

Airflow: The Power of stitching Services Together

based on experience with Google Composer

knapik@google.com

rafalbiegacz@google.com

Airflow Summit, July 2021



CONFIDENTIAL & PROPRIETARY
DO NOT SHARE PROPRIETARY

CONFIDENTIAL AND PROPRIETARY

DO NOT SHARE

DRAFT



Filip Knapik

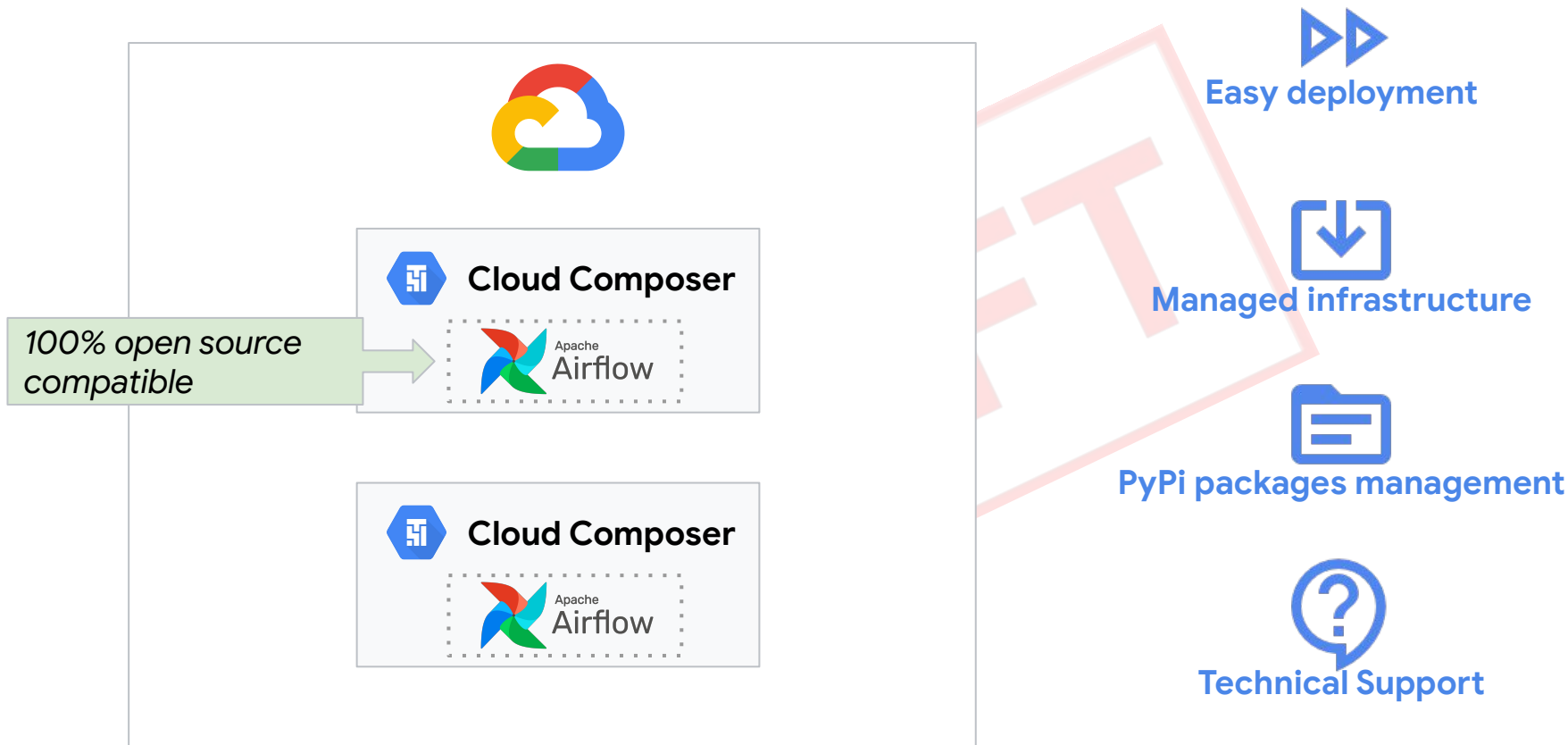
- Cloud Composer Product Manager at Google
- Working with Airflow for 2 years
- 18+ years of IT management experience
- Graduate of Computer Networks and Services at AGH University of Science and Technology in Cracow, Poland

Rafal Biegacz



- [Cloud Composer](#) Engineering Manager
- Has been working on Airflow for 2 years.
- Holds MSc degree in the field of Teleinformatics from Gdansk University of Technology
- Delivers Google Cloud Platform and cloud computing lectures to students of University of Warsaw and Technical University of Warsaw.

