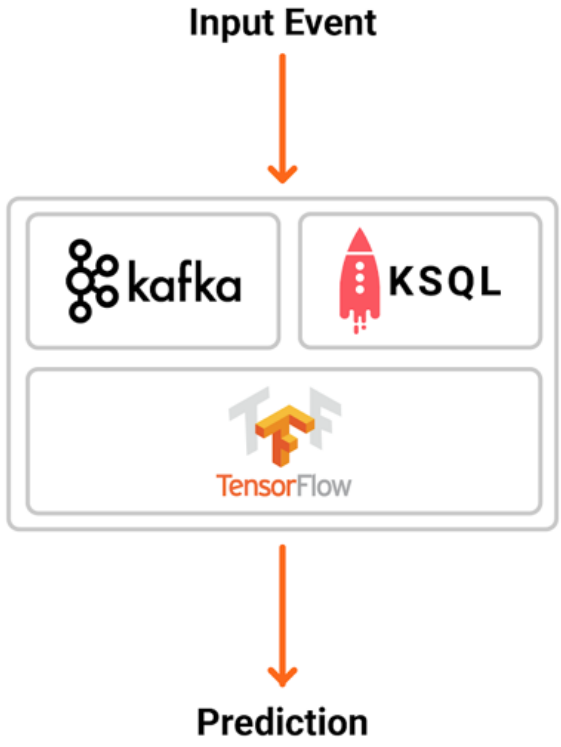
Embedded model deployment

On a high level, a machine learning lifecycle contains of two different parts:

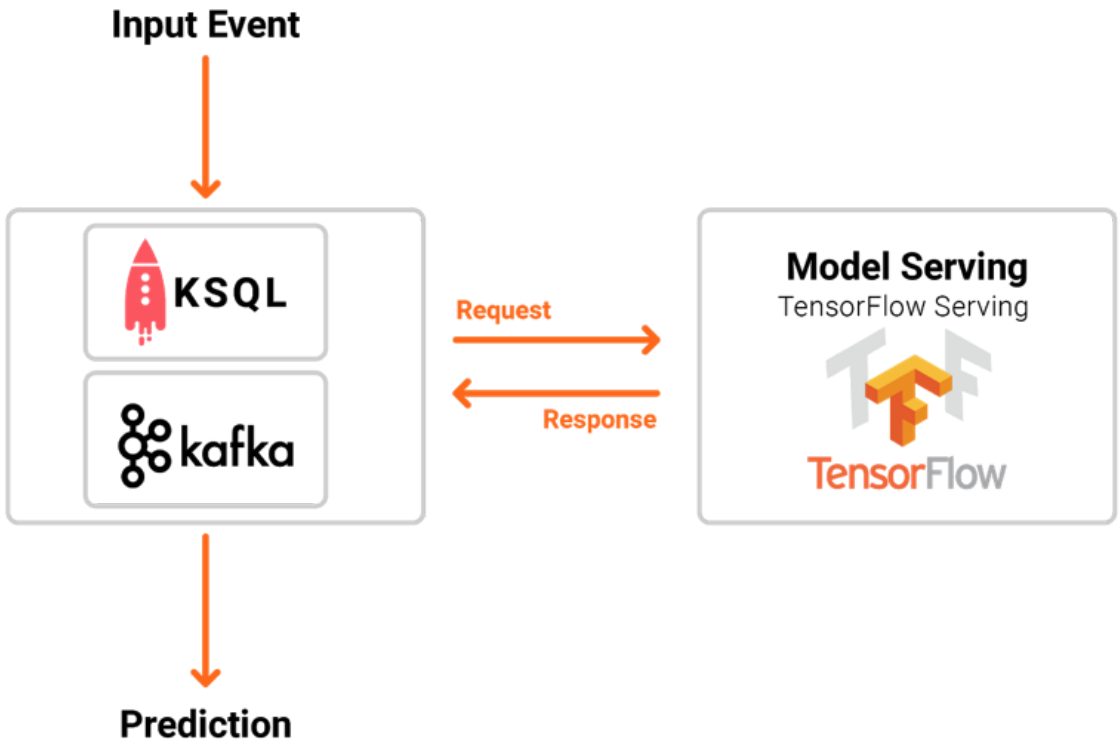
- Model training: In this step, we feed historical data into an algorithm to learn patterns from the past. The result is an analytic model.
- Generating predictions: In this step, we use an analytic model for making predictions on new events based on the learned pattern.

