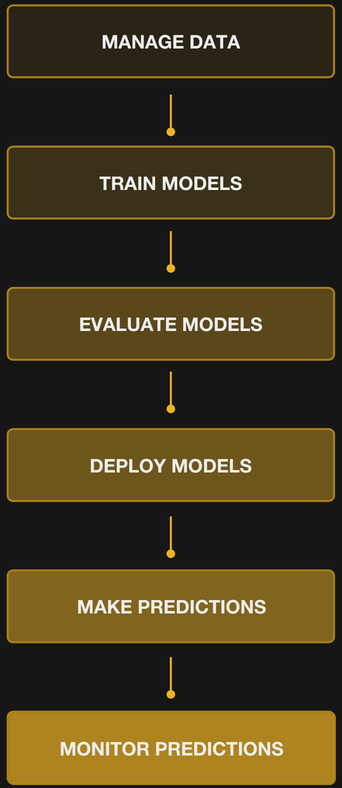
Michelangelo @ Uber

Enable engineers and data scientists across the company to easily build and deploy machine learning solutions at scale.

ML-as-a-service

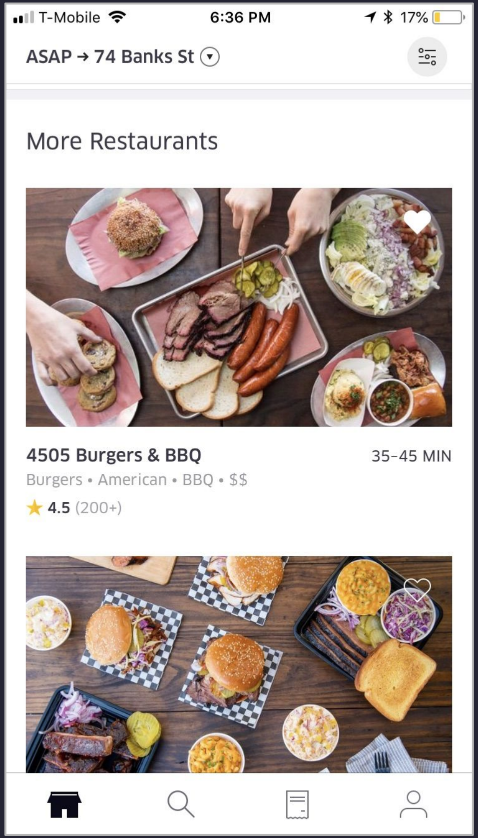
- Managing Data/Features
- Tools for managing, end-to-end, heterogenous training workflows
- Batch, online & mobile serving
- Feature and Model drift monitoring

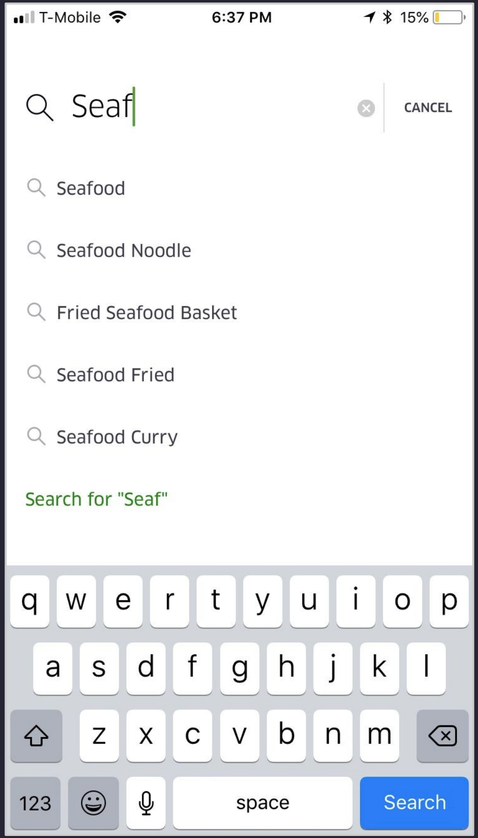


Uber

Feature Engineering @ Uber

- Example: ETA for EATS order
- Key ML *features*
  - How large is the order?
  - How busy is the restaurant?
  - How quick is the restaurant?
  - How busy is the traffic?





