Fish and Chips and Apache Kafka®

By Tibs (they / he)

Slides and accompanying material at
https://github.com/tibs/fish-and-chips-and-kafka-talk

tony.ibbs@aiven.io / @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*

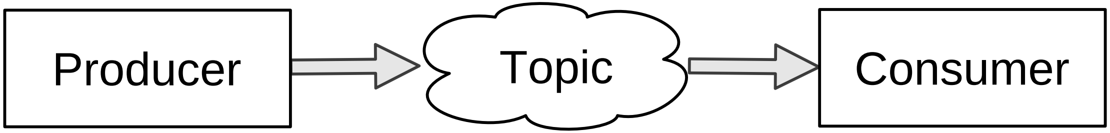
Producers send messages, *Consumers* read them.

Can have multiple Producers and Consumers

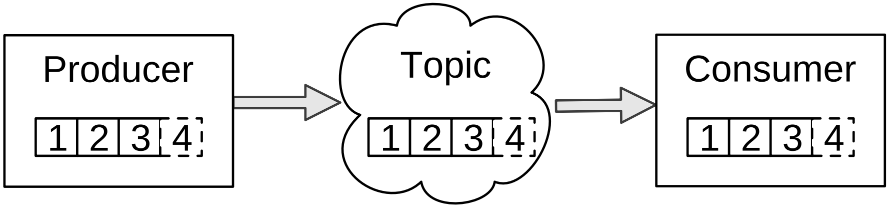
A Producer send a message to a named *Topic*, each Consumer reads from a single Topic

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

Producers, topics, consumers

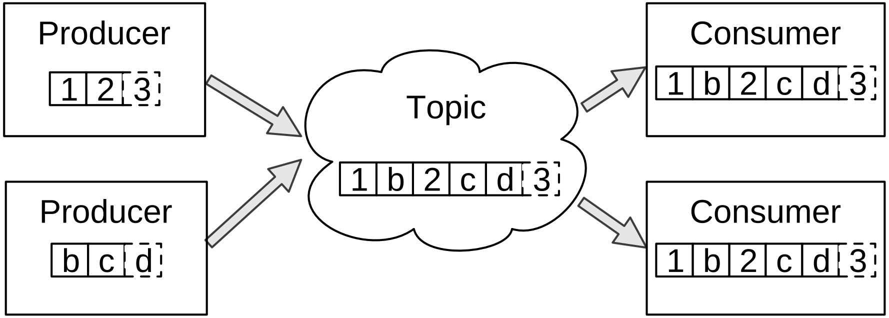


Events

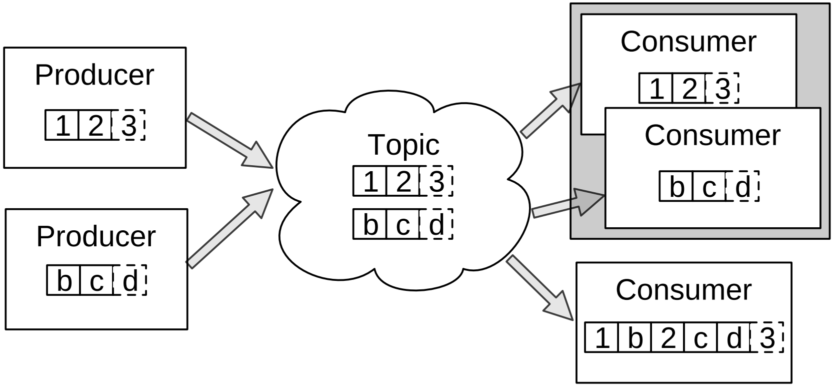


tony.ibbs@aiven.io / @much_of_a

Multiple produces, multiple consumers



Multiple partitions, consumer groups



tony.ibbs@aiven.io / @much_of_a

Let's model a fish-and-chip shop

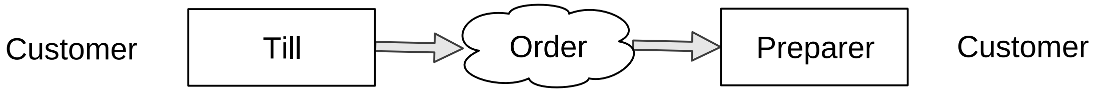
We start with a shop that

- 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"],
   ]
}
```

Show first demo

1 till, 1 food preparer

Libraries

`kafka-python`: https://github.com/dpkp/kafka-python

`aiokafka`: https://github.com/aio-libs/aiokafka

and

`Textual`: https://github.com/Textualize/textual

`Rich`: https://github.com/Textualize/rich

Code: Producer