Apache Airflow and Cloud Composer

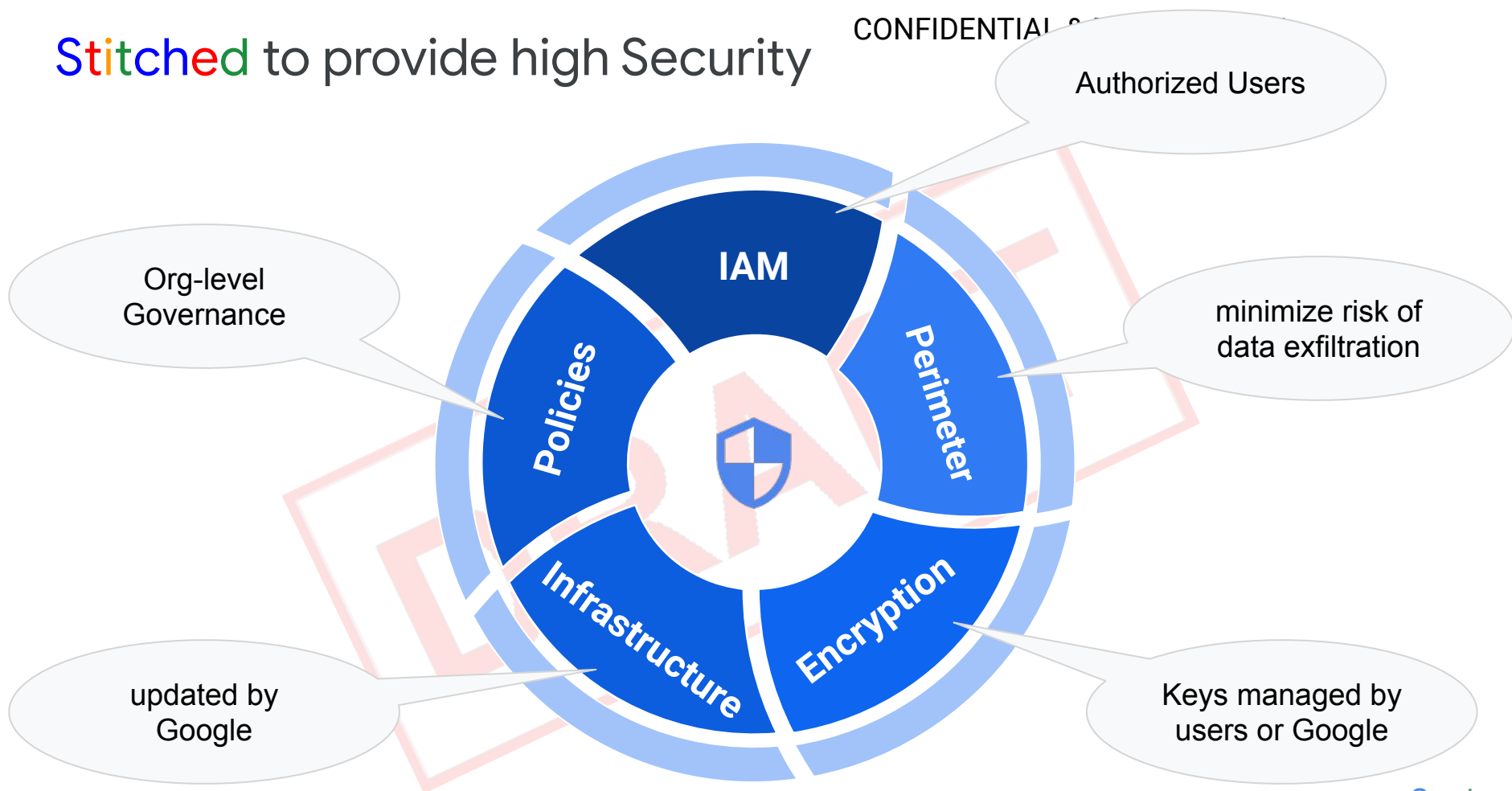


Managed Airflow Environments

Stitching many GCP services to provide
managed Airflow environments
so you can focus on
Airflow and DAG development/execution

