Journey to Cloud Data Engineering and Science

Airflow, Tensorflow, and GCP

Jeremy Gilmore
13 November 2018

zendesk

Agenda

BigDataMadison Meetup, 13 Nov, 2018

1. Introductions

2. Building the Enterprise Data Analytics Platform

3. Using the Enterprise Data Analytics Platform

4. Model Development, Deployment, and Enablement

5. Questions

Introduction

Jeremy Gilmore

Formal Education
- BA Economics
- MA Economics

Cloud Certifications
- AWS (2)
- GCP

Language Proficiencies
- Python
- R
- SQL
- Some JS
- Anything with contributions in Stack Overflow

Driving Ideology
“If we can reduce information costs we can make better decisions.”

Introduction

About Zendesk

We are the Infrastructure Behind Customer Support

Zendesk's family of products help organizations understand their customers, improve communication, and offer support when and where it's needed most.

support | chat | guide | talk | message | connect | explore

Driving Ideology
"Keeping it beautifully simple"

Zendesk Footprint
2000+ Employees
14 Global Locations
in Madison ~300

Our Problem

Disparate Data Systems

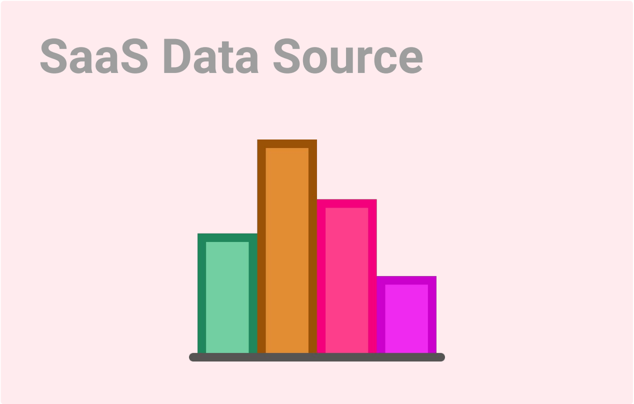
- Product backend not suitable for analytics
- Product Data Warehouse had outlived it's design
- Finance Data Warehouse was fragile
- 3rd party vendors' data isolated
- Data silos

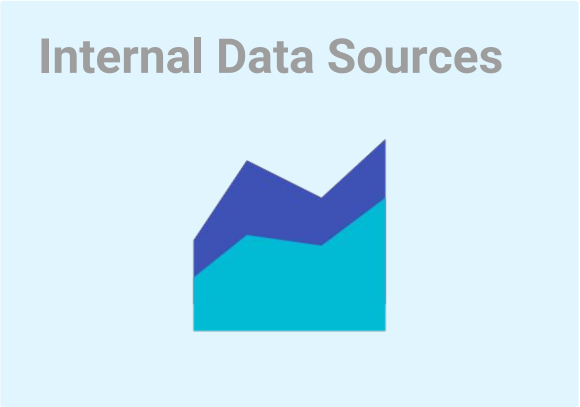
Which led to

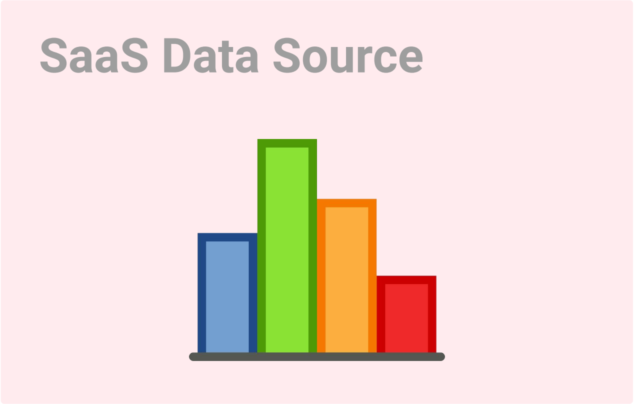
- Analyst silos
- Inconsistent analytics reporting
- Difficulty to evaluate across data sources and systems

