Cats v Dogs

Data Streaming on AWS

Alice Moyon

github.com/alicemoyon
linkedin.com/in/alicemoyon
alicemoyon.ie

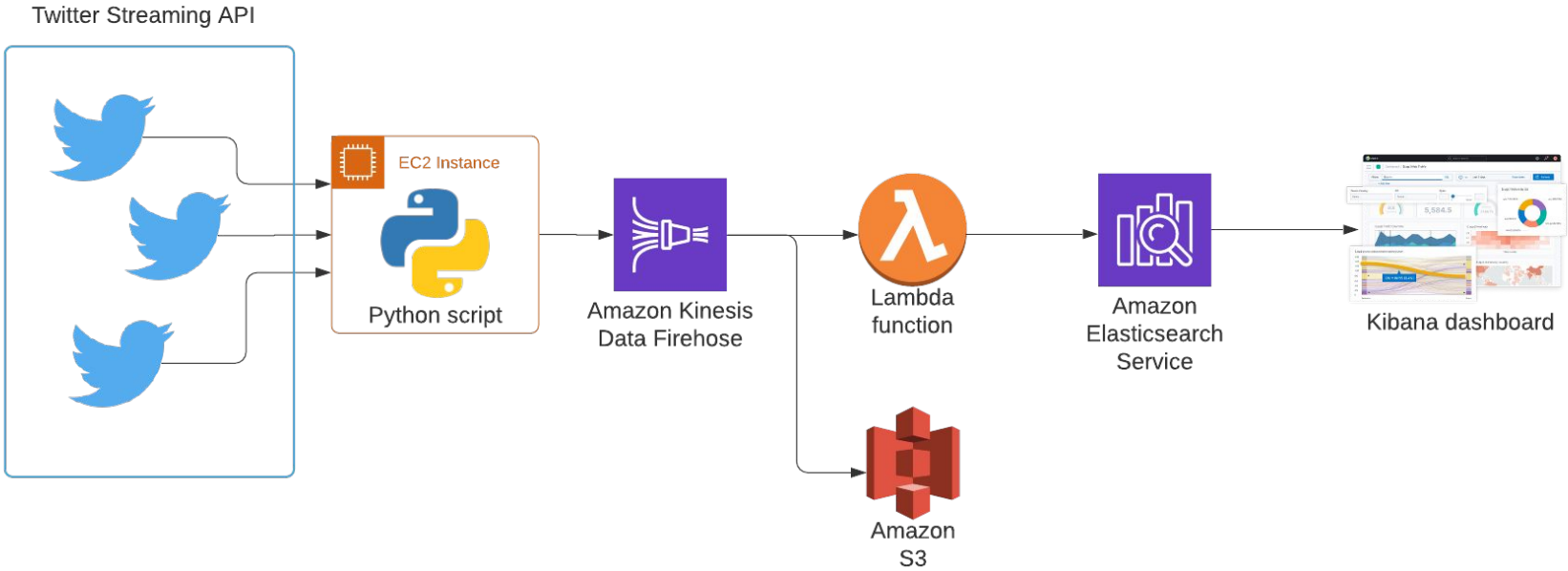
Project objectives

- Learn the fundamental components of a data streaming pipeline
- Find a simple, highly abstracted cloud-based method to analyse a data stream in real-time
- Visualise real-time data on a dashboard

Application objectives

- Stream tweets containing image or videos of cats and dogs
- Analyse tweets in real-time to determine most popular of the two
- Visualise evolution on a dashboard updated in real-time

Architecture



The new Twitter API v2

- Wide ranging and very well documented - a world of possibilities!
- Officials SDKs for Python, Node.js, Ruby + community libraries
- Two streaming options:
  - Sample Stream - random 1% sample of all tweets
  - Filtered Stream - filter with rules
- Returns Tweet objects in JSON
- Free, with some limitations:
  - Rate limits
    - 500,000 Tweets per month
    - 25 concurrent rules
    - 50 connections requests per 15-min window
    - 450 filter adds or deletes per 15-min window
  - Restricted features
    - Redundant connection & backfill
    - Advanced operators for rules (place, bio...)

The producer script - Python on EC2

- Sets up the filtering rules for the Twitter stream
- Opens a persistent HTTP connection to the API
  - Query parameters correspond to the tweet fields we want to retrieve.
- Puts the records into an AWS Firehose delivery stream as they come in through the API connection
- Issues encountered:
  - Trial & error with the query parameter syntax and filtering rules
  - Twitter bug: 0 bytes record every 20 minutes, requiring manual interrupt and reconnection
    ➔ Wrapped in a loop with automatic reconnection if any exceptions occur
- Developed and tested on local machine then deployed to an AWS EC2 instance (Ubuntu).

