Kafkaesque & Microbial

Peter Elger

@pelger

peter.elger@nearform.com

Stuff…

- Background
- Kafka + Zookeeper
- Kafkaesque
  - Node module implementing Kafka 0.8 protocol.
  - Tracking Kafka 0.9 protocol – not released yet.
- Microbial
  - Micro-services tool kit layered over Kafkaesque.

Now then what is a micro-service?

- Term first coined by Fred George
  - Checkout some awesome talks on You Tube

- The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services…
  - Martin Fowler

- A system component that any developer on the team can rewrite in a week or less
  - Richard Rodger

…And this Kafka thing ?

- Kafka is a distributed, partitioned, replicated commit log service.

- Producers publish messages to a Topics
  - Consumers subscribe to topics and process messages

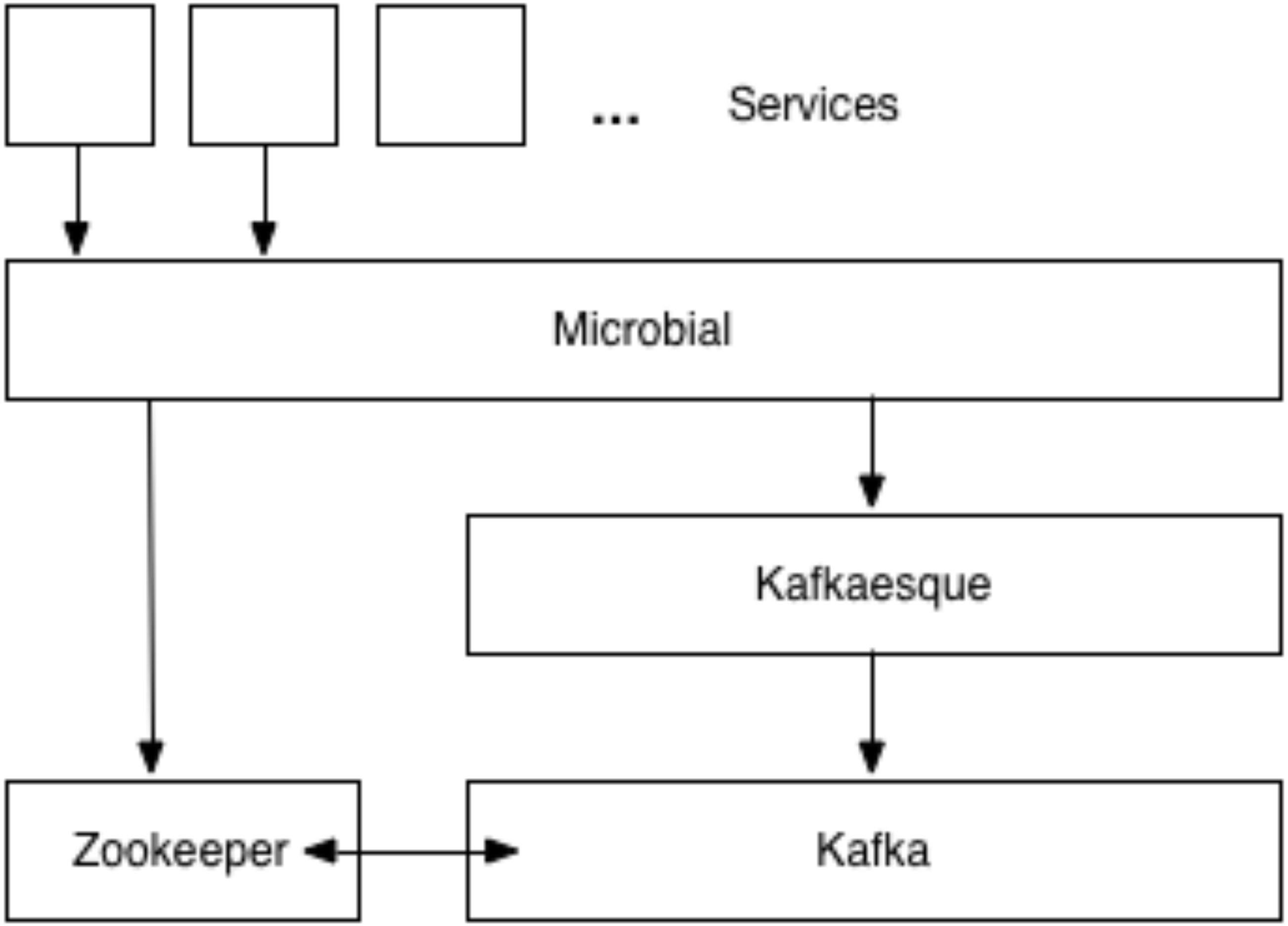
- Kafka is run as a cluster comprised of one or more servers each of which is called a broker.

- Grew out of project at linkedin

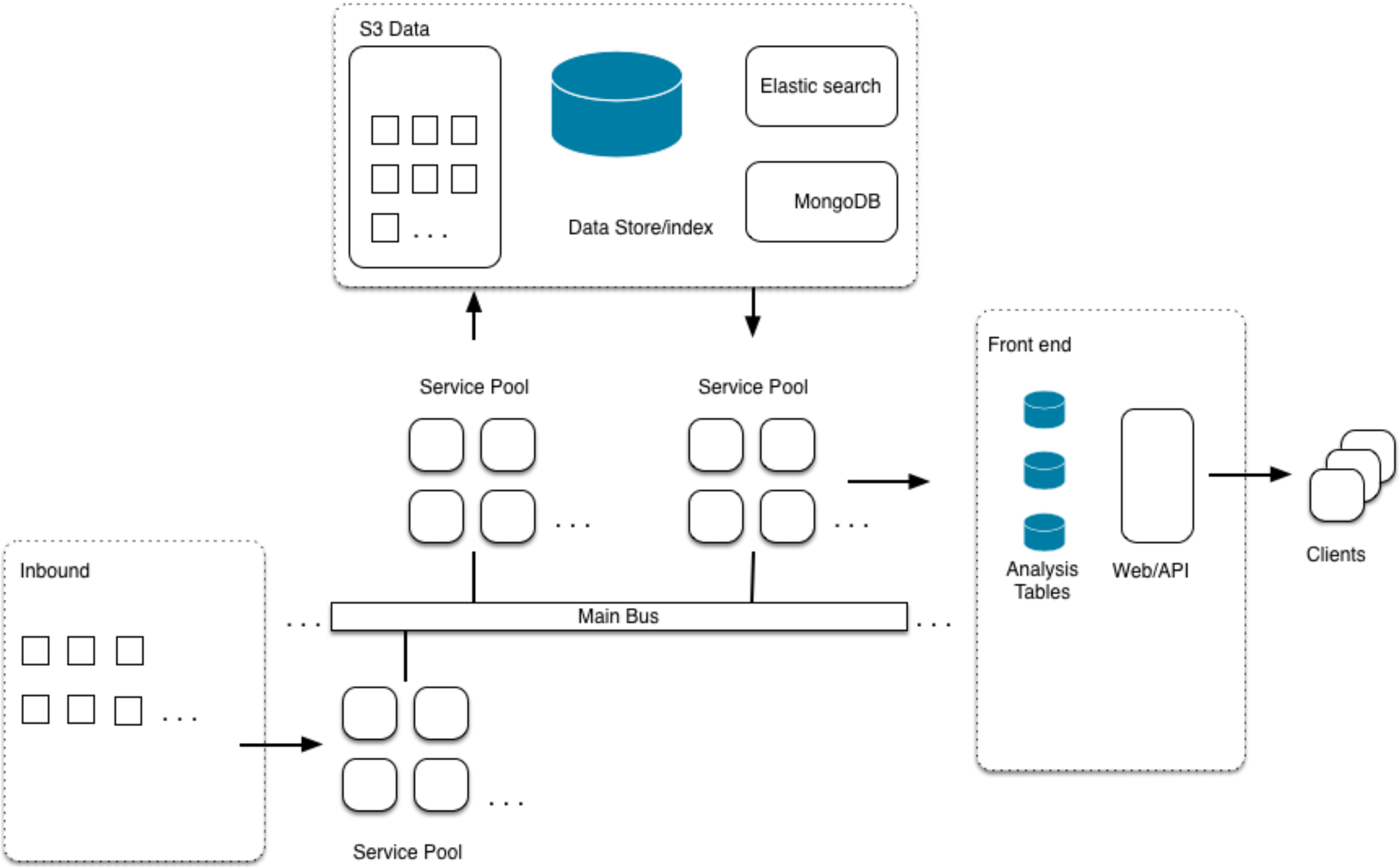
…Oh and Zookeeper ?