Machine learning is a continuous process, where we repeatedly improve and redeploy the analytic model over time.

Predictions can be performed in different ways within an application or microservice. One way is to embed an analytic model directly into a stream processing application, like an application that uses Kafka Streams. You could, for example, use the TensorFlow for Java API to load and apply models:



Alternatively, you could deploy the analytic models to a dedicated model server (like TensorFlow Serving), and use RPCs from the streaming application to the service (e.g., with HTTP or gRPC):



Both options have various tradeoffs.

Pros of a dedicated model server:

- Simple integration with existing technologies and organizational processes
- Easier to understand if you come from the non-streaming world
- Later migration to real streaming is also possible
- Model management built-in for deployment of different models including versioning, A/B testing, etc.

Cons of a dedicated model server:

- Often tied to specific machine learning technologies
- Often tied to a specific cloud provider and introduces lock-in
- Higher latency
- More complex security setups (remote communication through firewalls and authorization management)
- No offline inference (devices, edge processing, etc.)
- Couples the availability, scalability and latency/throughput of your stream processing application with the SLAs of the RPC interface
- Side effects (e.g., in case of failure) not covered by Kafka processing (e.g., exactly once)

Whether or not to go with a model serving infrastructure is a question that often comes up for various reasons, including latency and security considerations. Some of the trends discussed below like hybrid architectures or low latency requirements also force you to think about model deployment, and whether you should deploy models locally at the edge to process personally identifiable information (PII), or if you want to integrate with external AutoML services.

These two options and their tradeoffs are very important to understand so you can pick what is best for your use case and architecture.

Model deployment in Kafka applications

Kafka applications are event based, and leverage stream processing to continuously process input data. If you're using Kafka, then you can embed an analytic model natively in a Kafka Streams or KSQL application. There are various examples of Kafka Streams microservices embedding models built with TensorFlow, H2O or Deeplearning4j natively.

It is not always possible or feasible to embed analytic models directly due to architectural, security or organizational reasons. You can also choose to use RPC to perform model inference from your Kafka application (bearing in mind the the

pros and cons discussed above). You can visit my project for an example of gRPC integration between a Kafka Streams microservice and locally hosted TensorFlow Serving container for making predictions with a hosted TensorFlow model.

Hybrid cloud and on-prem architectures

Hybrid cloud architectures are very common for machine learning infrastructure. Training can happen with large sets of historical data in the public cloud or in a central data lake within your own datacenter. Model inference can be done anywhere.

In many scenarios, it totally makes sense to leverage the extreme scale and elasticity of public clouds. For example, you can spin up new large computing instances to train a neural network for a few days, then stop the instances. *Pay-as-you-go* is a perfect model, especially for deep learning.

In the cloud you can leverage specific processing units which would be expensive and often idle in your own datacenter. For example, Google's TPU (Tensor Processing Unit)—an application-specific integrated circuit (ASIC) designed from the ground up for machine learning—is one specific processor built just for deep

