

Go Big With Apache Kafka

Lorna Mitchell, Aiven



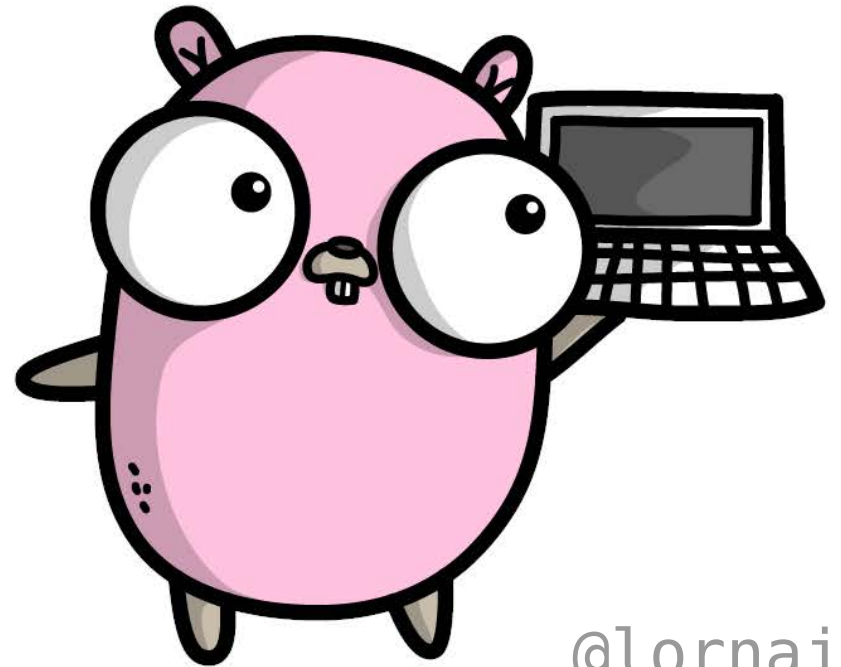
Apache Kafka

"Apache Kafka is an open-source distributed event streaming platform" - <https://kafka.apache.org>

- Designed for data streaming
- Real-time data for finance and industry
- Very scalable to handle large datasets



Modern, scalable, and fast remind you of anything?