- Originally a Hadoop sub project
- ZooKeeper is a centralized service for maintaining configuration information.
- distributed processes to coordinate with each other through a shared hierarchal namespace similar to a standard file system
- Replicated for HA
- Simple restful API

Logical Dependencies

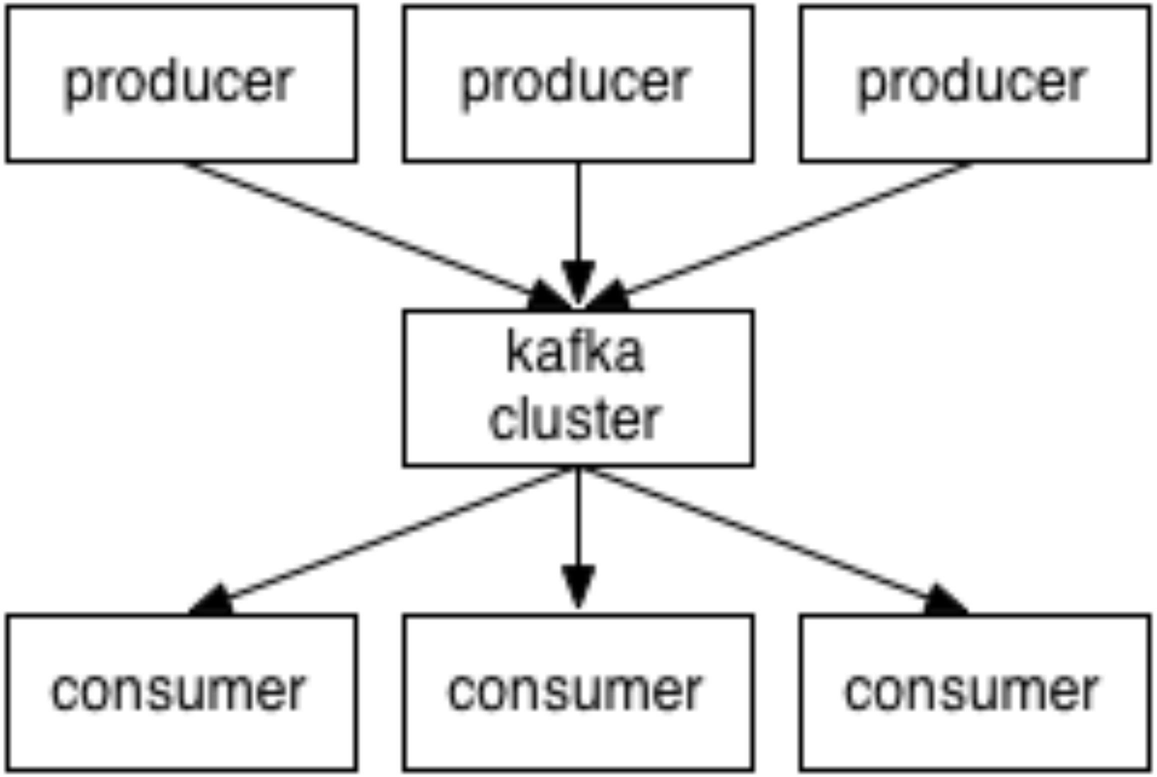


An Example System



Kafka

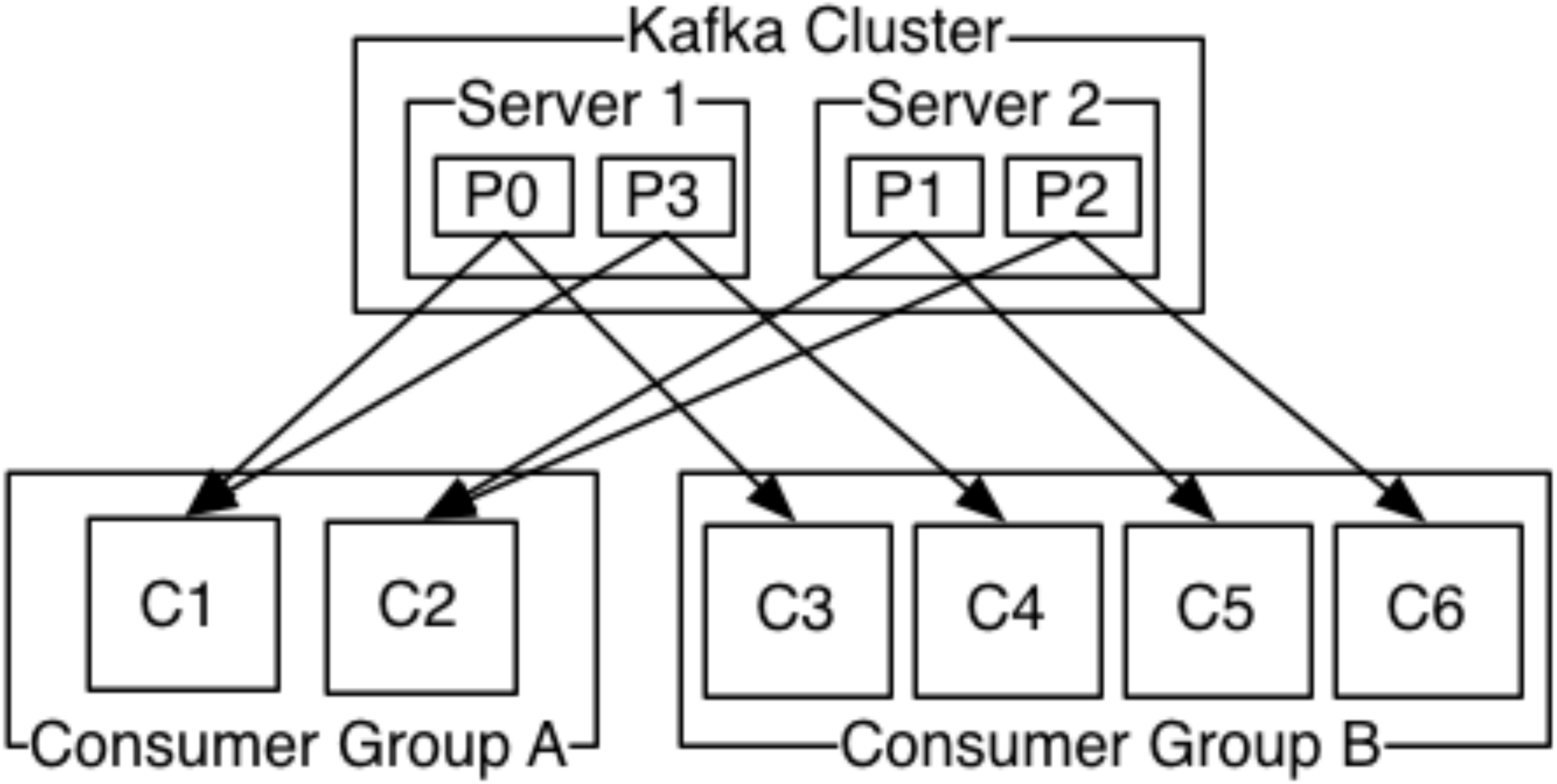
Kafka


producer
producer
producer
kafka cluster
consumer
consumer
consumer

Message Semantics

- Queue
  - pool of consumers read from the queue and each message goes to one only
- Pub/Sub
  - message broadcast to all consumers
- Kafka – provides consumer groups
  - each message published to a topic is delivered to one consumer instance within each subscribing consumer group
    - Pub/Sub and/or Queue

Consumer Groups

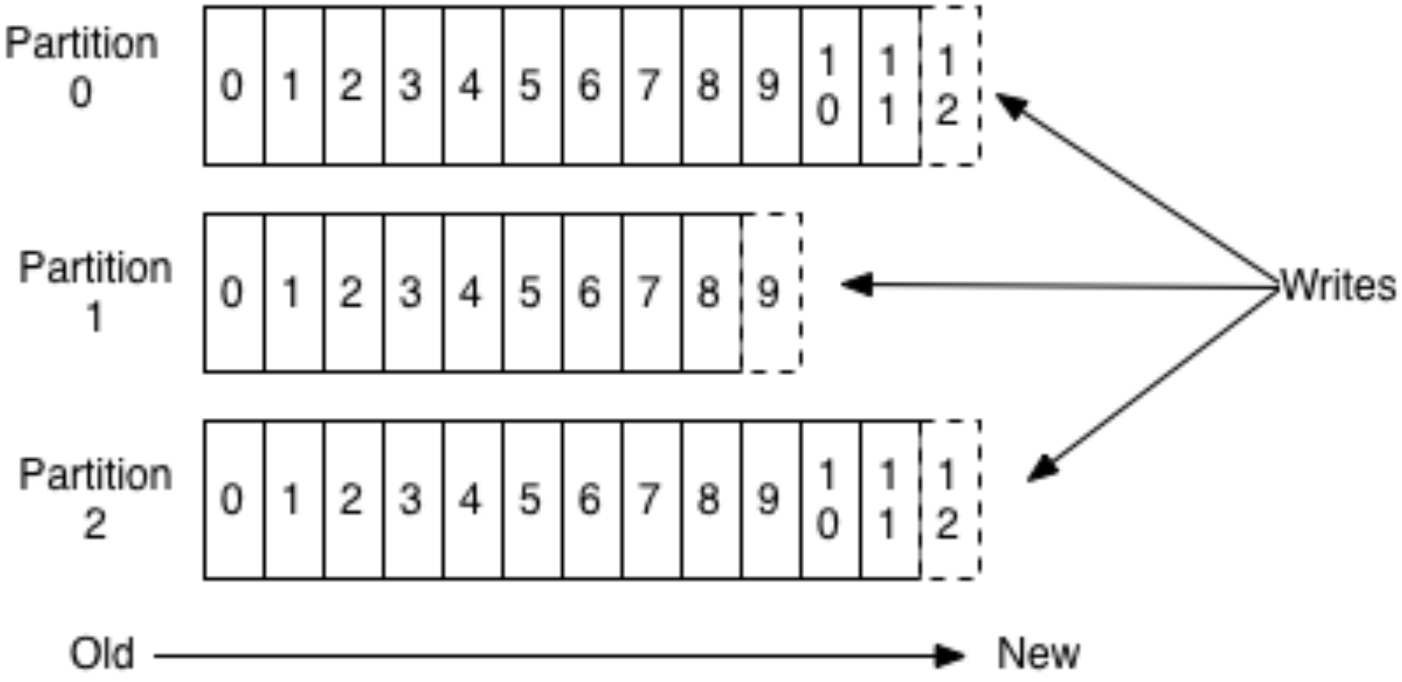


Why Kafka for microservices

- Central bus for system construction
  - Run locally or at scale
  - Supports queuing + pub/sub depending on configuration
- Commit log
  - Record all interactions
  - Replay interaction sequences
  - Build infrastructure services by reading history

Kafka - Topics

Anatomy of a Topic



Partition = ordered, immutable sequence of messages, continually appended to, i.e. a commit log.

Kafka – Producers and Consumers

- Producers decide which partitions to publish to in each topic
  - Round robin, Key based…

- Consumers belong to a consumer group
  - each message published to a topic is delivered to one consumer instance within each subscribing consumer group.

  - If all the consumer instances have the same consumer group - traditional queue balancing load.

  - if all the consumer instances have different consumer groups - publish-subscribe all messages are broadcast.

Kafka - Guarantees

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent.

- A consumer instance sees messages in the order they are stored in the log.

- For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any messages committed to the log.