Firehose + Lambda for in-stream processing

- Kinesis Data Firehose:
  - AWS Service that can ingest, transform and deliver streaming data in real-time to S3, Redshift, Elastic Search, HTTP endpoints + others
    - Serverless, fully managed
    - Automatically scales to match data throughput
    - Loads to destination within 60s
  - Very easy to test in the AWS console with default record provided
  - Associate with a Lambda function for in-stream processing before delivery to destination
- Lambda:
  - Transform and process data records
  - Must include decoding & encoding again
  - Outputs back to the Firehose stream for delivery to destination
- Delivery to AWS ElasticSearch Service + S3 backup
  - Issue: records are saved to S3 in their original state
- Monitoring: enable AWS CloudWatch + see failed records on S3

More on Lambda function

- Use Python, Java, Node.js, Go etc...
- Very versatile: can be used for lots of scenarios, triggered by specific events
- Create *Layers* to add Python packages used in the function
- Decode/Encode:
  - Records are Base64 encoded as they are ingested into the Firehose stream (bytes-to-text encoding)
  - Decode before processing, re-encode before returning the processed record to the stream

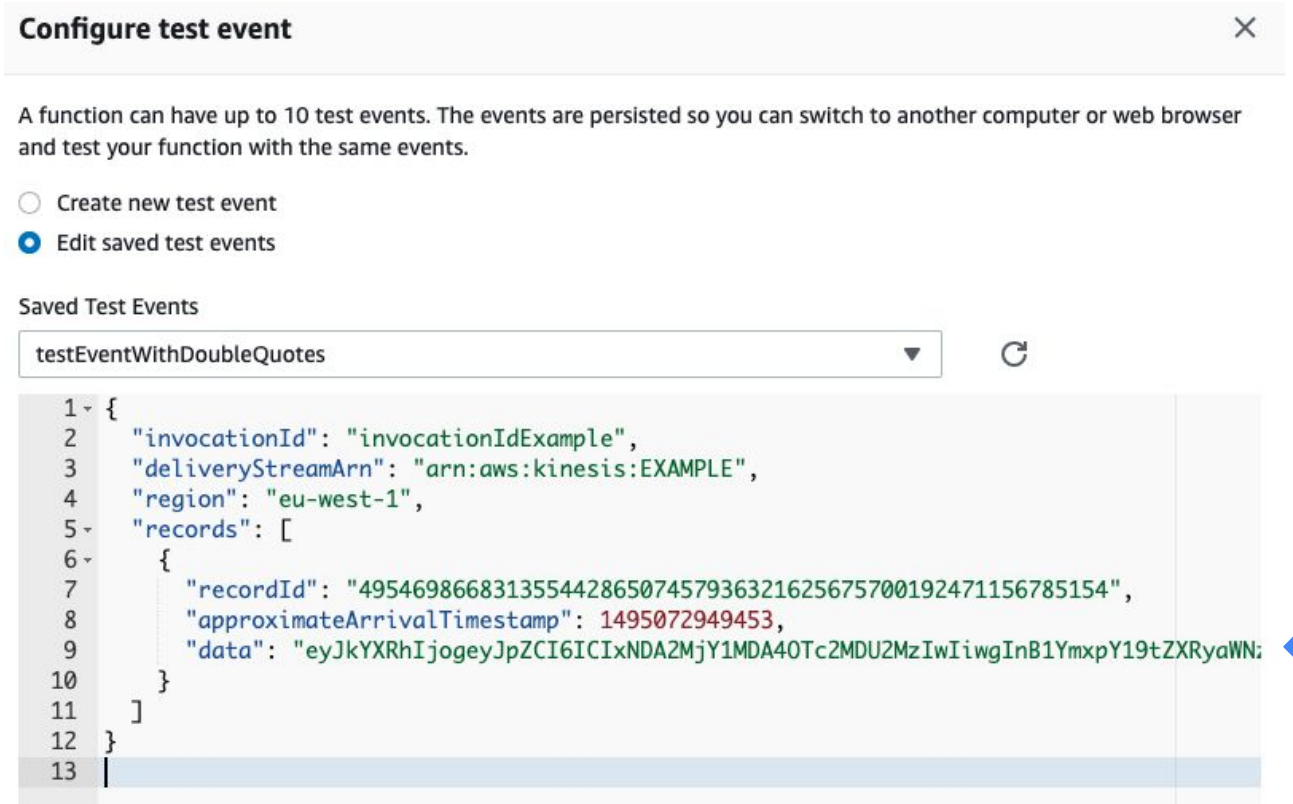
lambda_function × Execution results ×

```python
import base64
import json


def lambda_handler(event, context):
    output = []
    for record in event['records']:
        # decode the record data
        payload = base64.b64decode(record['data']).decode()
        # convert decoded string into json
        payload = json.loads(payload)
        print(type(payload))
        # clean up the record
        cleaned_payload = dict(tweet_id=payload['data']['id'],
                               retweets=payload['data']['public_metrics']['retweet_count'],
                               replies=payload['data']['public_metrics']['reply_count'],
                               likes=payload['data']['public_metrics']['like_count'],
                               category=payload['matching_rules'][0]['tag'],
                               timestamp=record['approximateArrivalTimestamp']
                              )
        # convert dictionary to json string and re-encode the payload
        new_payload = base64.b64encode(json.dumps(cleaned_payload).encode())
        output.append({
            'recordId': record['recordId'],
            'result': 'Ok',
            'data': new_payload
        })
    return {'records': output}

