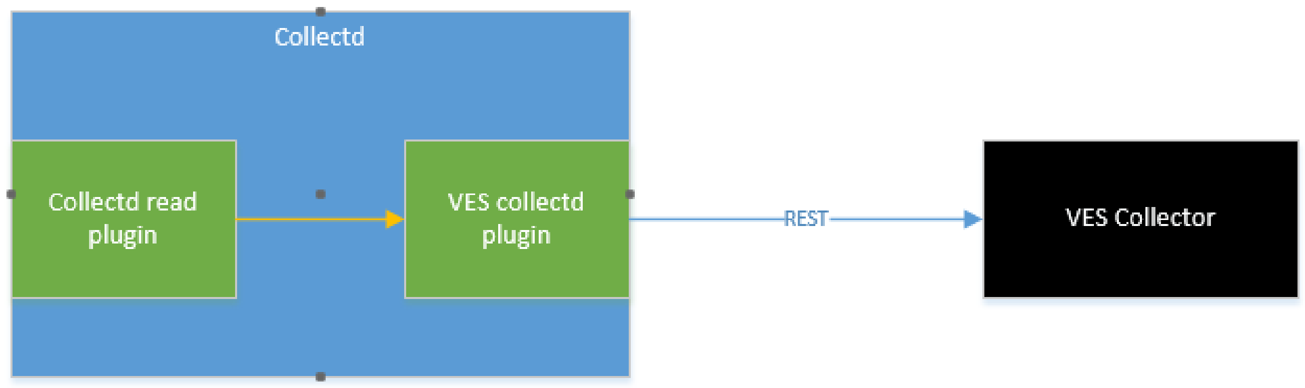
# VES plugin updates

## Old implementation

With the old collectd VES plugin implementation, the VES plugin:

1. Received telemetry directly from collectd.
2. Formatted a VES event with the received stats.
3. Submitted the stats directly to the VES collector



However this model forced an MIT license on the VES collectd plugin when we really needed it to be Apache v2 based. Also, everytime there was a VES schema update we had to make code changes to support the new schema.

## Updated implementation

The updated implementation to the collectd involves the following:

1. Separation of the VES plugin to a separate **application** so that it can be licensed appropriately.
2. Using a kafka messaging bus to transfer stats from collectd to the new application.
3. Using a YAML file to configure the mapping from collectd stats to VES events.

## Kafka Integration

The following section is from a readout from volodymyrx.mytnyk@intel.com on Kafka

### Kafka Overview

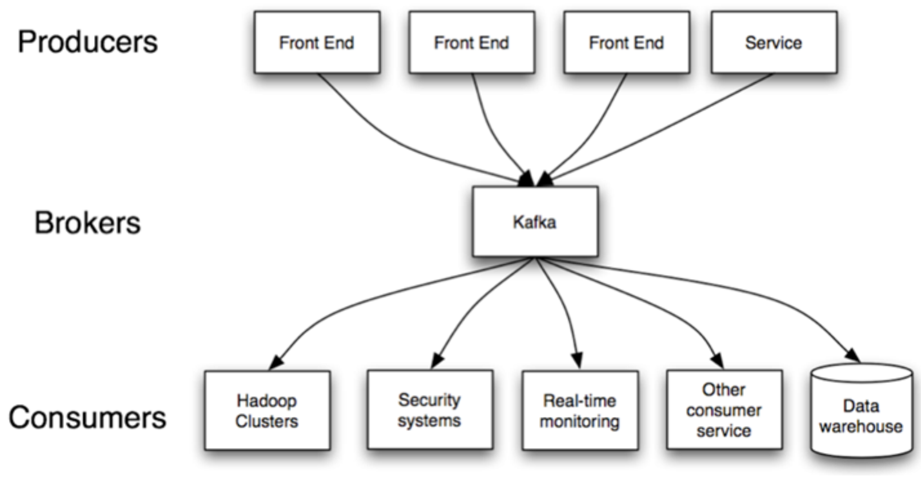
"Apache Kafka is a distributed commit log service that functions much like a publish/subscribe messaging system, but with better throughput, built-in partitioning, replication, and fault tolerance. Increasingly popular for log collection and stream processing" 0

Kafka core concepts:

"Kafka is run as a cluster on one or more servers.