Kafka – API/Protocol

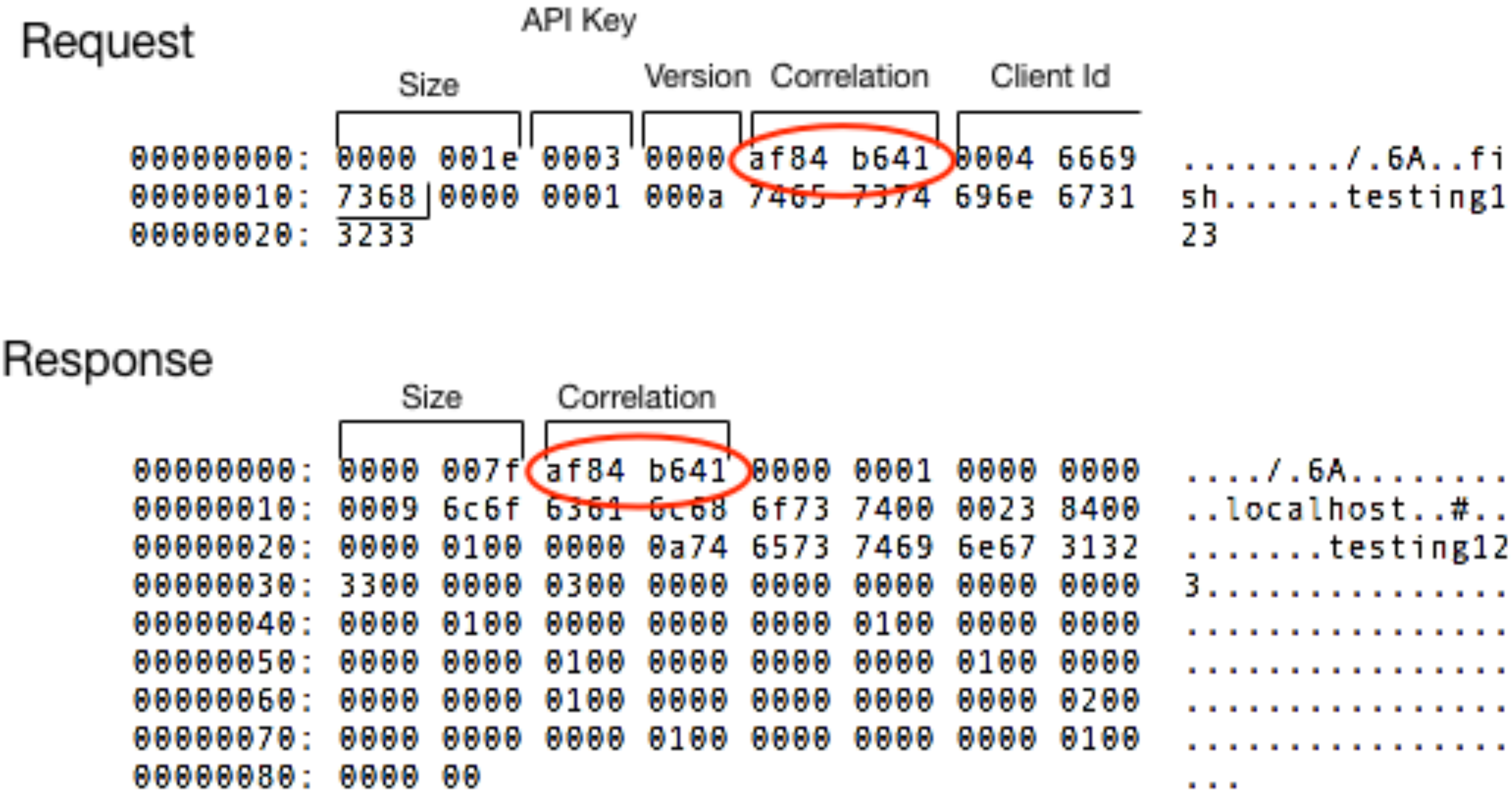
- Compact binary protocol providing following:
  - Metadata API - Detail on topics, partitions, leaders
  - Produce API - Publish messages to topics
  - Fetch API - pull messages from topics
  - Offset API - Read offset position by time
  - Offset Commit/Fetch API - Centralized offset management
    - **Due in 0.9**

Kafka – Offset Management – 0.9

- Remove the need for clients to manage offset position manually

- Currently most folks use zookeeper or similar to to manage offset position
  - See Kafka java client for example

Example packets



```
00000000: 0000 001e 0003 0000 af84 b641 0004 6669  ........./.6A..fi
00000010: 7368 0000 0001 000a 7465 7374 696e 6731  sh.......testing1
00000020: 3233                                     23
```

```
00000000: 0000 007f af84 b641 0000 0001 0000 0000  ..../.6A........
00000010: 0009 6c6f 6361 6c68 6f73 7400 0023 8400  ..localhost..#..
00000020: 0000 0100 0000 0a74 6573 7469 6e67 3132  ........testing12
00000030: 3300 0000 0300 0000 0000 0000 0000 0000  3...............
00000040: 0000 0100 0000 0000 0000 0100 0000 0000  ................
00000050: 0000 0000 0100 0000 0000 0000 0100 0000  ................
00000060: 0000 0000 0100 0000 0000 0000 0000 0200  ................
00000070: 0000 0000 0000 0100 0000 0000 0000 0100  ................
00000080: 0000 00                                  ...
```

