EUROPYTHON
2021 Jul 26 - Aug 1 Online

HIGH PERFORMANCE DATA PROCESSING WITH PYTHON, KAFKA AND ELASTICSEARCH

EUROPYTHON ONLINE 2021

Harshit Prasad / harshitprasad.com

INTRODUCTION

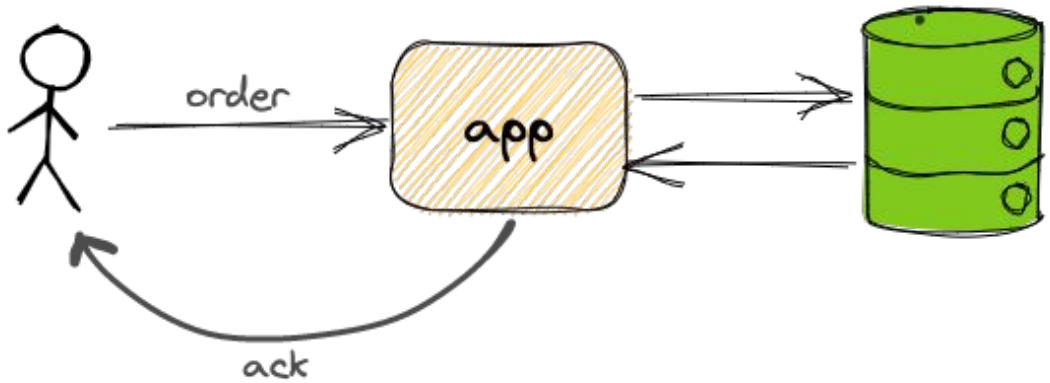
- Software Engineer at **Grofers - India**.
- Bachelors in Electronics and Communication Engineering from the **LNM Institute of Information Technology**
- Past GSoC'16 - **FOSSASIA**
- Past GSoC'17 - **CERN**
- Open Source Contributor.

TALK OUTLINE

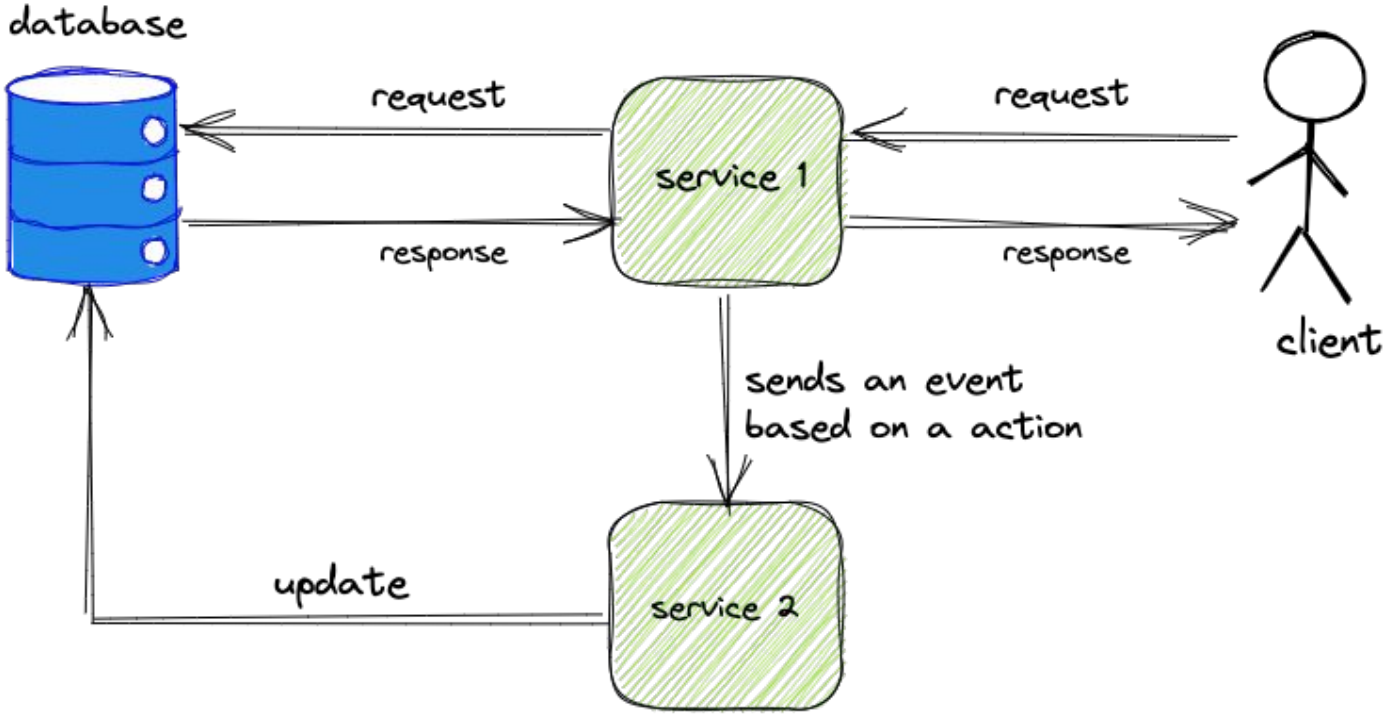
- Problem Statement
- Solution
- Workflow
- Code Walkthrough

