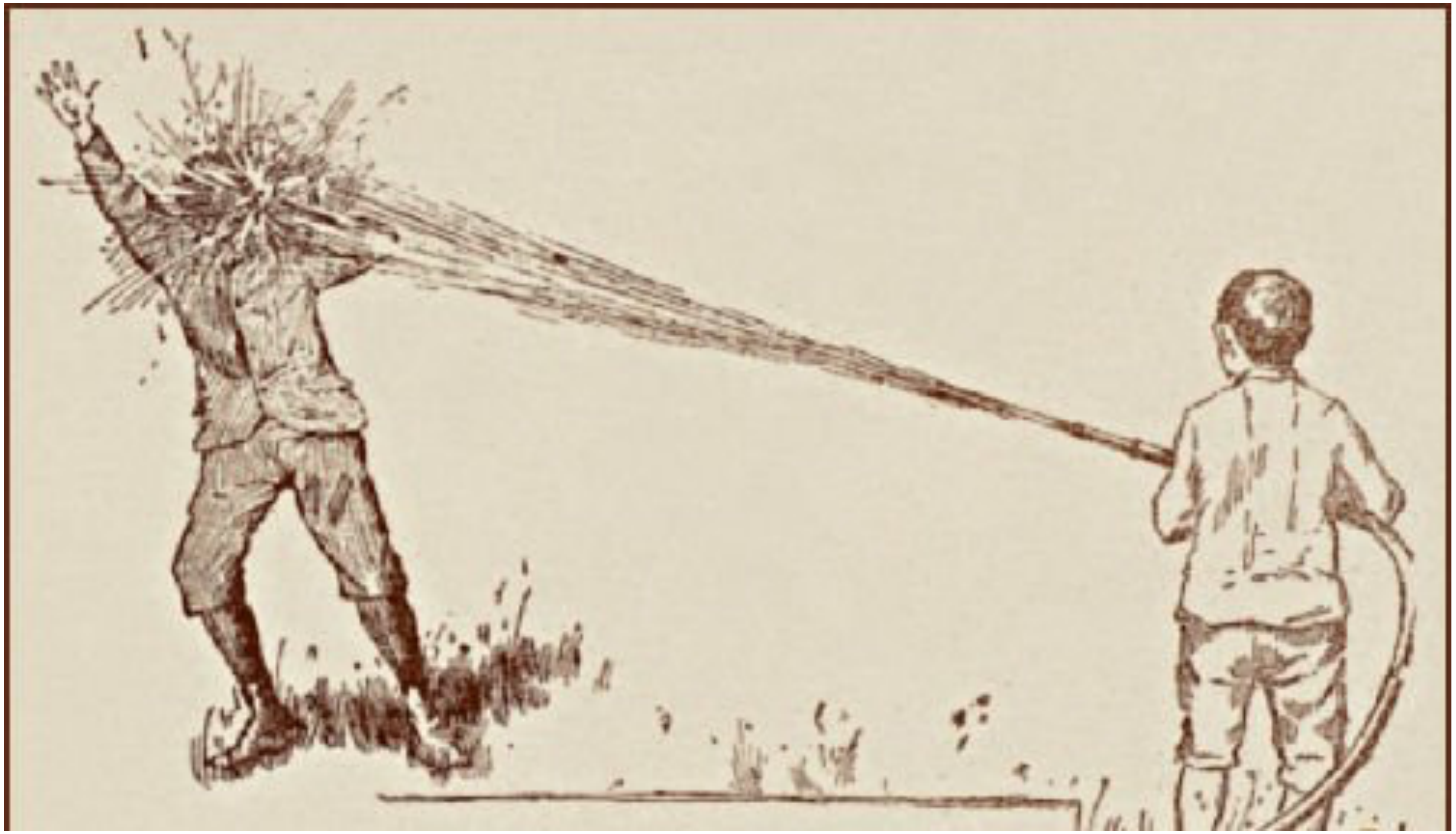
Let’s Drink from the Firehose

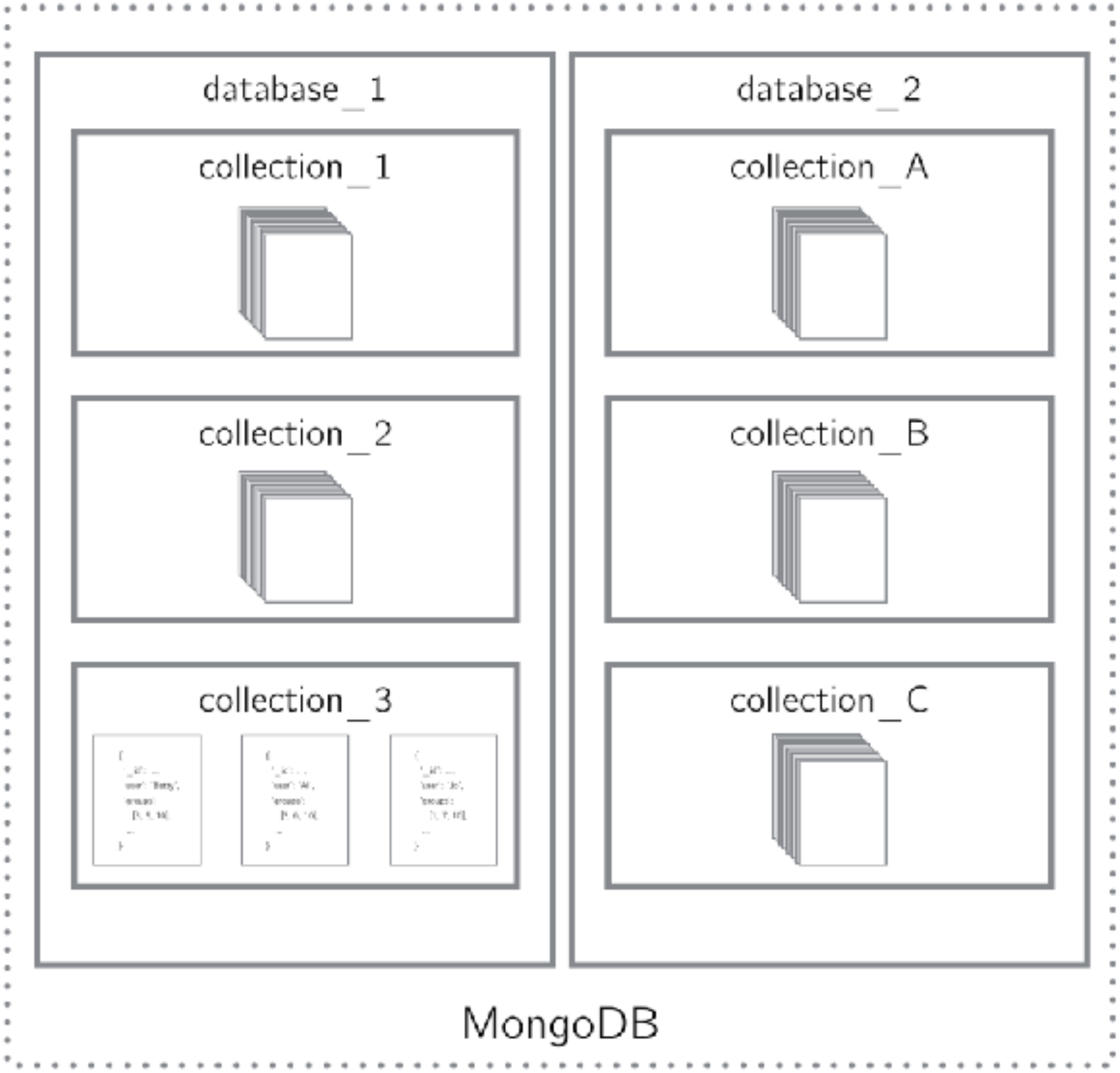


Twitter and MongoDB

MongoDB

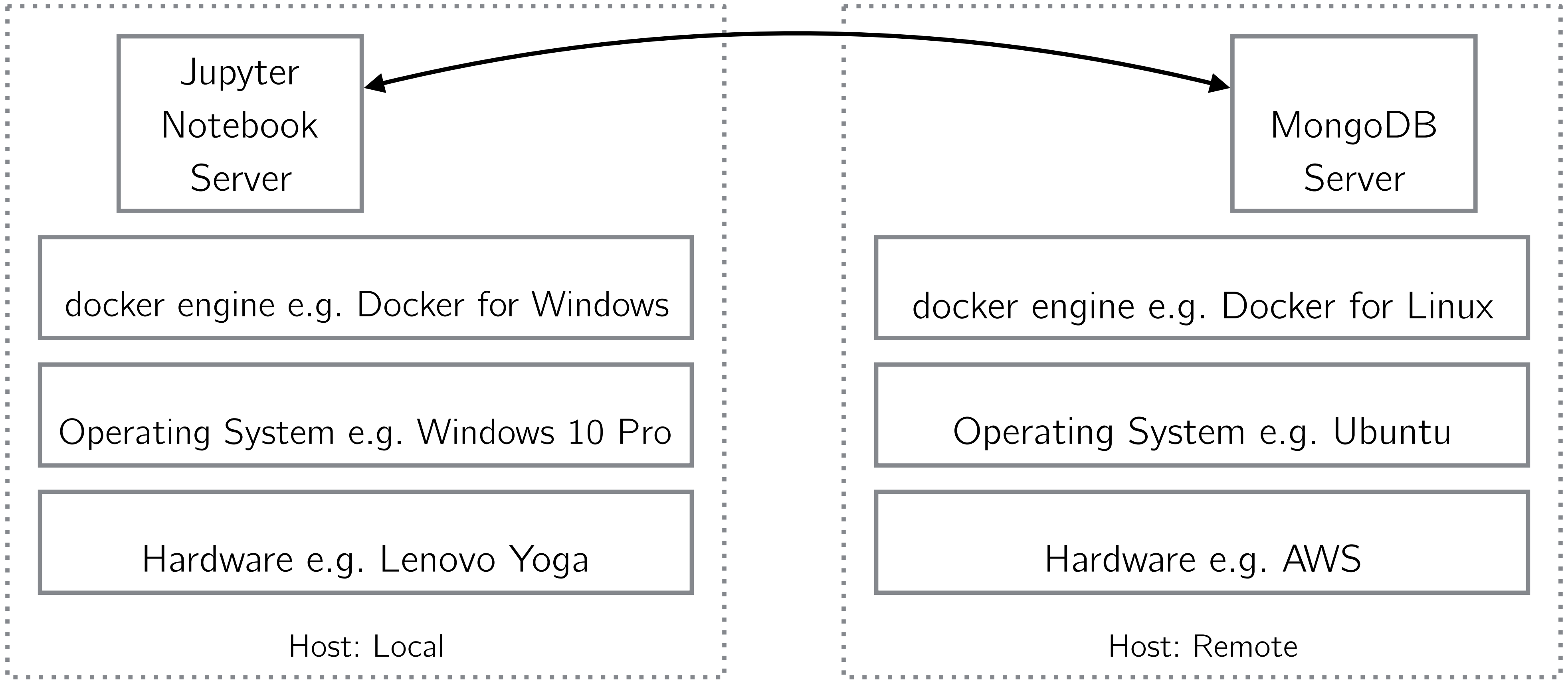
- A Database Server
- Contains Databases that contain Collections that contain Documents
- Documents are stored using a JSON-like format; play very well with `dict` objects

MongoDB


database_1
collection_1
collection_2
collection_3
database_2
collection_A
collection_B
collection_C
MongoDB

Blueprint

- Will use default images from Docker Hub for both services
- We will run them on two different systems
    - Jupyter - either local or on AWS
    - MongoDB - on a separate AWS server