Kafkaesque

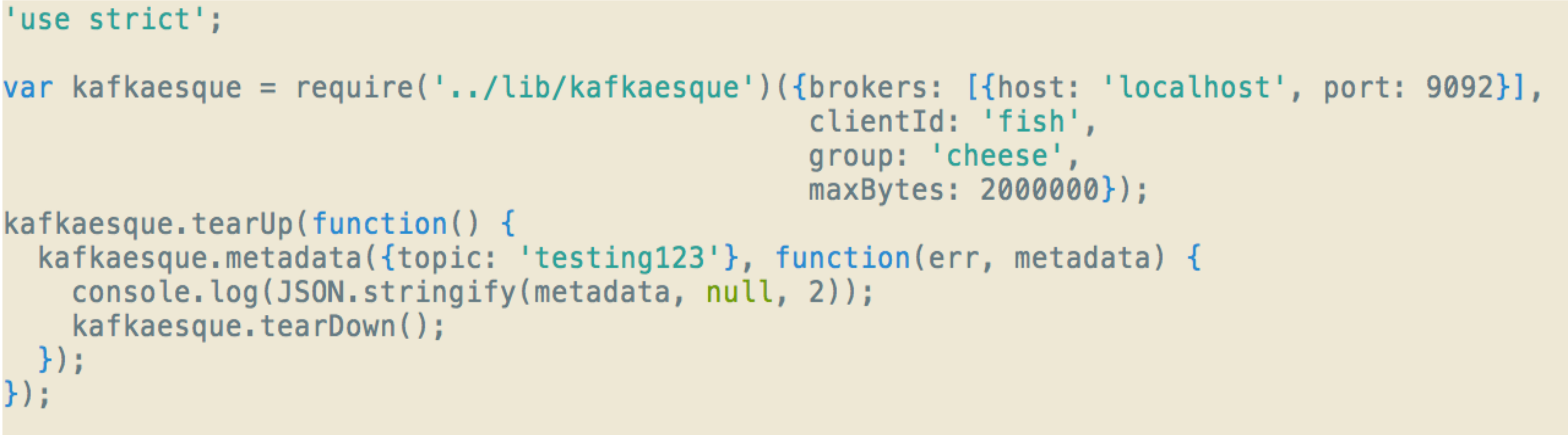
Kafkaesque

- npm install kafkaesque

- node module that implements a Kafka 0.8 client.
  - Tracking 0.9 development

- Some Example code…

Metadata request

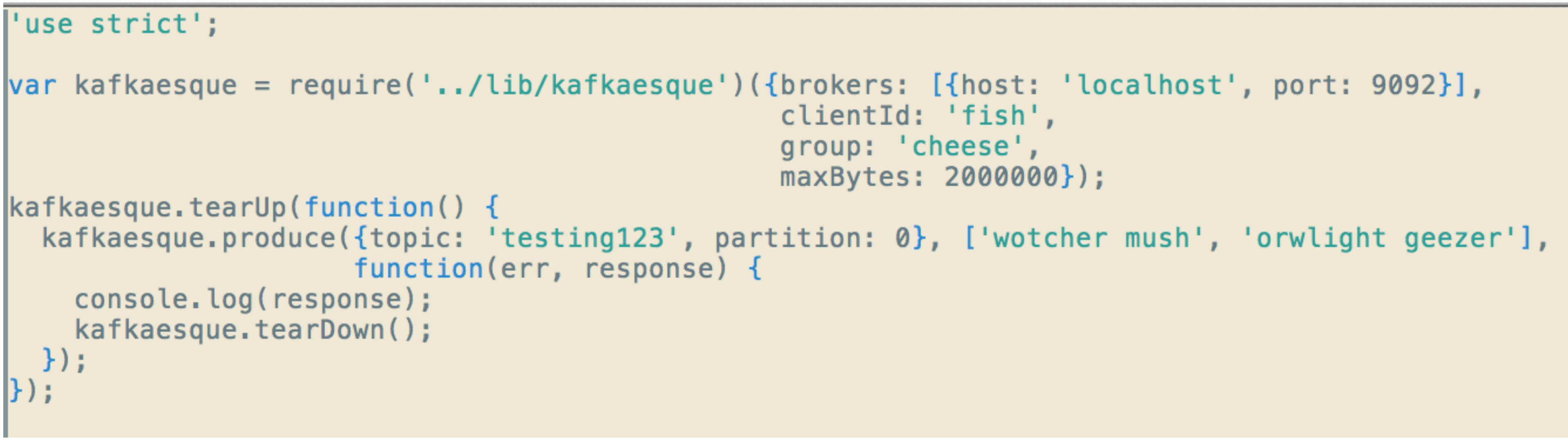


```
'use strict';

var kafkaesque = require('../lib/kafkaesque')({brokers: [{host: 'localhost', port: 9092}],
                                               clientId: 'fish',
                                               group: 'cheese',
                                               maxBytes: 2000000});

kafkaesque.tearUp(function() {
  kafkaesque.metadata({topic: 'testing123'}, function(err, metadata) {
    console.log(JSON.stringify(metadata, null, 2));
    kafkaesque.tearDown();
  });
});
```

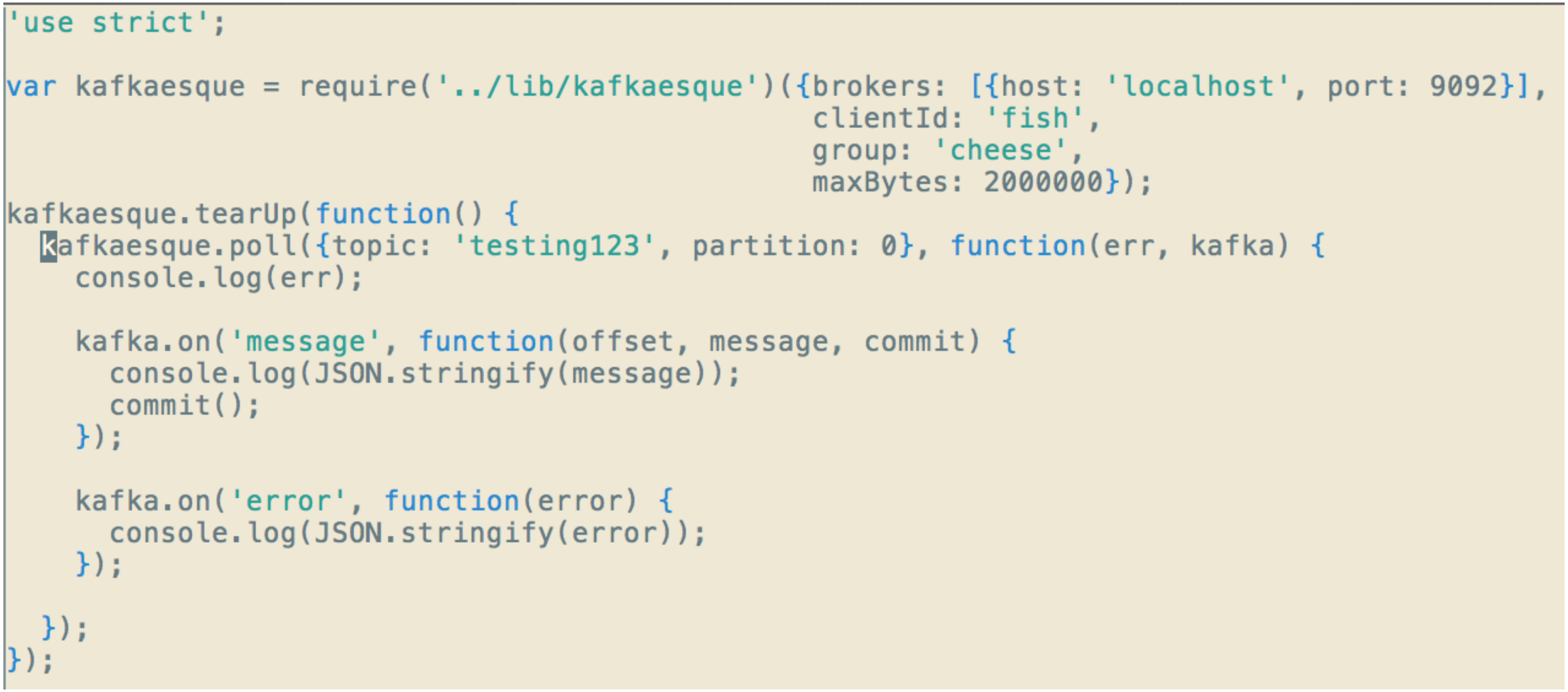
Producer Example



```
'use strict';

var kafkaesque = require('../lib/kafkaesque')({brokers: [{host: 'localhost', port: 9092}],
                                               clientId: 'fish',
                                               group: 'cheese',
                                               maxBytes: 2000000});
kafkaesque.tearUp(function() {
  kafkaesque.produce({topic: 'testing123', partition: 0}, ['wotcher mush', 'orwlight geezer'],
                     function(err, response) {
    console.log(response);
    kafkaesque.tearDown();
  });
});
```