PROBLEM STATEMENT

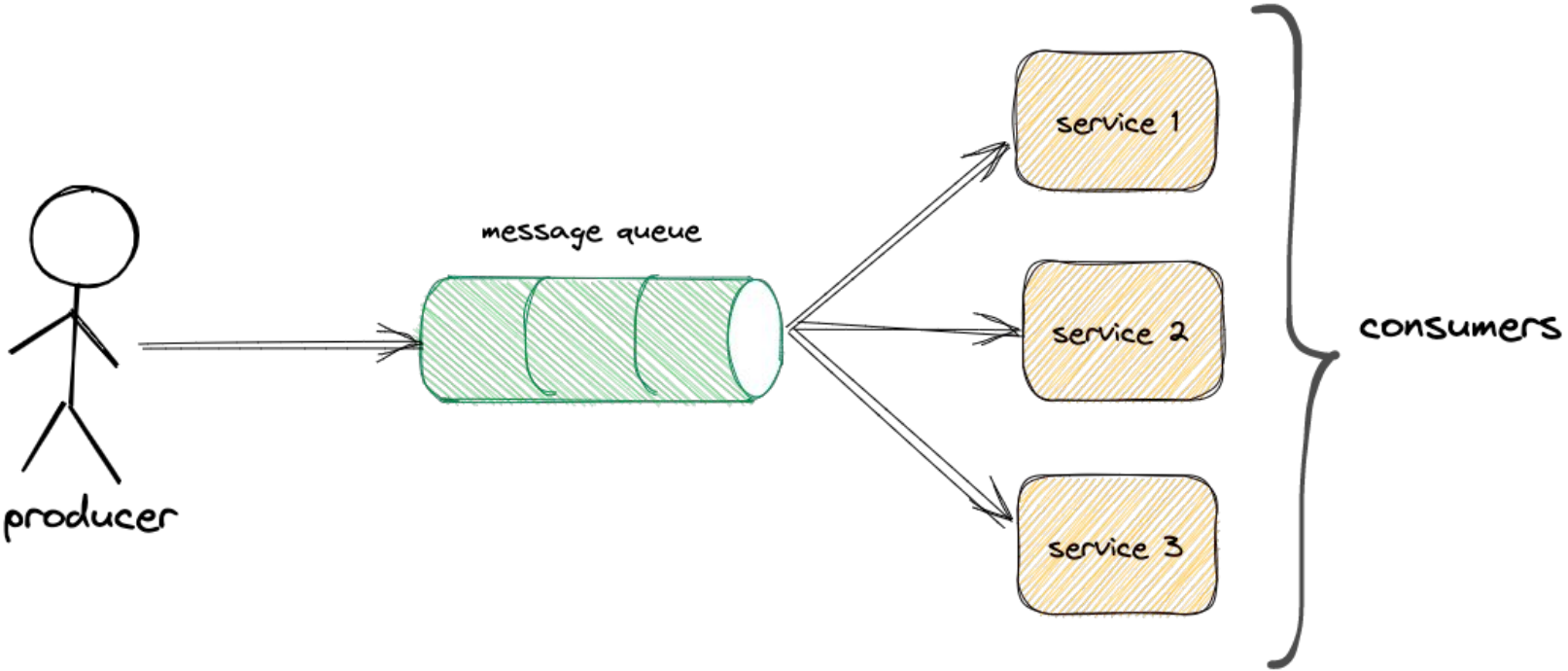
- Your own application
- Avg order rate X
- Peak hour order rate X * 1000