```python
from kafka import KafkaProducer

producer = kafka.KafkaProducer(
    bootstrap_servers=f"{HOST}:{SSL_PORT}",
    security_protocol="SSL",
    ssl_cafile=f'{certs_dir}/ca.pem',
    ssl_certfile=f'{certs_dir}/service.cert',
    ssl_keyfile=f'{certs_dir}/service.key',
    value_serializer=lambda v: json.dumps(v).encode('ascii'),

while SHOP_IS_OPEN:
    producer.send('ORDER', order)
```

Code: Consumer

```python
from kafka import KafkaConsumer

consumer = KafkaConsumer(
    "ORDER",
    bootstrap_servers=f"{HOST}:{SSL_PORT}",
    security_protocol="SSL",
    ssl_cafile="ca.pem",
    ssl_certfile="service.cert",
    ssl_keyfile="service.key",
    value_deserializer = lambda v: json.loads(v.decode('ascii')),
)

for msg in consumer:
    print(f'Message {msg.value}')
```

Code: Asynchronous - needs 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

```python
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)
```

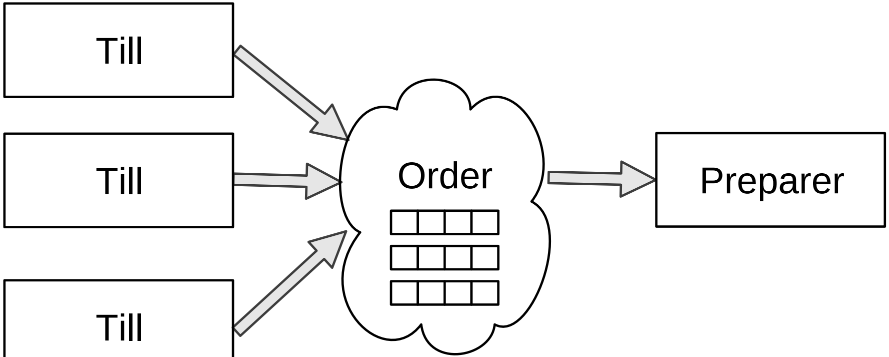
Code: Asynchronous Consumer

```
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:

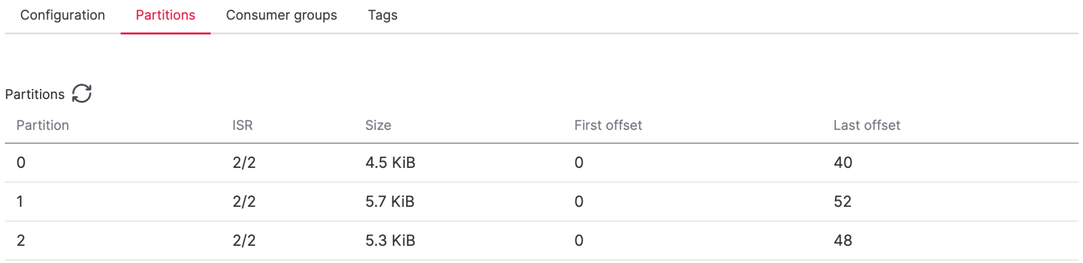
```
NewTopic(
    name='DEMO2-ORDERS',
    num_partitions=3,
    replication_factor=1,
)
```

