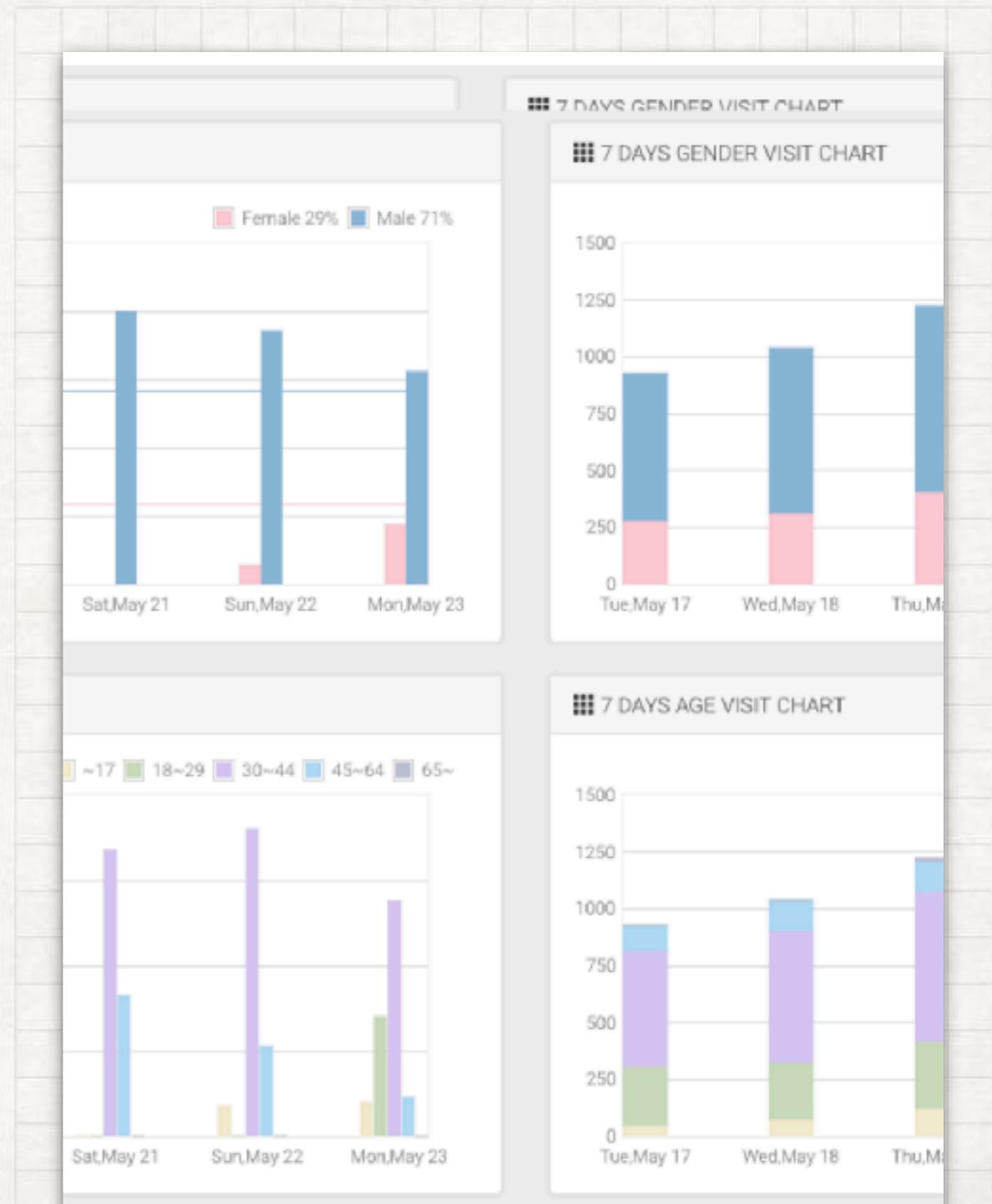


AWS를 활용한 얼굴분석 서비스 만들기

AWS 강남 소모임 - 오회근

개발사례 FACE ANALYSIS SERVICE



FACE ANALYSIS SERVICE

- 얼굴을 분석하여 성별과 나이등을 파악하여 마케팅 자료로 활용
- 영상분석 기능을 내장한 IP 카메라를 이용하여 얼굴수집
- 카메라의 개수가 늘어나므로 확장형 분산 시스템으로 개발
- 현상황에 맞게 최적의 설계를 하고 점차 규모에 맞추어 확장
- 개발기간: 1개월

결국 1달간 미친듯이 1차 완료.. -_-;;

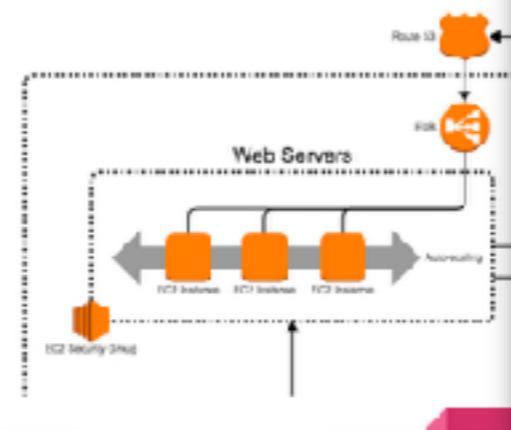
세부서비스

- **Face.In**
 - 카메라가 검출한 얼굴을 입력받아 S3와 SQS에 기록한다.
- **Face.Analysis.Worker**
 - SQS를 읽어 얼굴 분석 엔진을 이용하여 결과를 Face.DB에 전달한다.
- **Face.DB**
 - 분석결과를 DB에 저장하고 읽을 수 있는 API를 제공한다.
- **Face.Stats.Worker**
 - 분석한 결과를 주기적으로 통계를 낸다.
- **Face.Deploy**
 - 최종 결과물을 환경에 맞도록 패키징한다.