Jupyter Notebook Server
docker engine e.g. Docker for Windows
Operating System e.g. Windows 10 Pro
Hardware e.g. Lenovo Yoga
Host: Local
MongoDB Server
docker engine e.g. Docker for Linux
Operating System e.g. Ubuntu
Hardware e.g. AWS
Host: Remote

NoSQL

- MongoDB is a NoSQL database
- No schema required
- Just add a document (`dict`) to a collection

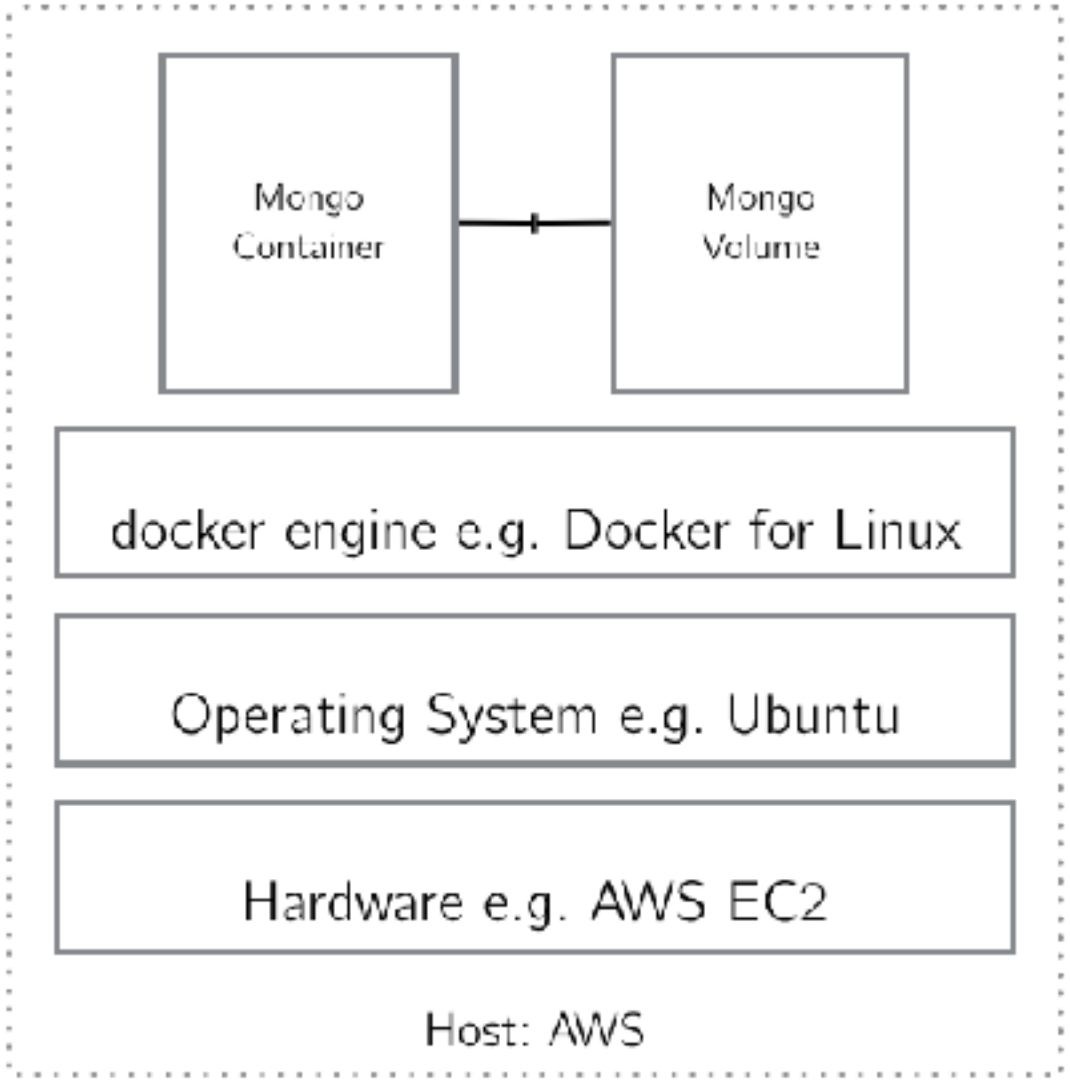
NoSQL

You == Han Solo

Your data == Chewbacca

Configuring Mongo on AWS

- Considerations:
  - Networking
    - Solve via AWS Security Groups
  - Data Persistence
    - Solve via Docker Volumes


Mongo Container
Mongo Volume
docker engine e.g. Docker for Linux
Operating System e.g. Ubuntu
Hardware e.g. AWS EC2
Host: AWS

Configuring Mongo on AWS

- Note that this configuration will use Mongo on a separate instance from the instance on which you are running Jupyter.
- If you are running Jupyter on AWS, you will need a second `t2.micro`.
- Accessing a database managed by Docker from a different AWS instance is actually easier than accessing a database managed by Docker on the same system.

Set up a new `t2.micro`

- From the AWS EC2 Dashboard, select "Launch Instance."
- On the Choose AMI tab, choose Ubuntu Server 16.04.
- On the Choose Instance Type tab, choose t2.micro.
- On the Add Storage tab, use the default setting of 8GB.
- On the Configure Security Group tab, choose "Create a new security group."
- a. Confirm that inbound SSH traffic can be accepted over port 22 from anywhere.
- b. Add a rule that accepts inbound traffic over port 2376 from anywhere. This port will allow you to pull images from Docker Hub.
- c. Add a rule that accepts inbound traffic over port 27016 from anywhere. This is the default port for accessing MongoDB.
- Review and launch an instance, taking care to confirm that you have access to the SSH keys stored with your AWS account.