Create 3 Till producers instead of 1

Show demo: multiple TILLs

Three tills, 3 partitions, 1 food preparer

Demo 2 partitions sizes

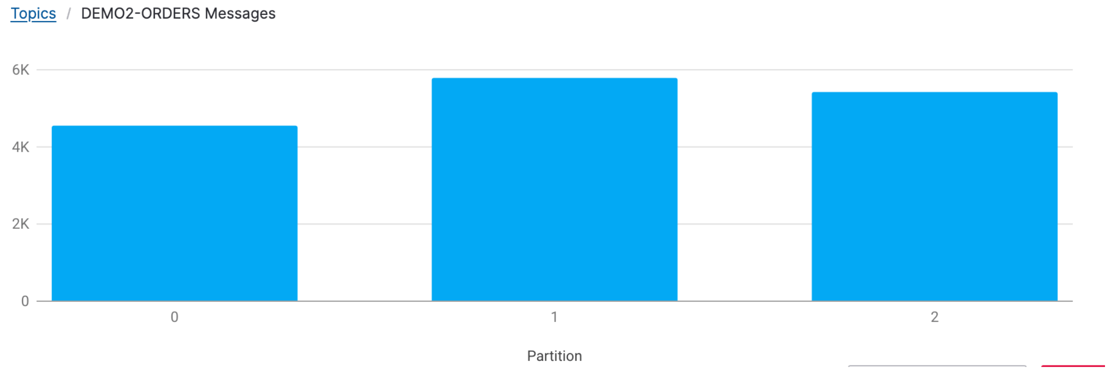


Partitions

Partition	ISR	Size	First offset	Last offset
0	2/2	4.5 KiB	0	40
1	2/2	5.7 KiB	0	52
2	2/2	5.3 KiB	0	48

