Building the Data Science Platform with Airflow @Near

Agenda

- Airflow - Development to Production - Architecture
- Airflow DAGs for different use cases
- Case study - Airflow for Engage and Insights
- Q&A

Near - Corporate Overview

Overview

2012
Established Year

USD $134Mn to date
$100Mn funding in July 2019

Global Presence
Strong presence in USA, EUR, SEA, ANZ, JPN

Scale/Data

1.6 Billion
Users

Across 44
Countries

70 Million+
Places

Marquee Investors

Telstra
CISCO
SEQUOIA CAPITAL
THE ENTREPRENEURS BEHIND THE ENTREPRENEURS
J.P.Morgan
global brain
GREATER PACIFIC

Acquisitions

2021 UM (formerly UberMedia) – provider of data intelligence and analytics solutions based in Pasadena, CA

2020 Teemo - Location intelligence company based in Paris

Nearverse – The Data Universe – Unified and Anonymized

A platform designed to merge online and offline consumer data to give brands the most holistic & actionable view of their consumers

1 Online data
100% Anonymized

Apps
In-app usage

Content
Content consumption pattern

Devices
Device make and model

near

2 Offline data
100% Anonymized

Mobility
People's location data from aggregators, partners, apps, public hotspots

Lifestyle
Places of Interest, Visitation patterns

Work or Home
Cross device usage

Online + offline data - Unified consumer view - Actionable insights - Unprecedented business value

What Near Does…

Enterprise Data

1 Sync And Aggregate

Places

e.g.: Stores, Zip Codes, Cities

People

e.g.: customers, Segments, Audiences

PROXIMA

2 Derive Intelligence

- Visitation / Footfall Trends
- Pathing – Before and After
- Trade Area / Home Locations
- Dwell Time / Distance Travelled

- Demographics
- Brand Affinity / Purchase Behavior
- Work Location / Employment Details
- Search / Social behavior

3 Act and Measure

- Site Selection
- Competitor Intelligence
- Trade Area Dynamics
- Supply Chain Optimization

- 360 Degree Customer Journey
- Customer Engagement
- Competitor Conquesting
- Attribution ROI

Airflow - Deployment Architecture

Data Science Models @ Near

Foundational

Cross Identity solves for linking user attributes to an individual using Graph Algorithms

Stay points identify pings when user is stationary

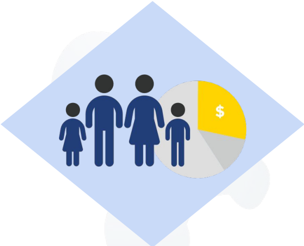
Visit History model user visit to Places of Interest - POIs accurately

Data Accuracy

PlaceMatrix improves Accurate Place Boundaries or Polygonal area of a Place-of-Interest

High Density Points are removed by a data-driven approach

User Behavior



Audience Estimates is a real-time survey about people at places

Properties about people and places helps in creating segments for marketing purposes

Deployment of Airflow

Edge Node

Airflow Configuration

1. Celery Executor
2. MySQL Database
3. Incoming WebHook
4. Python Env
5. Redis
6. Pools / connections

EMR Cluster

Jobs Deployed

1. Pyspark
2. Java Jars
3. Bash Scripts
4. Python

Benefits with Airflow

1. Ease of Deployment - Python Scripting
2. Scalability - Multiple jobs and execution (celery executor)
3. Operators - (BashOperator, PythonOperator, SparkSubmitOperator)
4. Webhooks - Slack, Github
5. Task Dependency Management
6. Connection with different systems - S3, Presto, Hive, Redis, Platform APIs
7. Proactive Monitoring / Alerting
8. Rerunning Failed Jobs (Idempotent)