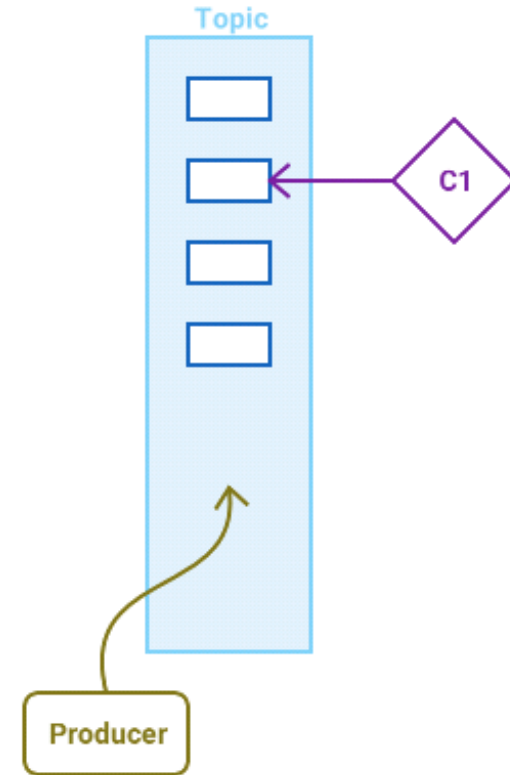
@lornajane

Kafka is a Log

We know about logs

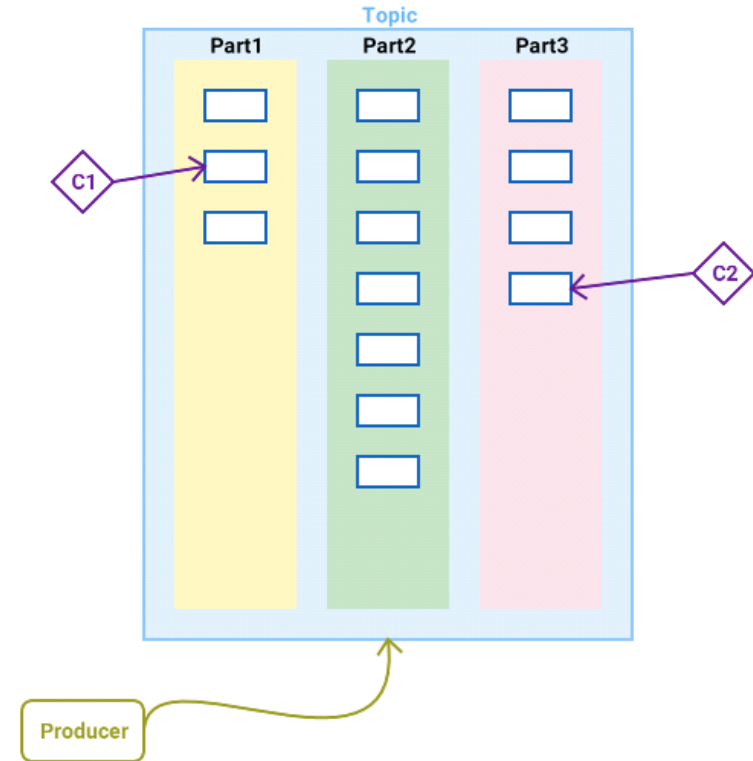
- append-only
- immutable

Producers send data.
Consumers fetch it.



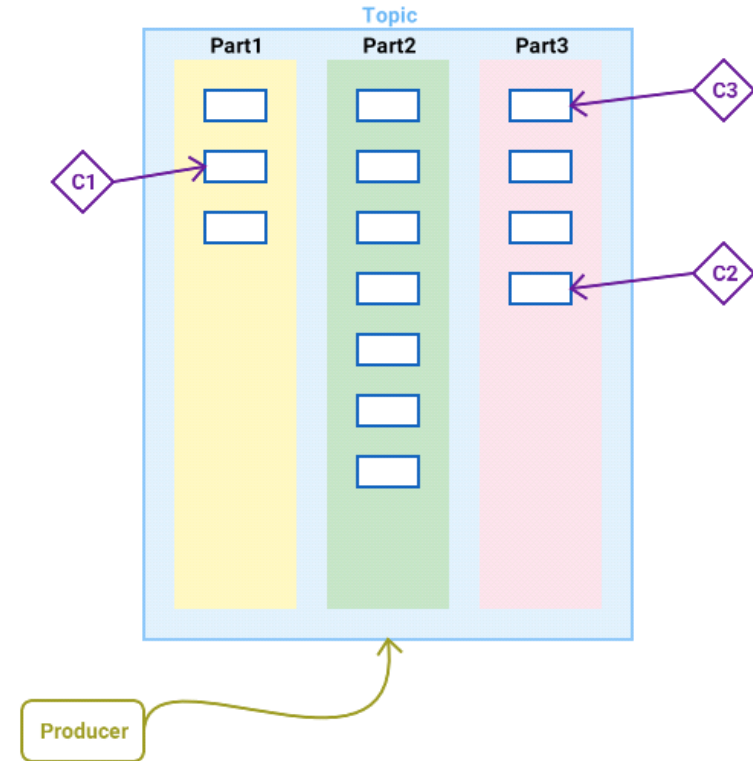
Topics and Partitions

- Topic is sharded across partitions
- Partition defined by key (usually)
- One consumer per consumer group per topic partition



Consumer Groups

- Different apps can read the same data
- Consumers belong to a "consumer group"
- One consumer per consumer group per topic partition



Replication Factors

- A topic has a "replication factor", this is how many copies of it will exist.
- Replication works at the partition level.



Let's Go

<https://github.com/aiven/thingum-industries/>



Let's Go

Three libraries in today's examples:

- `confluentinc/confluent-kafka-go`
- `riferrei/srclient`
- `actgardner/gogen-avro`



Example: Factory Sensors

Imaginary IoT application "Thingum Industries"

Which machine? What reading?

```
{  
  "machine": "MagicMaker3000",  
  "sensor": "oven_temp",  
  "value": 231,  
  "units": "C"  
}
```



@lornajane

Example: Producer

Connect to the broker

```
1      p, err := kafka.NewProducer(&kafka.ConfigMap{
2          "bootstrap.servers":      os.Getenv("BROKER_URI"),
3          "security.protocol":      "SSL",
4          "ssl.ca.location":        "../ca.pem",
5          "ssl.certificate.location": "../service.cert",
6          "ssl.key.location":       "../service.key",
7      })
8      if err != nil {
9          panic(err)
10     }
11     defer p.Close()
```