Configure the new `t2.micro`

- Take note of the IP address of the newly configured AWS instance.
- SSH into the instance using that IP address.
- Install Docker via a shell script.
- Add the ubuntu user to the docker group.
- Log out and back in.

```
 (local) $ ssh ubuntu@255.255.255.255
(remote) $ curl -sSL https://get.docker.com/ | sh
(remote) $ sudo usermod -aG docker  ubuntu
```

Run Mongo via Docker

- Pull the mongo image

  ```
  $ docker pull mongo
  ```

- Create a New Data Volume

  ```
  $ docker volume create mongo-dbstore
  ```

- Launch MongoDB as a Persistent Service

  ```
  $ docker run -d --name this_mongo \
                 -v mongo-dbstore:/data/db \
                 -p 27017:27017 mongo
  ```

Verify MongoDB Installation

- You can verify that you are running the mongo service by connecting to the running MongoDB via the MongoDB client, mongo, issued via `docker exec`.

- To do this, connect and then insert a trivial document to a mongo collection. You are inserting the JSON object `{"foo": 1}` into the collection test. You then search for the document you inserted using the `.find()` command.

```
$ docker exec -it this_mongo mongo
> db.test.insert({"foo":1})
> db.test.find()
```

Using MongoDB with Jupyter

- You will need to install the necessary Python library, `pymongo`

  ```
  !pip install pymongo
  ```

- This should be run from a Jupyter server that is not on the same AWS instance as your Mongo server.

pymongo

- `pymongo` is a Python module containing the MongoDB tools recommended for working with the database.
- You begin by instantiating a connection to MongoDB using `pymongo.MongoClient`.
- Here, you use the IP address of your AWS instance on which MongoDB is running.

```
from pymongo import MongoClient
client = MongoClient('255.255.255.255', 27016)
```

pymongo

- pymongo has a very useful “get or create” mechanism for both databases and collections.

```
client.database_names()
```

pymongo

- Databases and collections are accessed using either attribute-style (`client.database_ name`) or dictionary-style (`client['test-database']`).

- If the database exists, this method will return a reference to the existing database or collection ("get"). If the database does not exists, this method will create the database or collection and then return a reference to it ("create").

pymongo

- The creation happens at the time of insertion of a document.

```
db_ref = client.my_database
client.database_names()

coll_ref = db_ref.my_collection
client.database_names(), db_ref.collection_names()

sample_doc = {"name":"Joshua", "message":"Hi!",
'my_array' :[1,2,3,4,5,6,7,9]}
coll_ref.insert_one(sample_doc)

client.database_names(), db_ref.collection_names()
```

Mongo and Twitter

- To demonstrate a simple usage for MongoDB with Jupyter, you will implement a basic Twitter streamer that inserts captured tweets into a MongoDB collection.
- Twitter data represents an ideal use case for the NoSQL MongoDB.
- Each tweet obtained via the Twitter API is received as an unstructured nested JSON object.

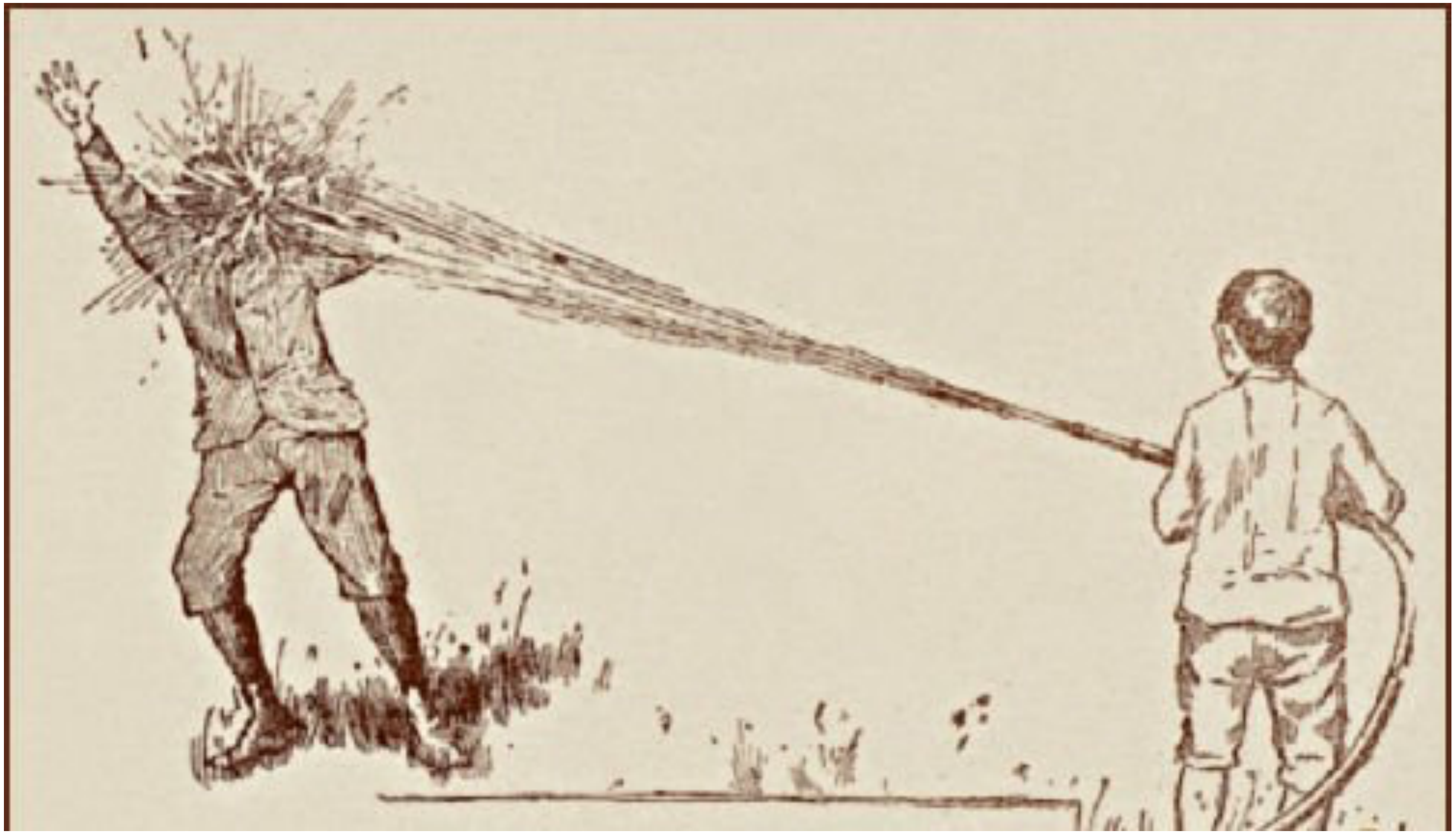
Mongo and Twitter

- Adding such an object to a SQL database would be a non-trivial task by any measure involving numerous foreign keys and JoinTables as the user seeks to manage each of the one-to-one, one-to-many, and many-to-one relationships built into the tweet.

