https://github.com/stealthly/go_kafka_client

Joe Stein

- Developer, Architect & Technologist
- Founder & Principal Consultant => Big Data Open Source Security LLC - http://stealth.ly

  Big Data Open Source Security LLC provides professional services and product solutions for the collection, storage, transfer, real-time analytics, batch processing and reporting for complex data streams, data sets and distributed systems. BDOSS is all about the "glue" and helping companies to not only figure out what Big Data Infrastructure Components to use but also how to change their existing (or build new) systems to work with them.

- Apache Kafka Committer & PMC member
- Blog & Podcast - http://allthingshadoop.com
- Twitter @allthingshadoop

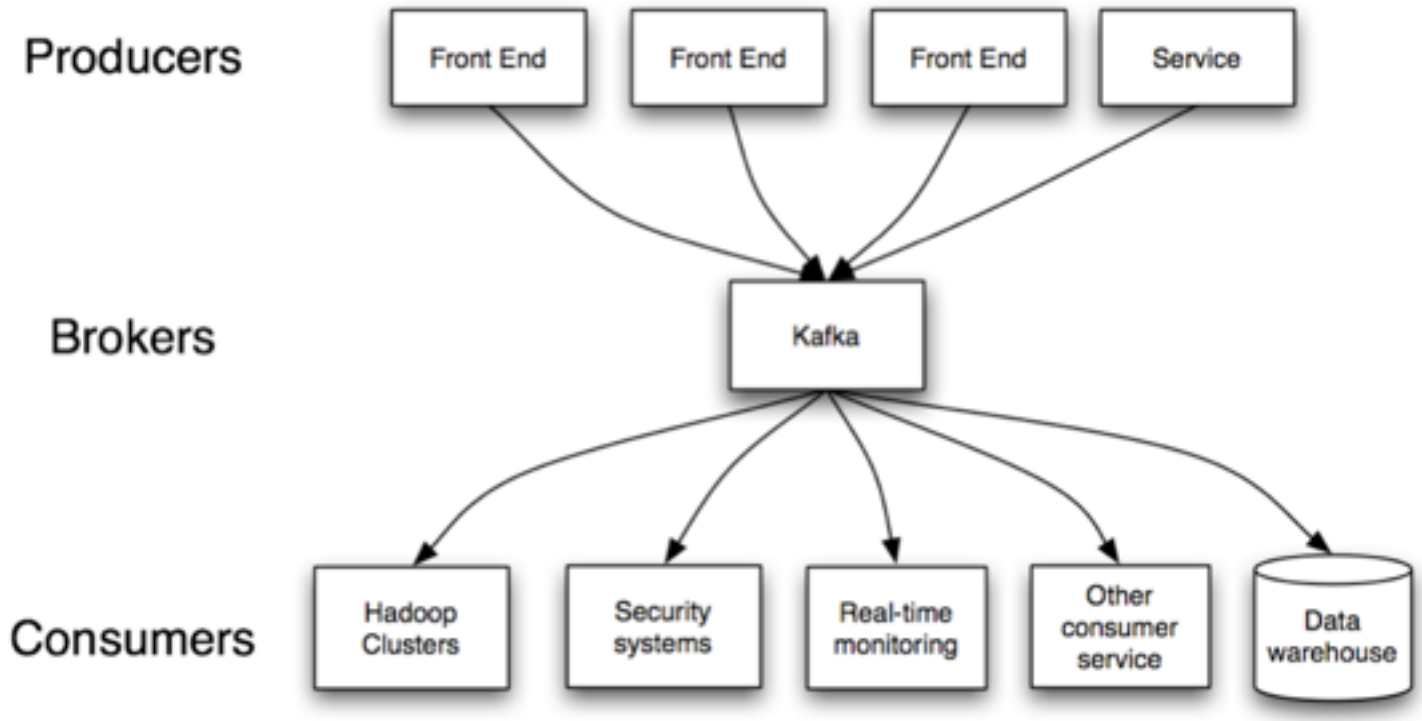
Overview

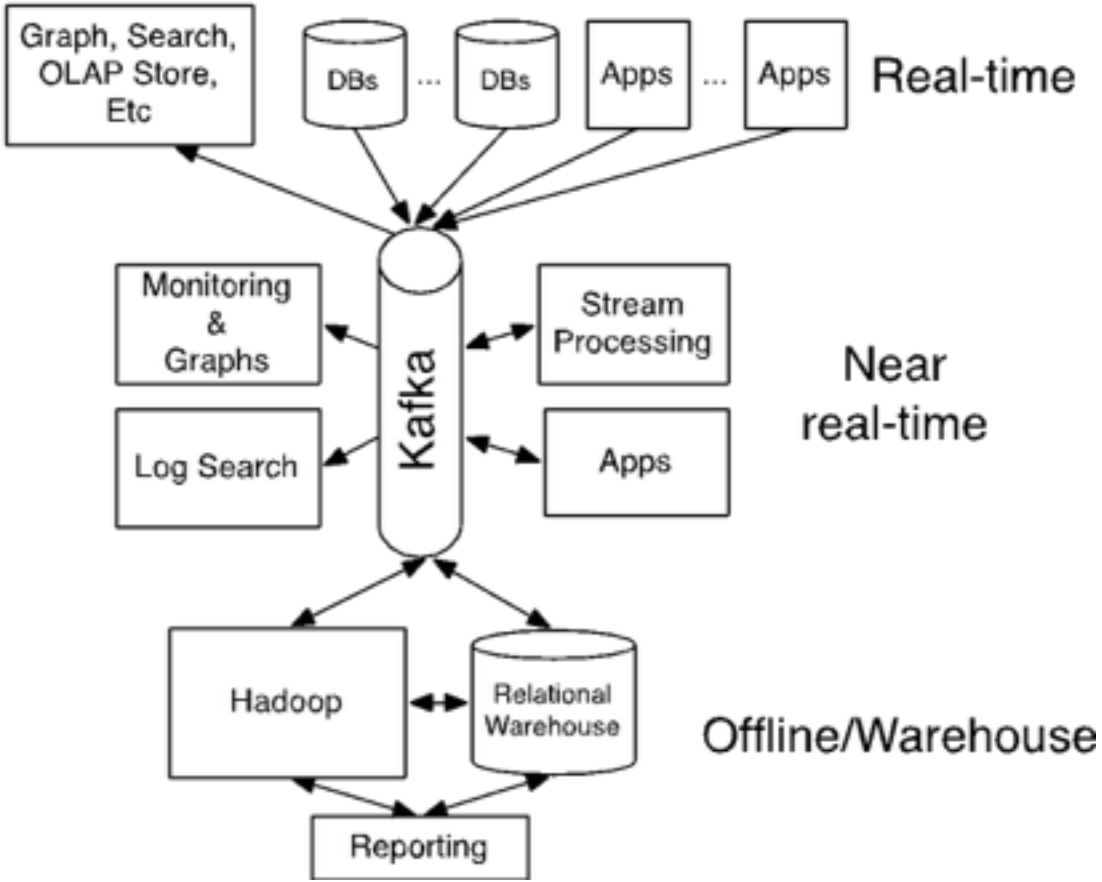
- Just enough Kafka
- Just enough Go
- Why a new Go Kafka Client
- Producers in Go
  - Code
  - Syslog
  - MirrorMaker
- Consumers in Go
  - Work management
  - Partition ownership
  - Blue / Green Deploys
  - Offset management
- Distributed Reactive Streams

Apache Kafka

- Apache Kafka
  - http://kafka.apache.org
- Apache Kafka Source Code
  - https://github.com/apache/kafka
- Documentation
  - http://kafka.apache.org/documentation.html
- FAQ
  - https://cwiki.apache.org/confluence/display/KAFKA/FAQ
- Wiki
  - https://cwiki.apache.org/confluence/display/KAFKA/Index