Example: Producer

Send a record to a topic

```
1      mySensorReading := avro.MachineSensor{
2          Machine: "MagicMaker4000",
3          Sensor:  "oven_temp",
4          Value:   (100*rand.Float32() + 150),
5          Units:   "C",
6      }
7      reading, _ := json.Marshal(mySensorReading)
8
9      p.Produce(&kafka.Message{
10         TopicPartition: kafka.TopicPartition{
11             Topic: &topic, Partition: kafka.PartitionAny},
12         Value: reading}, nil)
```



Example: Consumer

Read from a topic

```
1  c, err := kafka.NewConsumer(&kafka.ConfigMap{
2      "bootstrap.servers":      os.Getenv("BROKER_URI"),
3      "group.id":                "CG1",
4      "auto.offset.reset":      "earliest",
5  })
6  c.SubscribeTopics([]string{topic}, nil)
7
8  for {
9      msg, _ := c.ReadMessage(-1)
10     fmt.Printf("Message on %s: %s\n", msg.TopicPartition, string(msg.Value))
11 }
12
13 c.Close()
```



Kafka Tooling

Kafka itself ships with some useful shell scripts



Kafkacat: CLI Tool

<https://github.com/edenhill/kafkacat>

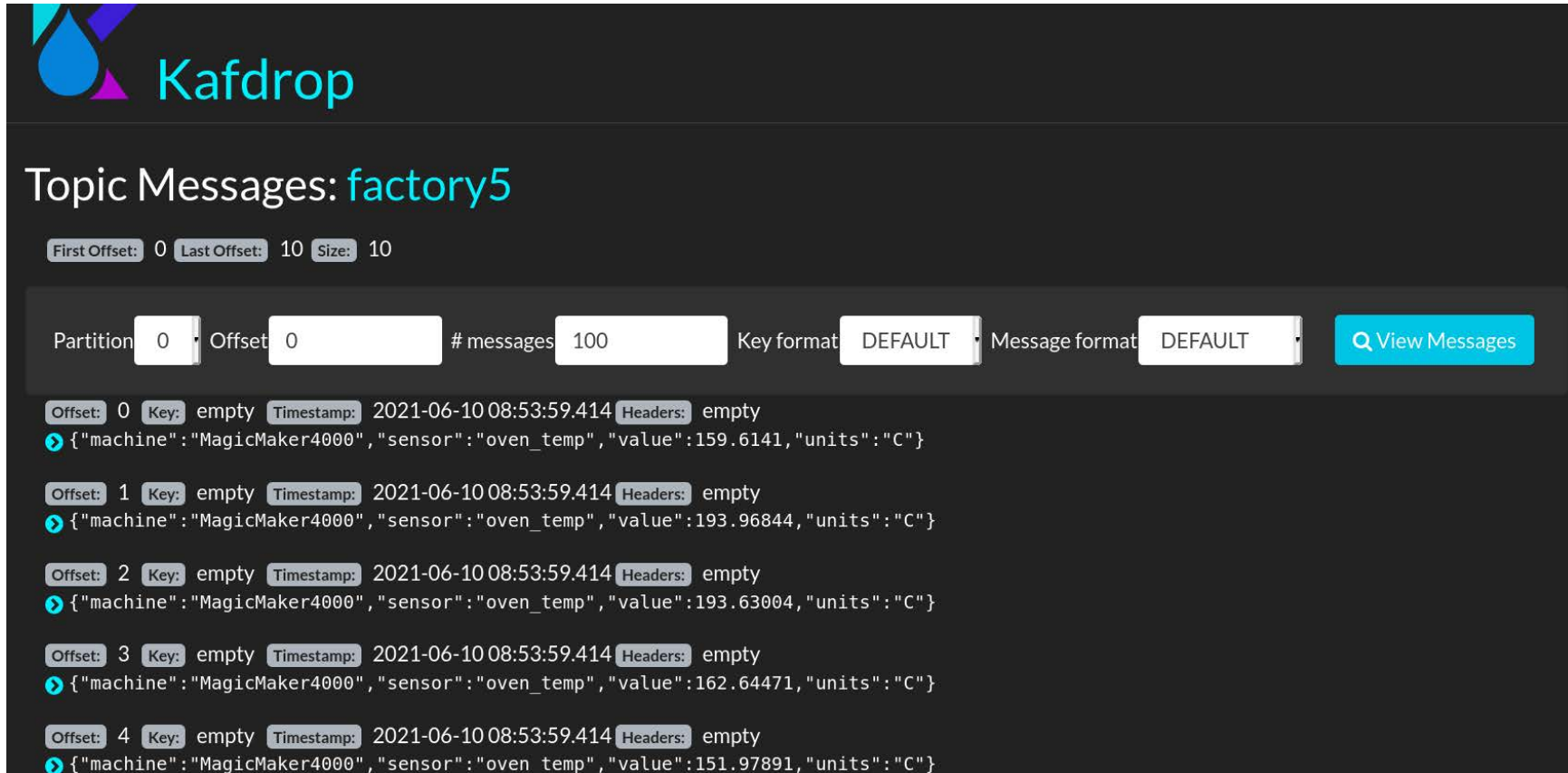
```
lornajane@fedora:~/thingum — kafkacat -F kafkacat.config -C -t factory5
lornajane@localhost:~/thingum [main]$ kafkacat -F kafkacat.config -C -t factory5
% Reading configuration from file kafkacat.config
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 159.6141, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 193.96844, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 193.63004, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 162.64471, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 151.97891, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 241.59236, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 186.5212, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 152.0553, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 230.40985, "units": "C"}
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 166.84143, "units": "C"}
% Reached end of topic factory5 [0] at offset 10
```