```

Testing the Lambda function



Encode your data record separately, then use the encoded string in your test event

Another option for in-stream analytics - Kinesis Data Analytics

Kinesis Data Analytics vs transformation Lambdas

Transformational Lambda	Kinesis Data Analytics
• Python + Pandas	• SQL
• Filter / aggregate	• Filter / aggregate
• Fixed window	• We control the window
• Great for data transformations per item	• Lets us look at the stream in chunks
• Cannot combine multiple streams	• Can combine multiple streams
• Not the best way to send output to another destination	• Can send output to another stream or other destinations

Source: DataCamp (see references slide)

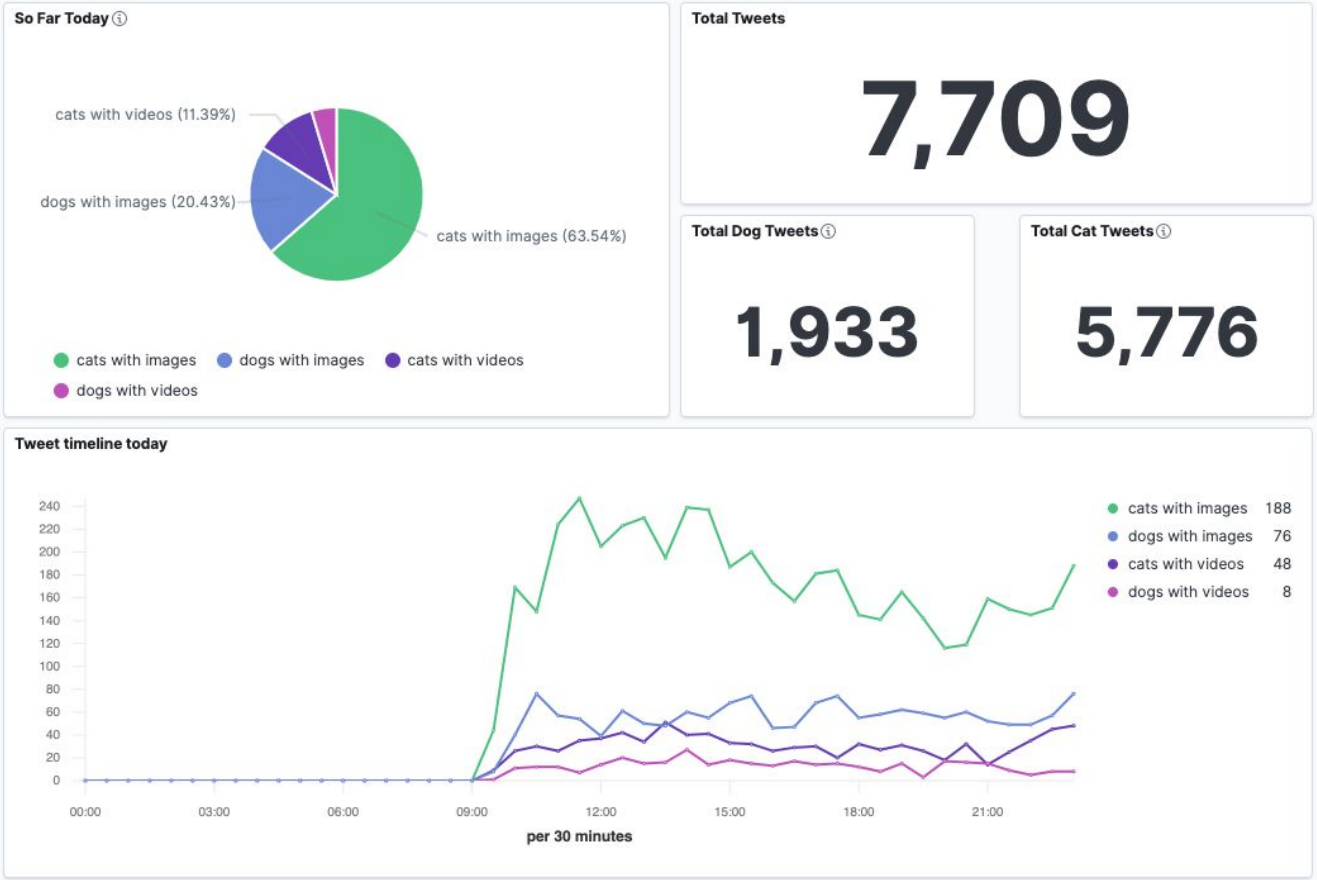
AWS Elasticsearch Service

- Seamless implementation, High level of abstraction with AWS ES
- So I know very little...but in short
  - AWS Service is fully managed to deploy and run an Elasticsearch cluster
  - Elasticsearch is a distributed, JSON-based search & analytics engine
  - Built-in *Kibana* for visualisation & near real-time dashboarding
  - Easy set-up through AWS Console, automatic *indexing* with Firehose
  - AWS version < full capabilities
- Not super intuitive at first but looks like it has amazing capabilities if you can dig deeper and explore
- Issues encountered:
  - Needs strict JSON format
  - Data type change in the index
  - No fine-grained access control with Free Tier ES

Kibana dashboard

- Very easy to create dashboard if only using basic aggregation
  - (once you figure out the jargon)
  - Index patterns (i.e. schemas) pulled from Elasticsearch
  - First, create individual visualisations
    - Great choice out of the box incl. geovisualisations
  - Then add visualisations to a dashboard
- Uses Elasticsearch index patterns
  - Automatically pulled from Elasticsearch
  - Watchout: data types cannot be edited
- For more, need to learn specific scripting language designed for Elasticsearch (Painless, Lucene and JSON style requests)