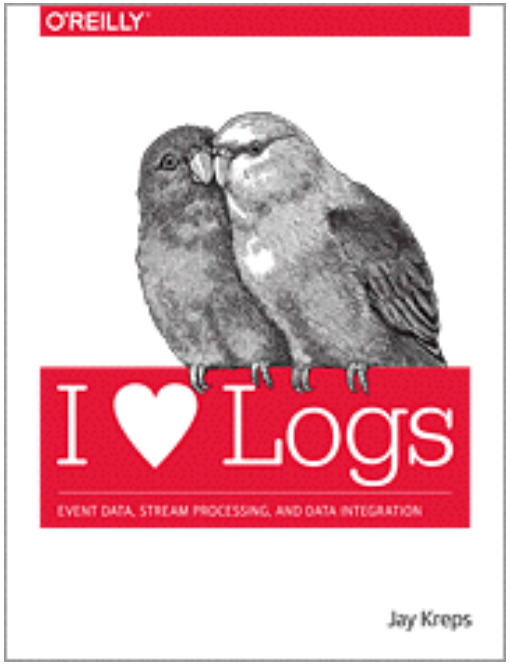
Kafka is a distributed, partitioned, replicated commit log service. It provides the functionality of a messaging system, but with a unique design.




Graph, Search, OLAP Store, Etc
DBs
...
DBs
Apps
...
Apps
Real-time
Monitoring & Graphs
Kafka
Stream Processing
Log Search
Apps
Near real-time
Hadoop
Relational Warehouse
Offline/Warehouse
Reporting

I heart logs

http://shop.oreilly.com/product/0636920034339.do?sortby=publicationDate



Really Quick Start (Go)

1) Install Vagrant http://www.vagrantup.com/
2) Install Virtual Box https://www.virtualbox.org/
3) git clone https://github.com/stealthly/go-kafka
4) cd go-kafka
5) vagrant up
6) vagrant ssh brokerOne
7) cd /vagrant
8) sudo ./test.sh

Just enough Go

- Produces statically linked native binaries without external dependencies
- Built in package management with source commit definition
- Share memory by communicating
- Code to read to start learning go http://www.somethingsimilar.com/2013/12/27/code-to-read-when-learning-go/

Why another Go client?

- Go Kafka Client https://github.com/stealthly/go_kafka_client wraps Sarama https://github.com/Shopify/sarama
- More real world use for producers
- High level consumer (lots more on this in a few minutes)
- The option to maybe wrap librdkafka https://github.com/edenhill/librdkafka the high perforamcen c/c++ library in the future

Go Kafka Client - Getting started

Prerequisites:

1. Install Golang http://golang.org/doc/install
2. Make sure env variables GOPATH and GOROOT exist and point to correct places
3. Install GPM https://github.com/pote/gpm
4. `go get github.com/stealthly/go_kafka_client && cd $GOPATH/src/github.com/stealthly/go_kafka_client`
5. `gpm install`

Optional (for all tests to work):

1. Install Docker https://docs.docker.com/installation/#installation
2. `cd $GOPATH/src/github.com/stealthly/go_kafka_client`
3. Build docker image: `docker build -t stealthly/go_kafka_client .`
4. `docker run -v $(pwd):/go_kafka_client stealthly/go_kafka_client`

After this is done you're ready to write some code!

For email support https://groups.google.com/forum/#!forum/kafka-clients

Producers

Producer Code

```
client, err := sarama.NewClient(uuid.New(), []string{brokerConnect}, sarama.NewClientConfig())
if err != nil {
        panic(err)
}

config := sarama.NewProducerConfig()
config.FlushMsgCount = flushMsgCount
config.FlushFrequency = flushFrequency
config.AckSuccesses = true
config.RequiredAcks = sarama.NoResponse //WaitForAll
config.MaxMessagesPerReq = maxMessagesPerReq
config.Timeout = 1000 * time.Millisecond
config.Compression = 2

producer, err := sarama.NewProducer(client, config)
if err != nil {
        panic(err)
}
```

Magic Happens Here =8^)

```
go func() {
	for {
		message := &sarama.MessageToSend{Topic: topic, Key: sarama.StringEncoder(fmt.Sprintf("%d", numMessage)), Value:
	sarama.StringEncoder(fmt.Sprintf("message %d!", numMessage))}
		numMessage++
		producer.Input() <- message
		time.Sleep(sleepTime)
	}
}()
go func() {
	for {
		select {
			case error := <-producer.Errors():
				saramaError <- error
			case success := <-producer.Successes():
				saramaSuccess <- success
			}
		}
}()
```

Syslog Producer

https://github.com/stealthly/go_kafka_client/tree/master/syslog

```
docker run -p --net=host stealthly/syslog --topic syslog --broker.list host:port
```

`--producer.config` - property file to configure embedded producers. *This parameter is optional*.