Uber DATA

Managing Features

One of the hardest problems in ML

- Finding good Features & labels
- Data in production: reliability, scale, low latency
- Data parity: training/serving skew
- Real-time features: traditional tools don't work

Uber

Palette Feature Store

Uber-specific *curated* and *crowd-sourced* feature database that is easy to use with machine learning projects.

One stop shop

- Search for features in single catalog/spec: *rider, driver, restaurant, trip, eaters, etc.*
- Define new features + create production pipelines from spec
- Share features across Uber: cut redundancy, use consistent data
- Enable tooling: Data Drift Detection, Auto Feature Selection, etc.

Uber

Feature Store Organization

Organized as <entity>:<feature-group>:<feature-name>:<join-key>

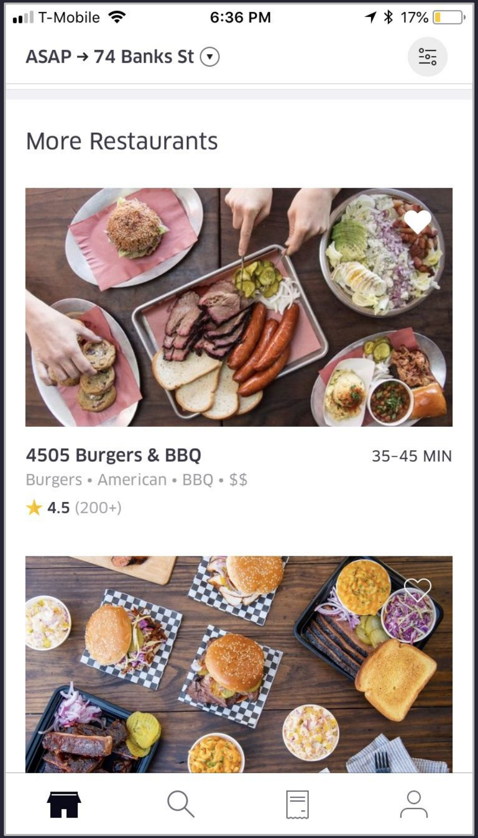
Eg. @palette:restaurant:realtime_group:orders_last_30min:restaurant_uuid

Backed by a dual datastore system: similarities to lambda

- Offline
  - Offline (Hive based) store for bulk access of features
  - Bulk retrieval of features across time
- Online
  - KV store (Cassandra) for serving latest known value
  - Supports lookup/join of latest feature values in real time
- Data synced between online & offline
  - Key to avoiding training/serving skew

EATS Features revisited

- How large is the order? ← Input
- How busy is the restaurant?
- How quick is the restaurant?
- How busy is the traffic?



