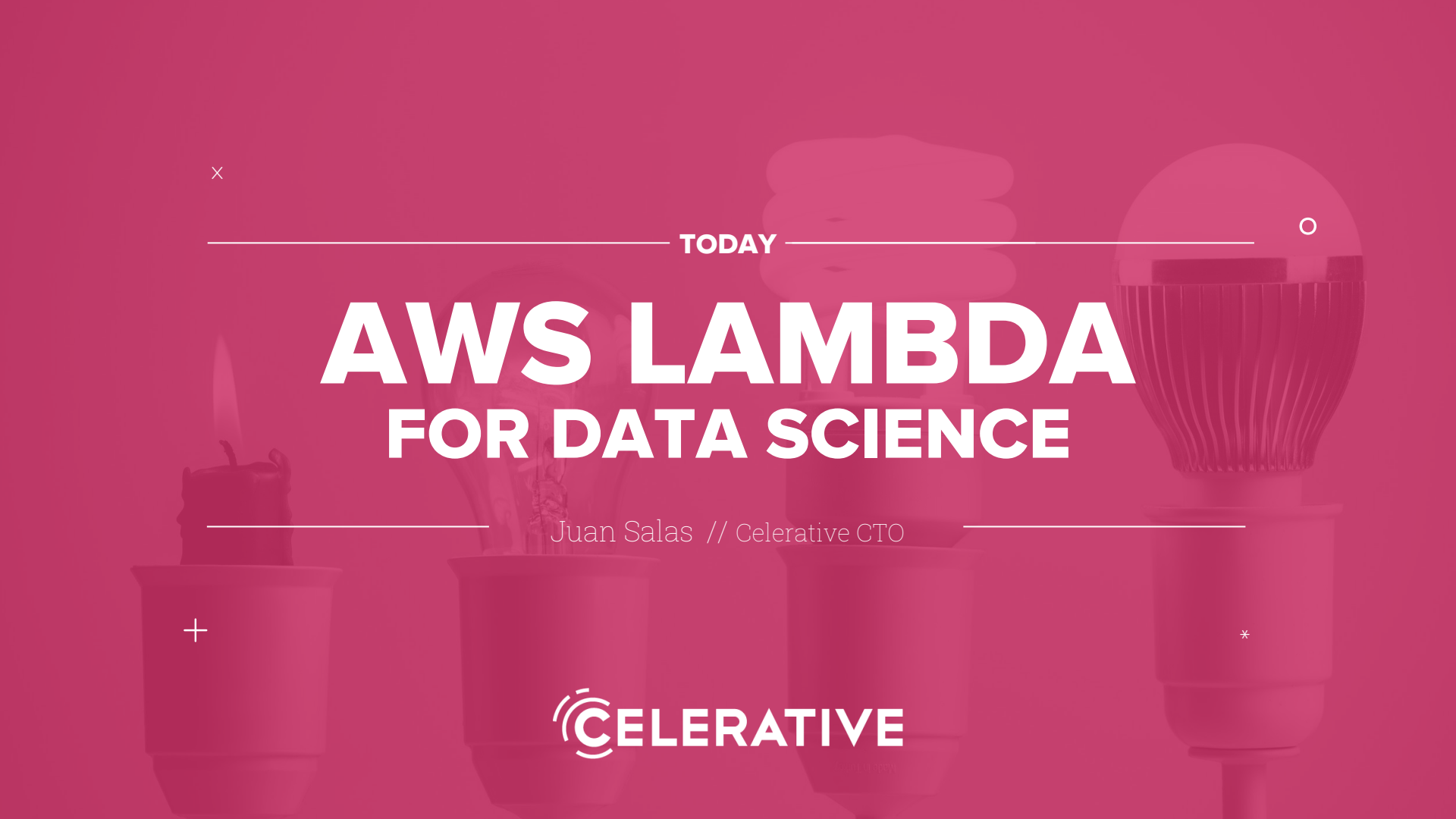

TODAY
AWS LAMBDA
FOR DATA SCIENCE
Juan Salas // Celerative CTO
CELERATIVE

OUR GOAL FOR TODAY

- What is **Amazon Lambda**?
- Why is it useful for **data scientists**?
- How can we **get started**? (Tutorial)