- Adding such an object to Mongo, on the other hand, is a trivial task.

- MongoDB's native Binary JSON (BSON) format was designed precisely to accept such an object.

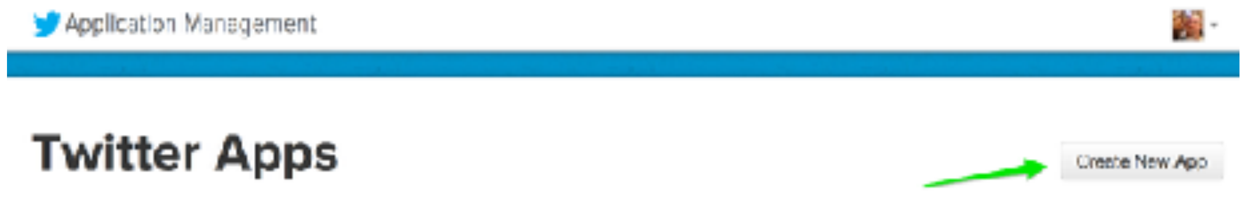
Let's Drink from the Firehose

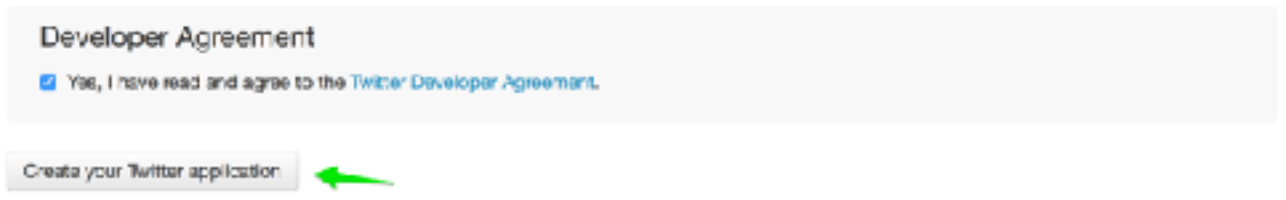


Obtain Twitter Credentials

- In order to follow along, you must obtain API credentials for accessing the Twitter API. This is done by creating a Twitter application.

- In order to do this, follow these steps:
  1. Visit https://apps.twitter.com and sign in.
  2. Choose “Create New App”.
  3. Give the new app a name, description, and website. For your purposes, the values of these responses are irrelevant, although the website will need to have a valid URL structure.
  4. Agree to the Developer Agreement and click “Create your Twitter Application”.



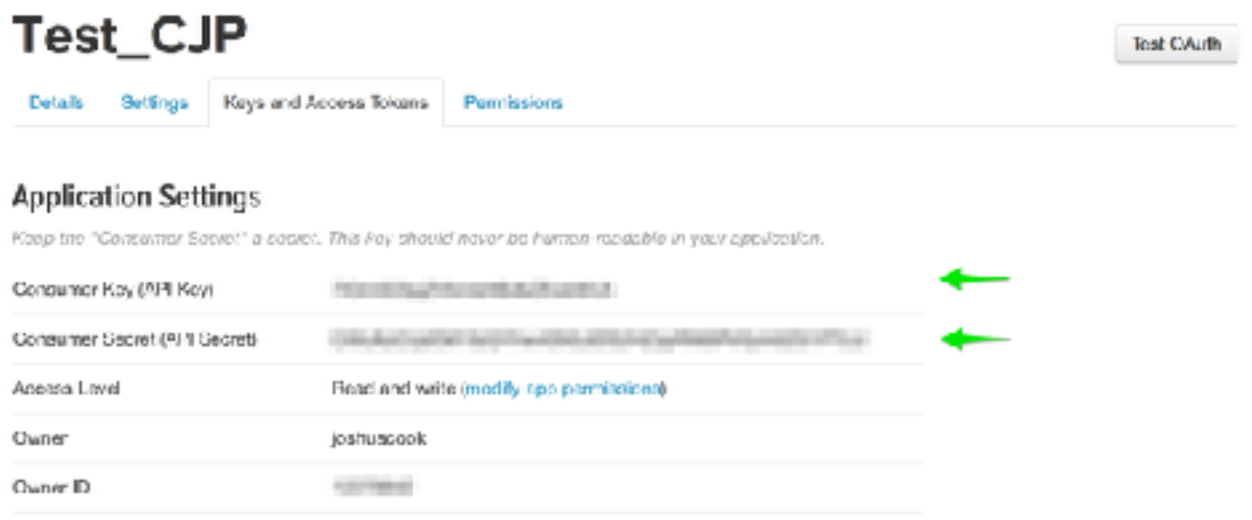


Yes, I have read and agree to the Twitter Developer Agreement.

Obtain Twitter Credentials

- Next, you will need to access your credentials on the “Keys and Access Tokens” tab.