Consumer Example



```
'use strict';

var kafkaesque = require('../lib/kafkaesque')({brokers: [{host: 'localhost', port: 9092}],
                                               clientId: 'fish',
                                               group: 'cheese',
                                               maxBytes: 2000000});
kafkaesque.tearUp(function() {
  kafkaesque.poll({topic: 'testing123', partition: 0}, function(err, kafka) {
    console.log(err);

    kafka.on('message', function(offset, message, commit) {
      console.log(JSON.stringify(message));
      commit();
    });

    kafka.on('error', function(error) {
      console.log(JSON.stringify(error));
    });

  });
});
```

Microbial

Microbial

- npm install microbial

- **NOT** a framework!

- Very lightweight toolkit to help with micro service construction

Microbial

- Set and retrieve system configuration from zookeeper

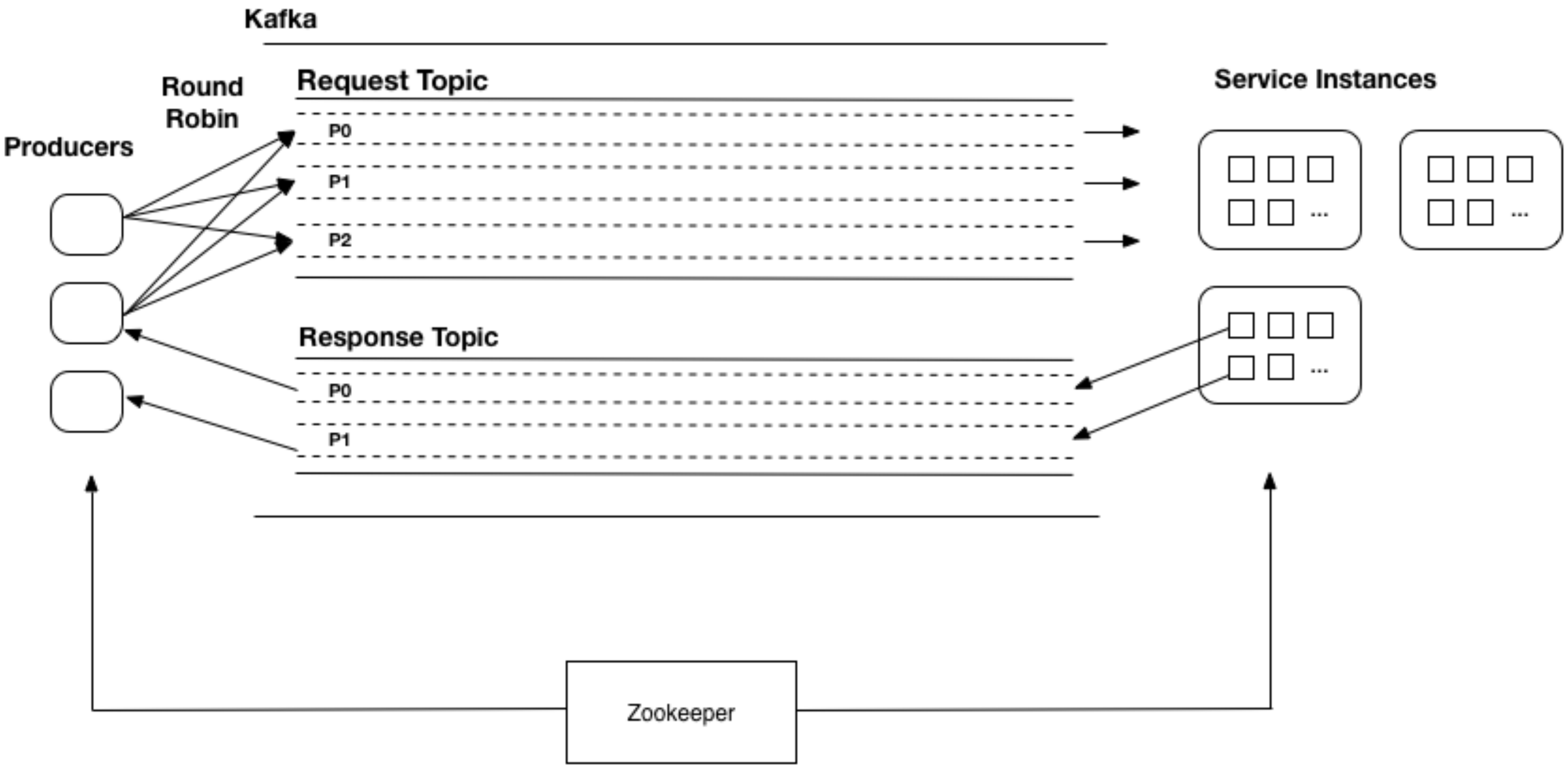
- Transparently handle Kafka offset management