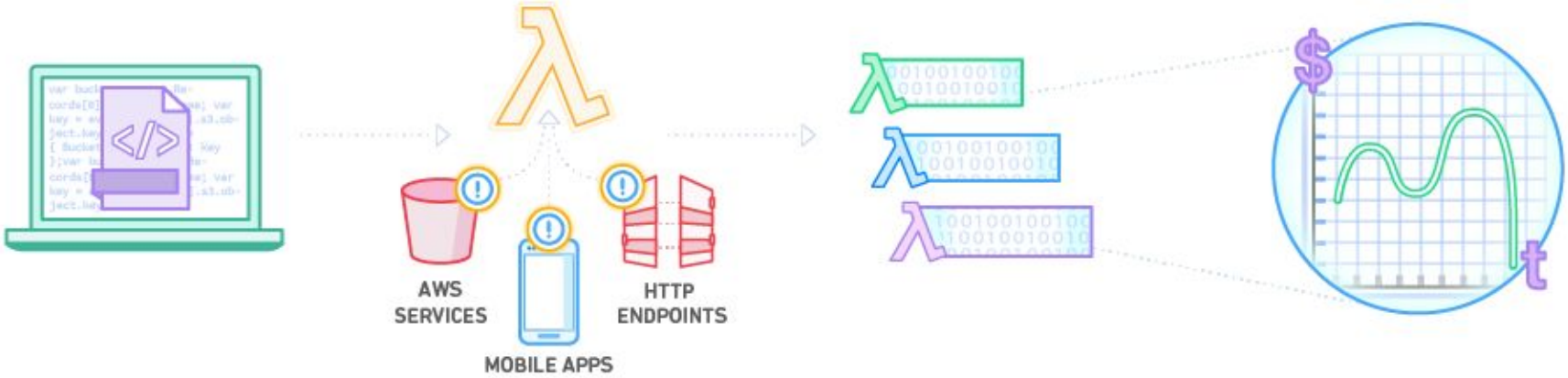
DISCLAIMER: We'll prioritize practical usage and intuition (let's make it fun!)

WHAT IS AWS LAMBDA?

WHAT IS AWS LAMBDA?

Stateless compute service for event-driven microservices

HOW DOES AWS LAMBDA WORK?



Upload your code to AWS Lambda

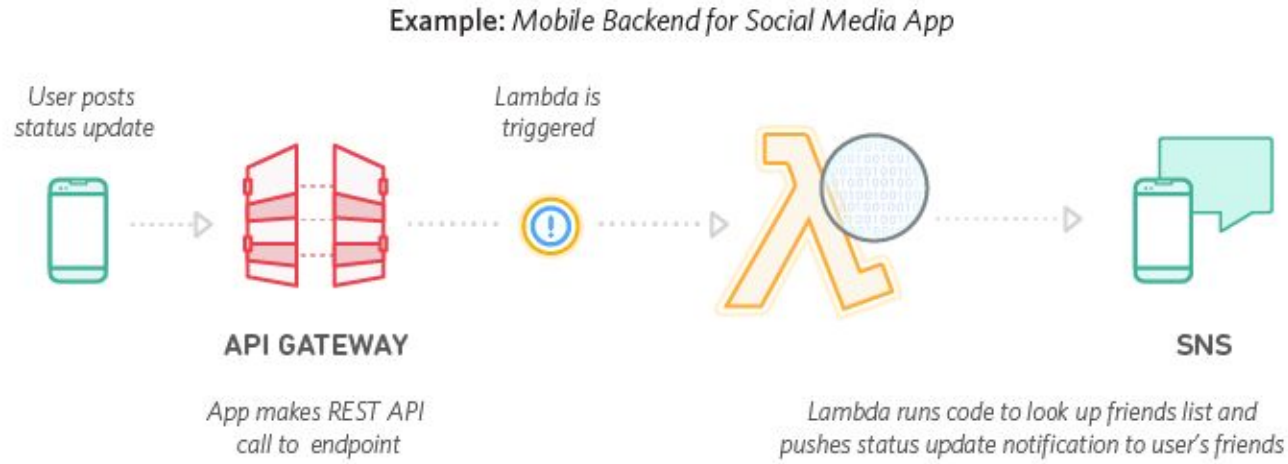
Set up your code to trigger from other AWS services, HTTP endpoints, or in-app activity