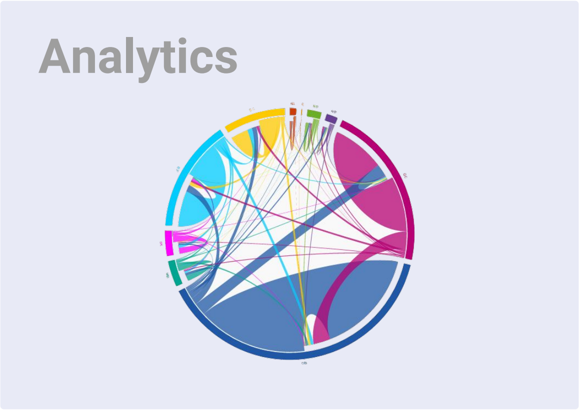
Resulting in these requirements

- ETL tool for multiple sources and data types
- Scalable for our growing business
- Scalable for our growing analyst community









Journey

Selecting the Analytics Database
- MySql/Postgres - No
- Redshift - No
- BigQuery - Yes

ETL - Moving data
- Data standards
- Manage changing schemas
- Standardize datetime (UTC)

Iterative design
- Repeatable builds
- Security focused
- Decisions were not made in isolation

About Airflow

my_dag.py

```
from operator_a import operator_aa
from operator_b import operator_bb

params_a = {...}
task_1 = operator_aa (
      params = params_a
      ...
)

params_b = {...}
task_2 = operator_bb (
      params = params_b
      ...
)

task_1.set_downstream(task_2)
```

operator_a.py

```
from hooks.bigquery_hook import bigquery_hook

class operator_aa (base_class):
      ...
```

operator_b.py

```
from hooks.gcs_hook import gcs_hook

class operator_bb (base_class):
      ...
```

bigquery_hook.py

```
from pandas_gbq_hook.gbq import GbqConnector

class bigquery_hook (base_class):
      ...
```

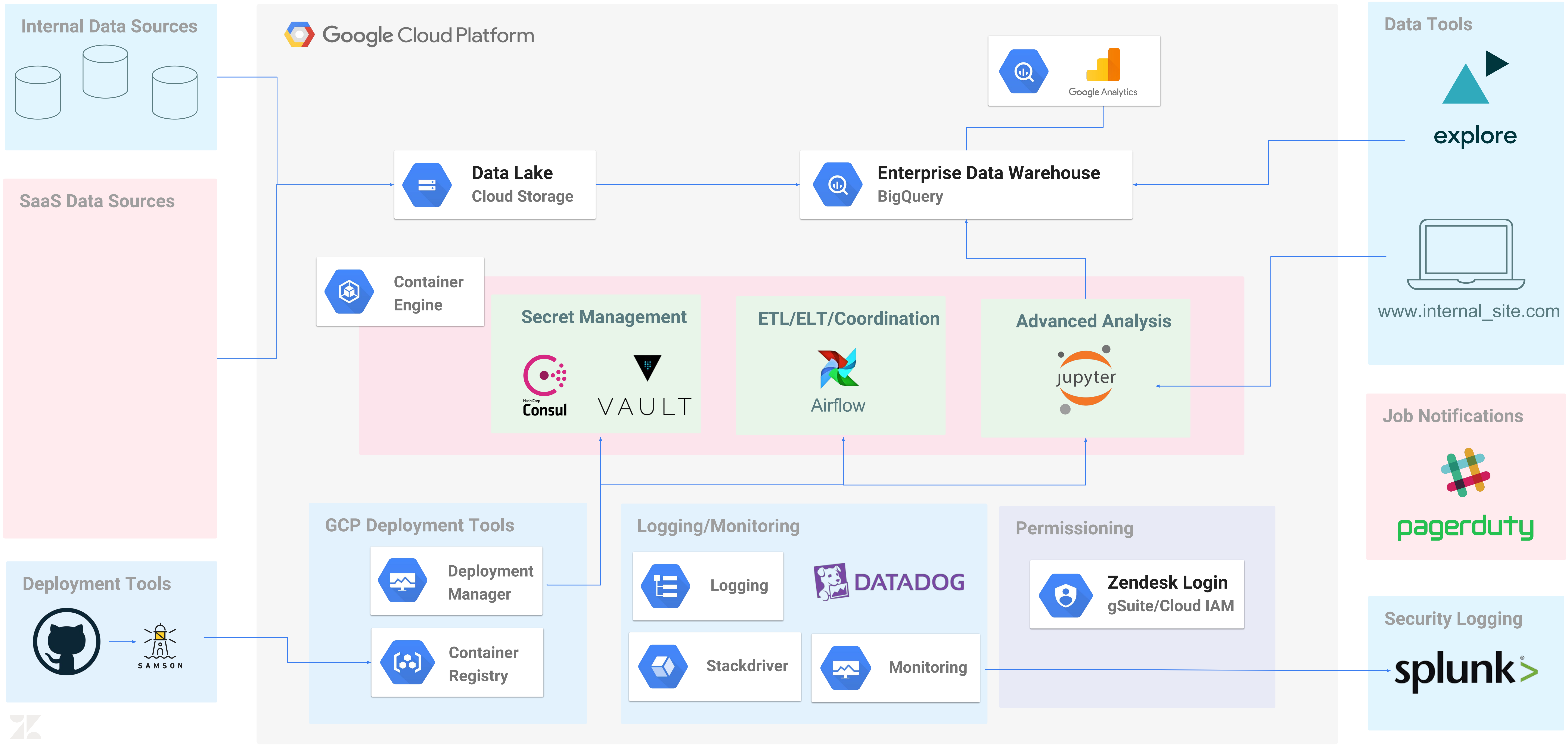
gcs_hook.py

```
from hooks.gcp_api_base_hook import GoogleCloudBaseHook

class gcs_hook (GoogleCloudBaseHook):
      ...
```