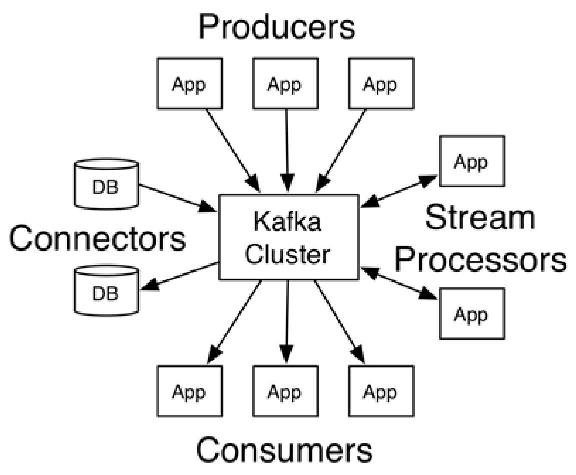
The Kafka cluster stores streams of *records* in categories called *topics*.

Each record consists of a key, a value, and a timestamp" 2.



- **"Producers** – consume the data feed and send it to Kafka for distribution to consumers". 1
- **"Consumers** – applications that subscribe to **topics**; for example, a custom application or any of the products listed at the bottom of this post". 1
- **"Brokers** – workers that take data from the producers and send it to the consumers. They handle replication as well". 1

## Kafka Use Cases

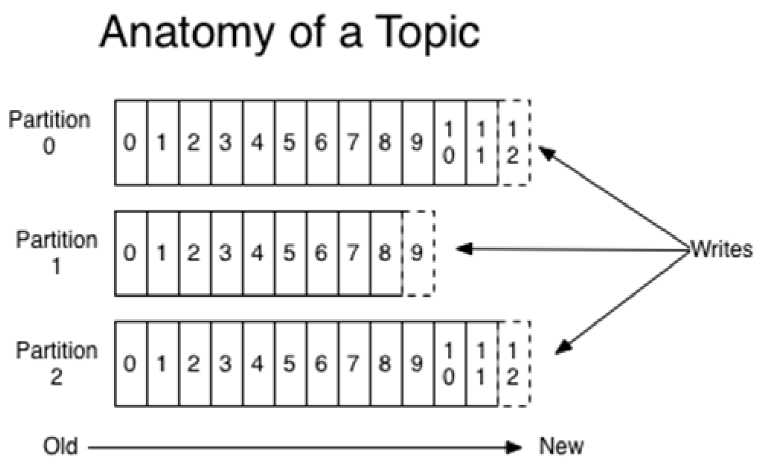


"Kafka has four core APIs:

- The Producer API allows an application to publish a stream of records to one or more Kafka topics.
- The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The Streams API allows an application to act as a *stream processor*, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

In Kafka the communication between the clients and the servers is done with a simple, high-performance, language agnostic TCP protocol. This protocol is versioned and maintains backwards compatibility with older version" 2.

## Topics and partitions



**"Topics** – categories for messages. They could be something like “apachelogs” or “clickstream””. 1 "Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it" 2.

**"Partitions** – the physical divisions of a topic, as shown in the graphic below. They are used for redundancy as partitions are spread over different storage servers" 1. "Each partition is an ordered, immutable sequence of records that is continually appended to—a structured commit log. The records in the partitions are each assigned a sequential id number called the *offset* that uniquely identifies each record within the partition" 2

"Producers publish data to the topics of their choice. The producer is responsible for choosing which record to assign to which partition within the topic. This can be done in a round-robin fashion simply to balance load or it can be done according to some semantic partition function (say based on some key in the record). More on the use of partitioning in a second" 2

