Bighead

Airbnb's End-to-End Machine Learning Infrastructure

Andrew Hoh
ML Infra @ Airbnb

Background

Design Goals

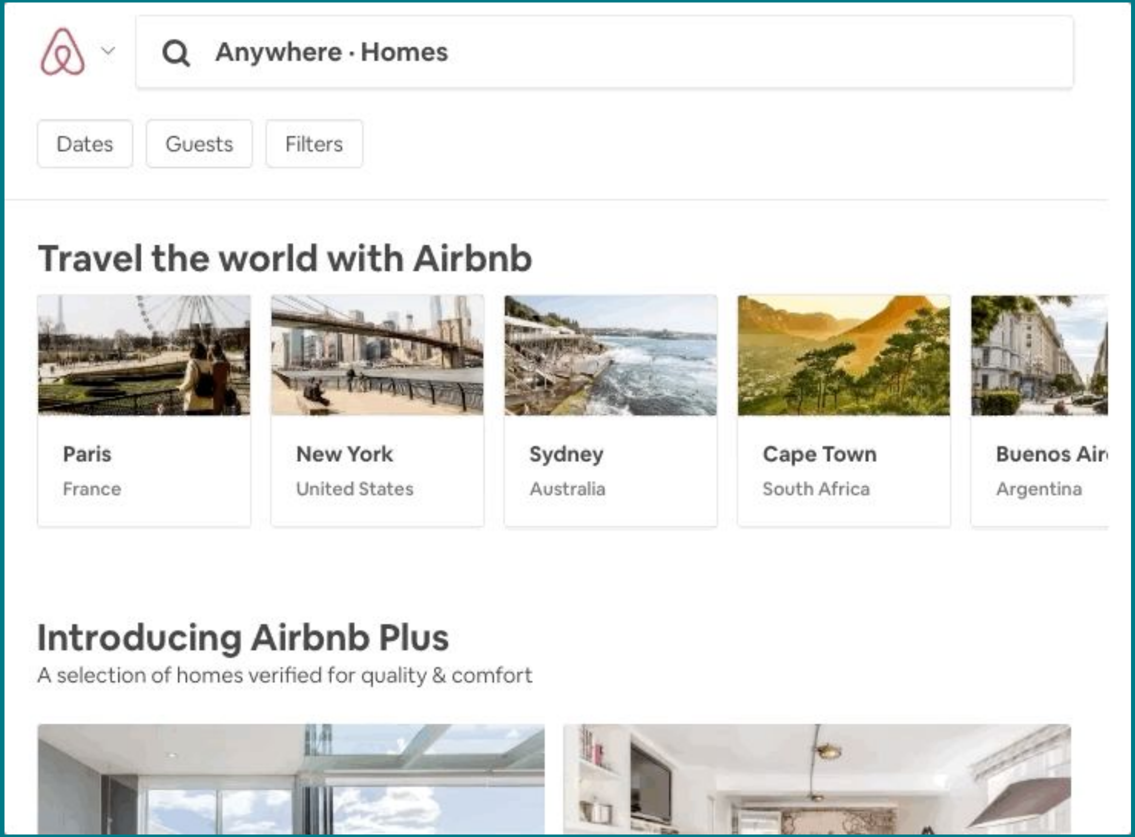
Architecture
Deep Dive

Open Source

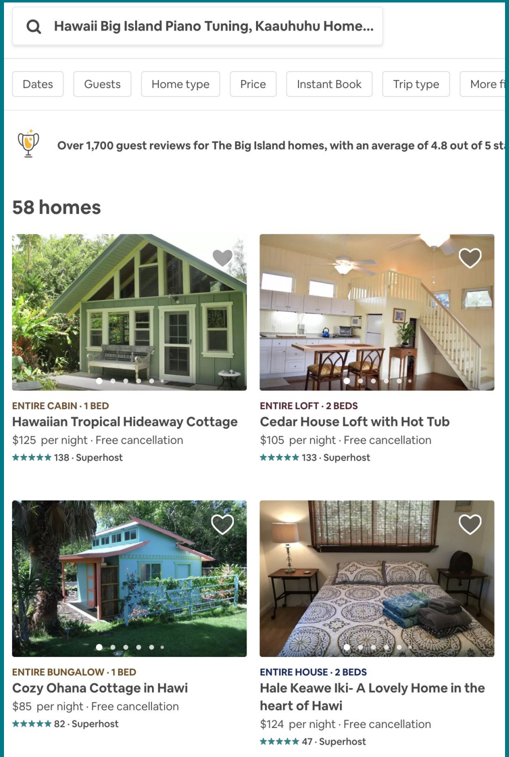
Background

Airbnb's Product