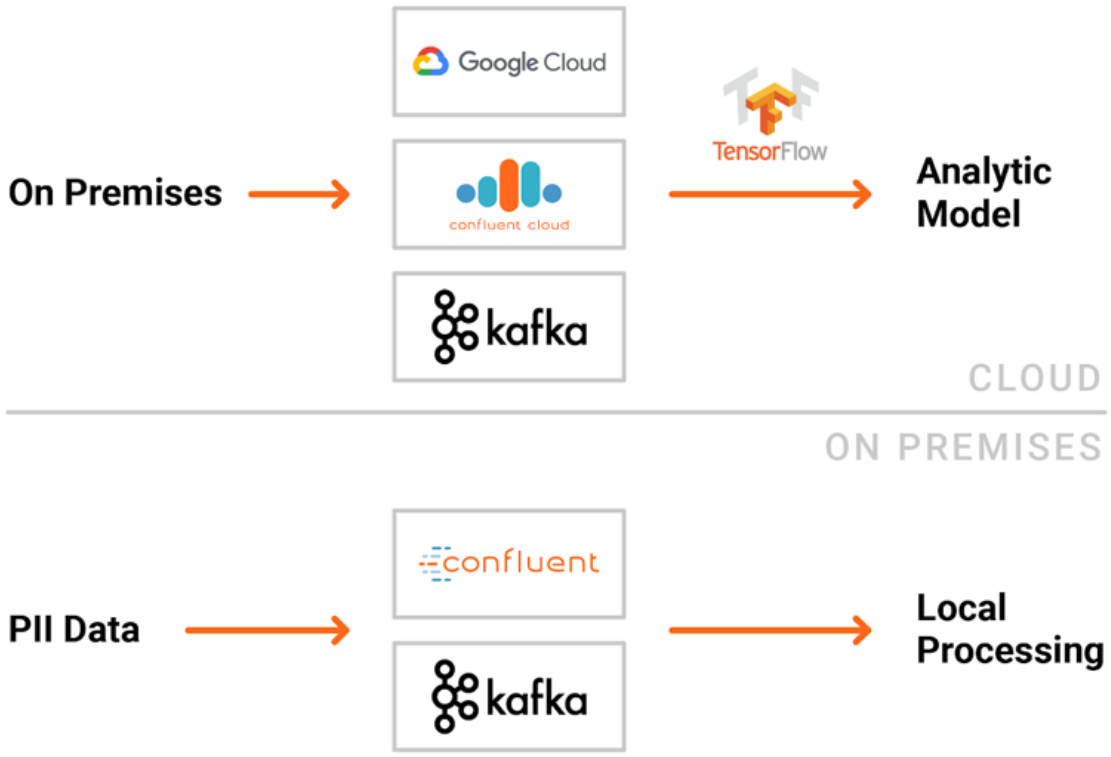
learning. TPUs do one thing well: matrix multiplication—the heart of deep learning—to train neural networks. Compare this to CPUs or GPUs which are used in a broader landscape.

If you need to keep data out of the public cloud, or if you want to build your own analytic infrastructure for bigger teams or departments, you might buy purpose-built hardware/software combinations for deep learning, such as Nvidia® DGX™ platforms for your own datacenter

Wherever you need it, model inference can be performed independent of the model training: in the public cloud, on prem in your data centers or on edge devices like Internet of Things (IoT) or mobile devices. Edge devices often have higher latency, limited bandwidth or poor connectivity.

Building hybrid cloud architectures with Kafka

Apache Kafka enables you to build cloud-independent infrastructure—either multi-cloud or hybrid architectures. You are not bound to specific cloud APIs or proprietary products. Here is an example of a hybrid Kafka infrastructure for training and deploying analytic models:



You can use Apache Kafka components like Kafka Connect for data ingestion and Kafka Streams or KSQL for the preprocessing of data. Model inference can also be done within a Kafka client including KSQL. The whole monitoring of the analytic infrastructure is often done with Apache Kafka, too. This includes technical metrics like latency and business information like model accuracy.

Often the data streams come from other data centers or clouds. A common scenario is to use a Kafka replication tool like MirrorMaker or Confluent Replicator to replicate the data from source Kafka clusters to the analytics environment in a reliable and scalable way.

Setup, configuration and operations of a reliable and scalable Kafka cluster requires a solid understanding of distributed systems and experience in running them.

As an alternative you can use a cloud service like Confluent Cloud, which provides Kafka as a service. You then build the Kafka client (e.g., Kafka's Java API, Kafka Streams or KSQL) and use the Kafka server side as an API. In your own data centers, tools like Confluent Operator and Confluent Control Center help to monitor, operate and manage your Kafka clusters.

Unleashing Apache Kafka and TensorFlow in the Cloud is a helpful resource for learning more about hybrid machine learning infrastructures leveraging the Apache Kafka ecosystem, and Confluent Cloud on GCP and AWS.