Face.In

- 카메리로 부디 얼굴 이미지와 카메라 정보를 POST 방식으로 전송받음.
- 전송받은 이미지는 S3에 저장하고 Presigned URL을 생성하여 SQS에 카메라 정보와 함께 저장한다.

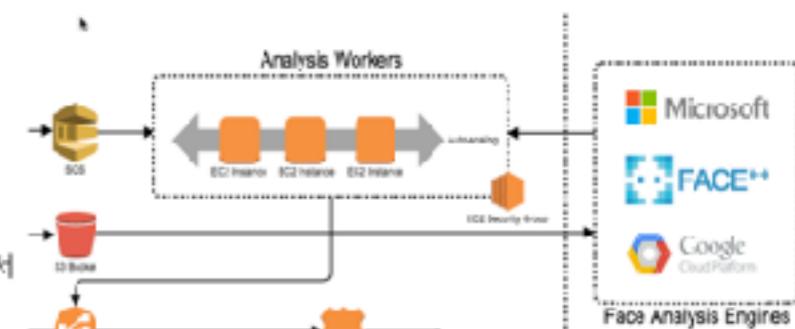
NodeJS은 스криプ트 레벨에서 단일쓰레드로 동작하여 동시에 여러 Request가 발생할 때에 내기시간이 발생한다.
Node Cluster를 이용하여 1 Instance당 3개의 Node Thread를 생성하였다.



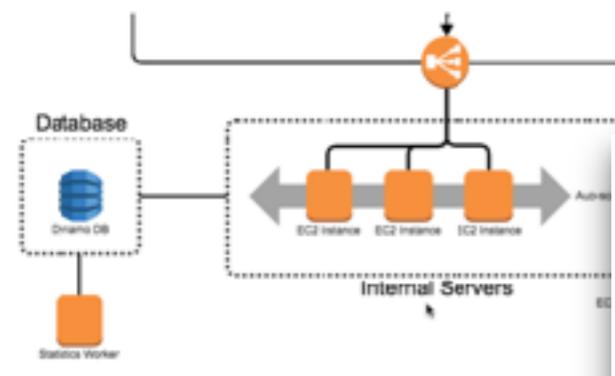
Face.Analysis.Worker

- SQS로부터 처리할 데이터를 가져와 외부 Engine을 사용하여 결과를 가져온다.
- 가져온 결과는 내부 API를 사용하여 DynamoDB에 저장한다.

외부의 엔진은 각기 성능 및 가격이 틀리다. 따라서 선택적으로 사용할 필요가 있다.
외부의 엔진들은 보통 API Key를 이용하는데 배포할 때에 유동적으로 적용할 수가 있도록 해야한다.



Face.DB



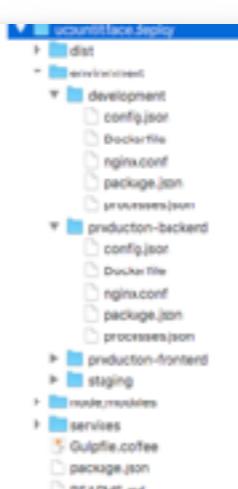
- 분석된 결과를 받아
- 통계된 데이터를 다

통계데이터를 제공하기 위해 VPC Peering을 통하여 인터넷에 노출된다.

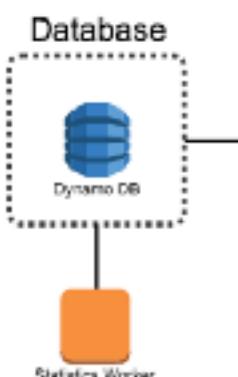
Face.Stats.Worker

- 분석 결과들을 주기적으로 통계를 내어 별도의 데이터를 생성한다.

카메라가 마주 아웃 캐모로 기느라 데이터



통계를
생성하는
필요한



Face.Deploy

- Atlassian Bamboo를 통하여 Deploy Project 내에 각 서비스들의 결과물을 취합한다.
- 각 서비스들이 취합된 Deploy Project는 각 Environment에 맞춰서 빌드되고 해당 Environment에 설정된 최종 결과물이 생성된다.
- Environment별 결과물은 AWS Beanstalk를 통하여 배포된다.
- AWS Credential과 얼굴분석엔진의 API 키는 배포시 환경변수를 통해 전달된다.

Staging Environment는 단일 Instance에서 모든 서비스를 들리고

Production은 Frontend/Backend 둘로 나누어 Frontend만 확장성을 확보하였다.

데이터가 많아질수록 더 세부적으로 구성할 필요가 있다.

```
14:16:46] gulp development
[14:16:47] Requiring external module coffee-script/register
[14:16:47] Using gulpfile `/work/upcloud/ucloud-face.deploy/Gulpfile.coffee'
[14:16:47] Starting 'development'...
[14:16:47] Starting 'clean'...
[14:16:47] Finished 'development' after 5.16 ms
[14:16:47] Finished 'clean' after 51 ms
[14:16:47] Starting 'build'...
[14:16:47] Finished 'build' after 70 ms
✓ ~/work/upcloud/ucloud-face.deploy [master 1]
```



일이
제일 좋아요.