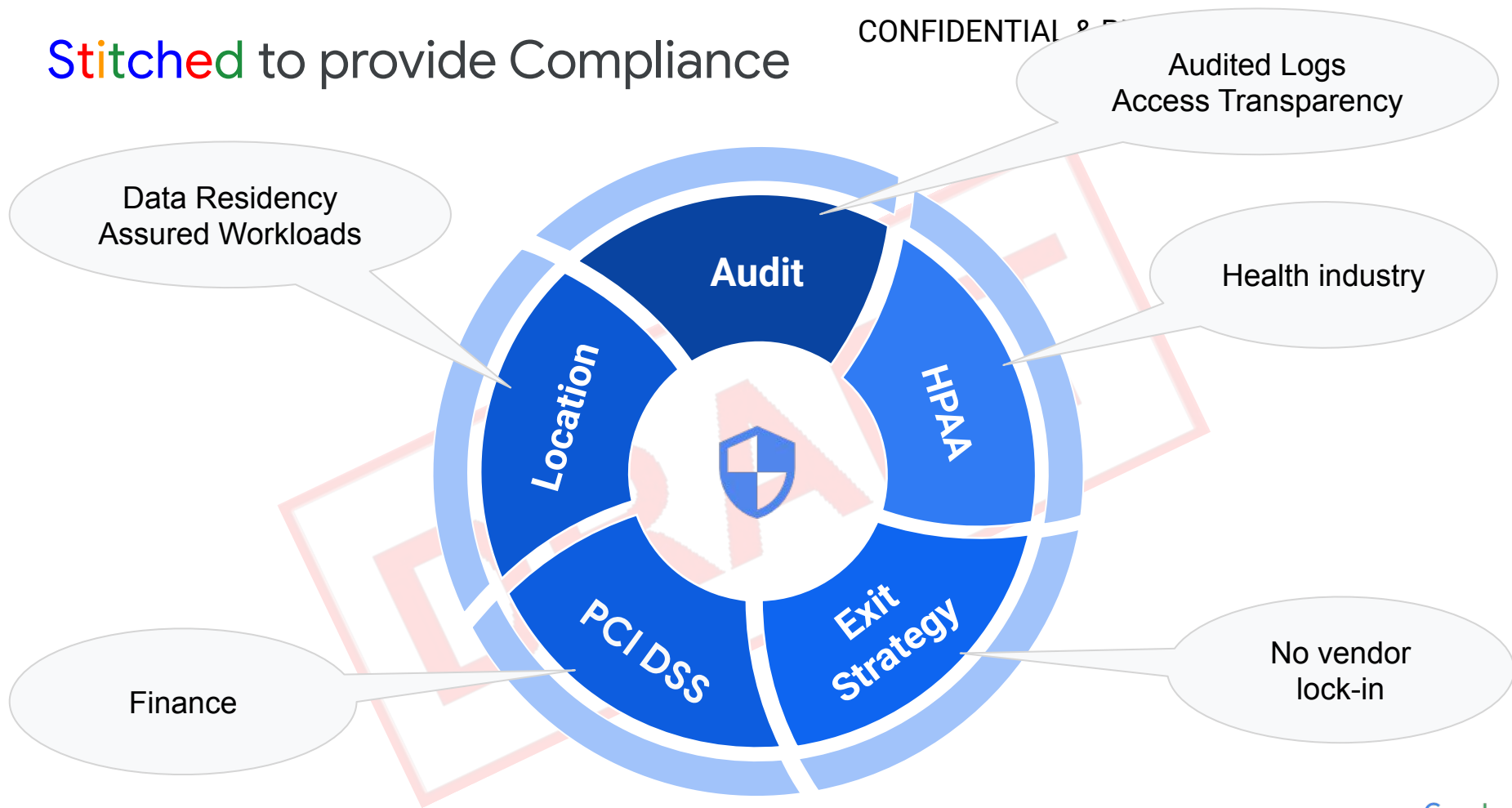
Stitched to provide high Security

CONFIDENTIAL



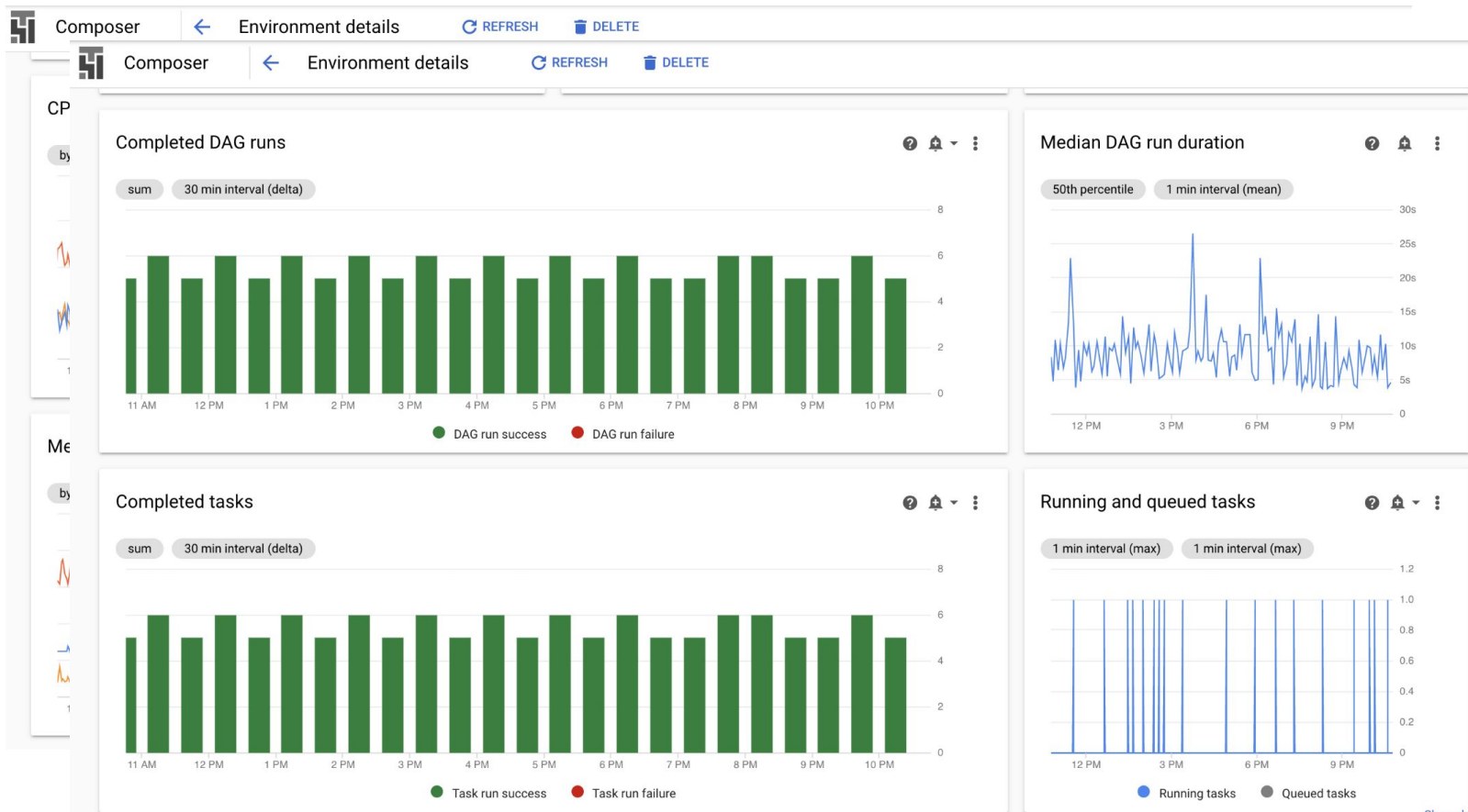
Stitched to provide Compliance

CONFIDENTIAL 8.5



Cloud Composer – Monitoring

CONFIDENTIAL & PROPRIETARY
Monitoring out of the box



Cloud Composer Logging

CONFIDENTIAL & PROPRIETARY
Logging out of the box



Composer



Environment details



REFRESH



DELETE



testenvironment1

This environment is running

MONITORING

LOGS

PREVIEW

ENVIRONMENT CONFIGURATION

AIRFLOW CONFIGURATION OVERRIDES

ENVIRONMENT VARIABLES

LABELS

PYPI PACKAGES

6 HOURS



All logs

Airflow logs

Scheduler



Workers



Web server



DAG processor manager



Composer logs

Composer agent



Build

Worker & Scheduler image



Web server image



Database operations



Monitoring



Infrastructure



SQL proxy

Redis

GCS syncd

Other

Logs Showing 100 messages

Severity

Default



Filter logs



2021-07-04 22:40:57.199 CEST	airflow-scheduler	[2021-07-04 20:40:57,199] {scheduler_job.py:1199} INFO - Executor reports execution of airflow_monitoring.echo e...
2021-07-04 22:40:57.660 CEST	airflow-scheduler	[2021-07-04 20:40:57,656] {scheduler_job.py:1226} INFO - Setting external_id for <TaskInstance: airflow_monitori...
2021-07-04 22:41:31.452 CEST	airflow-scheduler	DAGs # Errors Last Runtime Last Run
