Airflow the perfect match in our Analytics Pipeline
AGENDA

1. Why we met?
2. How we met?
3. The first date!
4. Fun dates!
5. Is there any dynamic in between?
6. Recap and conclusion
About LOVOO

- LOVOOO is a dating and social app and the place for chatting, live streaming, watching streams and getting to know people.
- Germany - Dresden & Berlin - 2011
- Acquired by The Meet Group (NASDAQ:MEET) in 2017
- Top 3 Dating App in Europe
- + 280 TB of Data
- ~ 6 TB Monthly Growth
- + 3 TB daily total aggregated data
- + 36 TB Swipes (162,824,303,474)
THE TEAM

Analytics

- 1 Head
- 6 Data Analysts
- 2 BI Architects
- Product
- Finance
- Marketing
- Talent Management
- Customer Insights
- CRM
What can you expect?

My main purpose today is to tell you about our journey with Airflow as well as a few different use cases that could also boost the work of your Analytics/BI team on a daily basis.

• Pieces of code (examples)

• Way too many screenshots
AGENDA

1. Why we met?
2. How we met?
3. The first date!
4. Fun dates!
5. Is there any dynamic in between?
6. Recap and conclusion
On-premise

- Cloudera
- Tableau
- Hadoop
- Hive
- Oozie
We went Cloud
Data Processing

Backend

mongoDB
EU-Bridge

Data Loading

Google Kubernetes

Pub-Sub

Google - Firebase

BigQuery

Airflow Composer

Cloud Storage

Payment Providers,
Appsumer, Adjust, CRM,
etc…

Google Sheets

Airflow Summit 2020

Sergio Camilo Fandiño Hernández
Senior Business Intelligence Architect @LOVOO
WHAT REALLY MATTERS...

Analytics

Data-Core

Payment Providers,
Appsumer, Adjust,
Redshift, etc…

Airflow Composer

Google Sheets

Cloud Storage

BigQuery

Python

R

Tableau

Google - Firebase

upsolver
AGENDA

1. Why we met?

2. How we met?

3. The first date!

4. Fun dates!

5. Is there any dynamic in between?

6. Recap and conclusion
Orchestration Tool

- Identify what is out there
- Costs?
- Scalability?
- Data sources compatibility?
- Knowledge/Human Resources?
Airflow

- Great community
- Game changer
- Mobile App
- Python
- BigQuery
Google Cloud Composer

- Fully Managed Airflow
- Scalable
- IAP - Secure
- Focus on building the Analytics data pipeline
- Ease of implementation
Google Cloud Composer

- Fully Managed Airflow

⭐ Alpha

This is an alpha release of Cloud Composer. This product might be changed in backward-incompatible ways and is not recommended for production use. It is not subject to any SLA or deprecation policy. This product is not intended for real-time usage in critical applications.

- Focus on building the Analytics data pipeline

- Ease of implementation

⚠ Confidential Material: This page is confidential. Do not share or discuss until authorized to do so.
AGENDA

1. Why we met?
2. How we met?
3. The first date!
4. Fun dates!
5. Is there any dynamic in between?
6. Recap and conclusion
TODO List

- SQL Scripts —> Data Modeling
- DAGs
- Permissions - Service Accounts
- Data Importers
- Create a Composer Environment
- How do we deploy? —> CI/CD
GROWING TOGETHER!

CI/CD

- YAML
- Cloud Build
- Trigger
- Checks Passed
- Version Control
- DAGs.py
- Importers
- SQL
- Slack
- Checks Passed

Cloud Composer

Cloud Storage

Cloud Storage
GROWING TOGETHER!

CI/CD

Google Cloud Build - GitHub

Build: master

Build logs
Status: WORKING
trigger name: trigger-714738f9-99be-41f0-8086-5739faeb2d4c

Google Cloud Build - GitHub

Build: master

Build logs
Status: SUCCESS
trigger name: trigger-714738f9-99be-41f0-8086-5739faeb2d4c
AGENDA

1. Why we met?
2. How we met?
3. The first date!
4. Fun dates!
5. Is there any dynamic in between?
6. Recap and conclusion
DAGs

- 26 DAGs
- Sub-DAGs
- Branching
- Jinja Templating
- Hooks
- Pools
- Trigger rules

Operators

<table>
<thead>
<tr>
<th>DAG</th>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>adevents-repair</td>
<td>None</td>
</tr>
<tr>
<td>airflow_monitoring</td>
<td>None</td>
</tr>
<tr>
<td>analytics_jobs</td>
<td>Daily</td>
</tr>
<tr>
<td>analytic_jobs_live</td>
<td>Daily</td>
</tr>
<tr>
<td>antispam-creditterm-detection</td>
<td>Daily</td>
</tr>
<tr>
<td>antispam-reputation-modeltraining</td>
<td>Daily</td>
</tr>
<tr>
<td>appuser-importer</td>
<td>Daily</td>
</tr>
<tr>
<td>appuser-importer-hayi</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_backend_msg_events</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_data_check</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_firestore_importer</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_firestore_live_events</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_marketing_events_jobs</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_marketing_jobs</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_payment_provider_api</td>
<td>Daily</td>
</tr>
<tr>
<td>bi_redshift_economy_data</td>
<td>Daily</td>
</tr>
</tbody>
</table>
The Core

