DATA+AI SUMMIT 2022

Okta's FIG Automation Library

Simplify Global DataOps and MLOps

Gregory Fee
Principal Architect, Okta

ORGANIZED BY databricks

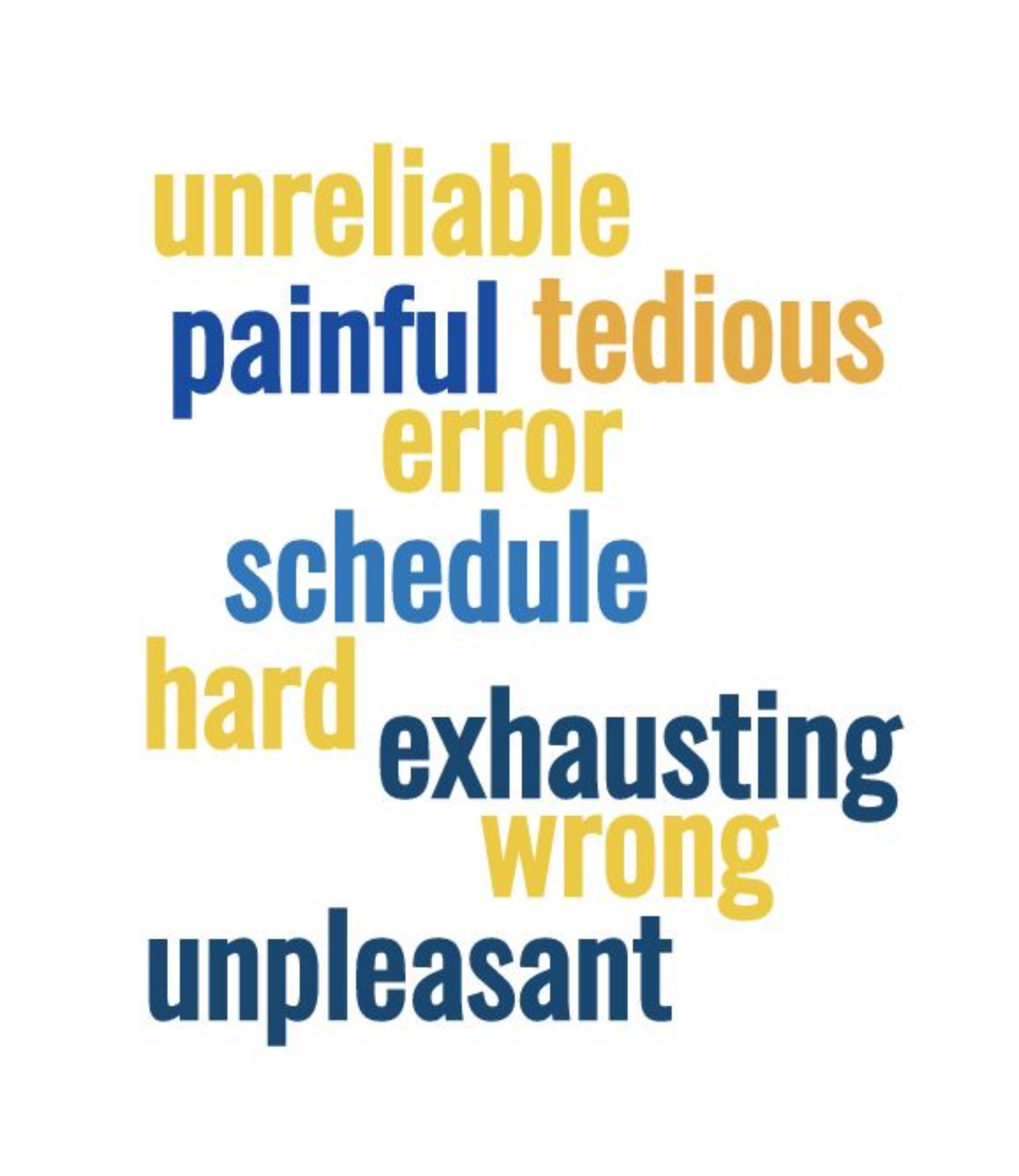
Agenda

- About Me
- The Typical Problems and Solutions I Have Seen
- The Approach at Okta using FIG

About Me

- Current Role: Principal Architect, Data Science @ Okta
- Previous Roles
  - Technical Lead, Data and ML @ Lyft
  - Lead Architect, Apex @ Salesforce
  - Various Security and Developer Tools Roles @ Microsoft
- Personal
  - Love hiking, warm weather, walks on the beach, and home improvements
  - Amateur Mentalist!