재미있나요?

그냥. 직접 만들어 봅시다!

최대한 간단하게

DIY 얼굴분석 서비스 만들기



잠깐! 조심하세요.

“ 앞으로 나오는 명령어들은
개념을 설명하기 위해 기술되어 있습니다.
그대로 따라서 동작하지 않을 수도 있습
니다.

그리고 보안에 취약한 내용이 있습니다!

이렇게 만들어 봅시다.

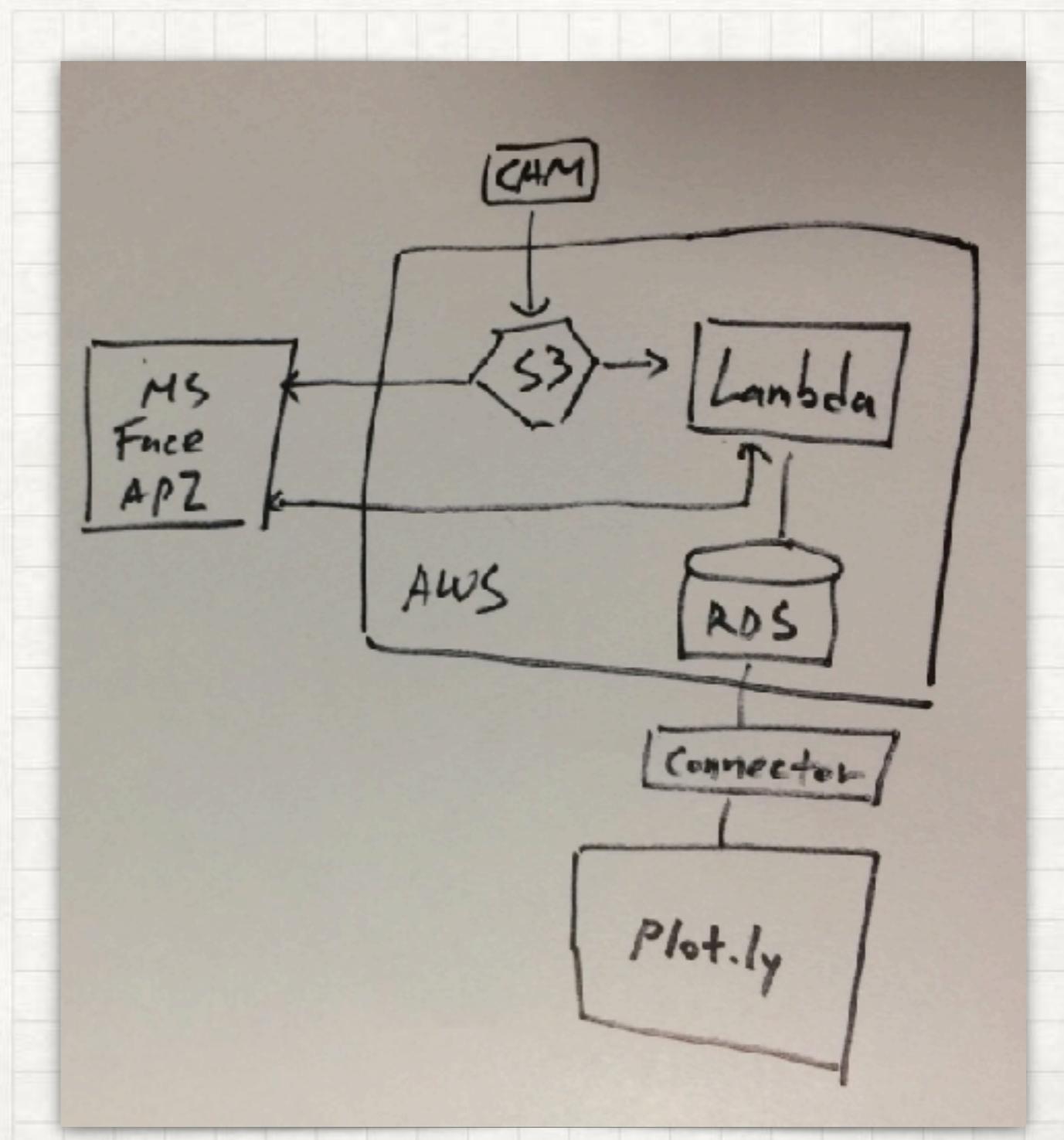
- 사진을 주기적으로 찍어 S3로 직접 올린다.
- 성별과 나이, 스마일 정도의 통계를 낸다.
- AWS에서 Serverless로 구성한다.
- 이왕이면 CLI를 이용해보자.
- 카메라는 값싸고 쉽게 구할 수 있는 것으로 만든다.
- 보안은 잠시 미루자. ::

준비물

- **라즈베리파이**
 - 집에 굴러다니던 Raspberry PI B+
 - USB 무선랜
- **Webcam**
 - 알리에서 \$3.5에 구입
- **휴대용 충전지**
 - 사은품으로 받은 것
- **스마트폰 (테더링용)**
 - 3G 무제한

역시 DIY는 손맛!