Lambda runs your code only when triggered, using only the compute resources needed

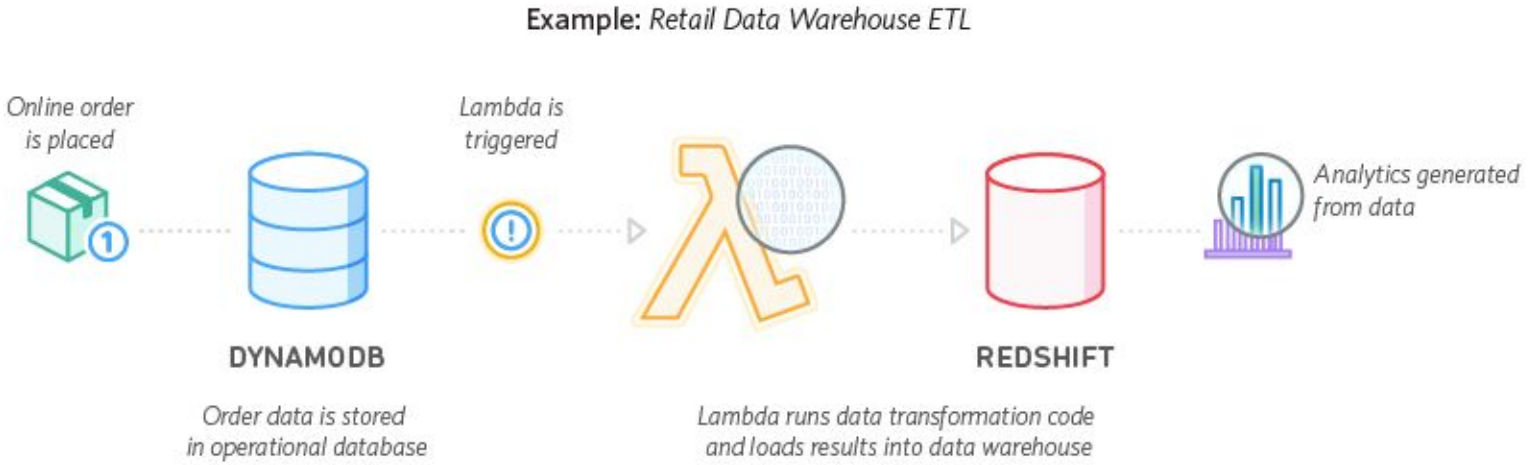
Pay just for the compute time you use

SUPPORTS:

EXAMPLE: MOBILE BACKEND

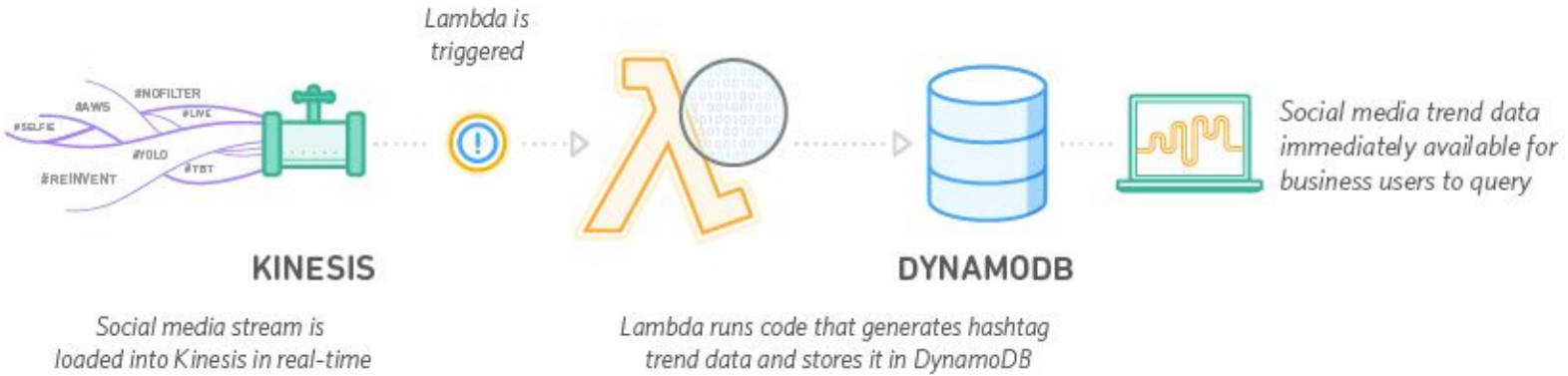


EXAMPLE: ETL


Example: Retail Data Warehouse ETL
Online order is placed
Lambda is triggered
Analytics generated from data
DYNAMODB
REDSHIFT
Order data is stored in operational database
Lambda runs data transformation code and loads results into data warehouse

EXAMPLE: SOCIAL MEDIA STREAMS

Example: *Analysis of Streaming Social Media Data*



Social media stream is loaded into Kinesis in real-time

Lambda runs code that generates hashtag trend data and stores it in DynamoDB

IT'S PRETTY COOL FOR DATA SCIENTISTS

- Supports **Python**.
- No **infrastructure** to manage, takes care of:
  - Scaling, monitoring, deployments, sys updates, etc.
- High **availability and scalability** by default
- Easy **parallelization**
- Fine grained **pricing**. Don't pay for idle!

THERE ARE SOME DRAWBACKS

- You give up some **control** (e.g. GPU, threading, filesystem, etc.).
- **Dependencies** not available with pip can be tricky.
- Package **size** constraints:
    - **Scikit-learn** depends on numpy and scipy, which in turn require C and Fortran (!!!) libraries. The package is too big.

(We'll see how to overcome this…)

LET'S GET STARTED!

TAKING LAMBDA FOR A RIDE

We will:

1. Deploy a basic **API**.
2. **Collect** some data from the Internet.
3. Use sklearn to make **predictions**.

CHALICE FRAMEWORK

Python Serverless Microframework for AWS.

Makes our life easier:

- CLI for deployment and management
- Easy interface to create APIs
- Automatic IAM policy generation
- Easy to maintain code structure

Also check out:

- Zappa (Flask)
- Serverless (Node.js)

More info:

- https://github.com/aws/chalice
- http://chalice.readthedocs.io/en/latest/

OPTIONAL: SETTING UP CREDENTIALS

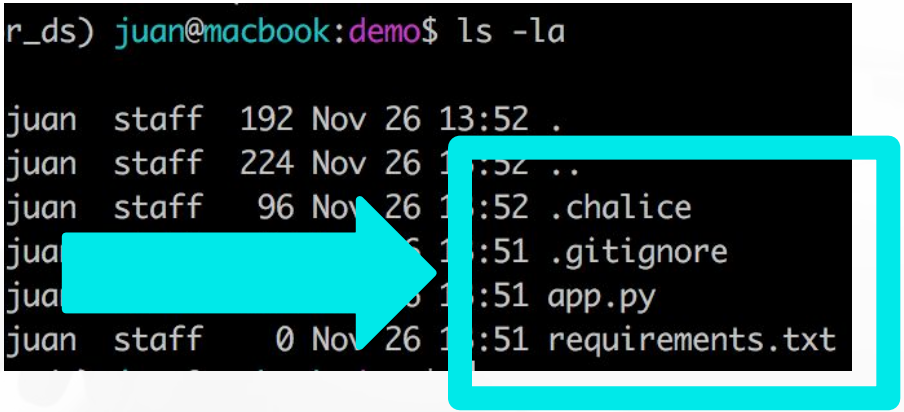
Set up your AWS credentials (skip if you already installed AWS cli or used boto):

```
$ mkdir ~/.aws
$ cat >> ~/.aws/config
[default]
aws_access_key_id=YOUR_ACCESS_KEY_HERE
aws_secret_access_key=YOUR_SECRET_ACCESS_KEY
region=YOUR_REGION (such as us-west-2, us-west-1, etc)
```

FIRST OF ALL...

Install Chalice and create a new project:

$ pip install chalice
$ chalice new-project demo && cd demo



HELLO WORLD!