Airflow Infrastructure

As of Q3 2018



What we learned during the build

Scaling
- Count of DAGs increases load on Airflow Scheduler
- Count of Airflow tasks increases overall run time
- Custom Kubernetes executor allows for horizontal scaling with dedicated pods per task
- Managing metadata
    - Use the db for Airflow or create your own metadata (we use Datadog)
- Airflow concurrency opportunities
- API limits - Understand API constraints
    - GCP BigQuery has some soft/hard limits

Alerting
- Three types:
    - Processing - successful job completion
    - Quantitative - expected volume
    - Qualitative - expected values
- Dynamic error handling
    - Don’t just alert, do something
- Independent processes

Make Templates
- DAG Template File
    - Define function that generates DAG
    - Import DAG list with DAG parameters (JSON/csv)
        - JSON config for each DAG
        - Add your own metadata
- Changes made to all DAGs made in one place
- Create common functions library

Dynamic DAG Template
- JSON for DAGs to include
- JSON config for each DAG
    - Dynamically generates tasks
    - Dependencies and DAG structure unique to each

Scaling the Platform

Scaling Across Multiple GCP Projects

- 500+ DAGs and counting
- API constraints mitigated with multiple projects
- Infrastructure as code can be deployed *easily*
- Cross project queries possible
- Allows for scalable security context definition
  - Data level sensitivities
  - Leave clear segregation
  - Less overhead to manage users

Invest in Data Curation

- Common transformations materialized as views
- Join across data sources
- Consistency in reporting/metrics across teams

Develop Data Science Models

- Deploy multiple models using various data sources
- Challenge or validate our existing hypotheses

Leverage Jupyterhub for More Use Cases

- Deploying notebooks as code
- Team collaboration/sharing

Develop Self-service Curation App

- Using dynamic DAG template
- Simple UI that allows non-technical users to create and manage their own DAGs and schedules
  - Considering open source