`--tcp.port` - TCP port to listen for incoming syslog messages. *Defaults to 5140*.

`--tcp.host` - TCP host to listen for incoming syslog messages. *Defaults to 0.0.0.0*.

`--udp.port` - UDP port to listen for incoming syslog messages. *Defaults to 5141*.

`--udp.host` - UDP host to listen for incoming syslog messages. *Defaults to 0.0.0.0*.

`--num.producers` - number of producer instances. This can be used to increase throughput. *Defaults to 1*

`--queue.size` - number of messages that are buffered for producing. *Defaults to 10000*.

`--log.level` - log level for built-in logger. Possible values are: `trace`, `debug`, `info`, `warn`, `error`, `critical`.*Defaults to info*.

`--max.procs` - maximum number of CPUs that can be executing simultaneously. *Defaults to runtime.NumCPU()*.

Syslog Producer + Metadata

option to transform via --source “anything” --tag key1=value1 --tag key2=value2 --log.type 3

```
package syslog.proto;
message LogLine
{
        message Tag
        {
                required string key = 1;
                required string value = 2;
        }
        required string line = 1;
        optional string source = 2 [default = ""];
        repeated Tag tag = 3;
        optional int64 logtypeid = 4 [default = 0];
        repeated int64 timings = 5;
}
```

MirrorMaker

https://github.com/stealthly/go_kafka_client/tree/master/mirrormaker

- Preserve order
- Preserve partition number
- Prefix destination topic (e.g. dc1_) so you know where it came from and avoid collision
- Everything else you expect

Consumer

Work management

- Fan out
- Sequential processing guarantees
- Ack communication from work to retry
- Failure “dead letter” so work can continue

Partition Ownership

- Not just “re-balance”
- Consistent single known state
- Strategies (plural)
  - Round Robin
  - Range
  - More to come! e.g. manual assignment

Blue / Green Deployments

Jim Plush & Sean Berry from CrowdStrike=> https://www.youtube.com/watch?v=abK2Q_aecxY

Offset Management / Bookkeeping

- At least once processing guarantee
- Plugable
- Per partition batch commit
- Exactly once processing, happens after work is done
- Always commits highest offset

Consumer Example

https://github.com/stealthly/go_kafka_client/blob/master/consumers/consumers.go

```
func main() {
		config, topic, numConsumers, graphiteConnect, graphiteFlushInterval := resolveConfig()
		consumers := make([]*kafkaClient.Consumer, numConsumers)
		for i := 0; i < numConsumers; i++ {
			consumers[i] = startNewConsumer(*config, topic)
		}
}
func startNewConsumer(config kafkaClient.ConsumerConfig, topic string) *kafkaClient.Consumer {
		config.Strategy = GetStrategy(config.Consumerid)
		config.WorkerFailureCallback = FailedCallback
		config.WorkerFailedAttemptCallback = FailedAttemptCallback
		consumer := kafkaClient.NewConsumer(&config)
		topics := map[string]int {topic : config.NumConsumerFetchers}
		go func() {
			consumer.StartStatic(topics)
		}()
		return consumer
}
```