Real-time streaming analytics

A primary requirement for many use cases is to process information whilst it is still relevant. It is important for all parts of a machine learning infrastructure:

- (Re-)Training analytic models with current information
- Predictions on new events in real time (often within milliseconds or seconds)
- Monitoring the whole infrastructure for model accuracy, infrastructure errors and other details
- Security-related tracking information like access control, auditing or provenance

The easiest and most reliable way to process data in a timely manner requires building real-time streaming analytics natively on top of Apache Kafka using Kafka clients such as Java, .NET, Go or Python. As well as using a Kafka client, you should look at using Kafka Streams, the stream processing runtime for Apache Kafka. This is a Java library that enables you to perform simple (and complex) stream processing within your Java applications.

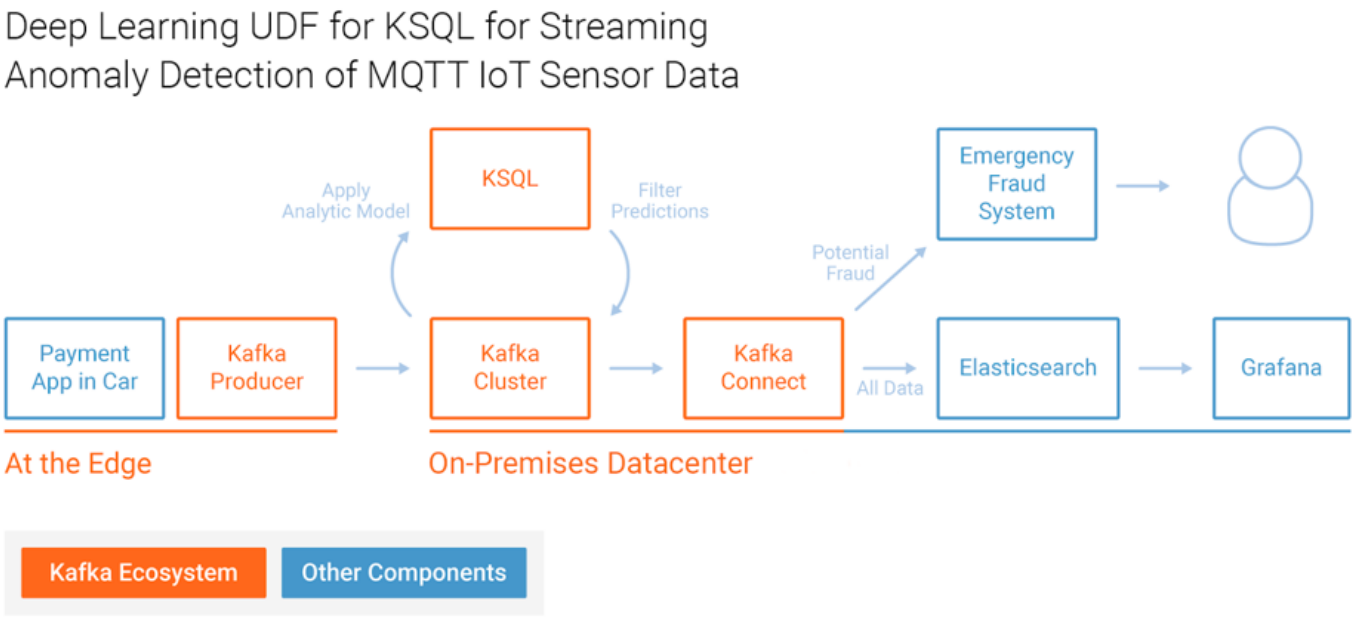
If you don't want to write Java or Scala, then there is also KSQL. KSQL is the streaming SQL engine for Apache Kafka. With KSQL you can write stream processing applications expressed in SQL. Read Neha Narkhede's blog post Introducing KSQL: Streaming SQL for Apache Kafka for more details. To get an idea of things that you can do with KSQL, have a look at the KSQL recipes, and our KSQL blogs.