Replica in Dev, Staging and Production

Environments we operate in

Local Execution

- EDA

Dev Environment

- Feature Engineering
- Model Training
- Performance Evaluation
- Airflow DAG

develop branch + feature branches

Staging Environment

- Github deployment with Jenkins
- Code Testing
- Packaging to egg / Docker

release v1.0 branch

Prod Environment

- CICD Pipeline
- Monitoring and Alerting
- Production API Integration

prod _deploy branch

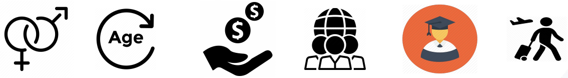
Data Science Models with Airflow

Batch Mode

Demographics

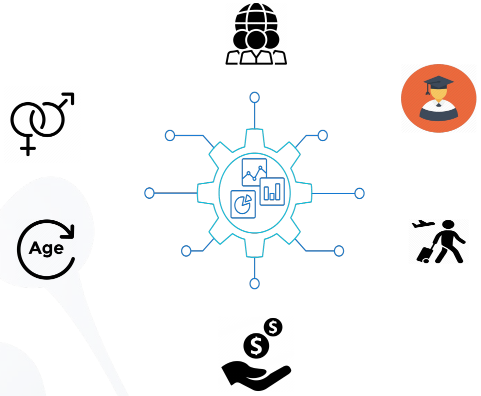
Brand Affinity / Propensity

Home Location



On Demand Mode

Aggregate Insights

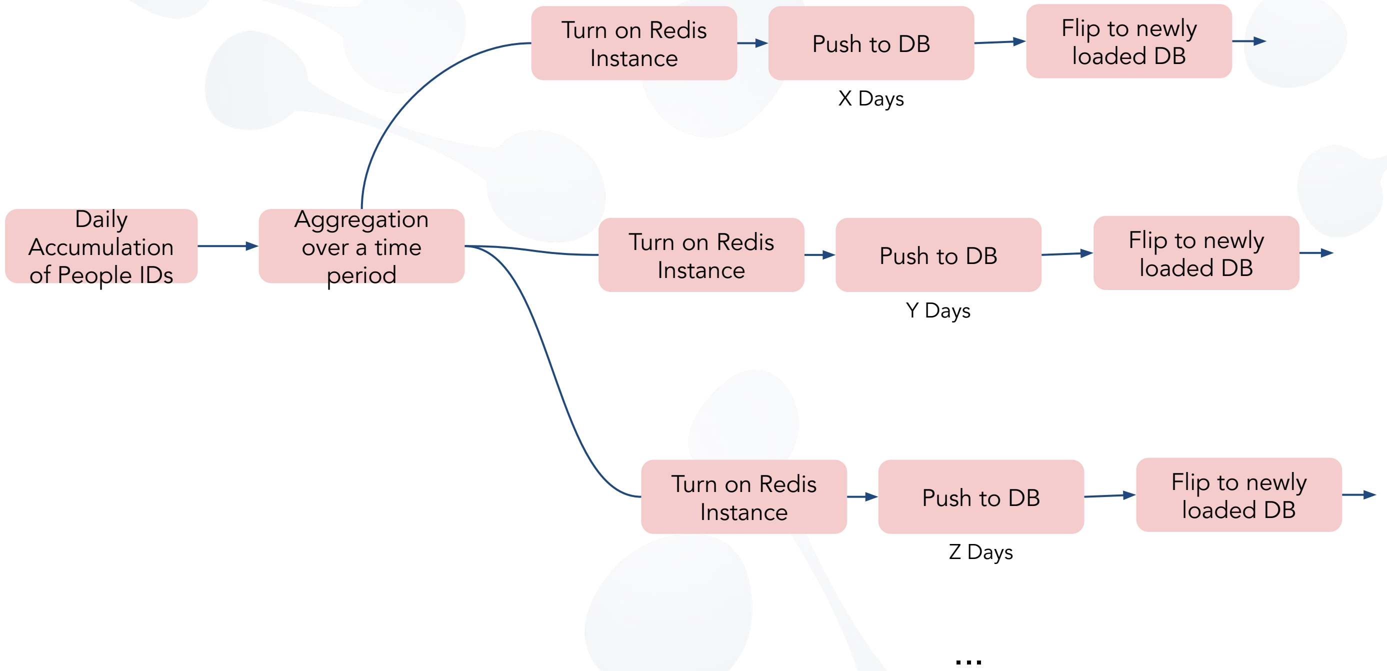


Dynamic Mode

Campaign Insights

Case Study - Audience Estimation, Engage DSP and Allspark Insights

Batch WorkFlow - Estimations



- Accurate estimation of people count at a given location.
- Input data from s3 and output to redis
- Daily updation of data for accurate predictions
- Ensure no downtime
- Automated start, stop and flipping of instances