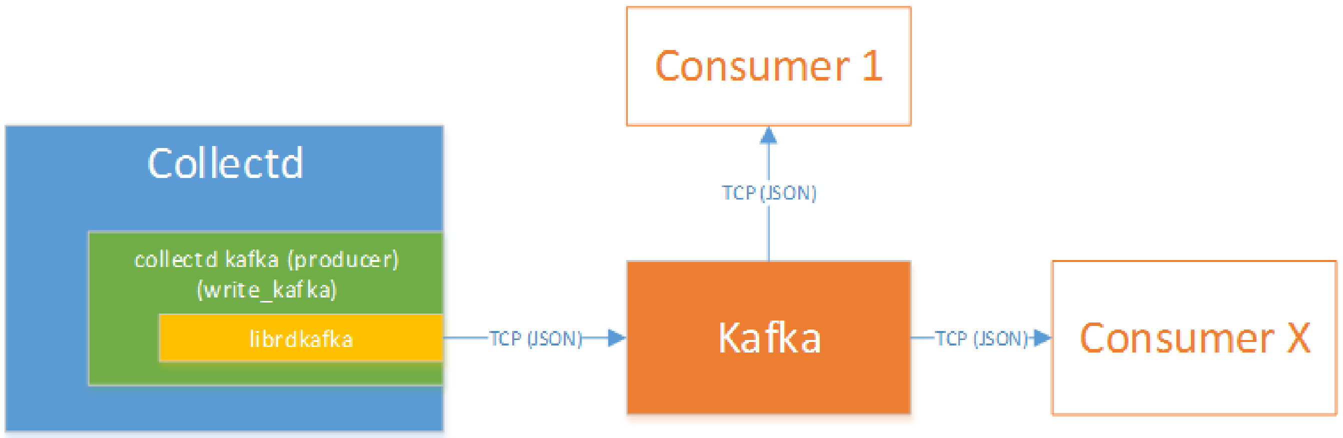
"The partitions of the log are distributed over the servers in the Kafka cluster with each server handling data and requests for a share of the partitions. Each partition is replicated across a configurable number of servers for fault tolerance. Each partition has one server which acts as the "leader" and zero or more servers which act as "followers". The leader handles all read and write requests for the partition while the followers passively replicate the leader. If the leader fails, one of the followers will automatically become the new leader. Each server acts as a leader for some of its partitions and a follower for others so load is well balanced within the cluster" 2.

"Consumers label themselves with a *consumer group* name, and each record published to a topic is delivered to one consumer instance within each subscribing consumer group. Consumer instances can be in separate processes or on separate machines. If all the consumer instances have the same consumer group, then the records will effectively be load balanced over the consumer instances. If all the consumer instances have different consumer groups, then each record will be broadcast to all the consumer processes" 2.

## Kafka Features

- Guaranties ordering within the partition.
- Stores all the records as a commit log (for configured interval time, etc.).
- Support partition replication (fault tolerance).
- Messages sent by a producer to a particular topic partition will be appended in the order they are sent.
- A consumer instance sees records in the order they are stored in the log.
- A consumer can reset to an older offset to reprocess data from the past or skip ahead to the most recent record and start consuming from "now".
- Balance messages between consumers in the group.
- **There is NO message prioritization support.**