KSQL and machine learning for preprocessing and model deployment

Processing streaming data with KSQL makes data preparation for machine learning both easy and scalable. You can write SQL statements to do filtering, enrichments, transformations, feature engineering or other tasks. Here is just one example to filter sensor data from multiple types of cars for a specific car model for further processing or analytics:

```
CREATE STREAM car_sensor AS
   SELECT car_id, event_id, car_model_id, sensor_input
   FROM car_sensor c
   LEFT JOIN car_models m ON c.car_model_id = m.car_model_id
   WHERE m.car_model_type ='Audi_A8';
```

Machine learning models are easily embedded in KSQL by building a user-defined function (UDF). I created the example Deep Learning UDF for KSQL for Streaming Anomaly Detection of MQTT IoT Sensor Data, where I apply a neural network—more precisely an unsupervised autoencoder—for sensor analytics to detect anomalies:



In this example, KSQL continuously processes millions of events from connected cars via MQTT integration to the Kafka cluster. MQTT is a publish / subscribe messaging protocol built for constrained devices and unreliable networks. It is often used in combination with Apache Kafka to integrate IoT devices with the rest of the enterprise. The autoencoder is used for predictive maintenance.

Real-time analytics combined with this car sensor data allows you to send anomalies to a warning or emergency system to act before the engine breaks. Other use cases for intelligent connected cars worth noting include optimized

routings and logistics, selling new features for a better digital car experience and loyalty services that integrate with restaurants and other shops on the streets.

While a KSQL UDF requires writing a bit of code, this has to be done only once by the developer. Afterwards, the end user can simply use the UDF within their KSQL statements like any built-in function. Here is the KSQL query from this example, using the `ANOMALY` UDF, which applies the TensorFlow model under the hood:

```
SELECT car_id, event_id, ANOMALY(sensor_input) FROM car_sensor;
```

Both KSQL and Kafka Streams, depending on your preference and requirements, are a perfect fit for machine learning infrastructures when it comes to preprocessing streaming data and doing model inference. KSQL lowers the entry barrier and allows you to realize streaming applications with simple SQL statements instead of writing source code.

Simplify building analytic models with AutoML

AutoML, a recent development in machine learning, allows you to automatically build different machine learning algorithms with various hyperparameter settings and features to choose and deploy the best analytic model. With AutoML, you can build analytic models without any knowledge about machine learning itself.

AutoML uses different implementations of algorithms like decision trees, clustering or neural networks to build and compare different models out of the box. You just upload or connect your historical datasets and click a few buttons to start the process.

AutoML is not perfect for every use case today, but you can easily improve many existing business processes without needing an expensive data scientist, which many organizations cannot afford. Of course, some experience and knowledge in machine learning or data processing is helpful. Even more important, however, is domain knowledge to interpret the data and models correctly.

DataRobot or Google's Cloud AutoML^{BETA} are two of several public cloud offerings in this space. H2O's AutoML is integrated into its open source machine learning framework, but they also offer a nice UI-focused commercial product called Driverless AI. I highly recommend spending at least 30 minutes to check out any AutoML tool, as it is really fascinating to see how AutoML tools evolve these days, even though AutoML is early on the maturity curve.

AutoML and the Apache Kafka ecosystem

One of the major advantages of using KSQL is that it enables you to process data and apply analytic models without writing complex source code in programming languages like Java. Instead, KSQL provides users with a SQL-like syntax in which to build stream processing applications. Together, KSQL and AutoML greatly simplify the process of building and deploying analytic models.

Most AutoML tools include a model server for deployment of their models. One way that you can access the analytic models is via a REST interface. As discussed in an earlier section, although an RPC call for making predictions is not a perfect solution for a scalable, event-driven architecture like Kafka, there is an alternative approach: several AutoML solutions allow you to export their generated models so that you can deploy them into your application.

AutoML in H2O's open source framework can be applied via several interfaces like R, Python, Scala and a web UI. Here is an example in R:

```
aml <- h2o.automl(x = x, y = y,
                  training_frame = train,
                  leaderboard_frame = test,
                  max_runtime_secs = 30)
```

It's similar to what you would do with H2O's Python API to build a logistic regression, decision tree or neural network in H2O. But this time, you use the higher level `automl` function, which automates many of the typical tasks. The result is generated Java code that you can easily embed into your Kafka Streams microservice, KSQL application (via a UDF) or any other Kafka client, as can be seen through TensorFlow or Deeplearning4j examples.

Scalable and flexible machine learning platforms

Tech giants are typically some years ahead of "traditional enterprises." They already built what you (have to) build today or tomorrow. Machine learning platforms are no exception.