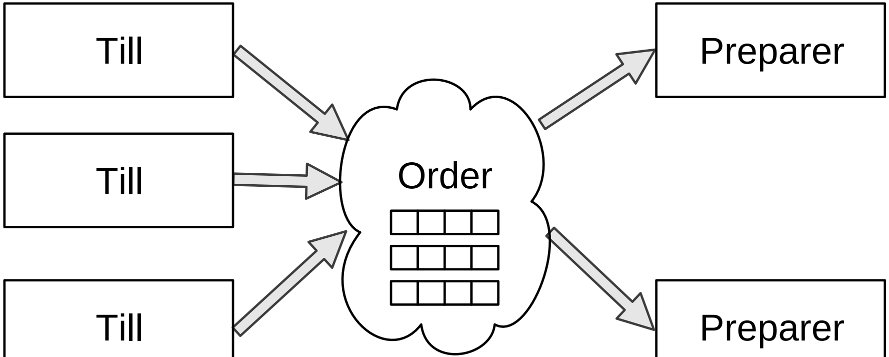
tony.ibbs@aiven.io / @much_of_a

Demo 2 partition barchart



tony.ibbs@aiven.io / @much_of_a

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)
```

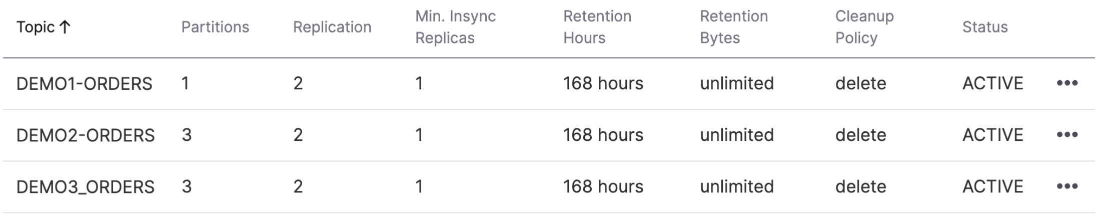
```
await producer.send(TOPIC_NAME, value=order, key='till')
```

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

Show demo: multiple TILLs and PREPARERS

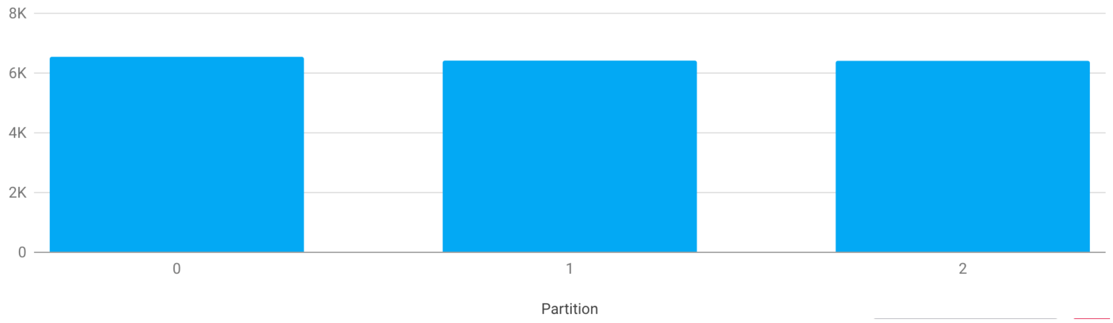
Three tills, 3 partitions, 2 food preparers

Demo 3 partitions sizes *Maybe*



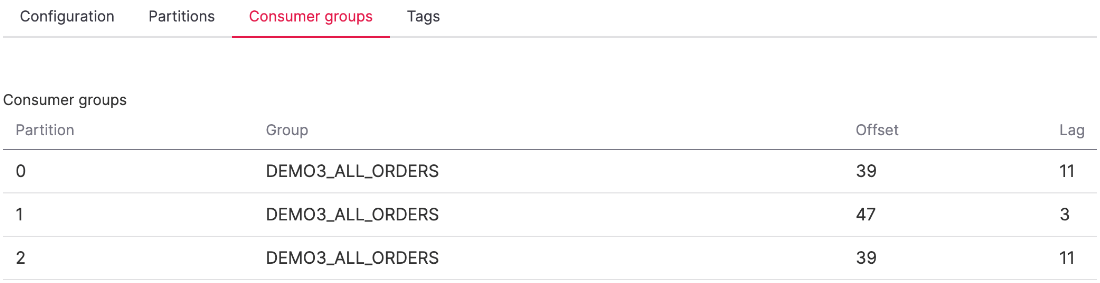
Topic ↑	Partitions	Replication	Min. Insync Replicas	Retention Hours	Retention Bytes	Cleanup Policy	Status	
DEMO1-ORDERS	1	2	1	168 hours	unlimited	delete	ACTIVE	•••
DEMO2-ORDERS	3	2	1	168 hours	unlimited	delete	ACTIVE	•••
DEMO3_ORDERS	3	2	1	168 hours	unlimited	delete	ACTIVE	•••

Demo 3 partition barchart *Maybe*



tony.ibbs@aiven.io / @much_of_a

Demo 3 consumer groups

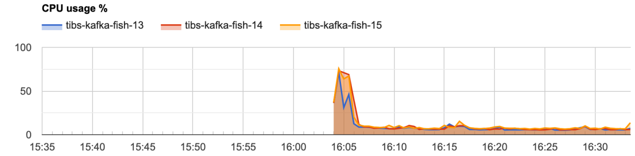


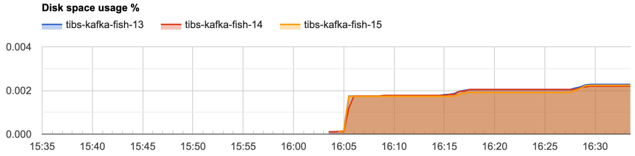
Configuration Partitions Consumer groups Tags

Consumer groups

Partition	Group	Offset	Lag
0	DEMO3_ALL_ORDERS	39	11
1	DEMO3_ALL_ORDERS	47	3
2	DEMO3_ALL_ORDERS	39	11

Demo 3 metrics





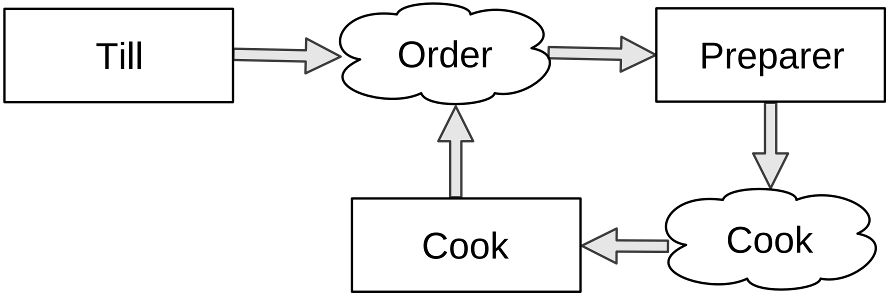
tony.ibbs@aiven.io / @much_of_a

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(TOPIC_NAME_COOK, 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(TOPIC_NAME_ORDERS, order)
```