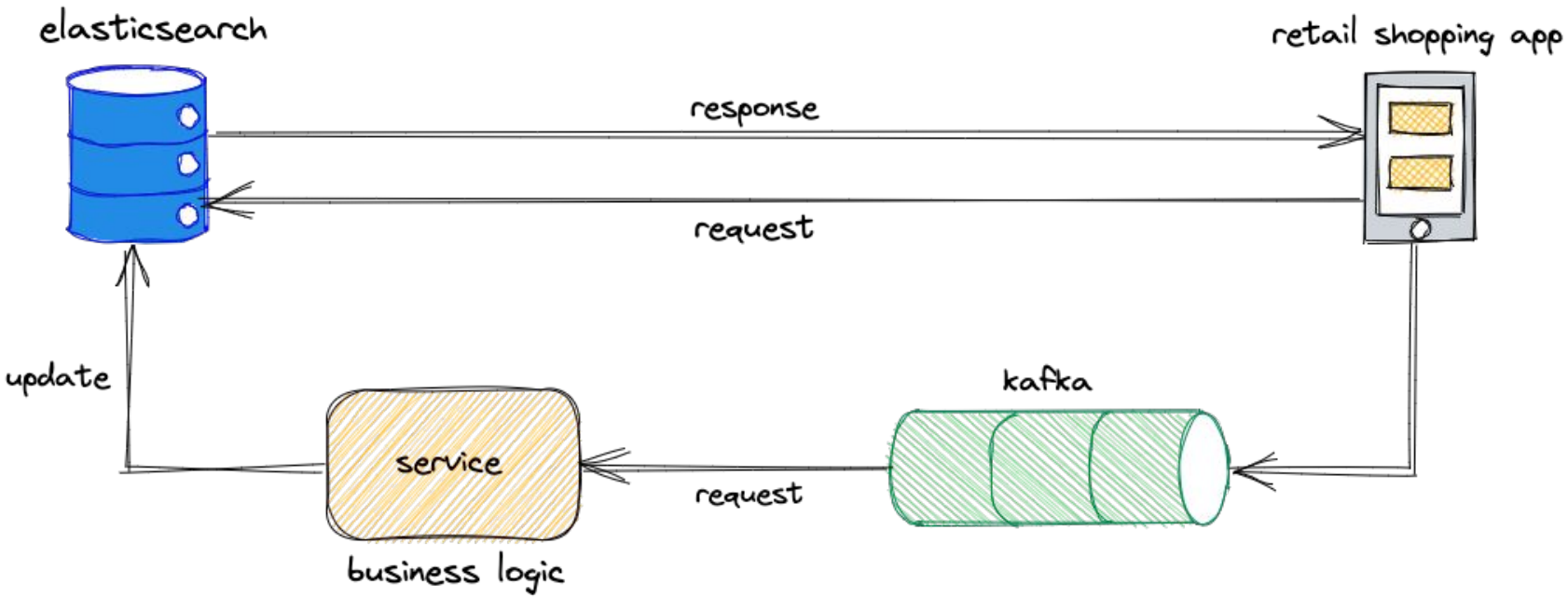
PROBLEM STATEMENT



PRODUCER CONSUMER MODEL



WORKFLOW


elasticsearch
retail shopping app
response
request
update
kafka
service
request
business logic