@lornajane

Kafdrop: Web UI

<https://github.com/obsidiandynamics/kafdrop>



The screenshot shows the Kafdrop web interface. At the top left is the Kafdrop logo, which consists of a stylized blue and purple 'K' next to the word 'Kafdrop' in a teal font. Below the logo, the text 'Topic Messages: factory5' is displayed. Underneath this, there are three small grey boxes: 'First Offset: 0', 'Last Offset: 10', and 'Size: 10'. A horizontal bar contains several input fields and buttons: 'Partition' with a dropdown set to '0', 'Offset' with a text input set to '0', '# messages' with a text input set to '100', 'Key format' with a dropdown set to 'DEFAULT', 'Message format' with a dropdown set to 'DEFAULT', and a blue button labeled 'View Messages' with a magnifying glass icon. Below this bar, five message entries are listed. Each entry has a blue play button icon, followed by its details: 'Offset', 'Key', 'Timestamp', and 'Headers'. The messages are JSON objects representing temperature sensor data from a 'MagicMaker4000'.

Kafdrop

Topic Messages: factory5

First Offset: 0 Last Offset: 10 Size: 10

Partition 0 Offset 0 # messages 100 Key format DEFAULT Message format DEFAULT View Messages

Offset: 0 Key: empty Timestamp: 2021-06-10 08:53:59.414 Headers: empty
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 159.6141, "units": "C"}

Offset: 1 Key: empty Timestamp: 2021-06-10 08:53:59.414 Headers: empty
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 193.96844, "units": "C"}

Offset: 2 Key: empty Timestamp: 2021-06-10 08:53:59.414 Headers: empty
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 193.63004, "units": "C"}

Offset: 3 Key: empty Timestamp: 2021-06-10 08:53:59.414 Headers: empty
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 162.64471, "units": "C"}

Offset: 4 Key: empty Timestamp: 2021-06-10 08:53:59.414 Headers: empty
{"machine": "MagicMaker4000", "sensor": "oven_temp", "value": 151.97891, "units": "C"}



@lornajane

Cloud Hosted Tools

messages 1-10 of 15 · page 1

Meta	Key	Value
OFFSET: +14 PARTITION: 0		<pre>{ "machine": "MagicMaker4000", "sensor": "oven_temp", "units": "C", "value": 174.65327 }</pre>
OFFSET: +13 PARTITION: 0		<pre>{ "machine": "MagicMaker4000", "sensor": "oven_temp", "units": "C", "value": 224.42029 }</pre>
OFFSET: +12 PARTITION: 0		<pre>{ "machine": "MagicMaker4000", "sensor": "oven_temp", "units": "C", "value": 176.1372 }</pre>
OFFSET: +11 PARTITION: 0		<pre>{ "machine": "MagicMaker4000", "sensor": "oven_temp", "units": "C", "value": 164.48169 }</pre>



Kafka and Schemas



Schemas

Schemas are great!