- AWS 서비스는 S3, Lambda, RDS만 사용
- Microsoft Face API 사용
- plot.ly를 이용하여 차트 생성



개발 환경

- Python 2.7 (virtualenv)
- pip
- emulambda (lambda 시뮬레이터)
- AWS Cli 설치 및 설정
- 라즈베리파이와 테더링 연결

BUCKET 생성

- 서울 리전에 funnyfaces bucket 생성
- lifecycle을 1일로 설정
- 모든 사용자들이 접근 가능하도록 설정
(주의): 실제로 서비스 할 경우 할 때에는 signed url등을 이용.)

```
$ aws s3 mb --region ap-northeast-2 s3://funnyfaces
make_bucket: funnyfaces

# lifecycle
$ aws s3api put-bucket-lifecycle --bucket funnyfaces \
--lifecycle-configuration '{
    "Rules": [
        {
            "Status": "Enabled",
            "Prefix": "",
            "Expiration": {
                "Days": 1
            }
        }
    ]
}

# public
$ aws s3api put-bucket-policy --bucket funnyfaces \
--policy '{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": { "AWS": "*" },
            "Action": [ "s3:GetObject" ],
            "Resource": [ "arn:aws:s3:::funnyfaces/*" ]
        }
    ]
}'
```

RASPBERRY PI

- 미리 준비해야 할 것들
 - Wifi 연결
 - USB WebCam 연결
 - fswebcam 설치 및 최적의 설정값
 - aws cli 설치 및 설정
 - 이미지를 S3에 전송하고 확인 한다.

```
# fswebcam -r 640x480 -S 7 \
--set "Backlight Compensation"=1 \
--set brightness=50% \
--set contrast=40% \
--set gamma=50% \
--set saturation=50% \
--set hue=50% \
--set sharpness=50% \
--no-banner test.jpg \
&& aws s3 cp test.jpg s3://funnyfaces

$ aws s3 ls s3://funnyfaces/test.jpg
2017-01-11 13:28:19      43329 test.jpg
```

IAM 설정

- Lambda 실행권한을 가진 funnyfaces-role을 추가
- funnyfaces bucket을 액세스 할 수 있는 policy 추가
- 해당 policy를 funnyfaces-role에 추가

```
$ aws iam create-role --role-name funnyfaces-role \
--assume-role-policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Principal": { "Service": "Lambda.amazonaws.com" },
        "Action": "sts:AssumeRole"
    }]
}'

$ aws iam put-role-policy \
--role-name funnyfaces-role \
--policy-name FunnyFacesS3FullAccess \
--policy-document '{
    "Version": "2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": "s3:*",
        "Resource": "arn:aws:s3:::funnyfaces/*"
    }]
}'

$ aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole \
--role-name funnyfaces-role
```

RDS 생성

- RDS를 생성
- Security Group에서 Inbound를 모두 허용
- RDS를 외부에 공개하는 건 위험!

Lambda가 rds에 접근하기 위해 서는 같은 VPC에 있어야한다. 하지만 이럴경우에 Lambda가 외부 망에 접근하기 위해서는 NAT Gateway가 필요함.

- 개발용으로만 사용해야함.
- 생성이 완료되고 endpoint를 가져옴

```
$ aws rds create-db-instance \
--db-instance-identifier FunnyFacesInstance \
--db-instance-class db.t2.micro \
--engine MySQL \
--allocated-storage 5 \
--no-publicly-accessible \
--db-name funnyfaces \
--master-username funnyfaces \
--master-user-password funnyfaces \
--backup-retention-period 0 \
--publicly-accessible
```