- You will need a total of four values:
  1. A consumer key (API Key)
  2. A consumer secret (API Secret)
  3. An access token
  4. An access token secret

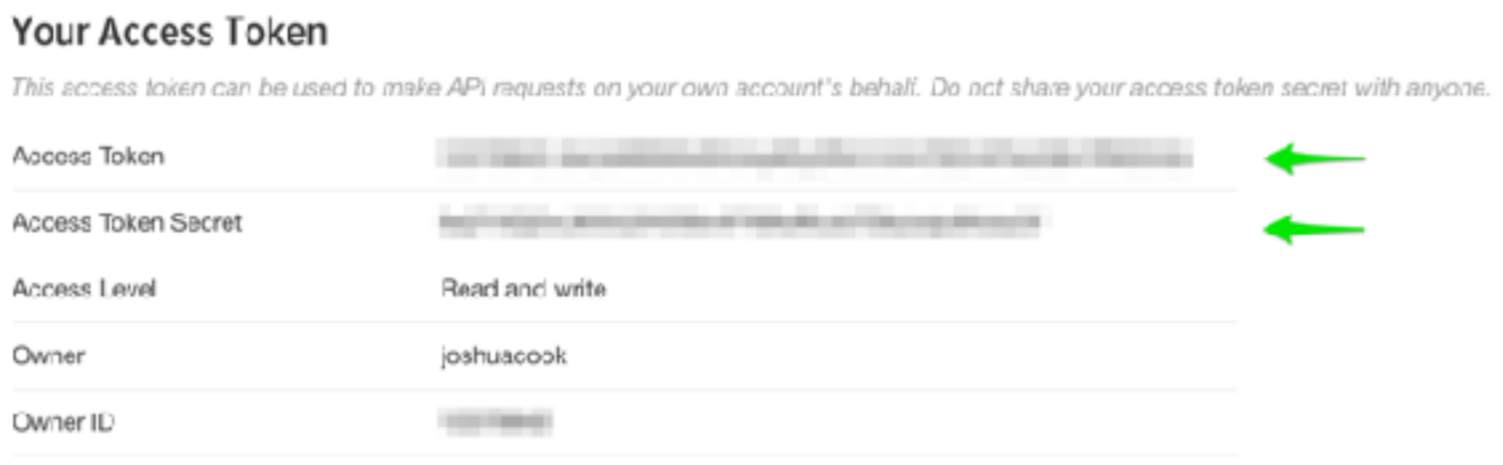


Test_CJP

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	[redacted]
Consumer Secret (API Secret)	[redacted]
Access Level	Read and write (modify app permissions)
Owner	joshuacook
Owner ID	[redacted]



Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	[redacted]
Access Token Secret	[redacted]
Access Level	Read and write
Owner	joshuacook
Owner ID	[redacted]

Load Twitter Credentials

- Load Twitter Credentials as Strings
- Replace this with your credentials:

```
CONSUMER_KEY = None
CONSUMER_SECRET = None
ACCESS_TOKEN = None
ACCESS_SECRET = None
```

Install the `twitter` library

- I prefer the `twitter` library over `tweepy`. I've found it to be better for streaming. Others have found tweeps better for historical data.

  ```
  !pip install twitter
  ```

Authentication

- You next instantiate a `twitter.OAuth` object using the Python twitter module and the credentials you have just loaded.

- You will use this object to facilitate your connection to Twitter's API.

```
from twitter import OAuth
oauth = OAuth(ACCESS_TOKEN, ACCESS_SECRET,
CONSUMER_KEY, CONSUMER_SECRET)
```