Code Snippets

```python
dag = DAG('get_dummy', default_args=default_args,
        schedule_interval=None, max_active_runs=5)

spark_conn_id = 'spark_default'


t1 = BashOperator(task_id='get_dummy_instance',
                  xcom_push=True,
                  bash_command='curl -X GET "dummy_instance_id"',
                  dag=dag
                  )

(t1)

def redis_instance(**kwargs):
    ti = kwargs['ti']
    if ti.xcom_pull('get_dummy_instance')=="x":
        return("y")
    elif ti.xcom_pull('get_dummy_instance')=="y":
        return("x")


pull_task = PythonOperator(
    task_id='flip_ dummy_instance',
    python_callable=redis_instance,
    provide_context=True,
    dag=dag)

trigger = TriggerDagRunOperator(
    trigger_dag_id="dummy_dag_{{ti.xcom_pull('flip_dummy_instance')}}",
    task_id='call_dummy_dag',  # Ensure this equals the dag_id of the DAG to trigger
    dag=dag
)

pull_task.set_upstream(t1)
trigger.set_upstream(pull_task)
```

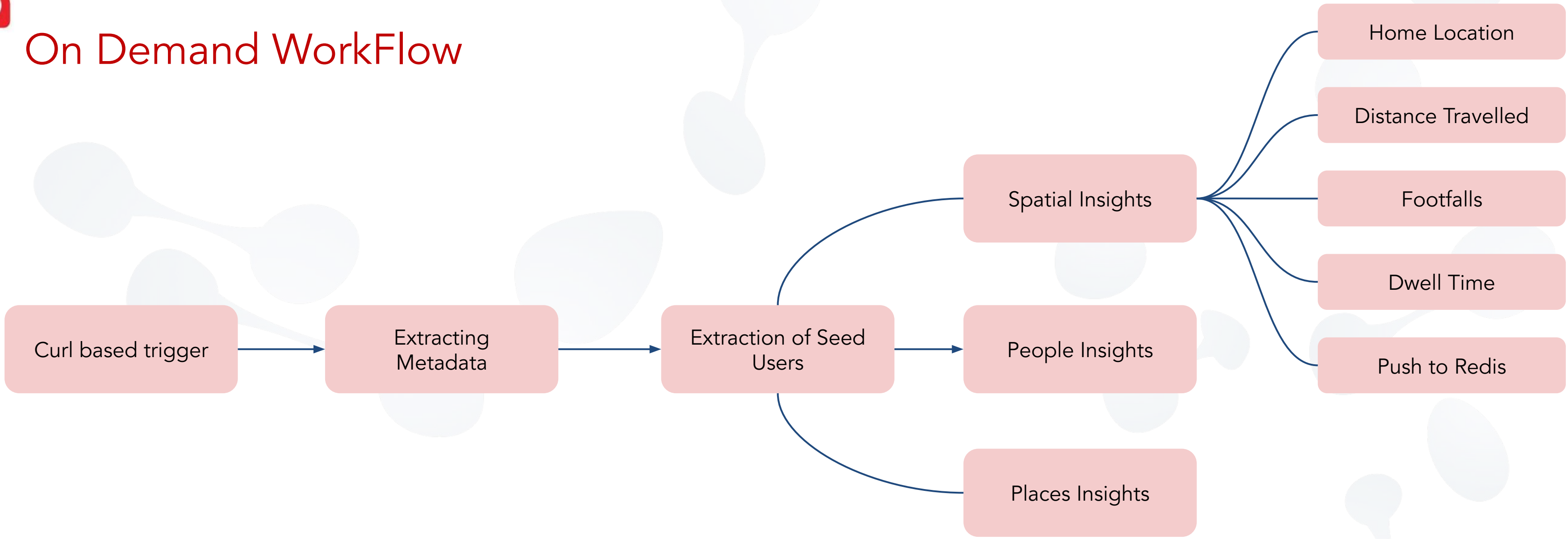
```python
def subDag1(parent_dag_id,sub_dag_id,default_args,spark_conn_id):

    dag_1 = DAG(parent_dag_id+"."+sub_dag_id, default_args=default_args,
        schedule_interval=None, max_active_runs=3)

    f1 = BashOperator(task_id='start_dummy_instance',
        bash_command='aws ec2 start-instances --dummy',
        dag=dag_1)


    f2 = BashOperator(task_id="delay_bash_task",dag=dag_1,bash_command="
        sleep 5m")


    f3 = SSHOperator(
        ssh_hook=ssh_hook,
        task_id='ssh_dummy_operator',
        command = "command to clear and restart redis",
        do_xcom_push=True,
        dag=dag_1)


    f2.set_upstream(f1)
    f3.set_upstream(f2)
```