Remember what it was like to build your ML pipeline


unreliable
painful
tedious
error
schedule
hard
exhausting
wrong
unpleasant

Great Tools But Something is Missing

- Quality of data and ML Tools is increasing rapidly
- Multitude of Commercial and Open Source offerings

But....

- Creating Data + ML Pipelines is still not a great experience

Required Technical Expertise

Bigger Companies

- Data Engineers
- ML Engineers
- MLOps
- DataOps
- DevOps
- Data Scientist
- Data Analyst

....and they still struggle

Smaller Companies

- Struggle even more with a fraction of the people and gaps in skill sets

ML Processes

What Do We Want?

- Data Pipelines
- Data Visualization
- Rapid Prototyping Environment
- Data Preparation
- ML Training
- ML Bulk Scoring
- ML Scoring Service
- ML Monitoring

How Are We Building It?

- Orchestration
- SQL/Spark
- Python/R
- Terraform
- So Much Glue Code

When I Started At Okta

Basic Infrastructure

- Data Warehouse in Snowflake with No ETL
- Sagemaker Notebooks
- Unsupervised streaming model in production

Small Team

- 1 DevOps
- 3 Data Scientists
- Me

Okta – Environment

Many Accounts

- 15+ Production Accounts
  - Services customer traffic
  - Multiple DBs + high volume events
- 4 Analytics Accounts
  - Collect data from production accounts
  - Support analytics/science/ML
    - Prototyping
    - Data Prep
    - Training/Batch Scoring

Requirements

- Support many environments without constant tweaking
- Support Data Scientists without much Data/ML Engineer/Ops
- Seamless transition from prototyping to production
- Reduce need to scale Data/ML/MLOps/Etc Engineers linearly with Data Scientists
- Reduce cost of porting to new technologies
    - Snowflake -> Spark
    - Sagemaker -> Pytorch
    - AWS Step Functions -> Flyte

Observations

- SQL mixes business logic and structural logic in a way that makes refactoring challenging
    - dbt is one attempt to fix this problem
- Glue code between systems is error-prone
- Multi-stage data + ML pipelines are difficult and time consuming to verify
- If the work is tedious then it can probably be automated

Use Case: Large Scale Threat Detection

Goal

- Identify large scale credential-based and other abuse style attacks
- Label all incoming traffic as malicious/legitimate based on this knowledge
- Create ML decision engine to identify and block malicious requests in real-time

Approach

- Create a shared set of request level features
- Create a set of weak labeling functions that each identify known malicious request types
- Combine labels to create a ground truth data set
- Create supervised ML models based on ground truth

Introducing FIG (Feature Infrastructure Generator)

What Do You Do?

- Specify your data transformation needs in a SQL-like configuration language
- Specify your model with Python just like you've been doing

What Do You Get?

- Auto-generated SQL for ad hoc queries
- Auto-generated data workflows
    - Daily ETL
    - Multi-day backfill
- Auto-generated Training/Scoring ML workflows
- Auto-generated full ML pipeline
    - Generate/transform data and (re)train models on a set schedule

ML Pipeline - Simplified

High Level Pipeline Steps

- Target two entities
  - IP address
  - Autonomous System Number (ASN)
- Aggregate data over 1 week
- Join entity aggregation data to every request
- Apply weak label functions and combine into ground truth label
- Train supervised ML model

Environment

- Requests generate an event with IP and ASN plus other data
- Events are stored in Snowflake
- Sagemaker used for ML training

FIG Configuration Language

Similarities to SQL

- Tabular data
- Row level data transforms
- Group-by and aggregation transforms

Differences from SQL

- Structured table types
  - Events, Features, Tables
- Temporal constructs
- ML Algorithm integration
- Data Quality Checks

