Fish and Chips and Apache Kafka®

By Tibs (they / he)

Slides and source code at
https://github.com/aiven-labs/fish-and-chips-and-kafka

Tutorial to come...

tony.ibbs@aiven.io / https://aiven.io/tibs / @much_of_a

What we'll cover

- Me and messaging and Apache Kafka®
- Fish and chips
    - How to talk to Kafka
    - Start with a simple model and work up
    - There's a demo you can play with afterwards
    - Some ideas for things you can do to extend the demos

Some message problems I've cared about

- between components on a Set Top Box
- configuration between microservices
- to / from Internet of Things devices, and their support systems

Kafka is a very good fit for the IoT cases, maybe less so for the others

What I want from messaging

- multiple producers *and* multiple consumers
- single delivery
- guaranteed delivery
- resumes safely if system crashes
- no back pressure handling (queue does not fill up)

Enter, Apache Kafka®

Kafka terms

Messages are *Events*

Producers send messages, *Consumers* read them.

Can have multiple Producers and Consumers

A Producer sends messages to one or more named *Topics*

Each Consumer reads from one or more Topics

Partitions can be used to "spread the load" within a Topic

Producers, topics, consumers

Events

Multiple producers, multiple consumers

Multiple partitions, consumer groups

Let's model a fish-and-chip shop

We start with a shop that

- just handles cod and chips
- which are always ready to be served

Glossary

- **Cod**: the traditional white fish for english fish-and-chip shops
- **Chips**: fatter, possibly soggier, french fries
- **Plaice**: a flat fish
- **Till**: a cash register

Serving a customer

An order

```
{
   "order": 271,
   "parts": [
       ["cod", "chips"],
       ["chips", "chips"],
   ]
}
```

Demo: 1 till, 1 food preparer

```
Till
Order 1: cod & chips
Order 2: chips
Order 3: chips
Order 4: chips
Order 5: cod & chips
Order 6: cod & chips, large chips
Order 7: cod & chips
Order 8: chips
Order 9: cod & chips, large chips
Order 10: cod & chips
Order 11: cod & chips
Order 12: cod & chips, large chips
Order 13: cod & chips, large chips
```

```
Food Preparer
Order ready 1: cod & chips
Order ready 2: chips
Order ready 3: chips
Order ready 4: chips
Order ready 5: cod & chips
Order ready 6: cod & chips, large chips
Order ready 7: cod & chips
Order ready 8: chips
Order ready 9: cod & chips, large chips
Order ready 10: cod & chips
Order ready 11: cod & chips
Order ready 12: cod & chips, large chips
Order ready 13: cod & chips, large chips
```

Libraries

To talk to Kafka and manage topics:

- `aiokafka`: https://github.com/aio-libs/aiokafka
- `kafka-python`: https://github.com/dpkp/kafka-python

To create our Terminal UI

- `Textual`: https://github.com/Textualize/textual
- `Rich`: https://github.com/Textualize/rich

Code: Create an SSL context

```
import aiokafka.helpers

context = aiokafka.helpers.create_ssl_context(
    cafile=CERTS_DIR / "ca.pem",
    certfile=CERTS_DIR / "service.cert",
    keyfile=CERTS_DIR / "service.key",
)
```

Code: Asynchronous Producer

```
from aiokafka import AIOKafkaProducer

producer = aiokafka.AIOKafkaProducer(
    bootstrap_servers=f"{HOST}:{SSL_PORT}",
    security_protocol="SSL",
    ssl_context=context,
    value_serializer=lambda v: json.dumps(v).encode('ascii'),
)

await producer.start()

while SHOP_IS_OPEN:
    await producer.send('ORDERS', message)
```

Code: Asynchronous Consumer

```
consumer = aiokafka.AIOKafkaConsumer(
    'ORDERS',
    bootstrap_servers=f"{HOST}:{SSL_PORT}",
    security_protocol="SSL",
    ssl_context=context,
    value_deserializer = lambda v: json.loads(v.decode('ascii')),
)

await consumer.start()

async for message in consumer:
    print(f'Received {message.value}')
```

More customers - add more TILLs

Customers now queue at multiple TILLs, each TILL is a Producer.