Code Snippets

```
for days in ['1']:
    host = "a1"
    port = 'p1'
    instance_id = "id1"

    t16 = SparkSubmitOperator(
        task_id = 'taskid1_' + param1 + '_' + param2 + '_' + param3,
        conn_id = spark_conn_id,
        name = 'job1_' + param1 +'_' + param2 + '_' + param3,
        num_executors = 12,
        executor_cores = 2,
        executor_memory = '20G',
        dag=dag_2,
        depends_on_past=True,
        env_vars = {'python_env'},
        jars= 'jar1.jar',
        java_class = 'org.abc.jar1',
        application='connector1.jar',
        application_args=[host,
        port,
        BASE_PATH + 'filepath/{}/..'.format(param1,param2,param3,
            exec_date_year, exec_date_month, exec_date_day)]
    )
```

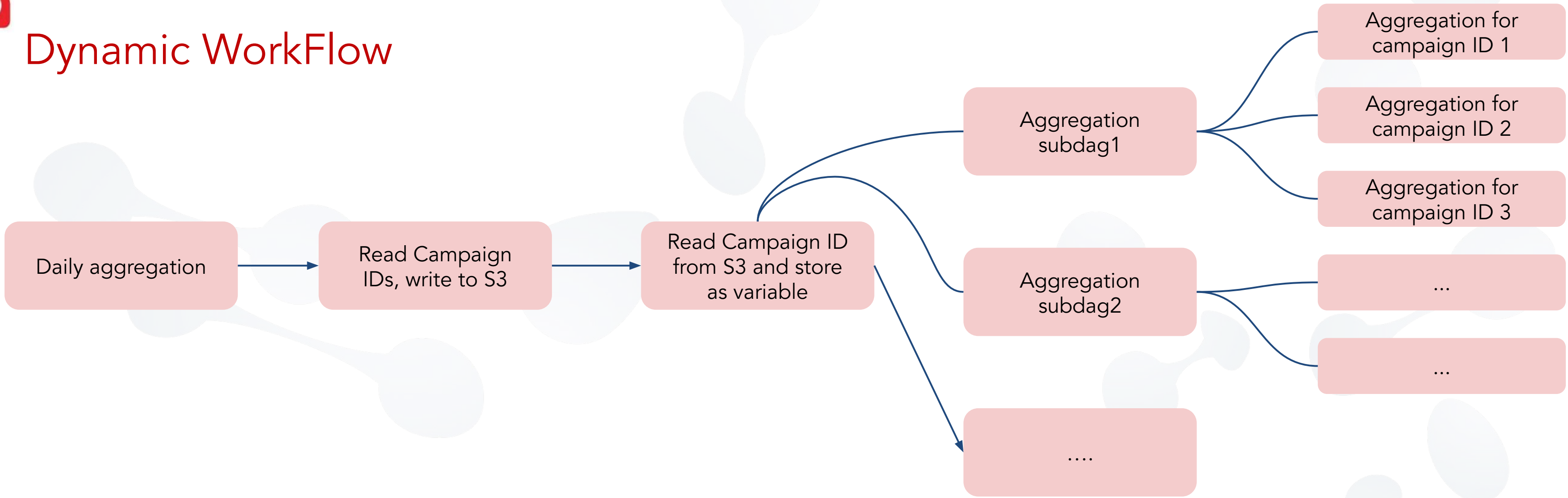
```
t17 = BashOperator(task_id='flip_instance',
    bash_command='curl -v -X PUT --data instancename',
    depends_on_past=True,
    dag=dag_2
)

t18 = BashOperator(task_id='stop_instance',
    bash_command='aws ec2 stop-instances instancename',
    dag=dag_2
)

t2.set_upstream(t1)
t3.set_upstream(t2)
```

