UNIVERSIDAD DE GRANADA

TwitterLocationFlow

Infraestructura Virtual

Luis Balderas Ruiz

ETSIIT
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

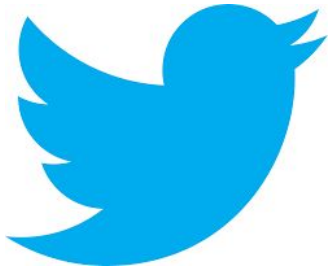
CONTENIDO

- Introducción
  - ¿Qué es?
  - Inspiración
  - Herramientas
- Almacenamiento de Datos
  - Primera aproximación
  - Gremlim CosmosDB
- API
- Tests
- Despliegues

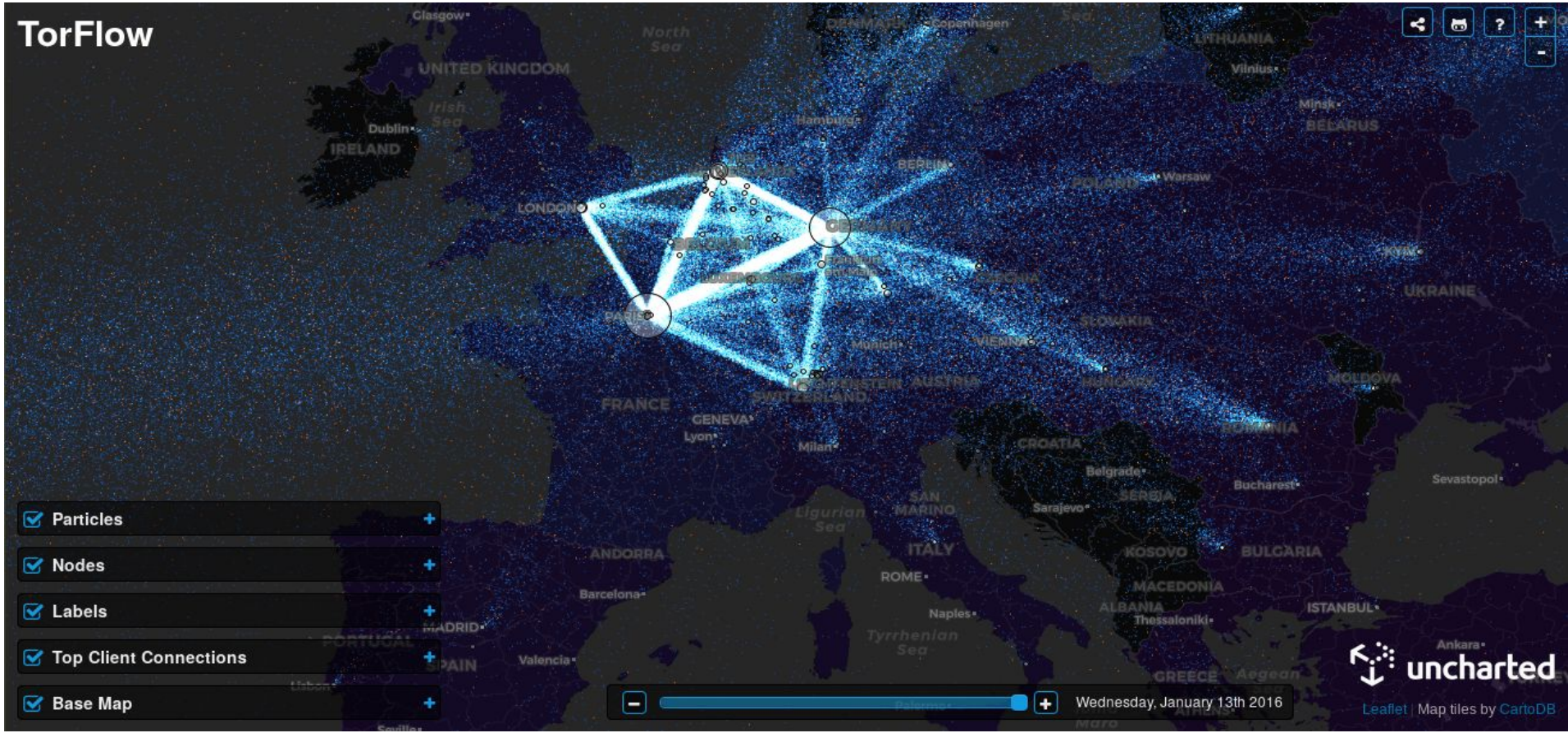
INTRODUCCIÓN

¿Qué es?