2021-07-04 22:42:42.535 CEST	airflow-scheduler	INFO - Finding 'running' jobs without a recent heartbeat
2021-07-04 22:42:42.536 CEST	airflow-scheduler	DAG_PROCESSOR_MANAGER_INFO - Finding 'running' jobs without a recent heartbeat
2021-07-04 22:46:31.580 CEST	airflow-scheduler	<TaskInstance: airflow_monitoring.echo 2021-07-04 20:46:22+00:00 [scheduled]>
2021-07-04 22:46:31.584 CEST	airflow-scheduler	[2021-07-04 20:46:31,584] {scheduler_job.py:970} INFO - Figuring out tasks to run in Pool(name=default_pool) wit...
2021-07-04 22:46:31.585 CEST	airflow-scheduler	[2021-07-04 20:46:31,584] {scheduler_job.py:998} INFO - DAG airflow_monitoring has 0/15 running and queued tasks
2021-07-04 22:46:31.585 CEST	airflow-scheduler	[2021-07-04 20:46:31,585] {scheduler_job.py:1063} INFO - Setting the following tasks to queued state:
2021-07-04 22:46:31.585 CEST	airflow-scheduler	<TaskInstance: airflow_monitoring.echo 2021-07-04 20:46:22+00:00 [scheduled]>
2021-07-04 22:46:31.589 CEST	airflow-scheduler	[2021-07-04 20:46:31,589] {scheduler_job.py:1105} INFO - Sending TaskInstanceKey(dag_id='airflow_monitoring', ta...
2021-07-04 22:46:31.590 CEST	airflow-scheduler	[2021-07-04 20:46:31,590] {base_executor.py:82} INFO - Adding to queue: ['airflow', 'tasks', 'run', 'airflow_mon...