Uber DATA

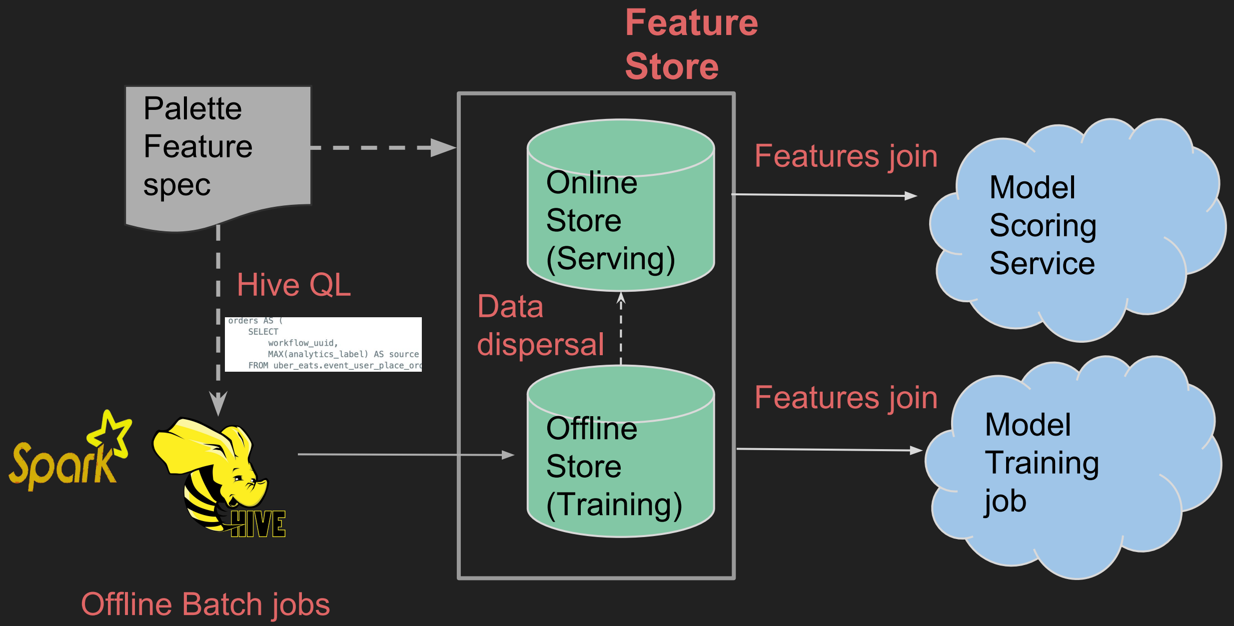
Creating Batch Features

General trends, not sensitive to exact time of event

Ingested from Hive queries or Spark jobs

How quick is the restaurant ?

- Aggregate trends
- Use Hive QL from warehouse
- *@palette:restaurant:batch_aggr:* ***prepTime****:rld*



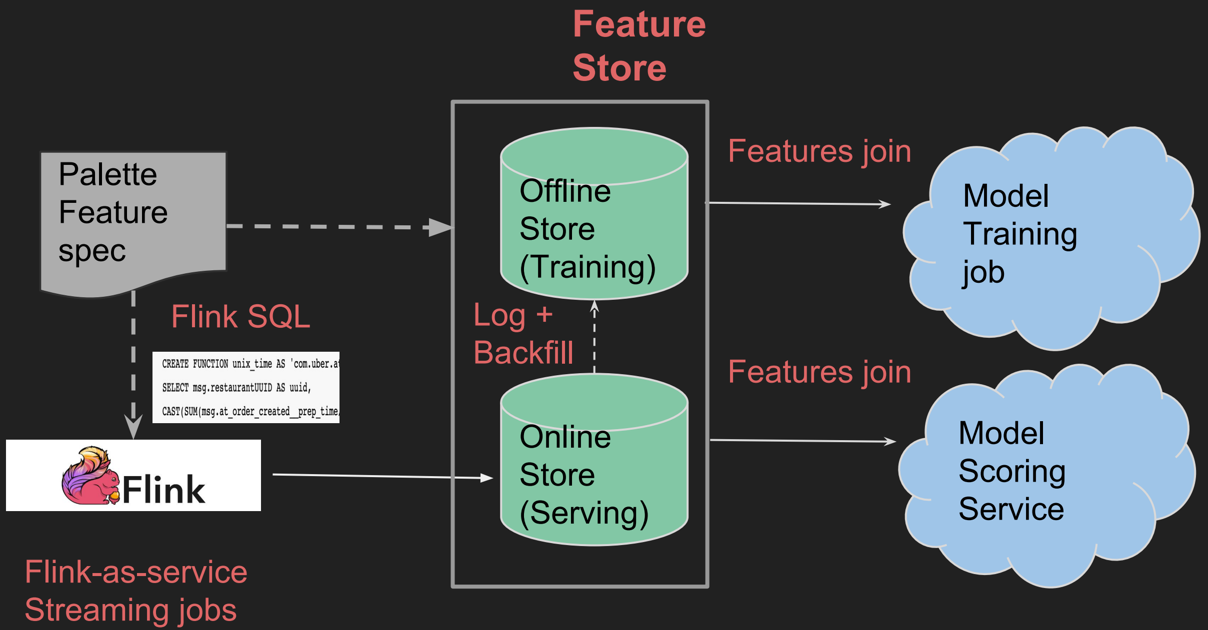
Creating Real-time Features

Features reflecting the latest state of the world

Ingest from streaming jobs

How busy is the restaurant ?