WHAT KIND OF DATA ARE WE UPDATING ?


```
{
  "_index" : "retail-catalogue",
  "_type" : "_doc",
  "_id" : "10",
  "_version" : 3,
  "_seq_no" : 4,
  "_primary_term" : 2,
  "found" : true,
  "_source" : {
    "product" : {
      "id" : 10,
      "name" : "Funny Farm House Ketchup",
      "category" : "Dips and Ketchups",
      "price" : 15.0,
      "stock" : 5
    },
    "update_ts" : 1625931472
  }
}
```

CODE WALKTHROUGH

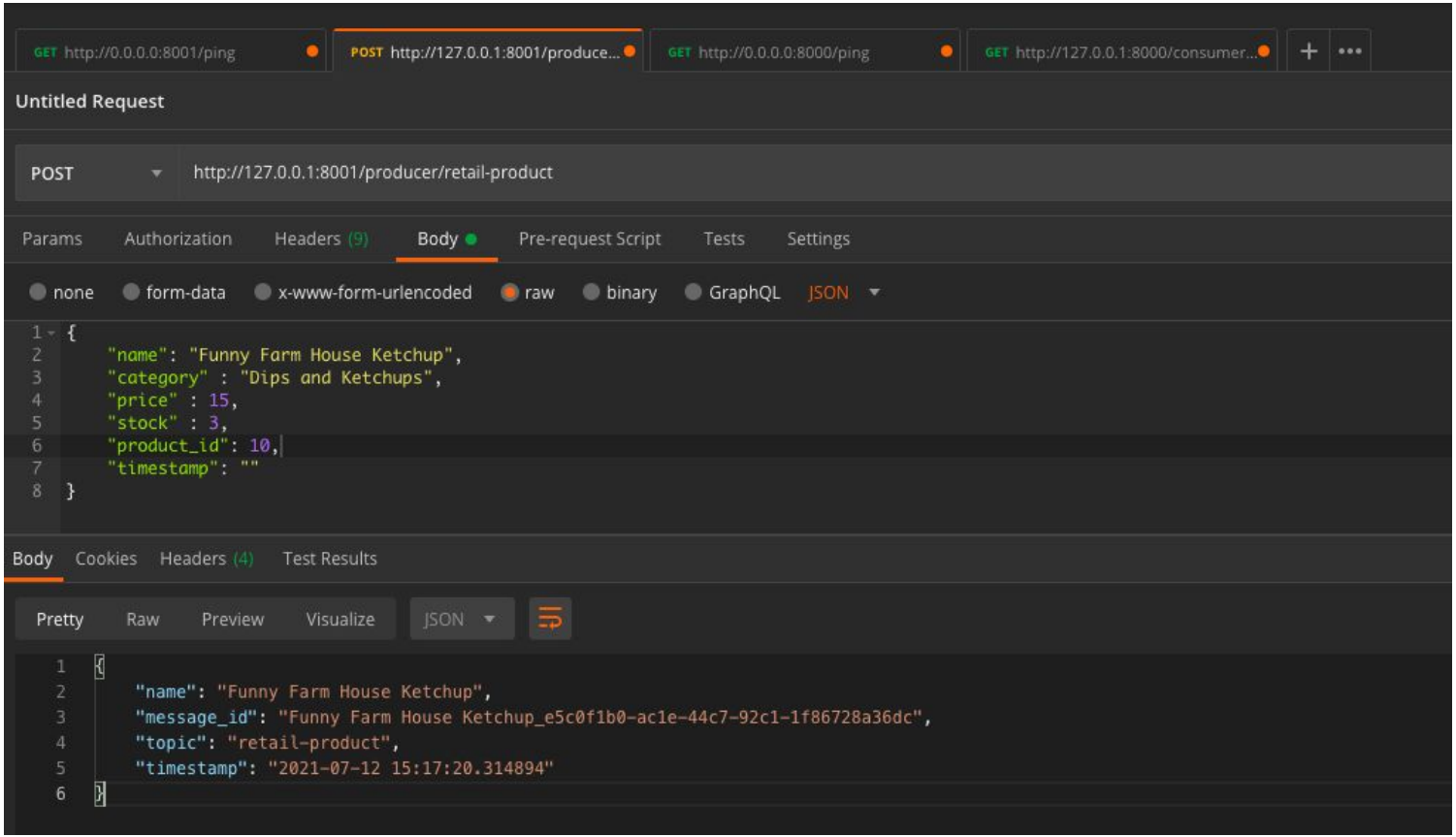
- Python based implementation of Producer Consumer model.
- Walkthrough of Producer Consumer implementation - like producing data, pushing it to Kafka and then again consuming from Kafka to send an update to the Elasticsearch datastore.
- Data Streamer:
  - Producer
  - Consumer