Documentation

- Data Provenance
- KPI Glossary
- Change Management

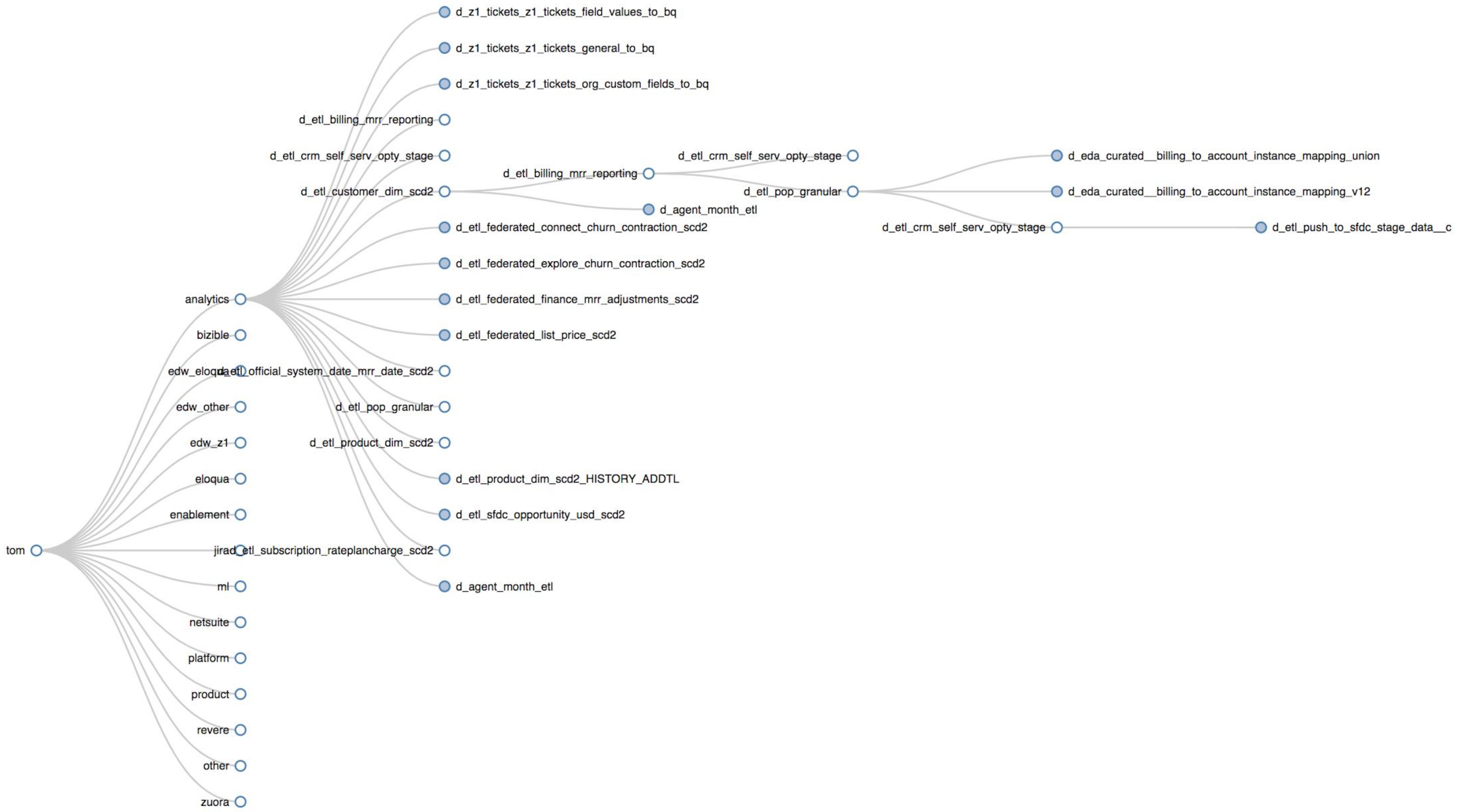
Platform Analytics

- Metadata stats
- User stats

Demo

Brought to you by:

Airflow DAG Dependencies



Developing Models

Business Context is Everything

- Understand your product/service
- Understand how your product is represented in data
    - Talk with engineers
    - Proper instrumentation is critical
- Understand limitations of data
    - Know what questions can and cannot be answered
- Throwing data at the problem may not solve the problem

Technical Details Matter

- Build standards that transcend any single model
- Use standard naming conventions *others* can read
- Use _ and __ not camelCase
- Parameterize datetime for file names, folders/buckets
- Know how your db rounds floats
- Use specific package and library versions
- Simplicity is better than complexity