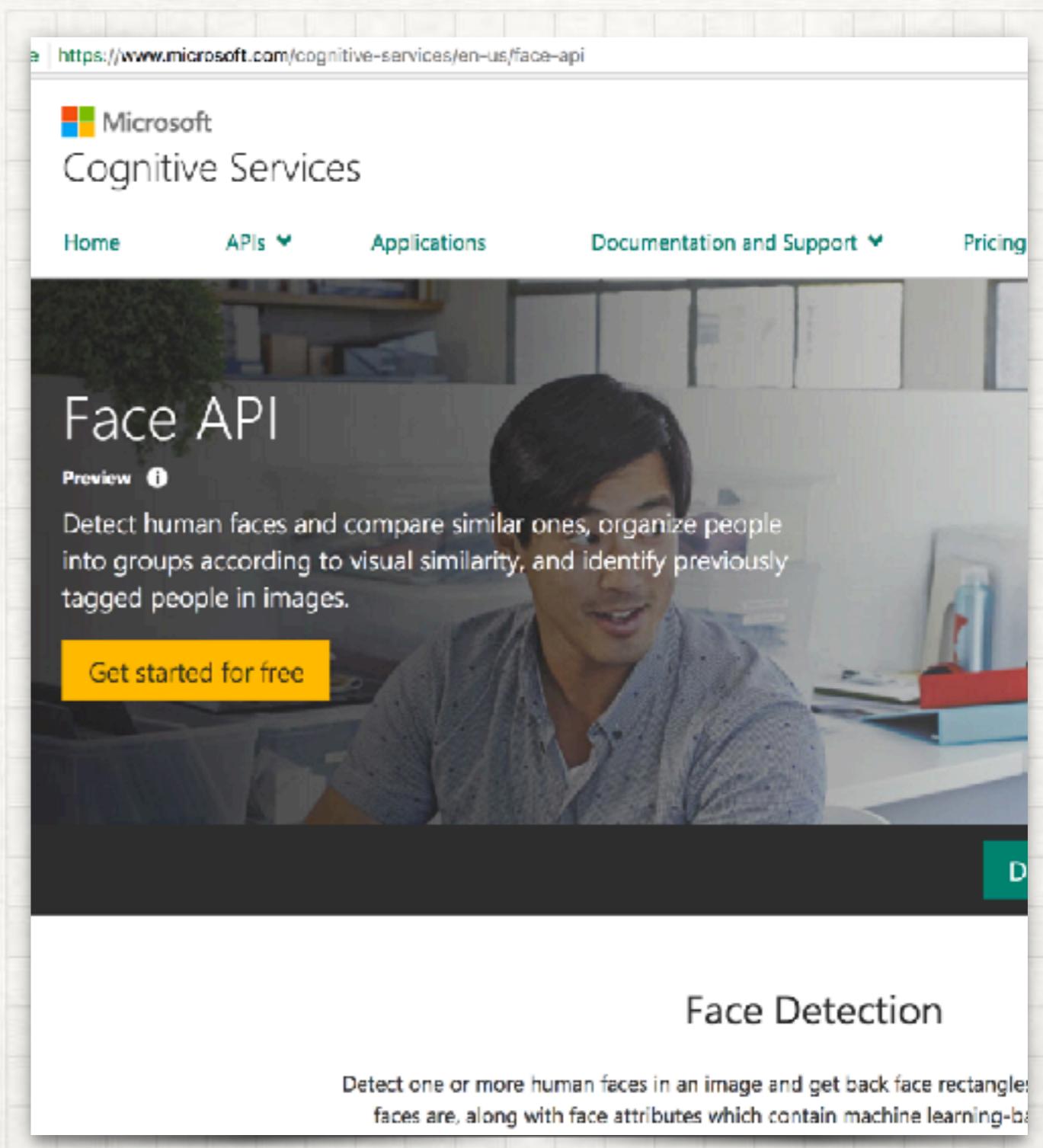
```
$ aws rds describe-db-instances \
--query "DBInstances[0].Endpoint.Address"
```

TABLE 생성

```
CREATE TABLE funnyfaces.faces_tbl (
    id INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
    face_id varchar(50) NOT NULL,
    smile SMALLINT(3) UNSIGNED NOT NULL,
    age TINYINT(3) UNSIGNED NOT NULL,
    male BOOL DEFAULT FALSE,
    female BOOL DEFAULT FALSE,
    img_url varchar(100) NOT NULL,
    created_at DATETIME DEFAULT NOW() NOT NULL,
    PRIMARY KEY(ID)
)
ENGINE=MyISAM
DEFAULT CHARSET=utf8
COLLATE=utf8_general_ci;
```

MICROSOFT FACE API

- Microsoft face api 페이지에서 API Key를 생성한다.
- 생성된 API Key는 별도 보관
- 월 3만건까지는 무료



CONFIG.PY

- face_api_key 설정
- rds_host 설정

```
face_api_key = "Your Key"
rds_host = "Your Host"
rds_port = 3306
rds_user = "funnyfaces"
rds_pwd = "funnyfaces"
rds_db = "funnyfaces"
```

FUNNYFACES.PY

```
from __future__ import print_function

# import json
import os
import urllib
import logging
import pymysql
import cognitive_face as CF

import config

FACE_API_KEY = os.getenv('FACE_API_KEY', config.face_api_key)
RDS_HOST = os.getenv('RDS_HOST', config.rds_host)
RDS_PORT = os.getenv('RDS_PORT', config.rds_port)
RDS_USER = os.getenv('RDS_USER', config.rds_user)
RDS_PWD = os.getenv('RDS_PWD', config.rds_pwd)
RDS_DB = os.getenv('RDS_DB', config.rds_db)

logger = logging.getLogger()
logger.setLevel(logging.INFO)
```

```
def write_data(s3url, faces):
    conn = pymysql.connect(RDS_HOST, user=RDS_USER, passwd=RDS_PWD, db=RDS_DB,
                           connect_timeout=5)
    query = [
        'INSERT INTO faces_tbl (face_id, smile, age, male, female, img_url)',
        'VALUES (%s, %s, %s, %s, %s, %s)'
    ]

    with conn.cursor() as cur:
        for face in faces:
            face_attr = face['faceAttributes']
            data = (face['faceId'], face_attr['smile']*100,
                    face_attr['age'],
                    face_attr['gender'] == 'male',
                    face_attr['gender'] == 'female',
                    s3url)
            cur.execute(query, data)
        conn.commit()

    return len(data)

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    region = event['Records'][0]['awsRegion']
    key = urllib.unquote_plus(
        event['Records'][0]['s3']['object']['key'].encode('utf8'))
    s3url = 'https://s3.{}.amazonaws.com/{}/{}'.format(region, bucket, key)

    CF.Key.set(FACE_API_KEY)
    result = CF.face.detect(s3url, attributes='age,gender,smile')

    item_count = write_data(s3url, result)
    return item_count
```