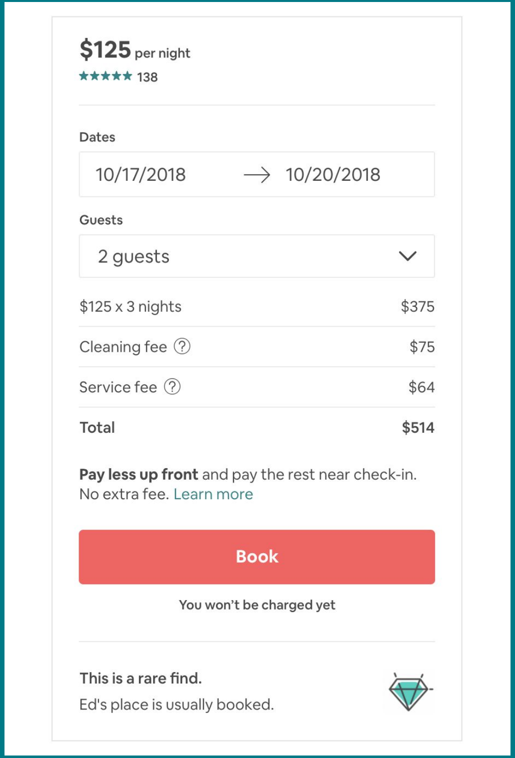
A global travel community that offers magical end-to-end trips, including where you stay, what you do and the people you meet.



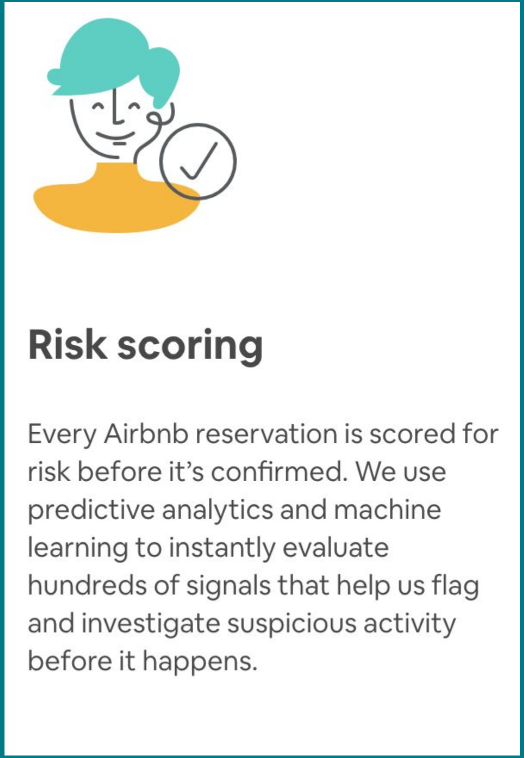
Airbnb is already driven by Machine Learning