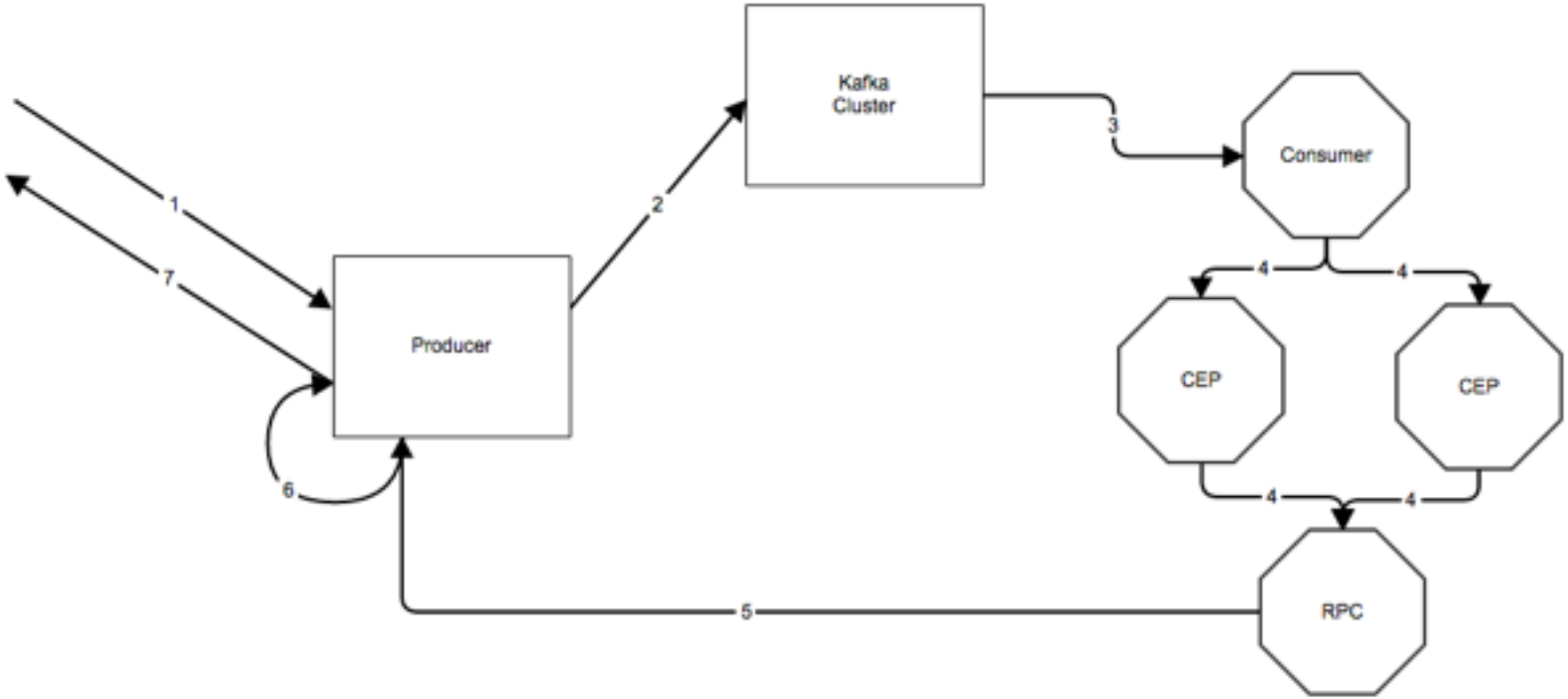
Consumer Example

```
func GetStrategy(consumerId string) func(*kafkaClient.Worker, *kafkaClient.Message, kafkaClient.TaskId) kafkaClient.WorkerResult
{
		consumeRate := metrics.NewRegisteredMeter(fmt.Sprintf("%s-ConsumeRate", consumerId), metrics.DefaultRegistry)
		return func(_ *kafkaClient.Worker, msg *kafkaClient.Message, id kafkaClient.TaskId) kafkaClient.WorkerResult {
				kafkaClient.Tracef("main", "Got a message: %s", string(msg.Value))
				consumeRate.Mark(1)


				return kafkaClient.NewSuccessfulResult(id)
		}
}
func FailedCallback(wm *kafkaClient.WorkerManager) kafkaClient.FailedDecision {
		kafkaClient.Info("main", "Failed callback")
		return kafkaClient.DoNotCommitOffsetAndStop
}
func FailedAttemptCallback(task *kafkaClient.Task, result kafkaClient.WorkerResult) kafkaClient.FailedDecision {
		kafkaClient.Info("main", "Failed attempt")
		return kafkaClient.CommitOffsetAndContinue
}
```

Distributed Reactive Streams

Distributed Reactive Streams



Questions?

/***

Joe Stein

Founder, Principal Consultant

Big Data Open Source Security LLC

http://www.stealth.ly

Twitter: @allthingshadoop

***/