DATA PRODUCER

```python
app = FastAPI()

loop = asyncio.get_event_loop()

aioproducer = AIOKafkaProducer(loop=loop,
                               client_id='retail-product',
                               bootstrap_servers=KAFKA_INSTANCE)


@app.on_event("startup")
async def startup_event():
    await aioproducer.start()


@app.on_event("shutdown")
async def shutdown_event():
    await aioproducer.stop()


@app.post("/producer/{topicname}", response_model=ProducerResponse)
async def kafka_produce(msg: ProducerMessage, topicname: str):
    """
    Produce a message into <topicname>
    This will produce a message into a Apache Kafka topic.
    """
    message_id = f"{msg.name}_{uuid.uuid4()}"
    logger.debug(msg)

    await aioproducer.send(topicname, json.dumps(msg.dict()).encode("ascii"))
    response = ProducerResponse(name=msg.name,
                                message_id=message_id,
                                topic=topicname,
                                timestamp=msg.timestamp)
    logger.info(response)
    return response
```

Initialization of AIOKafka Producer with KAFKA_INSTANCE

Producer sends message to Kafka. This message holds information related to product.

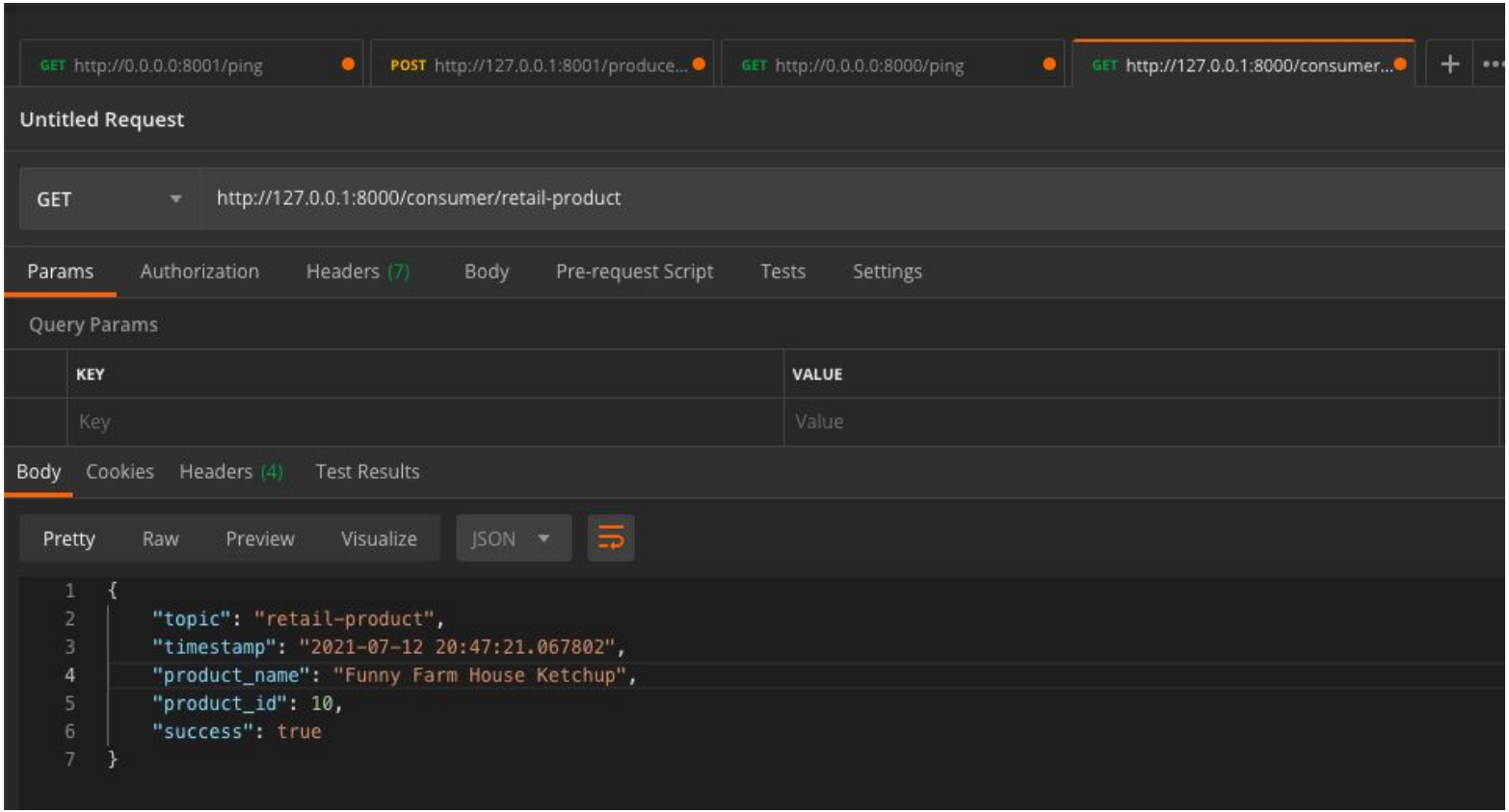

GET http://0.0.0.0:8001/ping
POST http://127.0.0.1:8001/produce...
GET http://0.0.0.0:8000/ping
GET http://127.0.0.1:8000/consumer...
Untitled Request
POST
http://127.0.0.1:8001/producer/retail-product
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"name": "Funny Farm House Ketchup",
"category" : "Dips and Ketchups",
"price" : 15,