- specify and enforce data format
- required by compression formats, e.g. Protobuf, Avro

Our favourite strongly-typed language really likes schemas



Avro Schemas

- Avro format requires a schema
 - message has schema version information
 - used to look up fieldnames and reconstruct payload
- Schema Registry holds the schema versions for each topic



Evolving Schemas

- Aim for backwards-compatible changes
 - to rename: add the new field, keep the old one
 - safe to add optional fields
- Each change is a new version
- Avro supports aliases and default values



Example: Avro Schema

Avro schema example for sensor data

```
{
  "namespace": "io.aiven.example",
  "type": "record",
  "name": "MachineSensor",
  "fields": [
    {
      "name": "machine", "type": "string",
      "doc": "The machine whose sensor this is"
    },
    {
      "name": "sensor", "type": "string", "doc": "Which sensor was read"
    },
    {
      "name": "value", "type": "float", "doc": "Sensor reading"
    },
    {
      "name": "units", "type": "string", "doc": "Measurement units"
    }
  ]
}
```



GoGen makes Avro Structs

gogen-avro avro machine_sensor.avsc

```
type MachineSensor struct {  
    // The machine whose sensor this is  
    Machine string `json:"machine"`  
    // Which sensor was read  
    Sensor string `json:"sensor"`  
    // Sensor reading  
    Value float32 `json:"value"`  
    // Measurement units  
    Units string `json:"units"`  
}
```



Example: Avro Producer

Add awareness of the schema registry, and turn the struct into bytes to add into the payload...

```
1  schemaRegistryClient := srclient.CreateSchemaRegistryClient(uri)
2  schema, err := schemaRegistryClient.GetLatestSchema(topic, false)
3
4  mySensorReading := avro.MachineSensor{
5      Machine: "MagicMaker4000",
6      Sensor:  "oven_temp",
7      Value:   (100*rand.Float32() + 150),
8      Units:   "C", }
9
10 var valueBytesBuffer bytes.Buffer
11 mySensorReading.Serialize(&valueBytesBuffer)
12 valueBytes := valueBytesBuffer.Bytes()
```



Example: Avro Producer

... then add the schema info, assemble and send.

```
1  schemaIDBytes := make([]byte, 4)
2  binary.BigEndian.PutUint32(schemaIDBytes, uint32(schema.ID()))
3
4  var recordValue []byte
5  recordValue = append(recordValue, byte(0))
6  recordValue = append(recordValue, schemaIDBytes...)
7  recordValue = append(recordValue, valueBytes...)
8
9  p.Produce(&kafka.Message{
10      TopicPartition: kafka.TopicPartition{
11          Topic: &topic, Partition: kafka.PartitionAny},
12      Value: recordValue, nil})
```



Describing Payloads



AsyncAPI for Apache Kafka

AsyncAPI describes event-driven architectures

<https://www.asyncapi.com>

We can describe the:

- brokers and auth
- topics
- payloads



@lornajane

Describing Payloads

The `channels` section of the AsyncAPI document

```
factorysensor:
  subscribe:
    operationId: MachineSensor
    summary: Data from the in-machine sensors
    bindings:
      kafka:
        clientId:
          type: string
    message:
      name: sensor-reading
      title: Sensor Reading
      schemaFormat: "application/vnd.apache.avro;version=1.9.0"
      payload:
        $ref: machine_sensor.avsc
```



Documenting Payloads

localhost:9092 KAFKA 1.0.0 DEVELOPMENT

Operations

SUB door - sensor

Door sensors (external and internal)

Open/closed state information from the doors.

Operation Bindings >

#sensor

Accepts the following message:

Door Sensor Reading door-sensor-data

Door sensor data

Payload ^

Object

location String

state

Allowed values: "open" "closed"

Additional properties are allowed.

Examples

Payload ^

Example #1

```
{  "location": "Car park",  "state": "open"}
```

Example #2

```
{  "location": "Roof-level fire exit",  "state": "closed"}
```



Go with Apache Kafka



Resources

- Repo: <https://github.com/aiven/thingum-industries>
- Aiven: <https://aiven.io> (free trial!)
- Karapace: <https://karapace.io>
- AsyncAPI: <https://asyncapi.com>
- Me: <https://lornajane.net>