- kafka topic with events
- perform realtime aggregations
- *@palette:restaurant:rt_aggr:**nMeal**:rld*



Uber

Bring Your Own Features

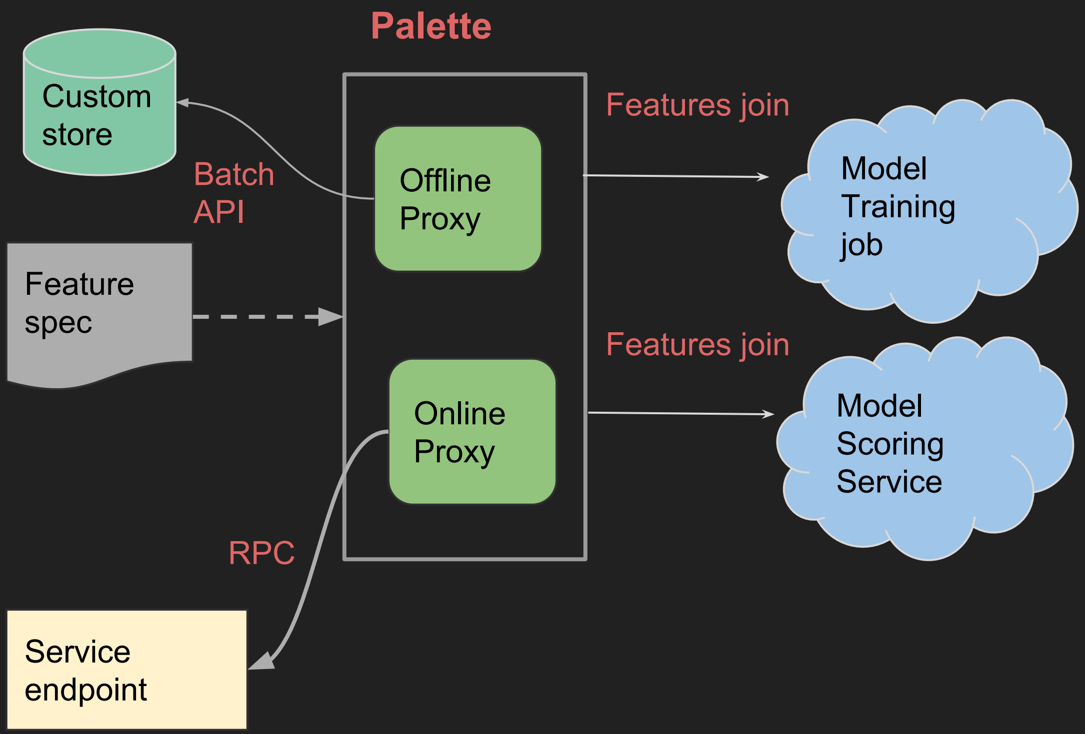
Feature maintained by customers

Mechanisms for hooking serving/training endpoints

Users maintain data parity

How busy is the region ?

- RPC: external traffic feed
- Log RPCs for training
- *@palette:region:traffic:**nBusy**:regionId*



Uber

Palette Feature Joins

Join *@basis* features with supplied *@palette* features into single feature vector

Join billion+ rows at points-in-time: dominates overhead

Join/Lookup 10s of tables at serving time at low latency

@basis features

order_id	nOrder	restaurant_uuid	latlong	Label ETA (training)	timestamp
1	4	uuid1	(10,20)	40m	t1
2	3	uuid2	(30,40)	35m	t2

@palette:restaurant:realtime_stats:nBusy:rId

@palette:region:stats:nBusy:regionId

Time + Key join

training/scoring feature vector

order_id	rId	latlong	Label ETA (training)	prepTime	..
1	uuid1	(10,20)	40m	20m	..
2	uuid2	(30,40)	35m	15m	..