The paper Hidden Technical Debt in Machine Learning Systems explains why building and deploying an analytic model is much more effort than just writing some machine learning code with technologies like Python and TensorFlow. You also need to take care of data collection, feature extraction, serving infrastructure, monitoring and other tasks—all by using a scalable and reliable infrastructure:

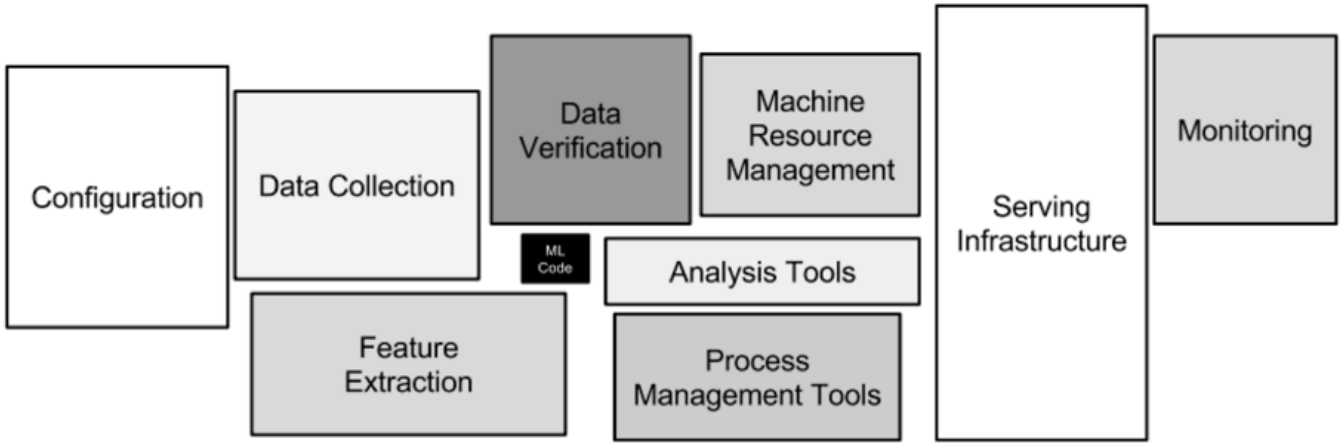


Figure 1. Hidden Technical Debt in Machine Learning Systems

In addition, tech giants show that one machine learning/deep learning framework is not sufficient for their use cases, and that machine learning is a fast growing space. A flexible machine learning architecture needs to support different technologies and frameworks. It also needs to scale well and be reliable if used for the most important business processes.

That's why many tech giants have built their own machine learning platforms, like Uber's Michelangelo, Netflix's Meson and Paypal's fraud detection platform. These machine learning platforms allow them to build and monitor powerful, scalable analytic models, but also stay flexible to choose the right machine learning technology for each use case.

Apache Kafka ecosystem for machine learning platforms

One of the reasons why Apache Kafka is so successful is due to its adoption by many tech giants.

Almost all the great Silicon Valley companies like LinkedIn, Netflix, Uber, eBay—*you name it*—blog and speak about their usage of Kafka as an event-driven central nervous system for their mission-critical applications. Many focus on the distributed event streaming platform for messaging, but we also see more and more adoption of components like Kafka Connect, Kafka Streams, REST Proxy, Schema Registry and KSQL.