"stock" : 3,
"product_id": 10,
"timestamp": ""
}
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"name": "Funny Farm House Ketchup",
"message_id": "Funny Farm House Ketchup_e5c0f1b0-ac1e-44c7-92c1-1f86728a36dc",
"topic": "retail-product",
"timestamp": "2021-07-12 15:17:20.314894"
}

DATA CONSUMER

```python
app = FastAPI()

loop = asyncio.get_event_loop()

es = AsyncElasticsearch(hosts=[ELASTICSEARCH_INSTANCE])

@app.get("/consumer/{topicname}")
async def kafka_consume(topicname: str):
    """
    Produce a message into <topicname>
    This will produce a message into a Apache Kafka topic.
    """
    consumer = AIOKafkaConsumer(topicname,
                                loop=loop,
                                client_id='retail-catalogue',
                                bootstrap_servers=KAFKA_INSTANCE,
                                enable_auto_commit=True,
                                value_deserializer=kafka_json_deserializer)

    await consumer.start()
    logger.debug('starting consumer task')

    retrieved_requests = []
    try:
        result = await consumer.getmany(timeout_ms=20000)
        logger.info(f"Got {len(result)} messages in {topicname}.")

        for _, messages in result.items():
            if messages:
                for message in messages:
                    retrieved_requests.append({"value": message.value})

        for request in retrieved_requests:
            value = request.get('value')
            ....
```

Initialization of AIOKafka Consumer with KAFKA_INSTANCE

Consume messages in a batch from “retail-product” topic.