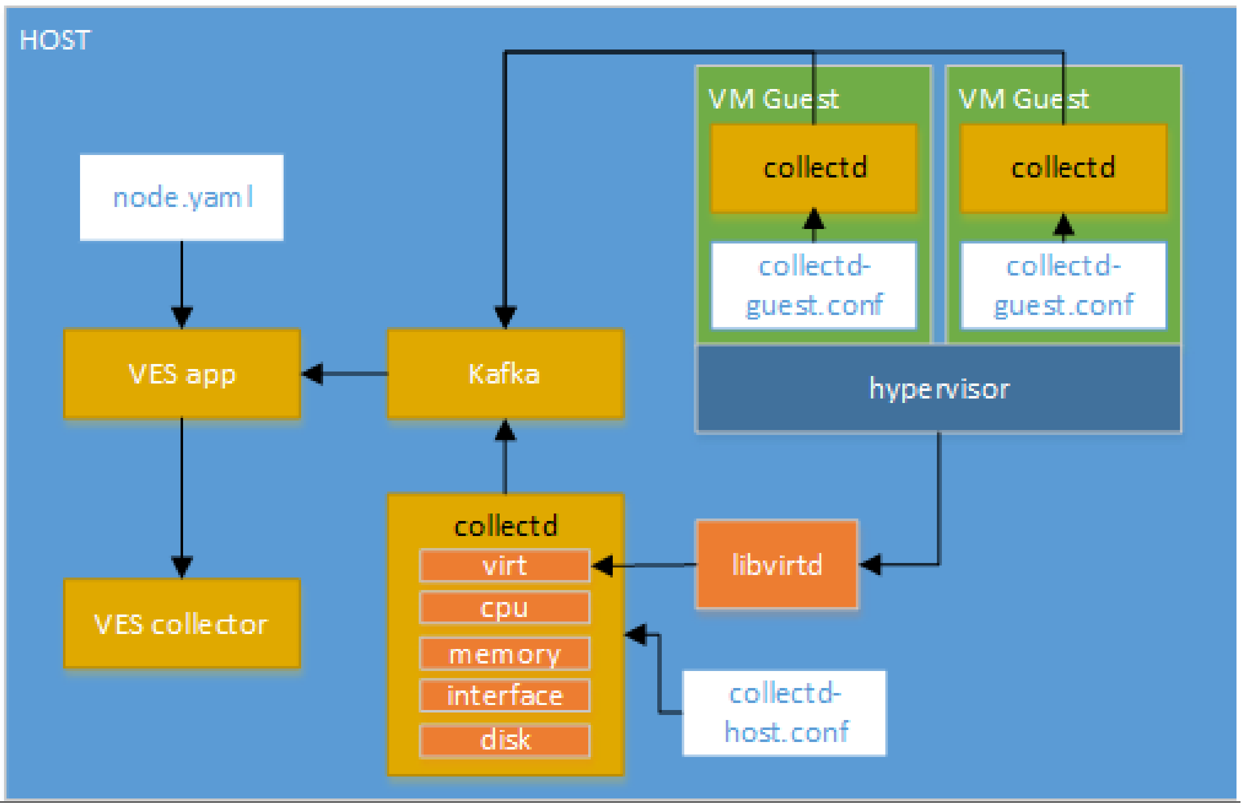
## Kafka - collectd



Collectd write_kafka plugin will send collectd metrics and values to a Kafka Broker.
The VES application uses Kafka Consumer to receive metrics from the Kafka Broker.

***<u>The new VES application is simply a consumer of the collectd topic.</u>***

## Collectd VES application with the new schema mapping:



Hardcoding of VES Schema fields has been removed from the implementation of the Collectd VES application. Instead you now provide a YAML file which describes how to map collectd fields/value lists to VES fields. The diagram above shows the usage example for the node.yaml configuration.

## Producer throughput:

50 million small (100 byte) records as quickly as possible.

| Test Case | Measurement |
|---|---|
| 1 producer thread, no replication | 821,557 records/sec (78.3 MB/sec) |
| 1 producer thread, 3 asynchronous replication | 786,980 records/sec (75.1 MB/sec) |
| 1 producer thread, 3 synchronous replication | 421,823 records/sec (40.2 MB/sec) |

| | |
|---|---|
| 3 producers, 3 async replication | 2,024,032 records/sec (193.0 MB/sec) |

### Consumer throughput

Consume 50 million messages.

| Test Case | Measurement |
|---|---|
| Single Consumer | 940,521 records/sec (89.7 MB/sec) |
| 3x Consumers | 2,615,968 records/sec (249.5 MB/sec) |
| End-to-end Latency | ~2 ms (median) |

## Collectd kafka notification support

**TODO**

### References:

0 https://www.cloudera.com/documentation/kafka/1-2-x/topics/kafka.html

1 https://anturis.com/blog/apache-kafka-an-essential-overview/

2 Documentation (http://kafka.apache.org/documentation.html)

Benchmarking Apache Kafka (https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines)

Robust high performance C/C++ library with full protocol support (https://github.com/edenhill/librdkafka)