APP.PY:

```
from chalice import Chalice

app = Chalice(app_name='demo')

@app.route('/')
def index():
    return {'hello': 'world'}
```

CONFIG.JSON:

```
{
  "version": "1.0",
  "app_name": "demo",
  "stages": {
    "dev": {
      "api_gateway_stage": "api"
    }
  }
}
```

IN TERMINAL:

```
$ chalice deploy

Creating role: demo-dev
Creating deployment package.
Creating lambda function: demo-dev
Initiating first time deployment.
Deploying to API Gateway stage: api
https://[…].amazonaws.com/api/
```

HELLO WORLD!

$ http GET **https://[…].amazonaws.com/api/**

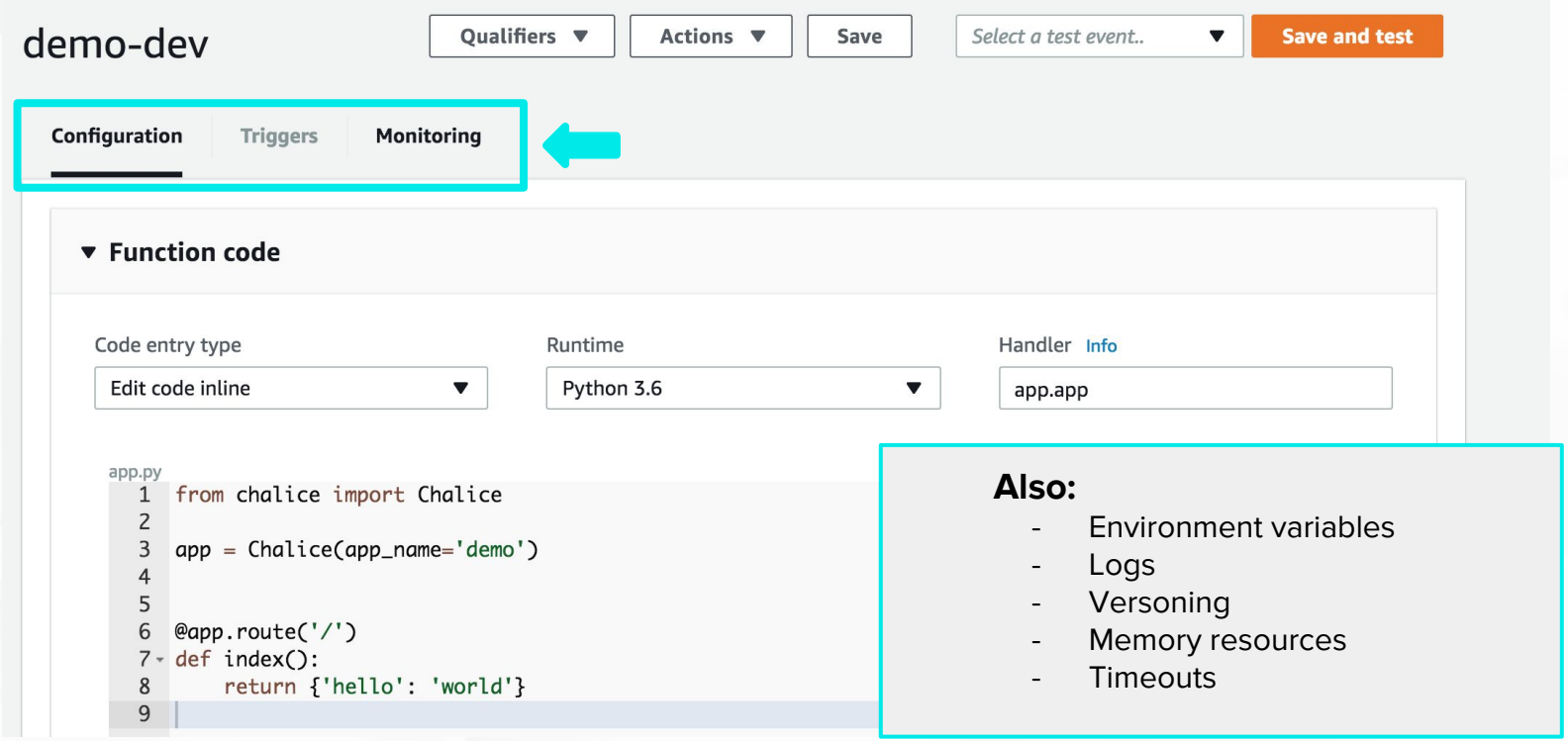
```
{
    “hello”: “world”
}
```

Note:

HTTPie is pretty cool

https://httpie.org/

HELLO WORLD!