2021-07-04 22:46:31.601 CEST	airflow-scheduler	[2021-07-04 20:46:31,601] {scheduler_job.py:1199} INFO - Executor reports execution of airflow_monitoring.echo e...
2021-07-04 22:46:31.616 CEST	airflow-scheduler	[2021-07-04 20:46:31,616] {scheduler_job.py:1226} INFO - Setting external_id for <TaskInstance: airflow_monitori...
2021-07-04 22:46:38.576 CEST	airflow-scheduler	[2021-07-04 20:46:38,576] {dagrun.py:454} INFO - Marking run <DagRun airflow_monitoring @ 2021-07-04 20:46:22+00...
2021-07-04 22:46:38.625 CEST	airflow-scheduler	[2021-07-04 20:46:38,624] {scheduler_job.py:1199} INFO - Executor reports execution of airflow_monitoring.echo e...
2021-07-04 22:47:03.967 CEST	airflow-scheduler	{dag_processing.py:1075} INFO - Failing jobs without heartbeat after 2021-07-04 20:40:58.632563+00:00
2021-07-04 22:48:05.033 CEST	airflow-scheduler	=====





Stitching Services Together

Stitching - Airflow's magic power

Build pipelines embracing different services is one of the biggest magic powers of Apache Airflow !

- Connecting to each other totally independent services
- End-2-end observability
- Symbiosis with other workflow technologies !!!

Stitching Enablers ...

Out-of-the-box library of Operators, Hooks and Sensors

Do-It-Yourself Operators

Containers

DRAFT

Airflow Operators, Hooks and Sensors

Apache Airflow only in its code base has:

- Operators: > 470
- Sensors: > 70
- Hooks: > 160

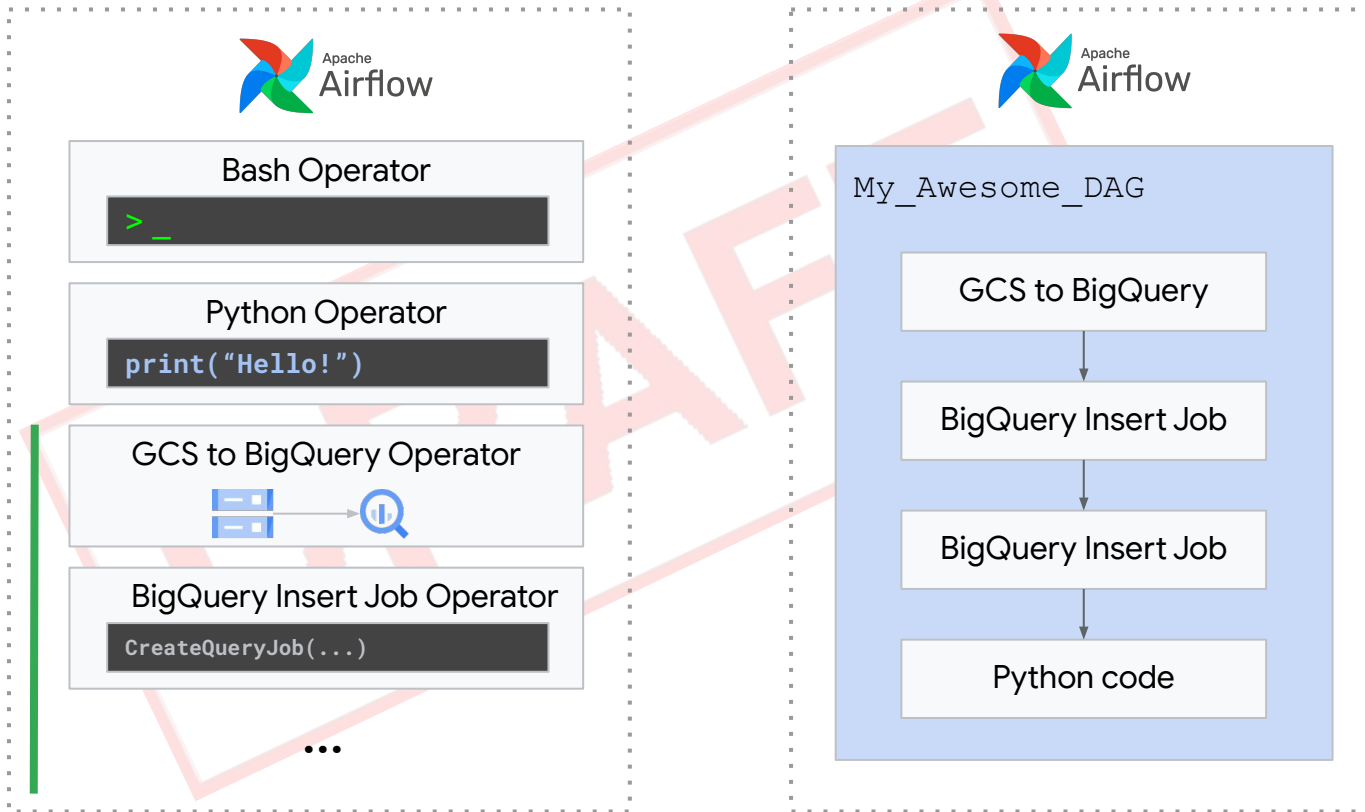
... from > 50 different providers like Amazon AWS™, Qubole™, Google Cloud Platform™, etc.

Google Operators/Hooks/Sensors...

- Operators: 304 (~65% of all operators)
- Sensors: 22 (> 25% of all sensors)
- Hooks: 50 (> 25% of all hooks)

The power of Airflow Operators

Airflow Provider
Packages
e.g. Google Cloud



Custom Operators

1. Code

```
from airflow.models.baseoperator import BaseOperator

class HelloOperator(BaseOperator):

    def __init__(
        self,
        name: str,
        **kwargs) -> None:
        super().__init__(**kwargs)
        self.name = name

    def execute(self, context):
        message = "Hello {}".format(self.name)
        print(message)
        return message
```

2. Upload to Airflow

Save
hello_operator.py
in
/plugins/
folder

3. Import in your DAG

```
from hello_operator import HelloOperator
```

What if I need special OS-level binaries?

1. Turn into a container

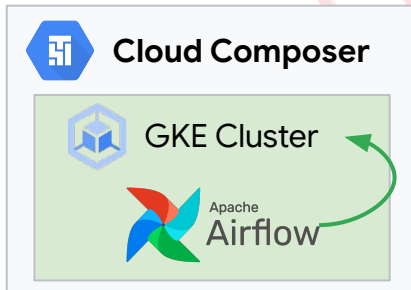
```
1 FROM python:3.8-slim
2
3 ENV APP_HOME /app
4 WORKDIR $APP_HOME
5
6 RUN apt -y update
7 RUN apt -y install gcc
8 ...
```

2. Build & push to a container repository (e.g. Google Artifact Registry)



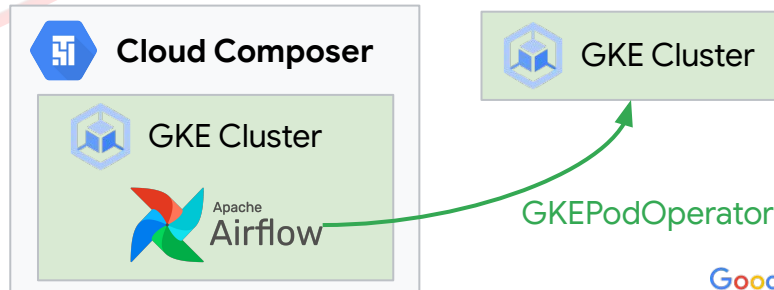
[https://\[region\]-docker.pkg.dev/\[project\]/\[image\]](https://[region]-docker.pkg.dev/[project]/[image])