Analytics - Workflow

Sub DAGs

- wait-for-user_merge
- wait-for-copy-non-enriched-events
- wait-for-device_merge
- wait-for-currency-importer
- wait-for-bi_firestore_importer
- wait-for-bi_backend_tmge_events
- wait-for-bi_redshift_economy_data

Pretty On The Outside...
The Core

Sub DAG
COMMUNICATION IS VITAL…

Reports!


Would you like to send the Report to all the recipients

Yes

Sergio Camilo Fandiño Hernández
Senior Business Intelligence Architect @LOVOO
COMMUNICATION IS VITAL…

Tableau Extracts

 bash_command
 1 curl -X POST http://35.240.226.12:8008

6 $tabcmd runschedule "Daily Extract Refreshes (9:00 AM)"
COMMUNICATION IS VITAL...

Is Airflow finished?

by the way, this is branching...
Error Alerting

'on_failure_callback': on_failure_callback,

def on_failure_callback(context):
    operator = SlackAPIPostOperator(
        task_id='notify_fail',
        channel='#the_channel',
        token='your_Slack_bot_token',
        username='airflow-bot',
        text= str('*Airflow requires an action* {} Task: {}).format(
            str(context['dag']), str(context['task_instance']))
    )

    return operator.execute(context=context)

Airflow requires an action  ->  DAG Name: <DAG: analytics_jobs.2_aggr_level>  -  Task Name: <TaskInstance: analytics_jobs.2_aggr_level.tableau_refresh_extracts 2020-06-22T05:00:00+00:00 [failed]>
Integrating Data Sources

**BEING FLEXIBLE IS A BIG FLEX!**

this code belongs to the DAG.py file

```python
t1a = PythonOperator(
    task_id='load_table_lovoo_transaction_groups_{}.format(i),
    python_callable=import_day_callable,
    provide_context=True,
    templates_dict={'exec_date': exec_date, 'table_name':'lovoo_transaction_groups'},
    dag=dag)
```
Integrating Data Sources

```python
from BI.redshift_importer import import_datalake_redshift_data

def import_day_callable(**kwargs):
    exec_date = kwargs.get('templates_dict').get('exec_date')
    table_name = kwargs.get('templates_dict').get('table_name')
    return import_datalake_redshift_data(table_name,
                                          'load_job_dataframe_to_bq',
                                          exec_date=exec_date)
```

BEING FLEXIBLE IS A BIG FLEX!
Integrating Data Sources

def postgreSQL_connection():
    try:
        # Using a Hook for getting the Redshift credentials from the Airflow connections
        connection = BaseHook.get_connection("redshift_tmg")
        password = connection.password
        host = connection.host
        dbname = connection.schema
        user = connection.login
        port = connection.port

        conn = psycopg2.connect("dbname='{dbname}' user='{user}' port='{port}' host='{host}' password='{password}'")
        cursor = conn.cursor()

    except Exception as e:
        print("I am unable to connect to the database: " + str(e))

    return cursor, conn

BEING FLEXIBLE IS A BIG FLEX!
Integrating Data Sources

```python
def import_datalake_redshift_data(table_name, method_type, exec_date, **kwargs):
    # Cursor & Connection
    cursor, conn = postgreSQL_connection()

    # Create dynamically a SQL query using the input parameters table_name and exec_date
    query = "select * from a_datalake{} where data_updated_at::date >= '{}'.format(table_name, exec_date)

    # Use the query to request the data using the cursor
    cursor.execute(query)

    # Use any method to upload the data to BigQuery
    df = cursor.fetchall()
    df = pd.DataFrame(df)
    job = client.load_table_from_dataframe(df, table_name, job_config=job_config)

    return whether it was successful or not
```

BEING FLEXIBLE IS A BIG FLEX!
Integrating Data Sources

2 Tables - 2 Days -> ELT in BQ
Data Importers

- Redshift
- Firebase (very dynamic)
- Google Cloud Storage (Adjust, Merger)
- Appsumer, Shopify, Paypal, AppStore, Adyen
- S3 Storage
AGENDA

1. Why we met?
2. How we met?
3. The first date!
4. Fun dates!
5. Is there any dynamic in between?
6. Recap and conclusion
Creating Tasks Dynamically
Creating Tasks Dynamically

1. Creating a plain text with meaningful structure
2. Create a task based on a PythonOperator
3. Define and write your Callable (your custom code)
YES, VERY DYNAMIC...