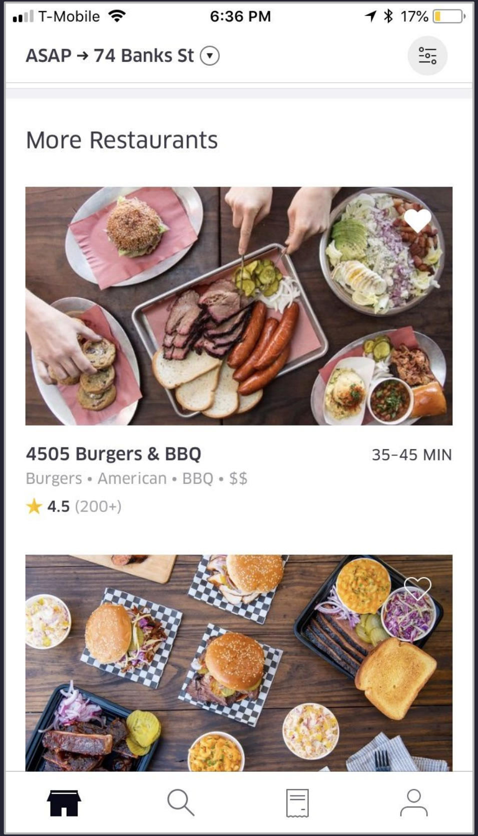
@palette:restaurant:agg_stats:prepTime:restaurant_uuid

rId	prepTime	timestamp
uuid1	20m	t1
uuid2	15m	t2

join_key = rId

Done with Feature Engineering ?

- Feature Store Features
  - *nOrder*: How large is the order? (*basis*)
  - *nMeal*: How busy is the restaurant? (*near real-time*)
  - *prepTime*: How quick is the restaurant? (*batch feature*)
  - *nBusy*: How busy is the traffic? (*external feature*)
- Ready to use ?
  - Model specific feature transformations
  - Chaining of features
- ***Feature Transformers***



Uber DATA

Feature Consumption

Feature Store Features

- *nOrder*: input feature
- *nMeal*: consume directly
- *prepTime*: needs transformation before use
- *nBusy*: input latlong but need regionId

Setting up consumption pipelines

- *nMeal: r_id -> nMeal*
- *prepTime: r_id -> prepTime -> featureImpute*
- *nBusy: r_id -> lat, log -> regionId(lat, log) -> nBusy*

In arbitrary order

Uber

Michelangelo Transformers

Transformer: Given a record defined a set of fields, add/modify/remove fields in the record

PipelineModel: A sequence of transformers

Spark ML: Transformer/PipelineModel on DataFrames

Michelangelo Transformers: extended transformer framework for both Apache Spark and Apache Spark-less environments

Estimator: Analyze the data and produce a transformer

Defining a Pipeline Model

Feature consumption

- Feature extraction: Palette feature retrieval expressed as a transform
- Feature mutation: Scala-like DSL for simple transforms
- Model-centric Feature Engineering: string indexer, one-hot encoder, threshold decision
- Result retrieval

Modeling

- Model inferencing (also Michelangelo Transformer)

Join Palette Features

Apply Feature Eng Rules

String Indexing

One-Hot Encoding

DL Inferencing

Result Retrieval

Michelangelo Transformers Example

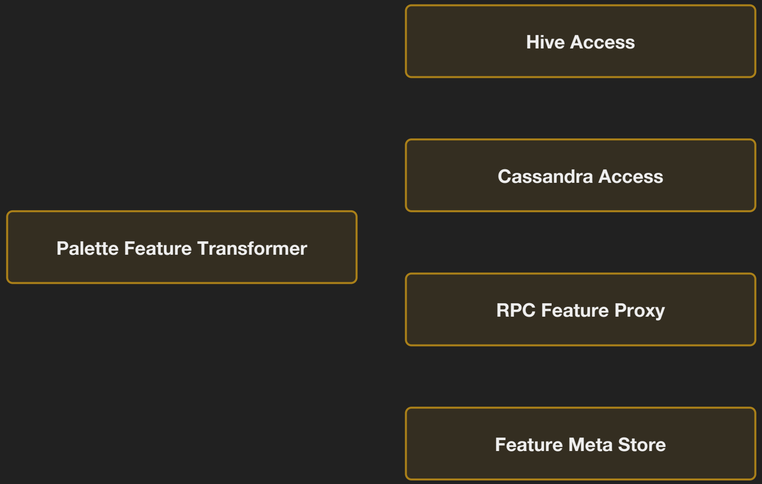
```scala
class MyEstimator(override val uid: String) extends Estimator[MyEstimator] with Params  with DefaultParamsWritable {

...

override def fit(dataset: Dataset[_]): MyModel = ...

}
```

```scala
class MyModel (override val uid: String) extends Model[MyModel] with MyModelParam with MLWritable with MATransformer {

 ...

 override def transform(dataset: Dataset[_]): DataFrame = ...

 override def scoreInstance(instance: util.Map[String, Object]): util.Map[String, Object] = ...

 }
```