Three tills

An order with multiple TILLs

```
{
   "order": 271,
   "till": 3,
   "parts": [
       ["cod", "chips"],
       ["chips", "chips"],
   ]
}
```

How we alter the code

When creating the topic for the demo, request 3 partitions:

```
NewTopic(
    name='DEMO2-ORDERS',
    num_partitions=3,
    replication_factor=1,
)
```

Create 3 Till producers instead of 1

Demo: 3 TILLs, 3 partitions, 1 PREPARER

```
Till 1
Order 8: cod & chips
Order 11: cod & chips
Order 14: cod & chips
Order 17: cod & chips
Order 20: cod & chips, large chips
Order 23: chips
Order 26: cod & chips

Till 2
Order 9: cod & chips
Order 12: cod & chips
Order 15: chips
Order 18: cod & chips
Order 21: cod & chips
Order 24: cod & chips, large chips
Order 27: cod & chips

Till 3
Order 10: cod & chips
Order 13: cod & chips, large chips
Order 16: chips
Order 19: cod & chips
Order 22: chips
Order 25: cod & chips
Order 28: cod & chips
```

```
Food Preparer
Order ready 1: chips
Order ready 2: chips
Order ready 3: chips
Order ready 4: cod & chips
Order ready 5: cod & chips
Order ready 6: cod & chips
Order ready 7: cod & chips
Order ready 8: cod & chips
Order ready 9: cod & chips
Order ready 10: cod & chips
Order ready 11: cod & chips
Order ready 12: cod & chips
Order ready 13: cod & chips, large chips
Order ready 14: cod & chips
Order ready 15: chips
Order 16: chips
```

Add multiple *consumers*

How we alter the code

Create 2 Food preparer consumers instead of 1

Consumers need to be in same *consumer group*

```
consumer = aiokafka.AIOKafkaConsumer(
    ...
    group_id=CONSUMER_GROUP,
    ...
```

Start consuming from a specific offset

If I run a demo more than once, there's a chance that a consumer might receive events from the previous demo. So we want to make sure that doesn't happen.

Various solutions - simplest for this case is to do:

```
await consumer.seek_to_end()
```

Sending to different partitions

```
await producer.send(TOPIC_NAME, value=order)
```

```
await producer.send(TOPIC_NAME, value=order, key='till')
```

```
await producer.send(TOPIC_NAME, value=order, partition=till_number-1)
```

Demo: 3 TILLs, 3 partitions, 2 PREPARERS

```
Till 1
Order 10: cod & chips
Order 13: cod & chips
Order 16: cod & chips
Order 20: cod & chips
Order 22: cod & chips, large chips
Order 26: cod & chips
Order 28: cod & chips

Till 2
Order 11: cod & chips
Order 14: cod & chips, large chips
Order 17: cod & chips
Order 19: cod & chips
Order 23: cod & chips
Order 25: chips
Order 29: cod & chips

Till 3
Order 12: cod & chips
Order 15: cod & chips
Order 18: cod & chips
Order 21: cod & chips
Order 24: cod & chips
Order 27: cod & chips
Order 30: chips
```

```
Food Preparer 1
Everyone is ready
Order ready 3: cod & chips
Order ready 5: cod & chips
Order ready 8: chips
Order ready 11: cod & chips
Order ready 14: cod & chips, large chips
Order ready 17: cod & chips
Order ready 19: cod & chips
Order ready 23: cod & chips
Order ready 25: chips
Order 29: cod & chips

Food Preparer 2
Order ready 6: cod & chips
Order ready 7: cod & chips
Order ready 9: cod & chips
Order ready 10: cod & chips
Order ready 12: cod & chips
Order ready 13: cod & chips
Order ready 16: cod & chips
Order ready 15: cod & chips
Order ready 18: cod & chips
Order ready 20: cod & chips
Order ready 21: cod & chips
Order 24: cod & chips
```

Web console

tibs-kafka-fish
Apache Kafka 3.4.1
EOL : 2024-05-13 : OK
Running
Nodes 3
Open support ticket
Delete service
Power off service
Overview
Integrations
Metrics
Logs
Users
Access Control List (ACL)
Quotas
Topics
Backups
Connectors
Connection information
Apache Kafka
Apache Kafka Connect
Apache Kafka REST
Schema Registry
Quick connect
Authentication Method
Client Certificate
SASL
Service URI
tibs-kafka-fish-devrel-tibs.aivencloud.com:10145

Topics in the web console

Topic	Partitions	Replication	Min. Insync Replicas	Retention Hours	Retention Bytes	Cleanup Policy	Status	
demo1-cod-and-chips	1	2	1	168 hours	unlimited	delete	ACTIVE	···
demo2-cod-and-chips	1	2	1	168 hours	unlimited	delete	ACTIVE	···
demo3-cod-and-chips	3	2	1	168 hours	unlimited	delete	ACTIVE	···
demo4-cook	1	2	1	168 hours	unlimited	delete	ACTIVE	···
demo4-orders	1	2	1	168 hours	unlimited	delete	ACTIVE	···

Demo 3 partition barchart

tony.ibbs@aiven.io / https://aiven.io/tibs / @much_of_a

Demo 3 consumer groups

Configuration | Partitions | Consumer groups | Tags

Consumer groups

Partition	Group	Offset	Lag
0	demo3-all-orders	767	166
1	demo3-all-orders	937	1
2	demo3-all-orders	797	138