- Microservicio que recoge las interacciones entre usuarios de Twitter entorno a una palabra o hashtag.
- Almacena los usuarios como nodos de un grafo cuyos arcos son las interacciones.



Inspiración: TorFlow



Herramientas

- Python
  - Flask
  - Gremlin-Python
- Gremlin CosmosDB (Microsoft Azure)
- TravisCL
- Heroku (Paas)
- Docker
- Zeit
- MIcrosoft Azure

Almacenamiento de Datos

Primera Aproximación

- Almacenamiento en JSON de los nodos
- Utilización de clase

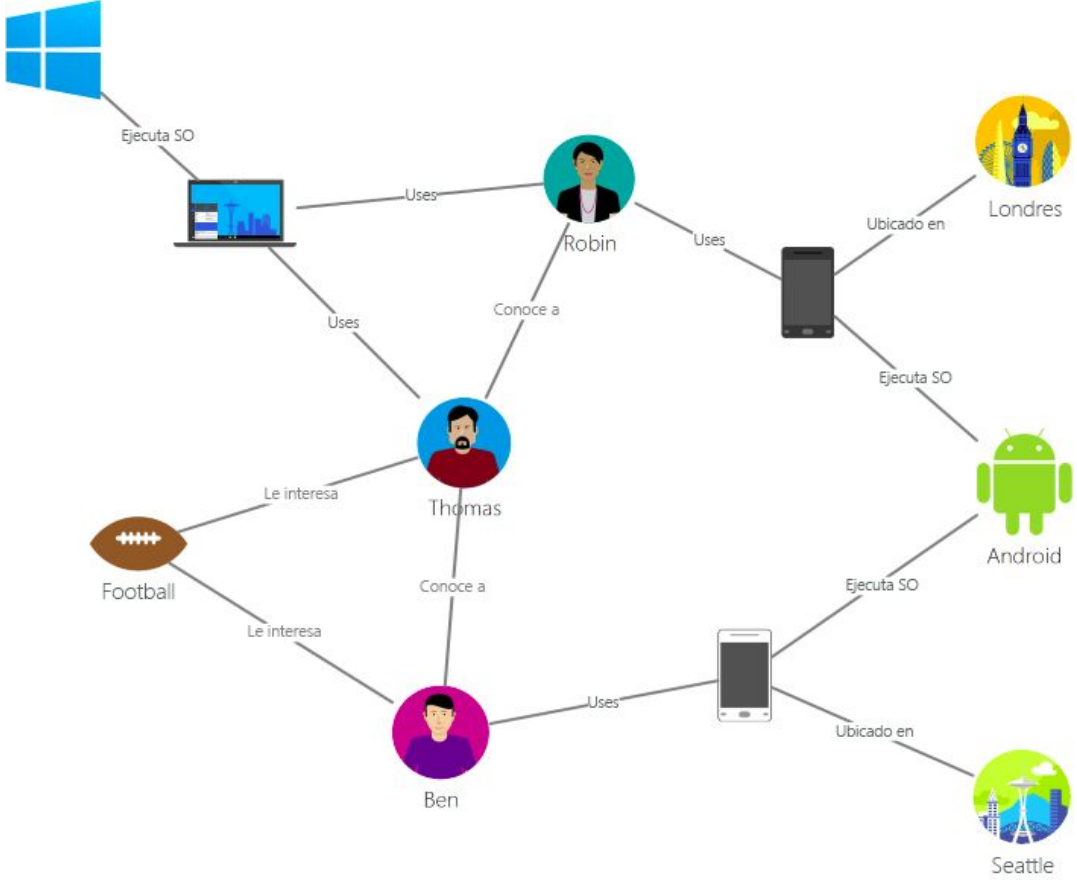
```
class TwitterGraph:
    """ Clase principal del proyecto que gestiona la estructura de datos de mi grafo.
        Basada en la representación de grafo dada en https://www.python-course.eu
        Añadidas algunas modificaciones
    """

    def __init__(self):
        """ Inicializa el objeto grafo """
        try: # Basado en el HitosIV.py de JJ/tests-python
            if os.path.isfile('nodes.json'):
                path = 'nodes.json'
            elif os.path.isfile('/data/nodes.json'):
                path = '/data/nodes.json'
            elif os.path.isfile('./data/nodes.json'):
                path = './data/nodes.json'
            elif os.path.isfile('../data/nodes.json'):
                path = '../data/nodes.json'
            else:
                raise IOError("No se encuentra 'nodes.json'")

            with open(path,"r") as data_file:
                self.__graph_dict = json.loads(data_file.read())
        except IOError as fallo:
            print("Error {:s} leyendo nodes.json".format(fallo))

    def vertices(self):
        """ Devuelve los vértices del grafo """
        return list(self.__graph_dict.keys())

    def aristas(self):
        """ Devuelve las aristas del grafo """
        return self.__generar_aristas()
```

```
def add_arista(self, arista):
    """ arista es un conjunto, tupla o lista """
    arista = set(arista)
    vertice1 = arista.pop()
    if arista:
        vertice2 = arista.pop()
    else:
        vertice2 = vertice1
    if vertice1 in self.__graph_dict:
        self.__graph_dict[vertice1].append(vertice2)
    else:
        self.__graph_dict[vertice1] = vertice2

def elimina_conexion(self, vertice1, vertice2):
    """ Elimina la arista entre vertice1 y vertice2 """
    aristas = self.__graph_dict[vertice1]
    if aristas:
        aristas.remove(vertice2)
        self.__graph_dict[vertice1] = aristas
    aristas = self.__graph_dict[vertice2]
    if aristas:
        aristas.remove(vertice1)
        self.__graph_dict[vertice2] = aristas

def elimina_vertice(self, vertice):
    """ Elimina un vértice del grafo (y todas sus conexiones) """
    if vertice in self.__graph_dict:
        del self.__graph_dict[vertice]
        for vertex in self.__graph_dict:
            if vertice in self.__graph_dict[vertex]:
                self.__graph_dict[vertex].remove(vertice)
```