- Simple API to initiate micro-service execution

- Micro-service pattern matching and execution

- Simple API for micro-service construction

Execution Pattern



Execution Pattern

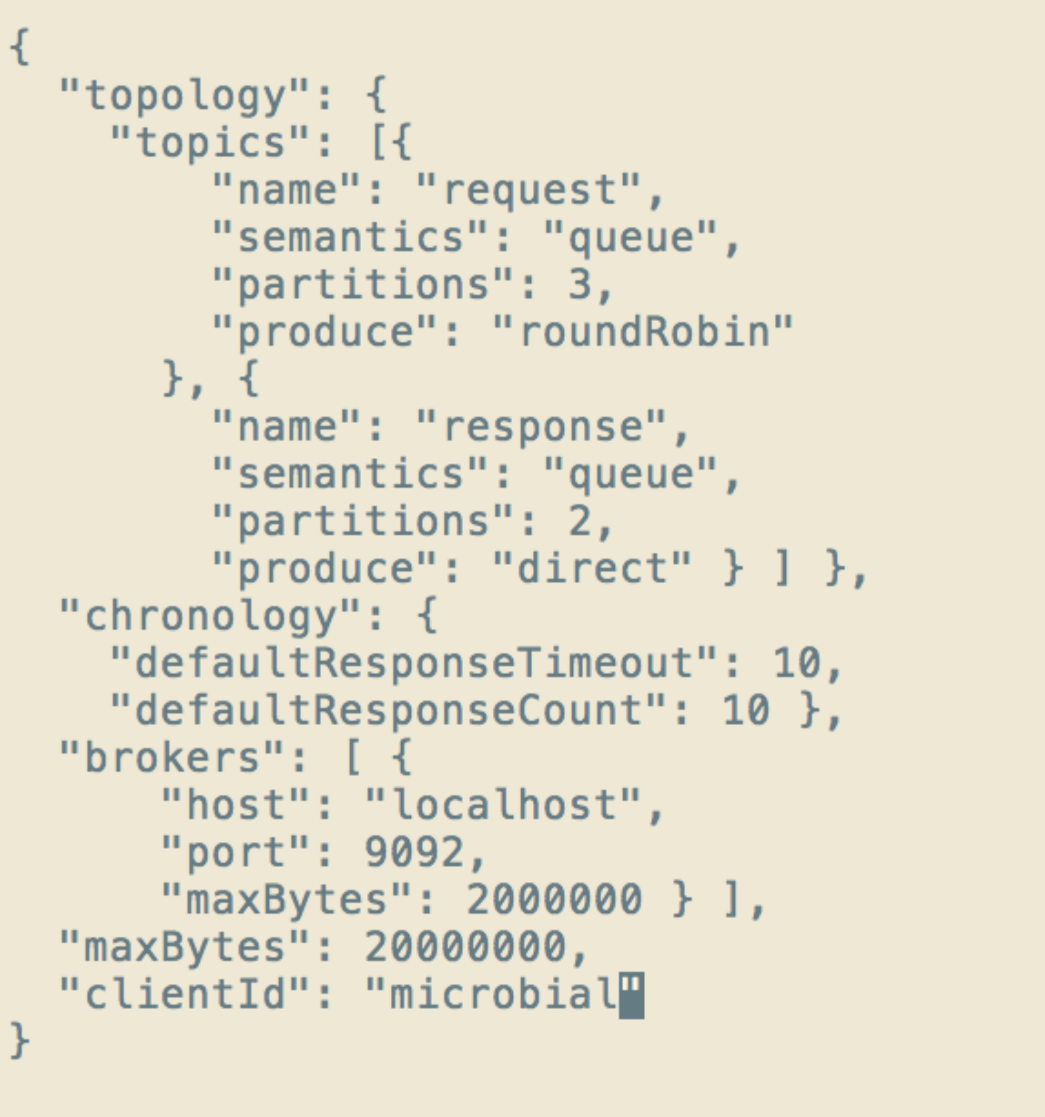
- Producers balance requests over partitions

- Producers place return address into call
  - {topic: <name>, partition: <partition>}

- Service nodes pattern match to execute the appropriate service or services

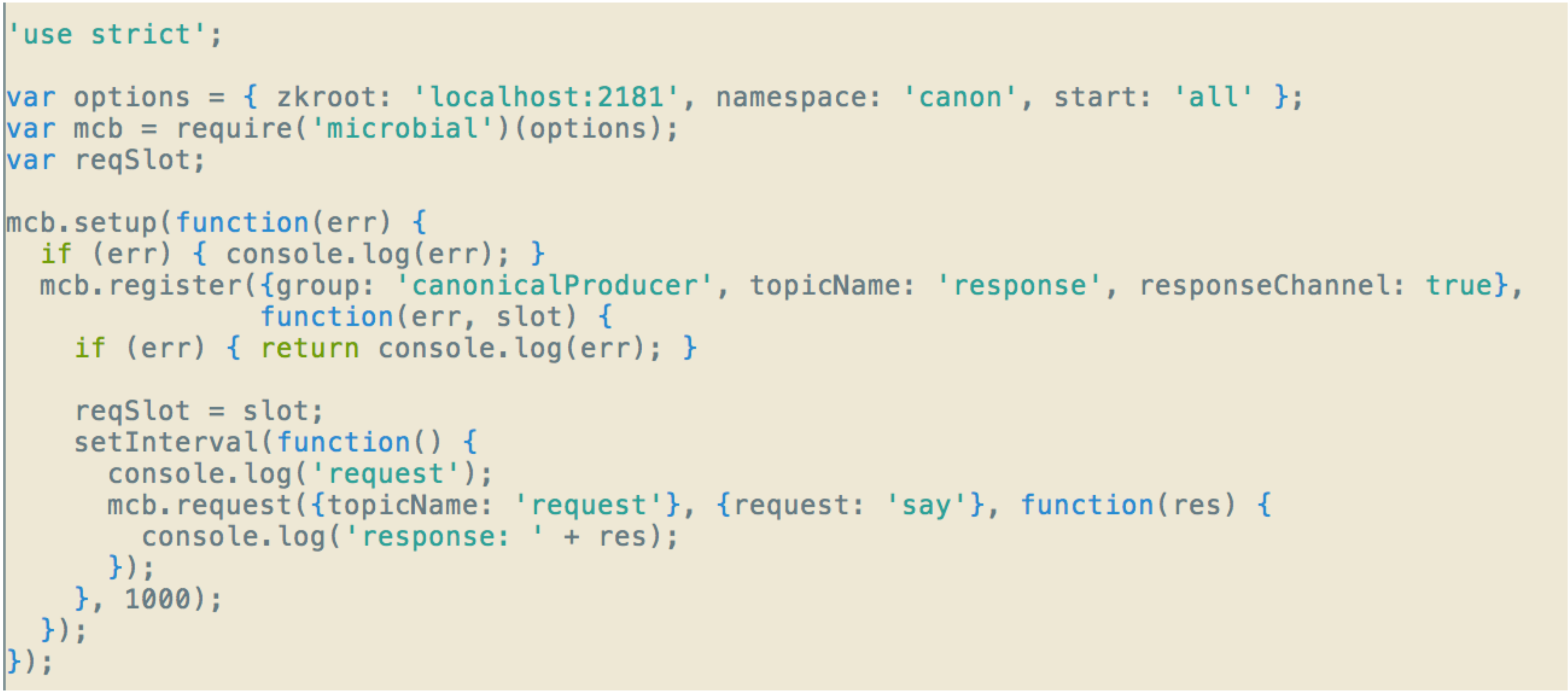
- Responses are placed onto the bus using the supplied return address

Microbial - Configuration



```
{
 "topology": {
  "topics": [{
    "name": "request",
    "semantics": "queue",
    "partitions": 3,
    "produce": "roundRobin"
   }, {
    "name": "response",
    "semantics": "queue",
    "partitions": 2,
    "produce": "direct" } ] },
 "chronology": {
  "defaultResponseTimeout": 10,
  "defaultResponseCount": 10 },
 "brokers": [ {
   "host": "localhost",
   "port": 9092,
   "maxBytes": 2000000 } ],
 "maxBytes": 20000000,
 "clientId": "microbial"
}
```

