Apache Kafka at Rocana

Machine Data Collection at Scale
Kafka Concepts
Every pair of systems requires:

- transformation between data models
- handling downtime and slowness in the target system
- understanding semantics for event delivery
  - at least once, at most once, exactly once
What is a log?

- A sequence of records with an absolute order
- Records have a logical “position” within the log
- Operations on a log are appending records and iterating over records
  - A record’s position within the log is immutable
How do you scale a log?

- Coordinating a single distributed log is slow
- Appending to multiple logs scales linearly
- A set of records is a **topic**, each log within a topic is a **partition**
- There's no absolute ordering of records in a topic, only an ordering within each partition
Replication

• Each partition has a single leader which is responsible for appending to it

• These appends can also be asynchronously replicated to other nodes
  • If the leader fails, a replica will be elected to become leader
  • The offsets of records on every replica are the same as the leader

• Consumers can’t read from replicas, adding replicas doesn’t improve throughput
Consumers

• Consumers are responsible for storing and requesting their own offsets within each partition
  • The consumer APIs can abstract away offset management
  • Kafka brokers retain records based on a configurable rule (size or time)

• Multiple consumer processes can work together in a consumer group

• Each consumer group will each see every record
  • Each consumer within a group will see an exclusive subset of the records
Producers

- Producers decide how events are routed to partitions
  - Round-robin and consistent hashing are typical options
  - Different producers may end up with different configurations

- Producers set the number of acknowledgements they require
  - They may require no ACKs, one ACK or ACKS from all replicas
  - ACKs don’t mean a record has been fsync’d

- It’s important to keep these configs consistent across all producers
Consistency

- Kafka supports replicating partitions across multiple nodes
  - Each partition has an In-Sync Replica (ISR) set
- The ISR can shrink to a single node and still accept writes
  - `min.insync.replicas` can be increased to prevent this
  - the producer also needs to request ACKs from every node in the ISR
- An out-of-date replica can be elected as the leader
  - This is disabled with `unclean.leader.election.enable`
- By default Kafka is an AP system, but it can be configured to be CP
Rocana Ops

• A scalable horizontally-scalable platform for data collection, storage and retrieval

• Applications and analytics designed for IT Operations use cases

• Built on open technologies: Hadoop, Kafka, Solr, Impala, Avro, Parquet

• Flexible and extensible

• Single-tenant: runs on premises or in the cloud
Producers
Rocana Agent

• The Rocana Agent is responsible for splitting records and extracting fields from a variety of data sources

• Records are held in memory to tolerate network partitions and Kafka failures
  • For some sources like syslog this can result in data loss
  • We recommend using the Agent to collect logs locally from disk

• Batching is crucial to achieve performance
  • Small batches result in excessive overhead
  • Batching is per-partition, memory use is also a concern for us

Event Encoding

- Records are encoded using Apache Avro
  - Avro has broad ecosystem support
  - A binary format reduces redundant data on the wire

- All data is encoded with a single Avro schema
  - Using a single schema simplifies consumers

- We require application-specific, first-class fields for our data model
  - Host, location, service, timestamp
  - Users can extend this with arbitrary key-value pairs
Schemas

• Kafka accepts any arbitrary byte array for records

• Enforcing a schema for data in Kafka is essential!
  • Transform data to a common intermediate form close to the source

• Plan for schema evolution from the start
  • Typically Kafka will contain a mix of legacy and current schemas

• Schema registries can help manage complexity with fast-evolving schemas
Consumers
Consumers

• Kafka enables an architecture with many distinct, purpose-built consumers

• We use multiple consumers in our platform:
  • persisting records to Solr and HDFS
  • computing metrics aggregates as Parquet in HDFS
  • scoring records against predicted models
  • updating an RDBMS with metadata

• Each consumer can be scaled, reconfigured and restarted independently
Consumer Lifecycle

- Consumers all follow a common life cycle:
  - read a batch of records from a given partition
  - transform records to the target data model
  - durably write the transformed records
  - persist the new offset for the partition to ZooKeeper

- In a crash failure, we may write a batch without checkpointing
  - This gives at-least-once semantics
  - Solr inserts and RDBMS updates are idempotent
Consumer Errors

- Kafka stores data durably, with retention configured by time or log size
- Retention limits the maximum downtime of consumers before losing data
- It’s possible to rewind offsets and re-consume individual partitions
- We also store the full set of events in HDFS
  - This allows for processing and reindexing at scale
- We use a dead-letter queue to store corrupt messages which can’t be decoded
In-line Transformation
Rocana Transformation

• We developed an extensible transformation engine for real-time data streams

• The transformation engine consumes from and produces to Kafka
  • Transformations are transparent to downstream consumers

• Transformation is limited to map and filter functions
  • This makes them trivially parallel

• The transformation engine includes common built-in actions
  • Filter, extract, flatten

http://www.slideshare.net/JoeyEcheverria/embeddable-data-transformation-for-real-time-streams
{
    load-event: {},
    build-metric: {
        gauge-mapping: {
            http.request.latency: "${convert:toDouble(attributes['latency'])}"  
        },
        destination: "latency_metric"
    },
    accumulate-output: {value: "${latency_metric}"},
    build-metric: {
        dynamic-counter-mapping: [
            "${string:format('http.status.%s.count', attributes['sc_status'])}"
        ],
        destination: "status_metric"
    }
    accumulate-output: {value: "${status_metric}"}
}
Customer Use Cases

• Extract performance metrics from Apache logs
• MaxMind GEO IP lookup
• Enrich records with host detail from a database
Monitoring

- Kafka buffers spikes in traffic if consumers can’t keep up
- It’s possible to build up a deficit if consumers are under-provisioned
  - Consumer processes can fail, with some partitions backing up
- Can be monitored indirectly with event rate in producers and consumers
  - Tools exist to monitor offsets in ZooKeeper, or in Kafka itself
Support

• CDH and HDP both include Kafka as a manageable component

• Confluent provides a Kafka-centric platform

• Kafka versions and APIs are moving quickly
  • Encryption and authentication were added recently
  • New APIs for consumers and producers
  • Group management re-architected
Thank You!

alan@rocana.com

@alanctgardner