Uber

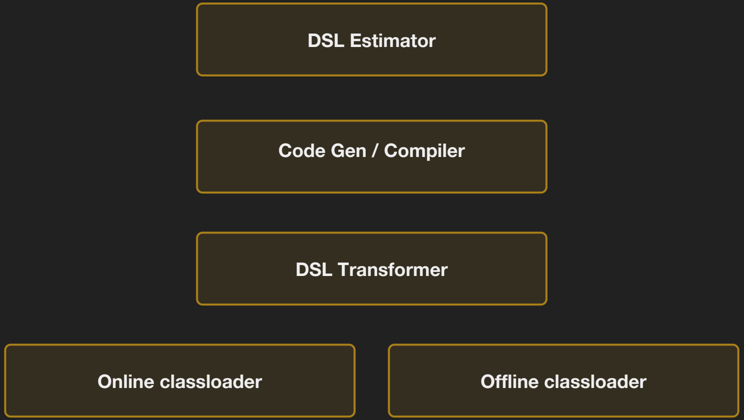
Palette retrieval as a Transformer

```
tx_p1 = PaletteTransformer([
  "@palette:restaurant:realtime_feature:nMeal:r_id",
  "@palette:restaurant:batch_feature:prepTime:r_id",
  "@palette:restaurant:property:lat:r_id",
  "@palette:restaurant:property:log:r_id"
])

tx_p2 = PaletteTransformer([
  "@palette:region:service_feature:nBusy:region_id"
])
```



DSL Estimator / Transformer

```
es_dsl1 = DSLEstimator(lambdas = [
  ["region_id", "regionId(@palette:restaurant:property:lat:r_id,
     @palette:restaurant:property:r_id"]
])

es_dsl2 = DSLEstimator(lambdas = [
  ["prepTime": nFill(nVal("@palette:restaurant:batch_feature:prepTime:r_id"),
                        avg("@palette:restaurant:batch_feature:prepTime:r_id")))"],
  ["nMeal": nVal("@palette:restaurant:realtime_feature:nMean:r_id")],
  ["nOrder": nVal("@basis:nOrder")],
  ["nBusy": nVal("@palette:region:service_feature:nBusy:region_id")]
])
```



Uber Eats Example Cont.

Computation order

- *nMeal: rId -> nMeal*
- *prepTime: rId -> prepTime -> featureImpute*
- *busyScale: rId -> lat, log -> regionId(lat, log) -> busyScale*

Palette Transformer
id -> nMeal
id -> prepTime
id -> lat, log

DSL Transformer
lag, log -> regionID

Palette Transformer
regionID -> nBusy

DSL Transformer
impute(nMeal)
impute(prepTime)

Dev Tools: Authoring and Debugging a Pipeline

Palette feature generation

- Apache Hive QL, Apache Flink SQL

Interactive authoring

- PySpark + iPython Jupyter notebook

Centralized model store

- Serialization / Deserialization (Spark ML, MLReadable/Writeable)
- Online and offline accessibility

```
basis_feature_sql = "..."
df = spark.sql(basis_feature_sql)

pipeline = Pipeline(stages=[tx_p1, es_dsl1, tx_p2, es_dsl2t, vec_asm, l_r)
pipeline_model = pipeline.fit(df)
scored_def = pipeline_model.transform(df)

model_id = MA_store.save_model(pipeline_model)

draft_id = MA_store.save_pipeline(basis_feature_sql, pipeline)
retrain_job = MA_API.train(draft_id, new_basis_feature_sql)
```

Uber

Takeaways

Feature Store: Batch, Realtime and External Features with online and offline parity

Offline scalability: Joins across billions of rows

Online serving latency: Parallel IO, fast storage with caching

Feature Transformers: Setup chains of transformations at training/serving time

Pipeline reliability and monitoring out-of-the-box