Creating Tasks Dynamically

```json
{
    "broadcast_start": {
        "id": "81ce53e3-2ca6-48a2-88f7-493f7fc2c364",
        "format": "json"
    },
    "broadcast_update": {
        "id": "de7cbe04-9b53-4b4c-bc18-4d065ed3830e",
        "format": "json"
    },
    "broadcast_view": {
        "id": "2a4a0093-baee-47a2-8017-aebdb469b1b1",
        "format": "json"
    },
    "broadcast_end_view": {
        "id": "3af65f80-94a3-42c4-8ed9-502134605d27",
        "format": "json"
    },
    "broadcast_end": {
        "id": "37fa9fd8-8384-41a7-83aa-1263814b3585",
        "format": "json"
    }
}
```
Creating Tasks Dynamically

this code belongs to the DAG.py file

```python
# Iterates over all the Mapping file and extracts the event name for generating all the task-events
for event_name in event_mapping:
    event_name_task = PythonOperator(
        task_id=str(event_name),
        provide_context=True,
        python_callable=run_import_day,
        templates_dict={'exec_date': exec_date, 'event_name': event_name,
                         'dataset': dataset, 'bucket_name': bucket_name},
        dag=dag)
```
YES, VERY DYNAMIC…

Creating Tasks Dynamically

# Function that will be called by the Python operator and will write a table partition in BQ

def run_import_day(**kwargs):
    dataset = 'events_input_analytics_tmrg_backend'
    bucket_name = 'lovoo-tmrg-transfer'
    import_gcs_to_bq(exec_date=kwargs.get('templates_dict').get('exec_date'),
                     event_name=kwargs.get('templates_dict').get('event_name'),
                     dataset=kwargs.get('templates_dict').get('dataset'),
                     bucket_name=kwargs.get('templates_dict').get('bucket_name'),
    )
Creating Tasks Dynamically

def import_gcs_to_bq(exec_date, event_name, dataset, bucket_name, **op_kwargs):
    # read the structured JSON file
    event_mapping = json.load(read_file)

    # mapping the id and the event_name
    id_event = event_mapping[event_name]['id']

    # gathering the blobs inside the bucket - array of paths
    path_array.append('gs://{0}/{1}/exec_date_file.json'.format(bucket_name, id_event))

    # BigQuery Job to Load the JSON files to a table
    load_job = bq_client.load_table_from_uri(
        tuple(path for path in path_array),
        table_dest, job_config=job_config
    )
Creating Tasks Dynamically

```python
def import_gcs_to_bq(exec_date, project_id, bucket_name, **op_kwargs):
    event_mapping = json.load(open('event_mapping.json'))

    # mapping the id and the event
    id_event = event_mapping

    # gathering the blobs in a list
    path_array.append('gs://{}{}'.format(bucket_name, id_event))

    load_job = bq_client.load_table_from_gcs
    load_job(path_array, table, job_config=job_config)
```

Dynamically this is your custom code (Pseudo-Code)
AGENDA

1. Why we met?
2. How we met?
3. The first date!
4. Fun dates!
5. Is there any dynamic in between?
6. Recap and conclusion
Recap and Conclusion

```python
return KubernetesPodOperator(
    startup_timeout_seconds=60 * 10,  # we need seconds here as int, 10min now
    task_id= 'apppsumer_import_' + iso_date.replace('-', '_'),
    namespace='default',
    image=task_kwargs.get('image'),
    cmd=task_kwargs.get('command'),
    secrets=[apppsumer_pass, service_account],
    env_vars=env_vars,
    name=task_kwargs.get('name'),
    is_delete_operator_pod=True,
    dag=dag,
    dt=dt,
    pool="apppsumer_pool",
    get_logs=True,
    resources=resources,
    affinity={
        'nodeAffinity': {
            # requiredDuringSchedulingIgnoredDuringExecution means in order
            # for a pod to be scheduled on a node, the node must have the
            # specified labels. However, if labels on a node change at
            # runtime such that the affinity rules on a pod are no longer
            # met, the pod will still continue to run on the node.
            'requiredDuringSchedulingIgnoredDuringExecution': {
                'nodeSelectorTerms': [[
                    'matchExpressions': [{
                        'key': 'KubernetesOperator',
                        'operator': 'In',
                        'values': [ 'true',
                    }]}}
```
Recap and Conclusion

- Using an Alpha version (Google Composer) in Production was challenging!
- Focus on what’s important - Google Cloud Composer
- Airflow leverages a bunch of Operators OOTB
- Always room for improvement
- No magic recipe to use - stay flexible
Gracias.

Feedback and Questions

LinkedIn: https://www.linkedin.com/in/fandinoherandez/

Email: sergio.fandino@lovoo.com