Also:

- Environment variables
- Logs
- Versoning
- Memory resources
- Timeouts

DATA COLLECTION

DATA COLLECTION

```
import logging
import requests

# We schedule this function to run every minute
@app.schedule('rate(1 minute)')
def log_ip(event):
    r = requests.get('http://httpbin.org/ip').json()

    # We will use the logs, but we could save in a DB.
    app.log.debug("IP: " + r['origin'])
    return {"status": "ok"}
```

EASTER EGG: Ask me about my capstone project!

DATA COLLECTION

CloudWatch > Log Groups > /aws/lambda/demo-dev-log_ip > 2017/11/27/[$LATEST]c03238d45d514f9aa1653ea5645f2031

Expand all Row Text

Filter events all 30s 5m 1h 6h 1d 1w custom

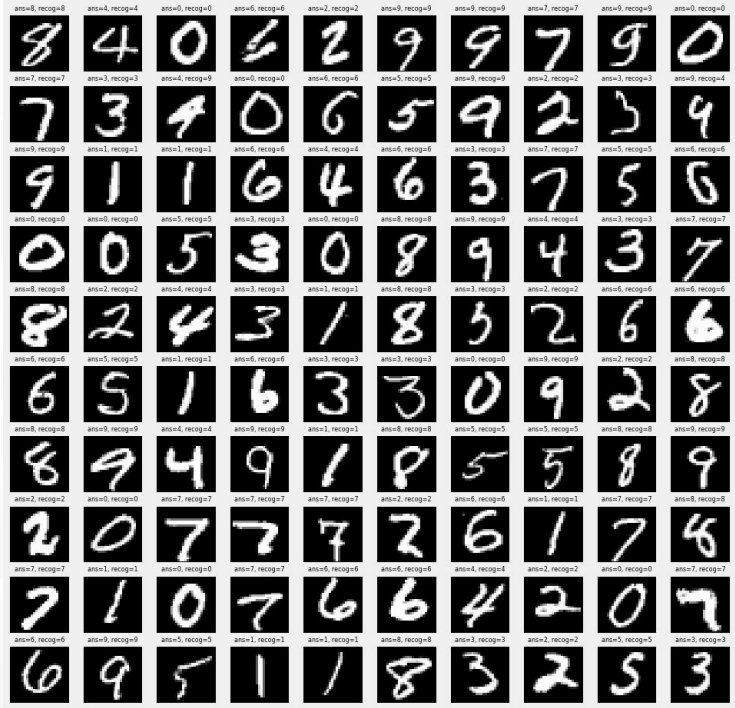
Time (UTC +00:00)	Message
2017-11-27	
	No older events found at the moment. Retry.
08:04:49	START RequestId: 2fe91ff7-d349-11e7-9e4a-b5d74bc03d6b Version: $LATEST
08:04:49	demo - DEBUG - IP: 34.201.59.206
08:04:49	END RequestId: 2fe91ff7-d349-11e7-9e4a-b5d74bc03d6b
08:04:49	REPORT RequestId: 2fe91ff7-d349-11e7-9e4a-b5d74bc03d6b Duration: 54.48 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 30 MB
08:05:35	START RequestId: bf1495f0-d349-11e7-8c89-b3076b5c7ea6 Version: $LATEST
08:05:35	demo - DEBUG - IP: 34.201.59.206