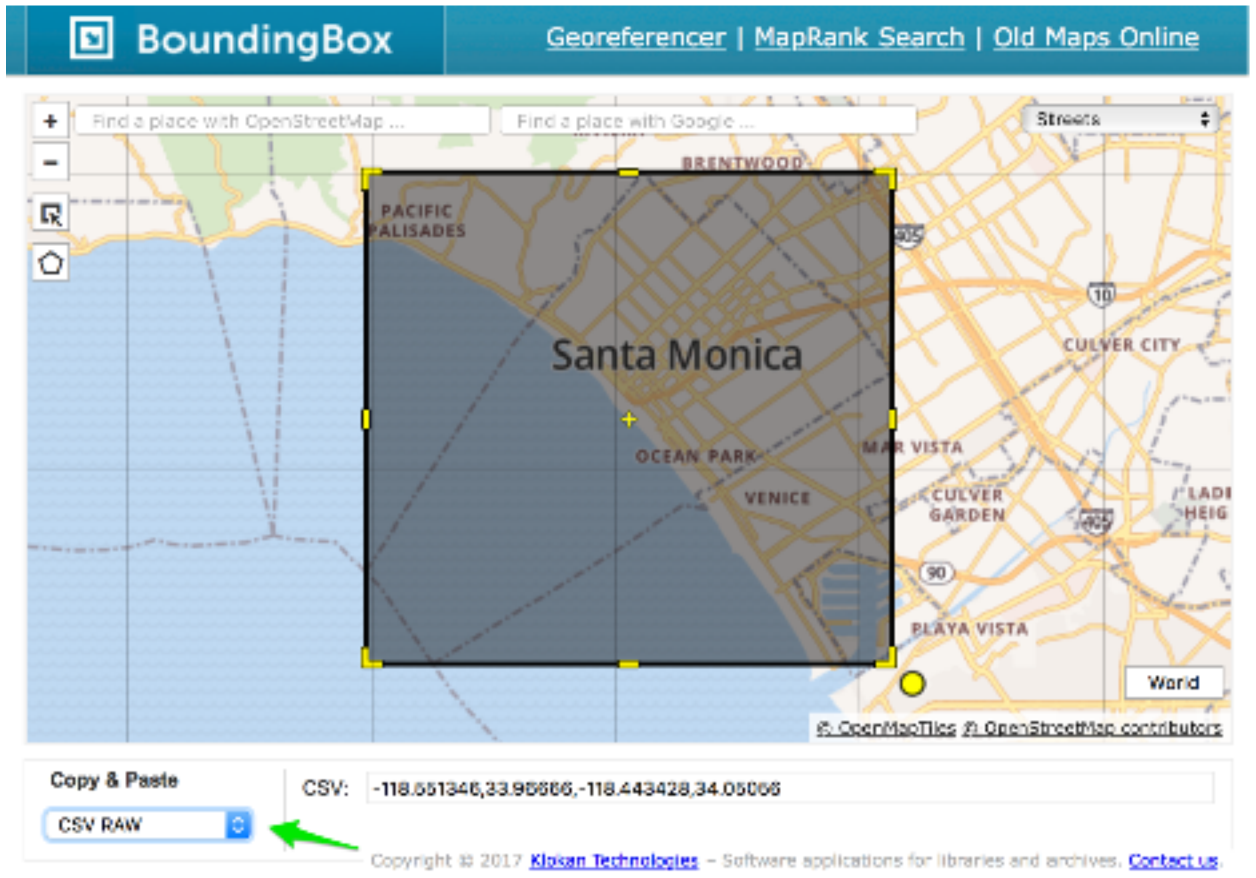
Collect Tweets by Geolocation

- For this example, you will be using Twitter's Public Stream.
- Applications that are able to connect to a streaming endpoint will receive a sample of public data flowing through Twitter and will be able to do so without polling or concern of API rate limits.
- In other words, the Public Stream is a safe and sanctioned way to collect a sample of live public tweets.
- That said, even this sample will return a great deal of unordered data.

Collect Tweets by Geolocation

- In order to provide a modicum of order to your Twitter stream, you will restrict incoming tweets using a geolocation bounding box, or bbox. You can easily obtain a bbox for a location of interest using the Klokantech BoundingBox Tool.

- Let's obtain a bbox for Santa Monica, California in the United States, making sure to select CSV Raw as the copy and paste format.


BoundingBox
Georeferencer | MapRank Search | Old Maps Online
Find a place with OpenStreetMap ...
Find a place with Google ...
Streets
BRENTWOOD
PACIFIC PALISADES
Santa Monica
OCEAN PARK
VENICE
MAR VISTA
CULVER CITY
CULVER GARDEN
PLAYA VISTA
World
© OpenMapTiles © OpenStreetMap contributors

Copy & Paste: CSV RAW

CSV: -118.551346,33.96666,-118.443428,34.05056

Copyright © 2017 Klokan Technologies – Software applications for libraries and archives. Contact us.

```
los_angeles_bbox = "-118.551346,33.96666,-118.443428,34.05056"
```

Instantiate a `TwitterStream`

- Finally, you instantiate a `twitter.TwitterStream` object you will use to collect tweets.
- `twitter.TwitterStream` provides an interface to the Twitter Stream API in Python.
- The result of calling a method on this object is an iterator that yields tweets decoded from the Twitter stream as JSON objects.

```
from twitter import TwitterStream

twitter_stream = TwitterStream(auth=oauth)
twitterator =
twitter_stream.statuses.filter(locations=los_angeles_bbox)
```

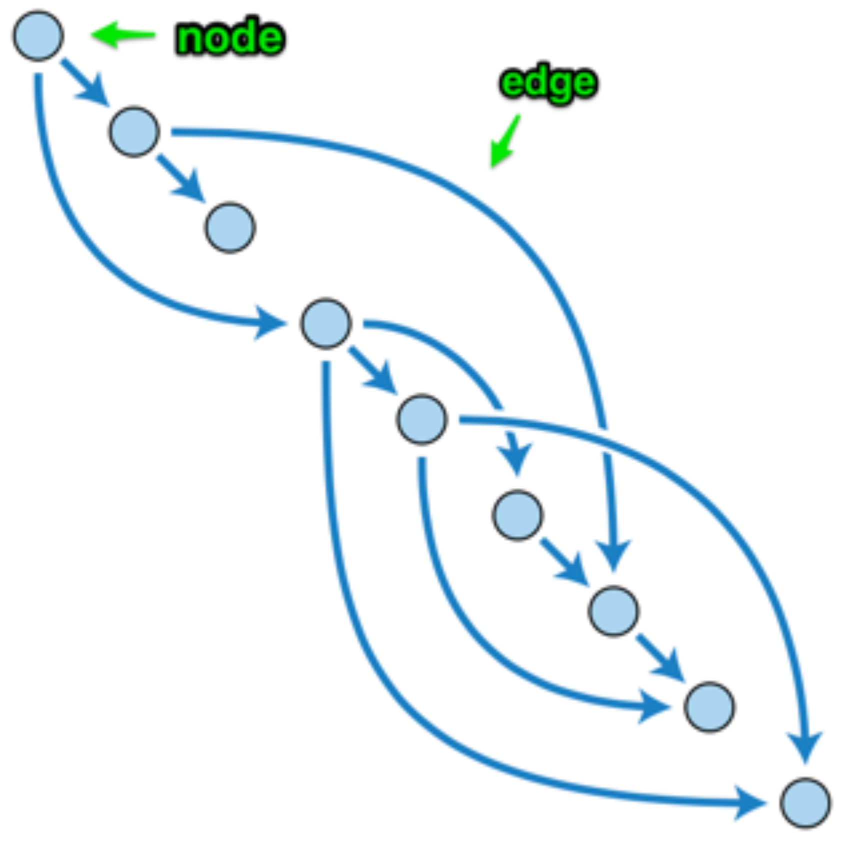
Insert Tweets Into Mongo

- Twitter is a wonderful source of messy, “real” data.
- Wrangling it into a database is where MongoDB truly shines.
- Using your `twitterator` object and the `.insert_one()` class function this can be done in a single line of code.

```
coll_ref.insert_one(next(twitterator))
coll_ref.count()
coll_ref.find_one()
```