  - Can create scripted fields for visualisations
  - Myriads of possibilities for customisation

So who's winning?



Estimated AWS cost for this project

- Firehose
  - Charged only for the data transmitted
  - N records * record size (round up to nearest 5KB)
  - $0.031 per GB for first 500TB/month
  - Here: 5KB * 450k records = approx. 2.25GB/mth ➜ **less than $1**
- Lambda
  - Charged based on number of requests and code execution duration
  - Need < 450k requests/mth, Free Tier includes 1M requests/mth ➜ **Free**
- AWS Elasticsearch service
  - Using Free Tier (instance type t2.small.elasticsearch) ➜ **Free**
- EC2
  - Free Tier t2.micro for 12 months ➜ **Free**
- S3
  - 450k PUT requests/month ➜ **$2.27**
  - Storage < 5GB ➜ **Free**

That was Phase 1.....

Possible future enhancements

- Including number of likes in the analysis
  - Loop to continually search for tweets already streamed and update their like count
- More complex processing
  - e.g. exclude promotional tweets, include more fields, sentiment analysis
- Additional data source
  - E.g. Reddit
  - In-stream image analysis with call to Computer Vision model
- From Elastic to a web page
- Explore Elastic & Kibana capabilities
  - More complex analysis in Elastic
  - More sophisticated dashboard visualisations
  - Conditional text based on the real-time info? Is that possible?

Questions / Discussion

- Has anybody used Elasticsearch or Kibana before?
- Or different tools for a similar use case?
- Any suggestions for improvements?
- What frameworks can be used to implement real-time updates in a web app?

Some useful references

- Twitter filtered stream API https://developer.twitter.com/en/docs/twitter-api/tweets/filtered-stream/introduction
- DataCamp tutorial https://learn.datacamp.com/courses/streaming-data-with-aws-kinesis-and-lambda
- Base64 Wikipedia definition https://en.wikipedia.org/wiki/Base64