GET http://0.0.0.0:8001/ping
POST http://127.0.0.1:8001/produce...
GET http://0.0.0.0:8000/ping
GET http://127.0.0.1:8000/consumer...
Untitled Request
GET
http://127.0.0.1:8000/consumer/retail-product
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Query Params
KEY
VALUE
Key
Value
Body
Cookies
Headers (4)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"topic": "retail-product",
"timestamp": "2021-07-12 20:47:21.067802",
"product_name": "Funny Farm House Ketchup",
"product_id": 10,
"success": true
}

DATA UPDATE

```
            ....
            product_id = value.get('product_id')
            price = value.get('price')
            stock = value.get('stock')

            logger.info("data before sending to elasticsearch for update")
            try:
                data = await es.get(index='retail-catalogue', id=product_id)
                logger.info("data: ", data['_source'])
            except TransportError as e:
                raise e

            logger.info("after update, new data in elasticsearch")
            data['_source']['product']['id'] = product_id
            data['_source']['product']['price'] = price
            data['_source']['product']['stock'] = stock

            product_name = data['_source']['product']['name']
            product_id = data['_source']['product']['id']

            update_data = data['_source']
            data_for_update = {'doc': {"product": update_data['product']}}

            logger.info("data going for update in elasticsearch")
            try:
                await es.update(index='retail-catalogue',
                                body=data_for_update,
                                id=product_id)

                response = ConsumerResponse(topic=topicname,
                                            timestamp=str(datetime.now()),
                                            product_name=product_name,
                                            product_id=product_id,
                                            success=True)

                logger.info(response)
            except TransportError as e:
                raise e

            return response

    except Exception as e:
        logger.error(
            f"Error when trying to consume request on topic {topicname}: {str(e)}"
        )
    finally:
        await consumer.stop()
```

Get the product doc which has to be update by “_id”

Update the product price and stock

Send update request to Elasticsearch to update the following product doc.

RESULT

```
{
  "_index" : "retail-catalogue",
  "_type" : "_doc",
  "_id" : "10",
  "_version" : 3,
  "_seq_no" : 4,
  "_primary_term" : 2,
  "found" : true,
  "_source" : {
    "product" : {
      "id" : 10,
      "name" : "Funny Farm House Ketchup",
      "category" : "Dips and Ketchups",
      "price" : 15.0,
      "stock" : 5
    },
    "update_ts" : 1625931472
  }
}
```


```
{
  "_index" : "retail-catalogue",
  "_type" : "_doc",
  "_id" : "10",
  "_version" : 4,
  "_seq_no" : 5,
  "_primary_term" : 2,
  "found" : true,
  "_source" : {
    "product" : {
      "id" : 10,
      "name" : "Funny Farm House Ketchup",
      "category" : "Dips and Ketchups",
      "price" : 15.0,
      "stock" : 3
    },
    "update_ts" : 1625931472
  }
}
```

CONCLUSION AND LEARNINGS

- We learnt about how Producer Consumer models work and their advantages to process millions of data.
- Deep dive in example of Producer Consumer working and how it can be used with as a background service.
- Understanding of how datastore is being updated and achieving high performance and other benefits using messaging queue.
- Python code implementation can be found in my git repository - https://github.com/harshit98/Retail-Updates-Streamer
- You can reach me out on Matrix / Email / Linkedin.

THAT'S ALL FOLKS, THANK YOU !

EUROPYTHON
2021 Jul 26 - Aug 1 Online

harshit98 harshit-prasad HarshitPrasad8 harshitprasad.com