FIG – Import Event Definition

- Requests are captured in an event named ‘user.session.start’
- Non-null data quality check on org_id
- Support for row-level transformations like ARRAY_SIZE
- Fields are unnested from a complex payload


```
{
    "name": "user_session_start",
    "event_name": "user.session.start",
    "fields": {
        "org_id": {
            "json": "org_id::string",
            "checks": "nonnull"
        },
        "actor_id": "actor:id::string",
        "actor_type": "actor:type",
        "ip_address": "client:ip_address::string",
        "ip_chain": "client:ip_chain",
        "ip_chain_len": {
            "function": "@ARRAY_SIZE(@PARSE_JSON($ip_chain))"
        },
        "ip_address_originating": "client:ip_chain[0]:ip::str
        "ip_address_connection": "client:ip_chain[0]:ip::str
        "raw_user_agent": "client:user_agent:raw_user_agent::
        "as_org": "security_context:as_org",
        "as_number": "security_context:as_number",
        "anonymizer_status": "metadata:ip_metadata:value:anon
        "hosting_facility": "metadata:ip_metadata:value:hosti
        "ip_routing_type": "metadata:ip_metadata:value:ipRout
        "org_type": "metadata:ip_metadata:value:org_type::str
        "residence": "metadata:ip_metadata:value:residence::b
        "proxy_level": "metadata:ip_metadata:value:proxy_leve
        "proxy_type": "metadata:ip_metadata:value:proxy_type:
        "device": "client:device",
        "browser": "client:user_agent:browser::string",
        "os": "client:user_agent:os::string",
```

FIG – Aggregate Features

- Aggregate a week of events
- Group by the autonomous system number
- Apply aggregation functions
  - Count distinct
  - Count
  - Sum
- Apply simple arithmetic functions
  - Divide

IP features is similar, but group by ip_address

```
{
    "name": "asn",
    "granularity": "autonomous_system_number",
    "granularity_type": "bigint",
    "period": [
        "weekly_by_day"
    ],
    "groups": [
        {
            "source": "user_session_start",
            "granularity_field": "as_number",
            "features": {
                "ip_address_count": {
                    "function": "@COUNT_DISTINCT($ip_address)",
                    "type": "bigint"
                },
                "login_count": {
                    "function": "@COUNT(*)",
                    "type": "bigint"
                },
                "failure_count": {
                    "function": "@SUM($is_failure)",
                    "type": "bigint"
                },
                "failure_rate": {
                    "function": "@DIVIDE($failure_count, $login_count)",
                    "type": "float"
                },
                "threat_suspected_count": {
                    "function": "@SUM($is_threat_suspected)",
                    "type": "bigint"
                },
                "unknown_actor_count": {
                    "function": "@SUM($is_unknown_actor)",
                    "type": "bigint"
                },
                "unknown_raw_user_agent_count": {
```

FIG – Request Features

- Use fields from event
- Join all ASN and IP features

Every request now includes the aggregated data from the preceding week automatically

```
{
    "name": "request",
    "granularity": "external_session_id",
    "granularity_type": "string",
    "groups": [
        {
            "source": "user_session_start",
            "granularity_field": "external_session_id",
            "features": {
                "timestamp": { "function": "$start_time", "type": "da
                "org_id": { "function": "$org_id", "type": "string" }
                "ip_address": { "function": "$ip_address", "type": "s
                "as_number": { "function": "$as_number", "type": "int
            }
        },
        {
            "source": "asn",
            "granularity_match": {
                "field": "as_number",
                "period": "weekly_by_day"
            },
            "features": "*"
        },
        {
            "source": "ip",
            "granularity_match": {
                "field": "client_ip",
                "period": "weekly_by_day"
            },
            "features": "*"
```