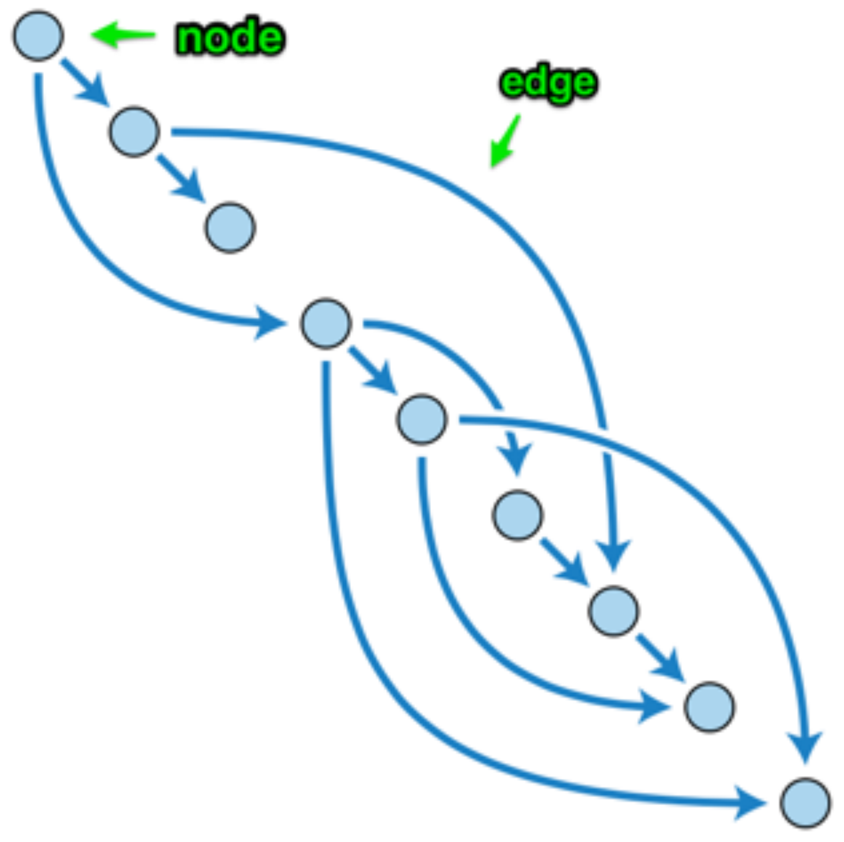
Directed Acyclic Graph

- A **graph** is a collection of nodes and edges
- A **directed graph** is a graph where each edge is directional.
- A **directed acyclic graph** (DAG) is a directed graph with no cycles.
  - Once we have passed through a node, we never return to it.



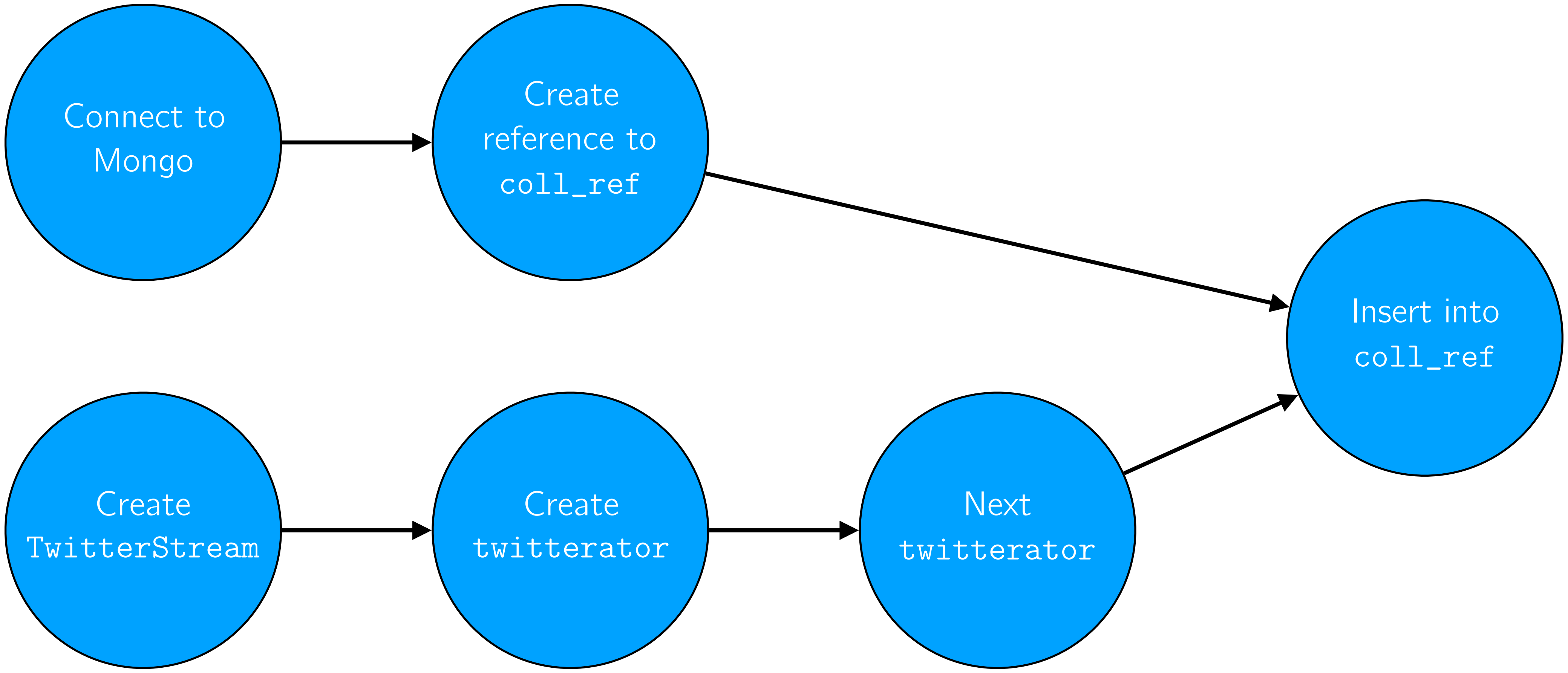
Directed Acyclic Graph

- The DAG is a great way to model a programmed process.



Directed Acyclic Graph

- The DAG for our Tweet Streamer is



Simple Parallelism

- What if I wanted to do more than one streamer at a time?
- Trivially, this can be done by running `n` notebooks, where `n` is the number of cpus on your computer.