DEPLOYING A MODEL

GENERATING PREDICTIONS

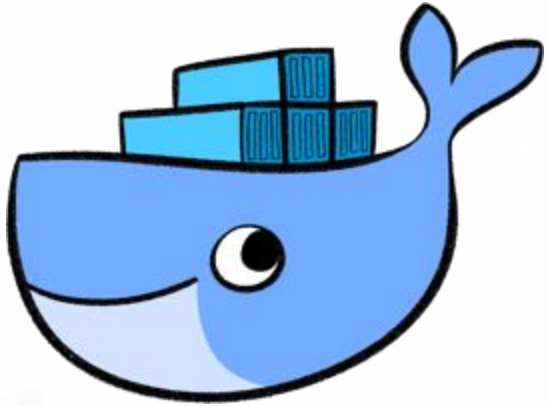
Let's say we trained a ML model offline:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import pickle

digits = datasets.load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X,y)

model = LogisticRegression()
model.fit(X_train, y_train)
pickle.dump(model, open("model.pkl", 'wb'))
```



sklearn's digits dataset:
1797 8x8 handwritten digits

USING SKLEARN WITH LAMBDA



Remember the Scikit-learn package size?

Docker to the rescue!

- Option 1: Create a container in ECS and send requests from Lambda (recommended):

  See: https://hub.docker.com/r/frolvlad/alpine-python-machinelearning/

- Option 2: Use Amazon Linux container to build a smaller version of sklearn.

- **See:** https://github.com/ryansb/sklearn-build-lambda

WHAT IF MY MODEL IS TOO BIG?

We can put our model in a **S3 bucket** and load it in the function using **boto3:**

```
import boto3
import pickle

S3 = boto3.client('s3', region_name='us-east-1')
BUCKET = 'bucket-name'
KEY = 'key-name'

response = S3.get_object(Bucket=BUCKET, Key=KEY)
body_string = response['Body'].read()
model = pickle.loads(body_string)
```

RECAP

RECAP

- We learnt what **AWS Lambda** is and what it's not.
- We used Lambda with **sklearn** to make predictions.
- We worked with pre-trained **models** on S3.
- We **collected data** using scheduled tasks.

GitHub: https://github.com/celerative/aws_lambda_for_data_science

THANK YOU!

juan.salas@celerative.com
jmsalas

44 Tehama St. / San Francisco / CA
us@celerative.com
celerative.com <<

42 nº 1389 / La Plata / Arg.
info@celerative.com
+54 (011) 527 56155

APPENDIX: USING SKLEARN WITH CHALICE

1. Get the Amazon Linux Docker image:

```
$ docker pull amazonlinux:2016.09
```

2. Clone a script that will build sklearn for us:

```
$ git clone https://github.com/ryansb/sklearn-build-lambda
$ cd sklearn-build-lambda
```

3. Run the Amazon Linux container and use it to run the script:

```
$ docker run -v $(pwd):/outputs -it amazonlinux:2016.09 \
    /bin/bash /outputs/build.sh
```

4. Unzip the contents of the resulting ***venv.zip*** in the ***vendor*** directory.

APPENDIX: USING SKLEARN WITH CHALICE

Add to app.py:

```
import os
import ctypes

# Use ctypes to support C data types, required for sklearn and numpy
for d, _, files in os.walk('lib'):
    for f in files:
        if f.endswith('.a'):
            continue
        ctypes.cdll.LoadLibrary(os.path.join(d, f))

import sklearn
```

APPENDIX: USING SKLEARN WITH CHALICE

Now we have:

- **requirements.txt:** pip dependencies except sklearn
- **chalicelib/model.pkl:** our pre-trained model
- **vendor/[venv.zip/*]:** our compressed scikit-learn package

Everything we put in the **chalicelib** and **vendor** directories will be recursively included in the deployment package (source code, json, binaries, etc.).

General rule: *chalicelib* for our own code and *vendor* for third-parties.

APPENDIX: USING SKLEARN WITH CHALICE

```
(aws_lambda_for_ds) juan@macbook:demo$ http POST localhost:8000/predict image="[0, 0, 0, 1, 15, 2, 0, 0, 0, 0, 0,
 6, 14, 0, 0, 0, 0, 0, 0, 11, 9, 0, 6, 0, 0, 0, 6, 15, 1, 11, 15, 0, 0, 5, 16, 14, 10, 16, 8, 0, 1, 15, 16, 16, 1
6, 16, 3, 0, 0, 3, 7, 5, 13, 11, 0, 0, 0, 0, 0, 0, 15, 3, 0, 0]"
HTTP/1.1 200 OK
Content-Length: 14
Content-Type: application/json
Date: Tue, 28 Nov 2017 00:57:08 GMT
Server: BaseHTTP/0.6 Python/3.6.2

{
    "class": [
        4
    ]
}
```