LAMBDA 테스트

- s3put_event.json 작성
- emulambda를 이용하여 테스트

```
{  
    "Records": [  
        {  
            "eventVersion": "2.0",  
            "eventTime": "1970-01-01T00:00:00.000Z",  
            "requestParameters": {  
                "sourceIPAddress": "127.0.0.1"  
            },  
            "s3": {  
                "configurationId": "testConfigRule",  
                "object": {  
                    "eTag": "0123456789abcdef0123456789abcdef",  
                    "sequencer": "0A1B2C3D4E5F678901",  
                    "key": "test.jpg",  
                    "size": 84278  
                },  
                "bucket": {  
                    "arn": "arn:aws:s3:::funnyfaces",  
                    "name": "funnyfaces",  
                    "ownerIdentity": {  
                        "principalId": "EXAMPLE"  
                    }  
                },  
                "s3SchemaVersion": "1.0"  
            },  
            ...  
        }  
    ]  
}
```

```
$ PYTHONPATH=packages \  
emulambda \  
-v funnyfaces.lambda_handler s3put_event.json
```

LAMBDA 생성

- 외부 라이브러리와 함께 압축
- Lambda 함수 생성
- Lambda 함수 업데이트도 가능
- 출력중 FunctionArn 을 기억

```
$ pip install cognitive_face logging pymysql -t packages
```

```
$ pushd packages;zip -r ../packages.zip *;popd  
$ zip packages.zip funnyfaces.py config.py
```

```
$ aws lambda create-function \  
  --region ap-northeast-2 \  
  --runtime python2.7 \  
  --role arn:aws:iam::550931752661:role/funnyfaces-role \  
  --description 'funnyfaces function' \  
  --timeout 10 \  
  --memory-size 128 \  
  --handler funnyfaces.lambda_handler \  
  --zip-file fileb://funnyfaces.zip \  
  --function-name GetFaceRecognition \  
  --environment Variables='{  
    "FACE_API_KEY": "d6bb77fe1e5421f907035c131855b49"  
  }'
```

```
# To update  
$ aws lambda update-function-code \  
  --function-name GetFaceRecognition \  
  --zip-file fileb://funnyfaces.zip
```

LAMBDA 실행

- S3에서 Lambda를 실행할 수 있도록 퍼미션 추가
- S3에 ObjectCreated가 되었을때 Lambda를 실행

```
$ aws lambda add-permission \
--function-name GetFaceRecognition \
--statement-id 1 \
--action "lambda:InvokeFunction" \
--principal s3.amazonaws.com \
--source-arn arn:aws:s3:::funnyfaces
```

```
$ aws s3api put-bucket-notification-configuration \
--bucket funnyfaces \
--notification-configuration '{
    "LambdaFunctionConfigurations": [
        {
            "Id": "funnyfaces-lambda",
            "Events": [ "s3:ObjectCreated:*" ],
            "LambdaFunctionArn": "arn:aws:lambda:ap-northeast-2:55093175266
        }
    ]
}'
```

RASPBERRY PI

- upload_cam.sh를 작성
- 주기적으로 카메라의 영상을 캡처하여 s3에 전송함
- nohup을 이용하여 접속이 끊어 졌을때에도 계속 동작할 수 있도록 함.

```
upload_cam.sh

#!/bin/bash

S3URL=s3://funnyfaces

for (( ; ; ))
do
    IMG=/tmp/$(date +%s).jpg
    fswebcam -r 640x480 -S 7 \
        --set "Backlight Compensation"=1 \
        --set brightness=50% \
        --set contrast=40% \
        --set gamma=50% \
        --set saturation=50% \
        --set hue=50% \
        --set sharpness=50% \
        --no-banner $IMG \
        >/dev/null 2>&1 \
    && aws s3 cp $IMG $S3URL \
    && rm -f $IMG
    sleep 5
done
```

```
$ nohup ./upload_cam.sh > upload_cam.log &

$ tail -f upload_cam.log
```

DATA 확인

```
SELECT CONVERT_TZ(  
    FROM_UNIXTIME(  
        UNIX_TIMESTAMP(created_at) - UNIX_TIMESTAMP(created_at) MOD 600  
, @@session.time_zone, '+9:00') as dt,  
    ROUND(AVG(smile/10)) as avg_smile,  
    ROUND(AVG(age)) as avg_age,  
    ROUND(SUM(male)) as male_count,  
    ROUND(SUM(female)) as female_count,  
    COUNT(*) as total_count  
FROM funnyfaces.faces_tbl  
WHERE DATE_ADD(created_at, INTERVAL 9 HOUR) >= '2017-01-10 23:00:00'  
GROUP BY UNIX_TIMESTAMP(created_at) DIV 600
```

count	female_count	total_count
0	4	
0	7	
0	5	
0	9	
0	13	
6	2017-01-10 23:00:00	7
7	2017-01-11 00:00:00	2
8	2017-01-11 00:10:00	0
9	2017-01-11 02:30:00	20
10	2017-01-11 02:40:00	54
11	2017-01-11 02:50:00	45
12	2017-01-11 03:00:00	16
13	2017-01-11 03:10:00	19
14	2017-01-11 11:00:00	91
15	2017-01-11 13:10:00	37
16	2017-01-11 13:20:00	0
17	2017-01-11 13:30:00	10
18	2017-01-11 13:40:00	2
19	2017-01-11 14:00:00	406
20	2017-01-11 14:10:00	90
21	2017-01-11 14:20:00	139

PLOT.LY

plotly Pricing Industries Products + CREATE SIGN IN SIGN UP UPGRADE français

Visualize Data, Together

Plotly is the modern platform for agile business intelligence and data science.