FIG – Labels

- Each request is labeled by a series of weak labeling function
  - Labeling can be performed by arbitrary boolean expressions, includes ML classifiers
- Weak labels are combined to form a strong label

```
{
    "name": "request_labels",
    "source": "request",
    "labels": [
        {
            "name": "ip_high_failure_rate",
            "type": "string",
            "description": "High failure rate for the IP address used f
            "cases": {
                "malicious": "@AND(@GREATERTHAN($ip__failure_rate, 0.9)
                "benign": "@AND(@LESSTHAN($ip__failure_rate, 0.5), @GRE
            }
        },
        {
            "name": "ip_high_missing_device_token_rate",
            "type": "string",
            "description": "High missing device token rate for the IP a
            "cases": {
                "malicious": "@AND(@GREATERTHAN($ip__other_device_token_
                "benign": "@AND(@LESSTHAN($ip__other_device_token_type_
            }
        },
        {
            "name": "plurality_vote",
            "type": "string",
            "description": "Most common label",
            "function": "@PLURALITY_VOTE(@ARRAY_CONSTRUCT($ip_high_fail
        }
```

FIG – Model

- Use features from request and the strong label to generate a supervised model
- Uses Sagemaker "bring your own model" to support any ML algorithm


```json
{
    "name": "request_threat_scorer",
    "location": "request_threat_scorer",
    "training_delay_delta_days": 3,
    "training_time_delta_days": 1,
    "input_features": [
        {
            "family_name": "request",
            "feature_names": {
                "session_definition": [
                    "ip_address",
                    "ip_chain_length",
                    "country",
                    "latitude",
                    "longitude",
                    "raw_user_agent",
                    "http_header_accept",
                    "http_header_accept_encoding",
                    "http_header_accept_language",
                    "browser",
                    "os",
                    "device",
                    "device_token_type",
                    "ssws_present_and_valid",
                    "outcome_result"
                ]
            }
        }
    ],
    "truth_label": {
        "source": "request_labels",
        "name": "plurality_vote",
    },
    "output_features": [
        "request_threat_score"
    ]
}
```

FIG – Usage

Sagemaker Notebook

- pip install feature_generator
- feature_generator.execute_workflow()
- Validation to detect errors before anything runs
- If it validates then it is guaranteed to run without errors

AWS Stats for full Okta Version

- 450+ features
- 6 models
- 45 Step Function Workflows
- 739 Total Step Function steps
- 883 Snowflake SQL queries
- 353 Lambdas
- Glue code to transfer data from Snowflake to S3 for Sagemaker
- 6 Containers for model code

Evolution

Add monthly aggregation period

Traditional Approach

- Cut'n'paste existing queries
- Edit the queries to target different the new period
- Add another join to main table
- Change unit tests to try to validate
- Execute and discover type-o
- Execute and discover that a month of data at once is too much data
- Try to refactor....

Evolution

Add monthly aggregation period

FIG

- Add new period to aggregation
- Refer to that point in feature family
- Run validation
- Execute and done

Generated SQL uses incremental aggregation to avoid inefficient execution on longer time periods

```
{
    "name": "asn",
    "granularity": "autonomous_system_number",
    "granularity_type": "bigint",
    "period": [
        "weekly_by_day",
        "monthly_by_day"
    ],
```

```
    {
        "source": "asn",
        "granularity_match": {
            "field": "as_number",
            "period": "weekly_by_day"
        },
        "features": "*"
    },
    {
        "source": "asn",
        "granularity_match": {
            "field": "as_number",
            "period": "monthly_by_day"
        },
        "features": "*"
    },
```

Key Takeaways

- Workflows and SQL
  - Great building blocks for creating ML pipelines
  - Clumsy and error-prone metaphors for specifying the ML pipelines
- Increase productivity by specifying in a higher level language
  - Tabular metaphor works well for scenarios where SQL is the traditional fit
  - Capture actions from different systems to allow generation of glue code
  - Capture error-prone areas that are likely to change often
  - Write validation rules to catch most frequent errors

FIG Future

Short-Term

- Support for Spark SQL as a target to reduce processing costs
- Scoped backfills
  - Backfill that only computes a newly added column or changed column definition
- Production Integration
  - Automatically validate and deploy trained models
  - Automatically upload generated features to Feature Service

Long-Term

- Real-time feature generation
- ML experiment and artifact tracking
- Bootstrapping FIG config from SQL and Pandas
- Open Source!

DATA+AI SUMMIT 2022

Thank you

Gregory Fee

Principal Architect, Data Science @ Okta