GREMLIN COSMOS DB

- Azure Cosmos DB es el servicio de base de datos con varios modelos y distribución global de Microsoft para aplicaciones críticas
- Gremlin API de Azure Cosmos DB se usa para almacenar y operar en los datos del grafo.
- Almacena:
  - Vértices: En mi caso, usuarios.
  - Bordes: Relaciones entre vértices.
  - Propiedades: Información acerca de los bordes y los vértices

GREMLIN COSMOS DB



- Imagen en

https://docs.microsoft.com/es-es/azure/cosmos-db/graph-introduction

GREMLIN COSMOS DB

- Es necesario crear una base de datos (db-iv-ugr) y grafo (grafo-iv-ugr)
- Se define usuario y contraseña para la conexión.

```
self.cliente = client.Client('wss://twitterlocationflow.gremlin.cosmosdb.azure.com:443/','g',
                             username = os.environ['CLIENT_USERNAME'],
                             password = os.environ['CLIENT_PASSWD'],
                             message_serializer = serializer.GraphSONSerializersV2d0())
```

¡ES NECESARIO CONTEMPLAR LA SEGURIDAD DE LA BASE DE DATOS!

API

API: Guía de Usuario

- Query status: `/status`
- Query all vertex saved: `/vertices`
- Add a vertex: `/vertices/add/<vertex>`
- Add an edge: `/aristas/add/<edge>`
- Remove a vertex: `/vertices/remove/<vertex>`
- Remove an edge between two vertex: `/aristas/remove/<ver1>/<ver2>`
- Count the number of vertex: `/vertices/cuenta_vertices`
- FInd a path between two vertex: `/vertices/find_path/<vert1>/<vert2>`

Tests

TravisCI

Unittest

- Se definen tests para la clase y para la API
- Los tests de la clase están totalmente relacionados con la base de datos

Despliegues

Heroku: https://iv1819-twitterlocationflow.herokuapp.com/

- Despliegue después de cada subida a Github (pasando los tests).
- Necesario *Procfile* con el siguiente formato:

<process type>: <command>

  - <process type>: Nombre del comando. Al ser web, es el único proceso que recibe tráfico HTTP externo
  - <command>: Comando que toda dyno del proceso debe ejecutar al iniciarse

```
1   web: gunicorn app:app
```

Docker: https://hub.docker.com/r/luisbalru/twitterlocationflow/

- Necesario Dockerfile
- Se crea nueva imagen después de cada push

¡NO SE DEBE COPIAR TODO!

```
1  FROM python:3.6
2
3  WORKDIR /twitterlocationflow
4
5  COPY . /twitterlocationflow
6
7  RUN pip install --trusted-host pypi.python.org -r requirements.txt
8
9  EXPOSE 80
10
11 CMD ["python3", "app.py"]
```

Heroku+Docker: https://docker-tlf.herokuapp.com

- Nuevo despliegue. Se crea un *heroku.yml*
- Instalación del plugin Manifest (para declarar la existencia de una app ya creada)
  - heroku plugins:install @heroku-cli/plugin-manifest
  -
- Logging
  - heroku container:login
- Construcción y subida de imágenes:
  - heroku container:push <process-type>
- Entrega:
  - heroku container:release web

Zeit: https://twitterlocationflow-wodotzcwyc.now.sh

- Necesario *now.json*
- En desuso y con muchos errores
- Poca documentación

No es práctico

Microsoft Azure: ¿Por qué?