Producer - example



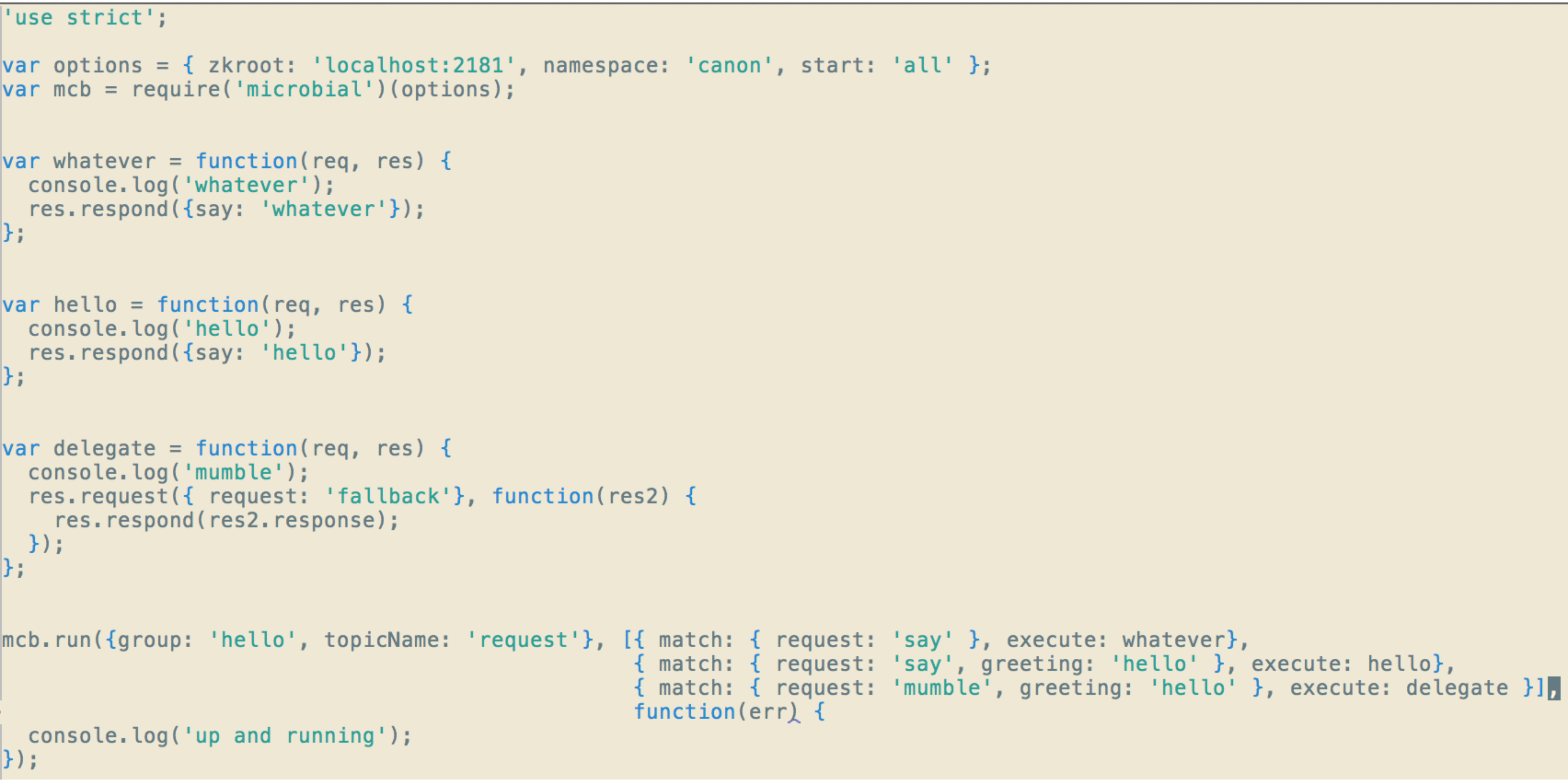
```
'use strict';

var options = { zkroot: 'localhost:2181', namespace: 'canon', start: 'all' };
var mcb = require('microbial')(options);
var reqSlot;

mcb.setup(function(err) {
  if (err) { console.log(err); }
  mcb.register({group: 'canonicalProducer', topicName: 'response', responseChannel: true},
               function(err, slot) {
    if (err) { return console.log(err); }

    reqSlot = slot;
    setInterval(function() {
      console.log('request');
      mcb.request({topicName: 'request'}, {request: 'say'}, function(res) {
        console.log('response: ' + res);
      });
    }, 1000);
  });
});
```

Micro-Service Example



```
'use strict';

var options = { zkroot: 'localhost:2181', namespace: 'canon', start: 'all' };
var mcb = require('microbial')(options);


var whatever = function(req, res) {
  console.log('whatever');
  res.respond({say: 'whatever'});
};


var hello = function(req, res) {
  console.log('hello');
  res.respond({say: 'hello'});
};


var delegate = function(req, res) {
  console.log('mumble');
  res.request({ request: 'fallback'}, function(res2) {
    res.respond(res2.response);
  });
};


mcb.run({group: 'hello', topicName: 'request'}, [{ match: { request: 'say' }, execute: whatever},
                                                 { match: { request: 'say', greeting: 'hello' }, execute: hello},
                                                 { match: { request: 'mumble', greeting: 'hello' }, execute: delegate }],
                                                 function(err) {
  console.log('up and running');
});
```

Ongoing…

- Track and update to 0.9
- Fold back learning from live development projects into infrastructure
- Build and develop infrastructural services to ease future development

Thank You !

Questions ?

Notes

Microservice Pathologies

- Broker dies
  - Problem: If the broker dies then there is no queue to post or receive messages from
  - Solution: Kafka is distributed and fault tolerant, each partition is replicated if a single node dies a new partition leader is elected and the system will continue to operate

Microservice Pathologies

- Consumer dies
  - Problem: If a consumer dies whilst processing a message, that message is lost.
  - Solution: Consumers will run as managed services. Kafka consumers must explicitly update the commit log with their position in the stream, this is done on a per consumer group basis. If the commit log is not updated due to consumer failure then the message is not lost and will be picked up on consumer restart.

Microservice Pathologies

- Consumer die repeatedly
  - Problem: The message causes the consumer to crash every time.
  - Solution: Consumers will run as managed services and will try and reprocess a message from the previous position in the commit log. This patology is solved by adding a maximum try count to the message, if the count is exceeded the consumer will discard the message – typically move it to the failed message store

Microservice Pathologies

- Producer dies
  - Problem: The producer dies.
  - Solution: Response messages from consumers wil be stored in Kafka, on producer restart responses will be picked up and processed by the producer.

Microservice Pathologies

- Consumer dies in call sequence
  - Problem: Consider the following: data save service and an indexing service. Which execute in sequence. Data is saved but the service dies before making a call to the indexer
  - Solution: Each service must send a response message over the bus. These responses will sit in the commit log and can be read by a watchdog process that will pair up requests with expected responses. Uncompleted sequences can be flagged and rerun to complete indexing.