On Demand WorkFlow



1. Event-driven approach to triggering the the airflow DAGs - Experimental API - pass parameters
2. Retriggers for failed jobs
3. Controlled dependency management (modular code)
4. Slack Alerts

Dynamic WorkFlow



1. Run on daily campaigns to generate insights
2. Campaign run time - 15-90 days
3. Dynamic Parameter passing - campaign IDs

Code Snippets

```
def x_function(**kwargs):
    s3 = s3fs.S3FileSystem(anon=False)
    run_date = datetime.strptime(kwargs["ds"], '%Y-%m-%d')
    bucket = s3.ls('demo.csv')
    for fileN in bucket:
        if "part" in fileN:
            df = pd.read_csv(s3.open('{}'.format(fileN),mode='rb'))
            df = df.replace(np.nan, 0)
            LineItemIDlist = df['ID1'].tolist()

            LineItemIDlist = [int(x) for x in LineItemIDlist if (x != 0)]
            split_size=len(LineItemIDlist)/5
            no_of_splits=math.ceil(split_size)

            line_item_ID_list_of_list= [LineItemIDlist[i:i + no_of_splits] for i in range(0, len(LineItemIDlist), no_of_splits)]

            CampaignIDlist = df['ID2'].tolist()
            CampaignIDlist = [int(x) for x in CampaignIDlist if (x != 0)]
            split_size=len(CampaignIDlist)/5
            no_of_splits=math.ceil(split_size)

            campaign_item_ID_list_of_list= [CampaignIDlist[i:i + no_of_splits] for i in range(0, len(CampaignIDlist), no_of_splits)]

            print (run_date.strftime('%Y-%m-%d'))
            Variable.set("execution_date_prod",run_date.strftime('%Y-%m-%d'))
            Variable.set("ID1_1_"+run_date.strftime('%Y-%m-%d'), line_item_ID_list_of_list[0])
            Variable.set("ID2_1_"+run_date.strftime('%Y-%m-%d'), campaign_item_ID_list_of_list[0])
            Variable.set("ID1_2_"+run_date.strftime('%Y-%m-%d'), line_item_ID_list_of_list[1])
            Variable.set("ID2_2_"+run_date.strftime('%Y-%m-%d'), campaign_item_ID_list_of_list[1])
            Variable.set("ID1_3_"+run_date.strftime('%Y-%m-%d'), line_item_ID_list_of_list[2])
            Variable.set("ID2_3_"+run_date.strftime('%Y-%m-%d'), campaign_item_ID_list_of_list[2])
            Variable.set("ID1_4_"+run_date.strftime('%Y-%m-%d'), line_item_ID_list_of_list[3])
            Variable.set("ID2_4_"+run_date.strftime('%Y-%m-%d'), campaign_item_ID_list_of_list[3])
            Variable.set("ID1_5_"+run_date.strftime('%Y-%m-%d'), line_item_ID_list_of_list[4])
            Variable.set("ID2_5_"+run_date.strftime('%Y-%m-%d'), campaign_item_ID_list_of_list[4])


t4= PythonOperator(
    task_id='x_task',
    python_callable=x_function,
    provide_context=True,
    dag=dag)
```

Conclusion and Future Work

Summary

1. Integration with newer data science models - reinforcement learning, deep learning, graphical networks
2. Upgrade to 2.0 - to improve security and use LDAP
3. Use of more hooks and operators
4. Dockerization
5. Use of Celery + Kubernetes Executor - CeleryKubernetes Executor