Don't Take Shortcuts

- Good science is good science
- Know what the algorithm is doing
- Be transparent and create repeatable results
- Documentation
- Know when to quit

Manage Resources Effectively

- Re-use namespace
- Build as much transformation into SQL as possible
- Build for failure breakpoints
- Do testing on different machines
- Evaluate cost ($ and performance) of solution options

Pushing Models to Production

Model Flow in Airflow

Step 1 - Parameterized BigQuery query

Step 2 - Data transformations in Python

Step 3 - Save output to Google Cloud Storage

Step 4 - Call ML Engine
- Stored model input data
- Stored TensorFlow model with version

Step 5 - Retrieve output data
- Check for errors
- Add metadata
- Join results with some attributes of inputs

Step 6 - Send results to BigQuery

Step 7 - Send ML output to trigger events

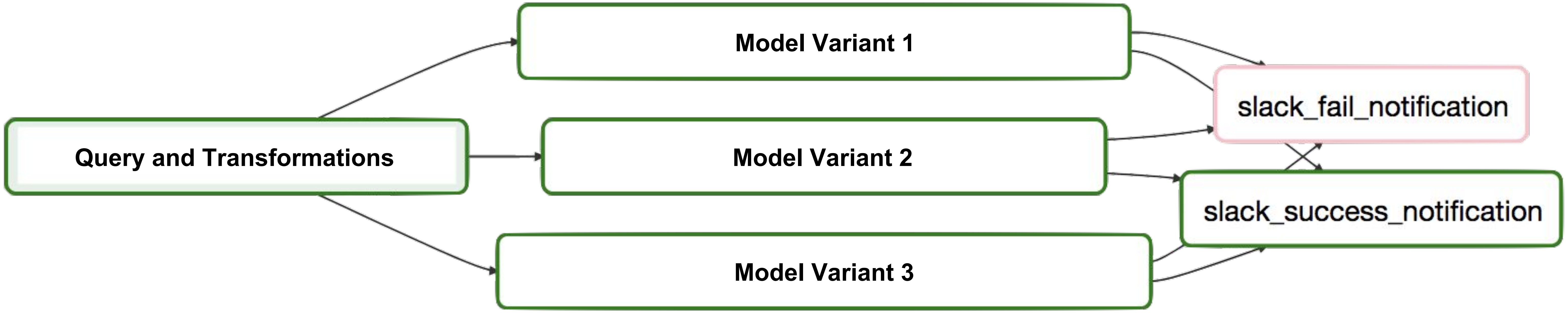
Step 8 - Send output to other SaaS tools

A machine learning model not in production is a point in time analysis regardless of the algorithm used.

‘production’ = automated

Model in Production (using Airflow)

Example



Production and Enablement

Models in Production

The Life of a Model

- Models in production are living things
- Model drift? develop thresholds
- Using the model output
- Develop experiments
- Integrate with third party systems
- Reporting efficacy
- Additional inputs - did behavior change

Testing which leads to action

- Alerts
    - Threshold/Variance
    - Record count
    - Detect drift
- Automated queries/triggers

Model Refresh Process

- Automatic triggers that warrant re-train
- Cyclical refresh as product/data changes

Enablement

Delivered with every release

- Documentation
- Integration
- Training
- Support

Partnership with other teams

- Collaboration should be a part of model development
- Teams need to understand model output

Pushing to other systems

- Existence of a score does not add value unless available
- Automation is key
- Common naming conventions
- Similar model output format for simplicity/interpretability

Training Business Users

- Know your audience
- Transparency

We Are Hiring

Data Scientist
You get to learn and use
- Airflow
- BigQuery
- Jupyterhub
- Tensorflow
- ML Engine

You bring to the team
- Advanced SQL and analytics skills
- Experience in business domains
- Enthusiasm and curiosity

Enterprise Data Openings
https://www.zendesk.com/jobs/
Search for Data & Analytics

Madison
https://www.zendesk.com/jobs/madison/

Find Emily Souder!
esouder@zendesk.com

zendesk