Demo with COOK

1 till, 1 food preparer, 1 COOK

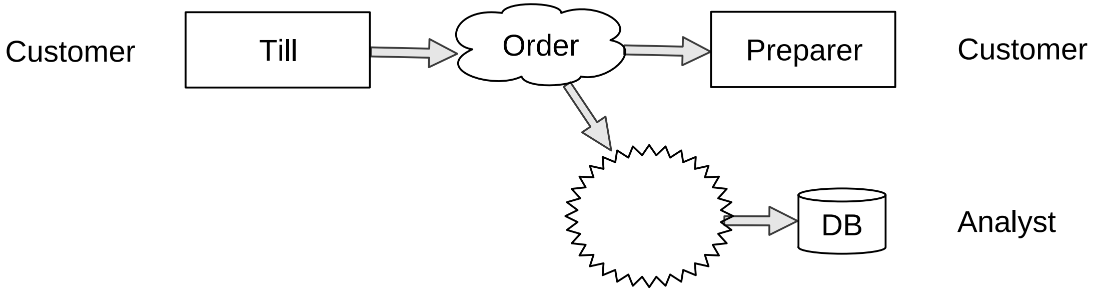
Summary so far

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

We know how to scale with multiple Producers and Consumers

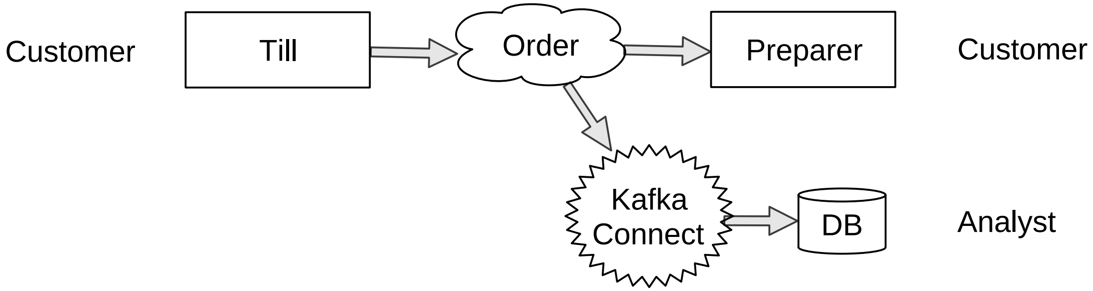
We made a simple model for orders with plaice

Homework 1: Adding an ANALYST



tony.ibbs@aiven.io / @much_of_a

Using Kafka Connect



tony.ibbs@aiven.io / @much_of_a

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.

Homework 2: 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 cache. If there's enough "stuff" to make the order up, decrements the cache appropriately, and that's done

If not, sends the order to the COOK

COOK updates the cache - for `plaice`, adds as many as are needed, for the others, if they go below a threshold, adds a standard quantity back in ("cooking in batches"). Then sends the order back to the [ORDER] topic

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 using Kafka Connectors to share data with other data users

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

tony.ibbs@aiven.io / @much_of_a

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://console.aiven.io/signup/email

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/tibs/fish-and-chips-and-kafka-talk

tony.ibbs@aiven.io / @much_of_a