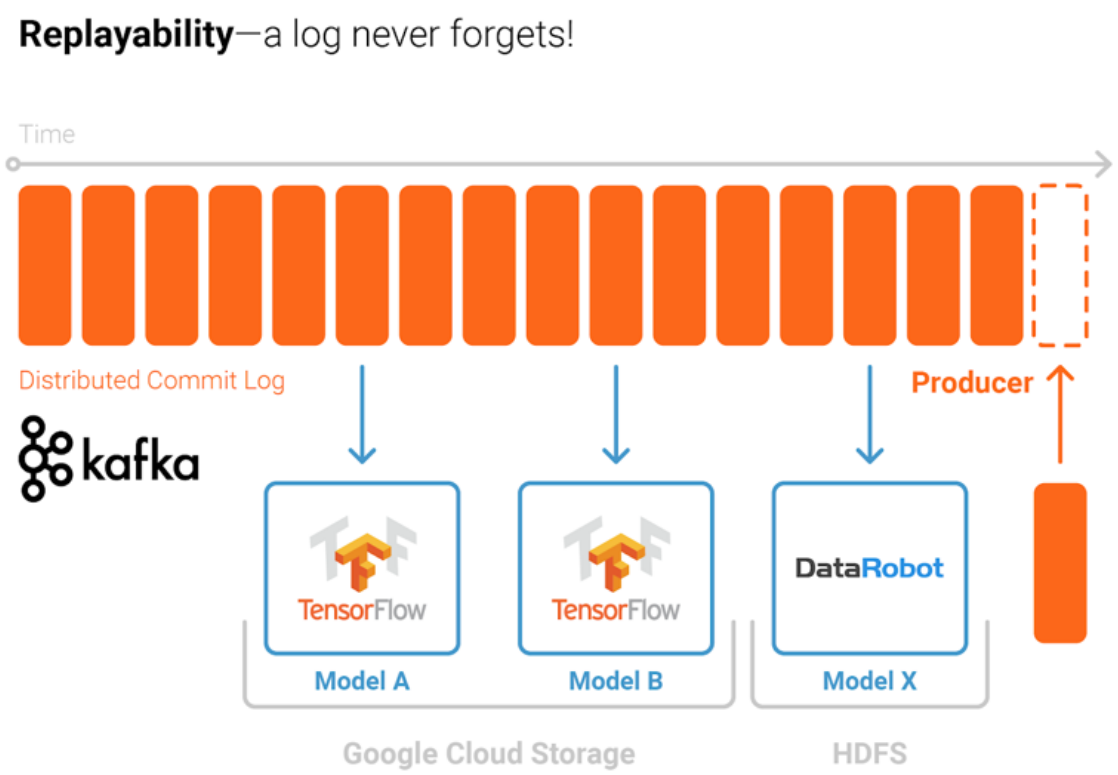
As discussed above in this post, Kafka is a logical fit for a machine learning platform. Training, monitoring, deployment, inference, configuration, A/B testing, etc. That's probably why Uber, Netflix and many others use Kafka already as a central component in their machine learning infrastructure, too.

Kafka allows easy, simple deployments of machine learning tasks without the need for another big data cluster, yet it is flexible in integrating with other systems. If you rely on your batch data processing in Hadoop or want to do machine learning processing with Spark, just connect it to Kafka via Kafka Connect.

For some real-world machine learning architectures, take a look at projects built with the Confluent Platform:

- Severstal, a manufacturing company, which uses Kafka and machine learning for predictive maintenance to reduce downtime with real-time streaming data
- Celmatix, a biotech company, uses Kafka and machine learning to make informed and proactive reproductive health decisions by leveraging real-time genomics data
- Centrica Hive, a company providing solutions and services for connected homes, leverages Kafka Streams and machine learning to create predictive heating failure alerts and to optimize schedule predictions

With the Kafka ecosystem as base of your machine learning infrastructure, you are not forced to use just one specific technology. One of the great design concepts of Kafka is that you can re-process data again and again from its distributed commit log:



You can use different technologies for training analytic models, deploy the models to any Kafka-native or external client applications, build a central monitoring and

auditing system and stay ready for future innovations in the machine learning space. All at scale, reliable and fault tolerant, as decoupled systems, loosely connected with Apache Kafka as the nervous system.

Apache Kafka as key to flexible and future-ready machine learning infrastructure

The Apache Kafka ecosystem is a perfect match for a machine learning architecture. Pick the components you need to build a scalable, reliable platform that is independent of a specific on-prem/cloud infrastructure or machine learning technology.

You can build an integration pipeline for model training and deploy the analytic models for real-time inference and monitoring within Kafka applications. This does not change with new trends like hybrid architectures or AutoML. Quite the opposite, in fact: With Kafka as the central but distributed and scalable layer, you stay flexible and ready for new technologies and concepts of the future.

Please let me know what you think. Do you already use Kafka in the machine learning space? What components in addition to Kafka do you use? Share your thoughts in a comment below.

And if you haven't already, be sure to download Confluent Platform, the leading distribution of Apache Kafka, to get started with KSQL.