Try free edition Buy now for \$33/mo.*

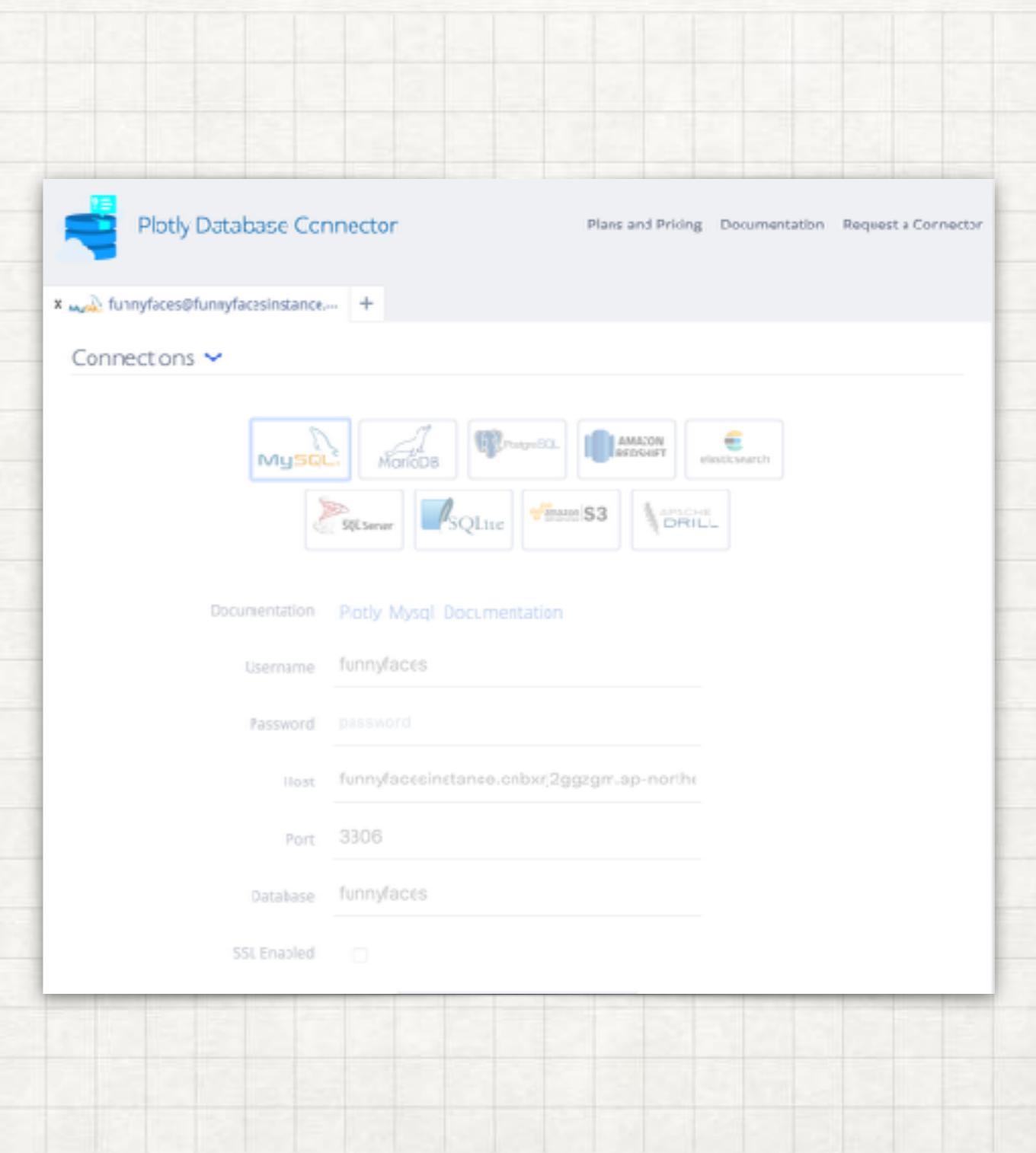
Make something amazing.

Make a Chart Make a Presentation Make a Dashboard

*Billed annually. Special prices for University students and instructors are available.

PLOT.LY DATABASE CONNECTOR

- DataBase Connector를 자신의 PC에 설치하여 Database의 값을 plot.ly로 전송한다.
- 전송된 값은 Chart로 표시 가능



SQL QUERY

Age/Smile Time

```
SELECT
    CONVERT_TZ(
        FROM_UNIXTIME(UNIX_TIMESTAMP(created_at) - UNIX_TIMESTAMP(created_at) MOD 600),
        @@session.time_zone, '+9:00') as dt,
    ROUND(AVG(smile/10)) as avg_smile,
    ROUND(AVG(age)) as avg_age,
    ROUND(SUM(male)) as male_count,
    ROUND(SUM(female)) as female_count,
    COUNT(*) as total_count
FROM funnyfaces.faces_tbl
WHERE DATE_ADD(created_at, INTERVAL 9 HOUR) >= '2017-01-10 23:00:00'
GROUP BY UNIX_TIMESTAMP(created_at) DIV 600
```