- Mala experiencia anterior con Google Cloud
- Ética: software libre + financiación para educación
- Grandes herramientas a nuestra disposición (por ejemplo CosmosDB)
- Extensa documentación y buena compatibilidad con Python

Microsoft Azure: Orquestación de la Máquina Virtual con Vagrant

- Elemento clave: Vagrantfile
- Características de mi máquina
  - Ubuntu Server 16.04 LTS
  - Localización: Sur del Reino Unido
  - Standard_F1(1 core, 2GB RAM)
  - Dummy-box (sin ella, falla)
  - Red virtual privada para aislar el
  - servicio de bases de datos.

```
Vagrant.configure('2') do |configuracion|

  configuracion.vm.box = 'mv-iv'
  configuracion.vm.box_url = 'https://github.com/msopentech/vagrant-azure/raw/master/dummy.box'
  configuracion.vm.network "private_network", guest: 80, host: 80
  configuracion.ssh.private_key_path = '~/.ssh/id_rsa'

  configuracion.vm.provider :azure do |az, override|
    az.vm_image_urn = 'Canonical:UbuntuServer:16.04-LTS:latest'
    az.vm_size = 'Standard_F1'
    az.location = 'uksouth'
    az.tcp_endpoints = '80'
    az.vm_name = "twitterlocationflow"
    az.resource_group_name= "tlf-iv"
    az.tenant_id = ENV["AZURE_TENANT_ID"]
    az.client_id = ENV["AZURE_CLIENT_ID"]
    az.client_secret = ENV["AZURE_CLIENT_SECRET"]
    az.subscription_id = ENV["AZURE_SUBSCRIPTION_ID"]

  end

  configuracion.vm.provision "ansible", run:"always" do |ansible|
    ansible.playbook = "./provision/playbook.yml"

  end

end
```

Microsoft Azure: Provisionamiento con Ansible

- Instalación del software más general
- *playboook.yml*
- Llamada a procedimiento desde Vagratfile
- Para provisionar, *vagrant provision*

```
- hosts: all
  remote_user: vagrant
  tasks:
    - name: Instalar git
      become: true
      apt: name=git state=installed update_cache=true

    - name: Instalar python3 y extras
      become: true
      apt: name={{ item }} state=installed update_cache=true
      with_items:
        - python3
        - python3-setuptools
        - python3-pip

    - name: Actualizar sistema
      become: true
      command: apt update

    - name: Autoremove
      become: true
      command: apt autoremove

    - name: Update pip
      command: pip3 install --upgrade pip
```

Microsoft Azure: Despliegue con Fabric

```
def InstalarApp():
    run('git clone https://github.com/luisbalru/TwitterLocationFlow.git')
    with cd('TwitterLocationFlow'):
        run('pip3 install --user -r requirements.txt')


def ActualizarApp():
    with cd('TwitterLocationFlow'):
        run('git pull')
        run('pip3 install --user -r requirements.txt')


def IniciarApp():
     # Iniciamos el servicio web.
    ActualizarApp()
    with shell_env(C_URL='wss://twitterlocationflow.gremlin.cosmosdb.azure.com:443',C_USER=os.environ['CLIENT_USERNAME'], C_PASS =
        run('cd TwitterLocationFlow && python src/genera_credenciales.py $C_URL $C_USER $C_PASS')
        sudo('nohup gunicorn app:app -b 0.0.0.0:80 & echo $! > pid.txt')

def KillApp():
    with cd('TwitterLocationFlow'):
        sudo('kill -9 `cat pid.txt`')
```

Microsoft Azure: Despliegue con Fabric

genera_credenciales.py:

```
1 import json
2 import sys
3
4 C_URL = sys.argv[1]
5 C_USER = sys.argv[2]
6 C_PASS = sys.argv[3]
7
8 data = {'C_URL': C_URL, 'C_USER' : C_USER, 'C_PASS' : C_PASS}
9
0 with open('data.txt', 'w') as outfile:
1         json.dump(data,outfile)
```

¿Preguntas?

¡Muchas gracias!