3a. Use KubernetesPodOperator



OR

3b. Use GKEPodOperator



Example Airflow Use Cases

Information about other services used with Cloud Composer (rafal)

BigQuery,

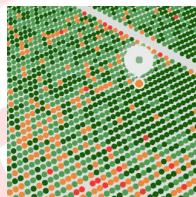
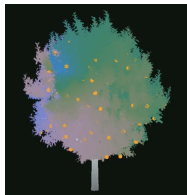
Dataflow, Datafusion, CloudSQL,

// Graphically

DRAFT

SreeTree Use case

Real Use Case: Take pictures of trees and send recommendations to the farmers



Example Composer DAG



Collecting the data

Processing and moving
the data

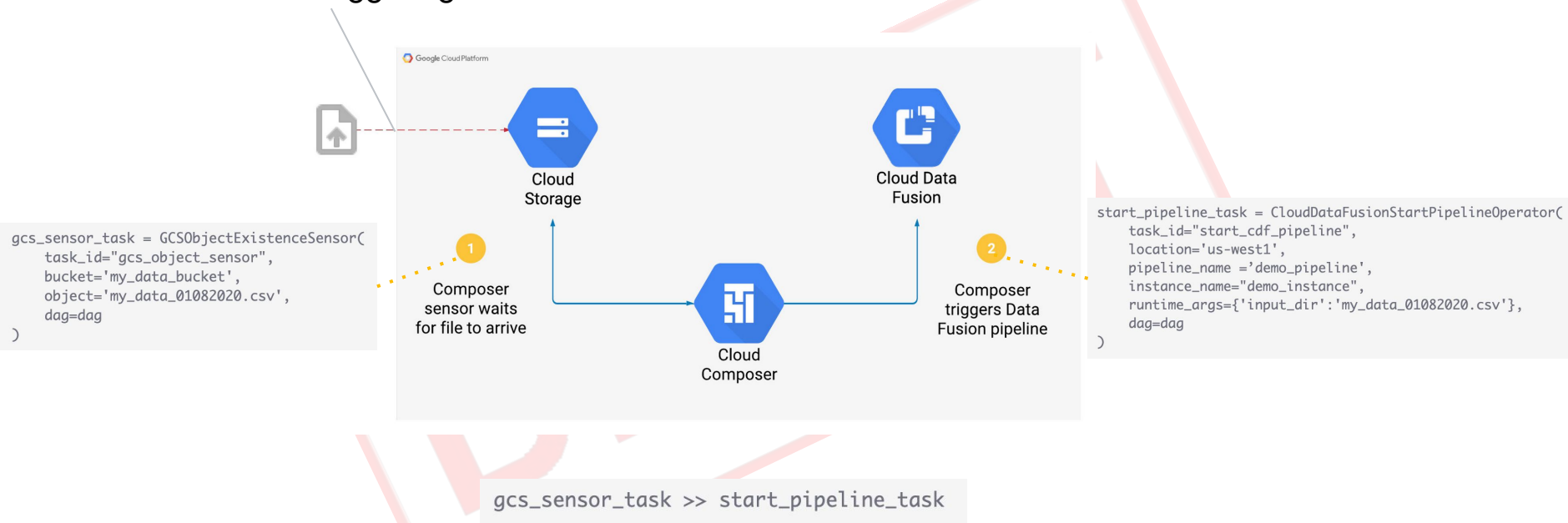
Choosing and applying the
right AI model

Write the
outcome back



Automating runs of Data Fusion pipelines

Event-based triggering



Loading & enriching data from a transactional system (Filip)

