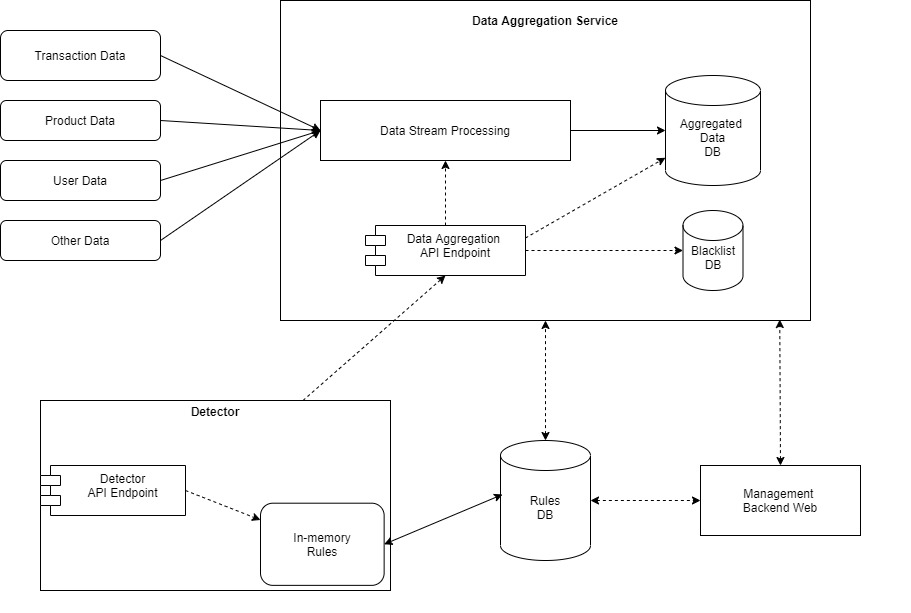
**RULE-BASED DETECTOR SYSTEM**

**Architecture Overview**



**Description**

**Data Aggregation Service**

Data from various sources such as Transaction, Product, User and changes history of them are streamed to data pipeline and processed, transformed to aggregated data. The aggregated data then stored to may be various kinds of databases, such as one for very fast query and one for storage purpose.

In this component, we can use Kafka and Kafka Stream to process and aggregate data such as number of transactions of a credit card in last 5 minutes… For applying dynamic logic of processing, we build the Kafka Stream processor capable of hot loading logic rules from Rule DB.

There is a database to store Blacklist data such as credit card, customer… Data of this may come from Stream Processor, Detector or manually input using Management Backend Web.

Data Aggregation Service exposes some gRPC API to the world. The API returns aggregated data needed for Fraud Detector. The data query should be very fast so we can use a high perforamce key-value database such as Redis.

With Kakfa Stream, the gRPC API can directly query data from Stream State, that should boost the performance of data reading.

**Detector**

This expose gRPC/Rest API for client to determine if a transaction is fraud or not. The handling flow when a client request comes is

* Detector queries high-performance Data Aggregation API to retrieve aggregated features related to the transaction.
* Use loaded fraud-checking rules in memory to run on transaction data and aggregated data for verifying if the transaction is fraud or go to go.
* Return the result to client.

The checking rules is hot-loading to memory for performance purpose. All the rules are stored in a database, such as MySQL and user can manage those rule using Management Backend Web.

**Rules**

We can business rules of fraud-checking are spread cross 2 modules: Data Stream Processing and Detector. The former lets the data process know how data shoud be aggregated for the input of the Detector. The later is logic to determine fraud transaction.

For hot deploy rules, we keep a timestamp of the last time fetching updated rules.

We build a feature of DryRun mode and Percentage-base running for each rule. So users can deploy a rule as DryRun mode to verify performance of the rule without any impact. Similarly, user can deploy the rule to apply for from 1% to 100% facts. This is very helpful for A/B testing if need.

**Machine Learning**

We may think machine learning is a better way than rule engine for fraud detecting. Fraud criminals are smart, variable and have many methods that can go out of scope that rules can prevent. Preventing fraud needs intelligent decision making and machine learning is a trend, no doubt one day we will put it into our system.

We will build a ML model or more that take features includes transaction data and aggregated data as input and predict a score of fraud risk as the output. The Detector will include the score into the fact then run rules for final result. We should consider the performance impact when using ML.

There is an idea of replacing rule base logic by ML models. That is interesting. But we have to make sure the ML model has capable of accurately preventing fraud transactions like rule-based solution. Can we train a ML model faster than fraud criminals? Is our training data accurate enough? Can we train a ML model “smart” enough to cover all complicated things that rules are covering?