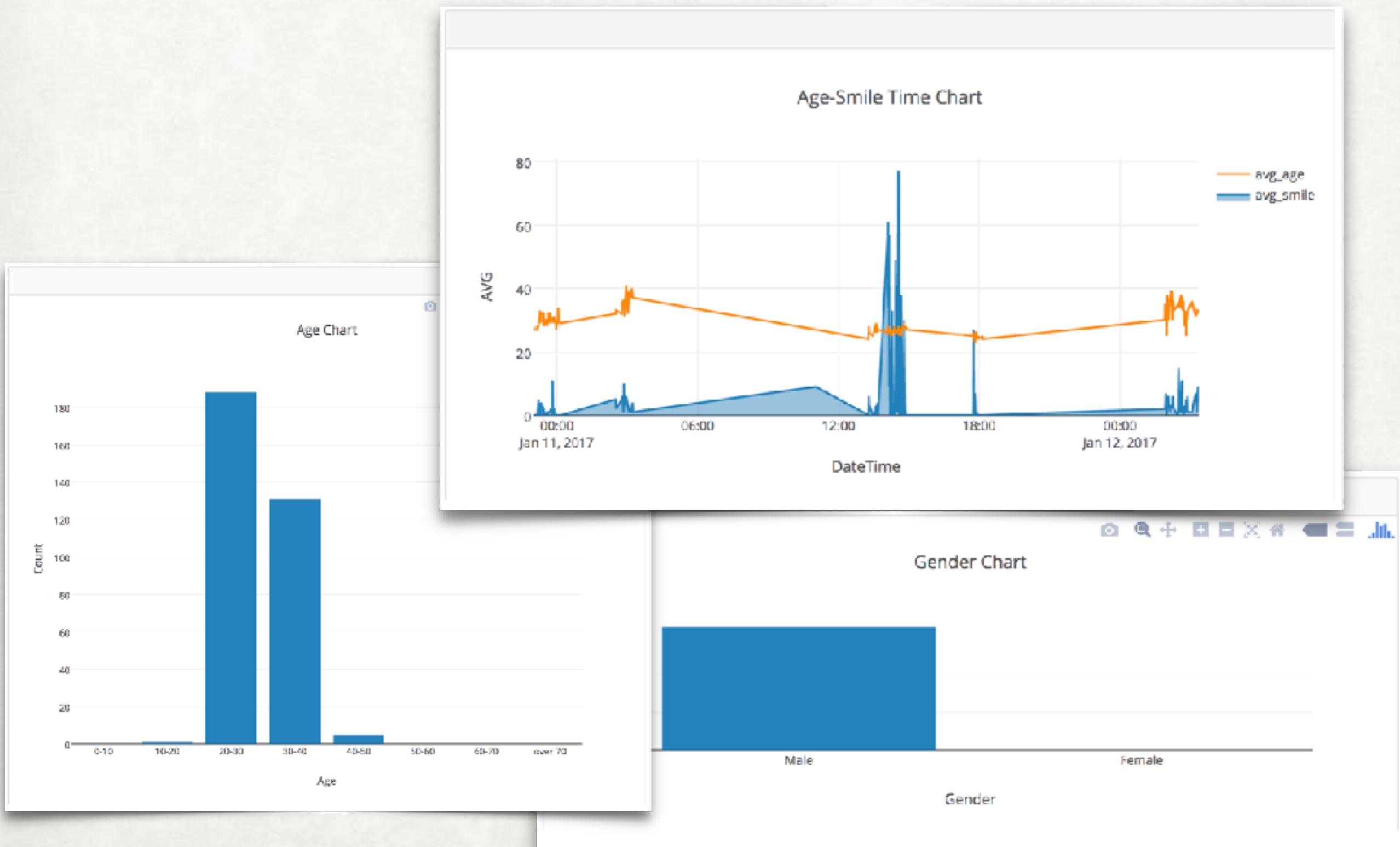
gender

Age

```
SELECT '0-10' label, count(CASE WHEN age BETWEEN 0 AND 10 THEN 1 END) from faces_tbl
UNION ALL
SELECT '10-20' label, count(CASE WHEN age BETWEEN 10 AND 20 THEN 1 END) value from faces_tbl
UNION ALL
SELECT '20-30' label, count(CASE WHEN age BETWEEN 21 AND 30 THEN 1 END) value from faces_tbl
UNION ALL
SELECT '30-40' label, count(CASE WHEN age BETWEEN 31 AND 40 THEN 1 END) value from faces_tbl
UNION ALL
SELECT '40-50' label, count(CASE WHEN age BETWEEN 41 AND 50 THEN 1 END) value from faces_tbl
UNION ALL
SELECT '50-60' label, count(CASE WHEN age BETWEEN 51 AND 60 THEN 1 END) value from faces_tbl
UNION ALL
SELECT '60-70' label, count(CASE WHEN age BETWEEN 61 AND 70 THEN 1 END) value from faces_tbl
UNION ALL
SELECT 'over 70' label, count(CASE WHEN age > 70 THEN 1 END) value from faces_tbl
```

```
SELECT 'Male' label, SUM(male) count from faces_tbl
UNION ALL
SELECT 'Female' label, SUM(female) count from faces_tbl
```

PLOT.LY DASHBOARD



개선할 점

- Raspberry Pi에서 opencv를 활용하여 스냅샷을 전송할 시점을 정의
- Raspberry Pi 통신 및 전송 자동화
- S3의 Bucket을 비공개로 하고 Signed URL을 활용하여 Face API 사용
- Lambda와 RDS를 같은 VPC에 놓고 Nat Gateway를 이용하여 외부와 통신
- API Gateway와 Lambda를 활용하여 API 제공

별거 없다!

-끝-

github: <https://github.com/harryoh/funnyfaces.git>

plot.ly: <https://plot.ly/dashboard/harryohf19d:6/view>