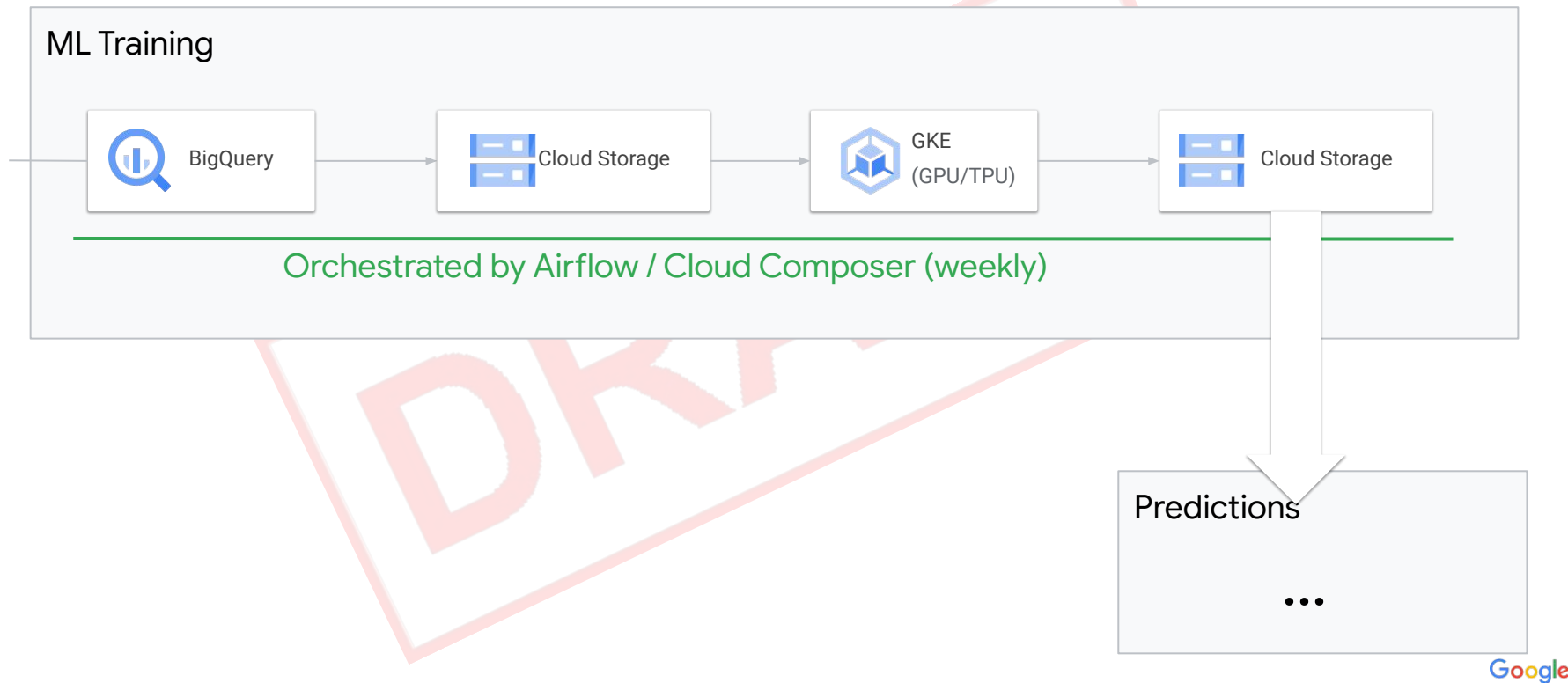
Retailer triggering Airflow DAG to:

1. Load data from Salesforce to BigQuery at a scheduled interval
2. Process the data in BigQuery using SQL queries (ELT)
3. Generate data marts that users access using BI tools



Change to CRM : Salesforce

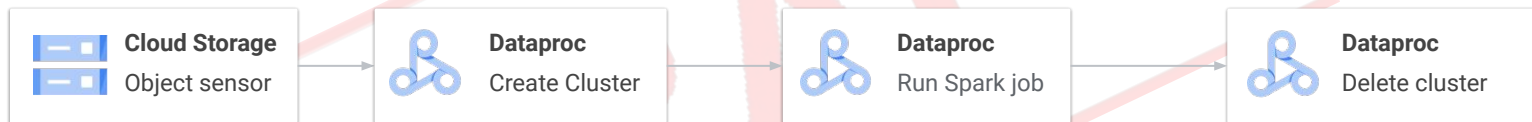
Machine Learning training



Big Data Spark jobs in ephemeral clusters

Large operator of a marketplace service:

1. Trigger a DAG when the file arrives
2. Create a Dataproc cluster
3. Run a job and push its result to GCS
4. Delete a Dataproc cluster (ephemeral!)



Orchestrated by Airflow / Cloud Composer

CloudML Use Case

CONFIDENTIAL & PROPRIETARY

DRAFT

<format of all slides should be equal>

DRAFT

Summary



Why Airflow?

Large operator of a marketplace service:

1. Rich integrations
2. Extensibility with own operators
3. Ability to schedule custom non-python tasks
4. Stitching services with Airflow provides
 - a. Observability
 - b. Easier troubleshooting
 - c. Simpler change management
 - d. Integrated security

DRAFT



Thank You