Demo 3 metrics

Showing CPU usage, but there's also disk space usage, disk iops (read and write), 5 minute load average, memory usage, and network receive/transmit

Cod or plaice

Plaice needs to be cooked

So we need a COOK to cook it

Participant changes - add COOK

An order with plaice

```
{
   "order": 271,
   "till": 3,
   "parts": [
       ["cod", "chips"],
       ["chips", "chips"],
       ["plaice", "chips"],
   ]
}
```

Gets turned into...

```
{
   "order": 271,
   "till": 3,
   "parts": [
       ["cod", "chips"],
       ["chips", "chips"],
       ["plaice", "chips"],
   ],
   "ready": <boolean>
}
```

Code changes to the PREPARER

```
def all_order_available(self, order):
    if 'ready' not in order:
        all_items = itertools.chain(*order['order'])
        order['ready'] = 'plaice' not in all_items
    return order['ready']
```

```
order_available = self.all_order_available(order)
if not order_available:
    await self.producer.send(COOK_TOPIC, order)
```

In the new COOK

```
async for message in consumer:
    ...
    # "Cook" the (plaice in the) order
    await asyncio.sleep(random.uniform(COOK_FREQ_MIN, COOK_FREQ_MAX))
    # It's important to remember to mark the order as ready now!
    # (forgetting to do that means the order will keep going round the loop)
    order['ready'] = True
    await self.producer.send(ORDERS_TOPIC, order)
```

Demo: with COOK

```
Till
Order 3: cod & chips
Order 4: cod & chips
Order 5: chips
Order 6: cod & chips
Order 7: cod & chips, plaice & chips
Order 8: cod & chips
Order 9: cod & chips, plaice & chips
Order 10: cod & chips, plaice & chips
Order 11: cod & chips, plaice & chips
Order 12: chips
Order 13: cod & chips, plaice & chips
```

```
Food Preparer
Order ready 5: ✓ chips
Order ready 6: ✓ cod & chips
COOK  7: cod & chips, plaice & chips
Order ready 8: ✓ cod & chips
COOK  9: cod & chips, plaice & chips
COOK  10: cod & chips, plaice & chips
Order ready 7: ✓ cod & chips, plaice & chips
COOK  11: cod & chips, plaice & chips
Order ready 12: ✓ chips
Order ready 9: ✓ cod & chips, plaice & chips
COOK  13: cod & chips, plaice & chips
```

```
Cook
Cooked 7: ✓ cod & chips, plaice & chips
Order 7 available
Cooked 9: ✓ cod & chips, plaice & chips
Order 9 available
Cooking 10: cod & chips, plaice & chips
```

Summary so far

We know how to model the ordering and serving of our cod and chips

We know how to scale with multiple Producers and Consumers

We made a simple model for orders with plaice

Homework 1: Model cooking the fish and chips

Use a Redis cache to simulate contents of the hot cabinet

Redis has entries for the hot cabinet content, keyed by `cod`, (portions of) `chips` and `plaice`. We start with 0 for all of them.

Using the cache

PREPARER compares the order to the counts in the (hot cabinet) cache.

- If there's enough, decrement the cache appropriately, order's done
- If not, sends the order to the COOK

COOK updates the cache

- For `plaice`, adds as many as are needed
- For `cod` and `chips`, cook enough to stock the hot cabinet
- Then sends the order back to the [ORDER] topic

Homework 2: Adding an ANALYST

Using Kafka Connect

How I would do it

The Aiven developer documentation has instructions on how to do this at https://docs.aiven.io/docs/products/kafka/kafka-connect/howto/jdbc-sink.html

- Create an appropriate PostgreSQL database and table
- Make sure that the Kafka service has Kafka Connect enabled
- Use the Aiven web console to setup a JDBC sink connector to send events to PG

And then add code to the Python demo to query PostgreSQL and make some sort of report over time.

Final summary

We know how to model the ordering and serving of our cod and chips

We know how to scale with multiple Producers and Consumers

We made a simple model for orders with plaice

We talked briefly about how one might model the hot cabinet in more detail

We talked briefly about using Kafka Connectors to share data with other data users

Acknowledgements

Apache, Apache Kafka, Kafka, and the Kafka logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries

Postgres and PostgreSQL are trademarks or registered trademarks of the PostgreSQL Community Association of Canada, and used with their permission

Redis is a registered trademark of Redis Ltd. Any rights therein are reserved to Redis Ltd.

Fin

Get a free trial of Aiven services at
https://go.aiven.io/tibs-signup

Also, we're hiring! See https://aiven.io/careers

Written in reStructuredText, converted to PDF using
rst2pdf

Slides and accompanying material at
https://github.com/aiven-labs/fish-and-chips-and-kafka

tony.ibbs@aiven.io / https://aiven.io/tibs / @much_of_a