Search Ranking



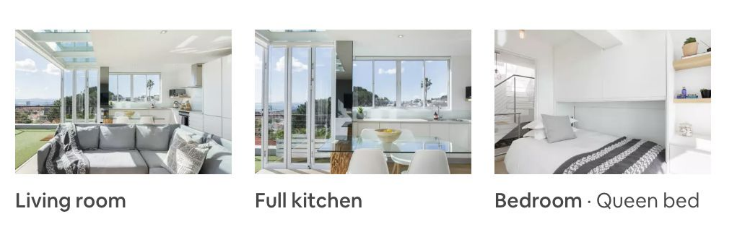
Smart Pricing

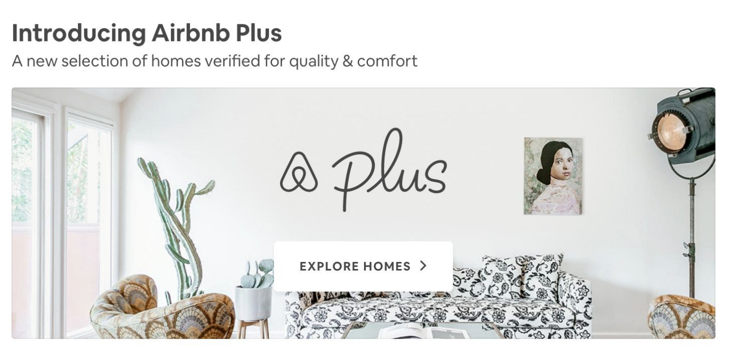


Fraud Detection

But there are *many* more opportunities for ML

- Paid Growth - Hosts
- Classifying / Categorizing Listings
- Experience Ranking + Personalization
- Room Type Categorizations
- Customer Service Ticket Routing
- Airbnb Plus
- Listing Photo Quality
- Object Detection - Amenities
- ....





Intrinsic Complexities with Machine Learning

- Understanding the business domain
- Selecting the appropriate Model
- Selecting the appropriate Features
- Fine tuning

Incidental Complexities with Machine Learning

- Integrating with Airbnb's Data Warehouse
- Scaling model training & serving
- Keeping consistency between: Prototyping vs Production, Training vs Inference
- Keeping track of multiple models, versions, experiments
- Supporting iteration on ML models

→ ML models take on average <u>8 to 12 weeks</u> to build

→ ML workflows tended to be slow, fragmented, and brittle

The ML Infrastructure Team addresses these challenges

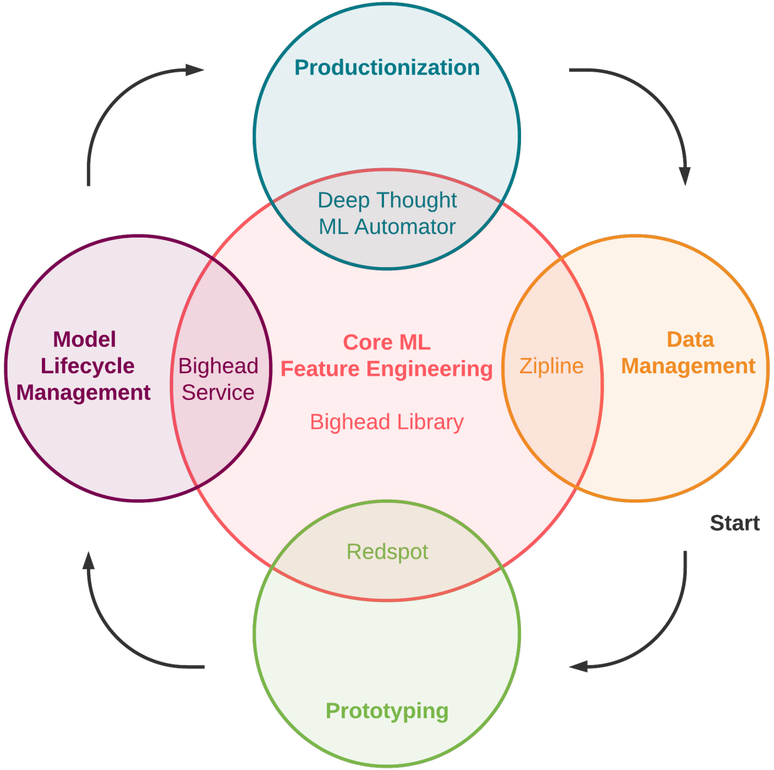
Vision

Airbnb routinely ships ML-powered features throughout the product.

Mission

Equip Airbnb with shared technology to build *production-ready* ML applications with no *incidental complexity*.

Supporting the Full ML Lifecycle



Bighead: Design Goals

Seamless

Versatile

Consistent

Scalable

Seamless

- Easy to prototype, easy to productionize
- Same workflow across different frameworks

Versatile

- Supports all major ML frameworks

- Meets various requirements
  - Online and Offline
  - Data size
  - SLA
  - GPU training
  - Scheduled and Ad hoc

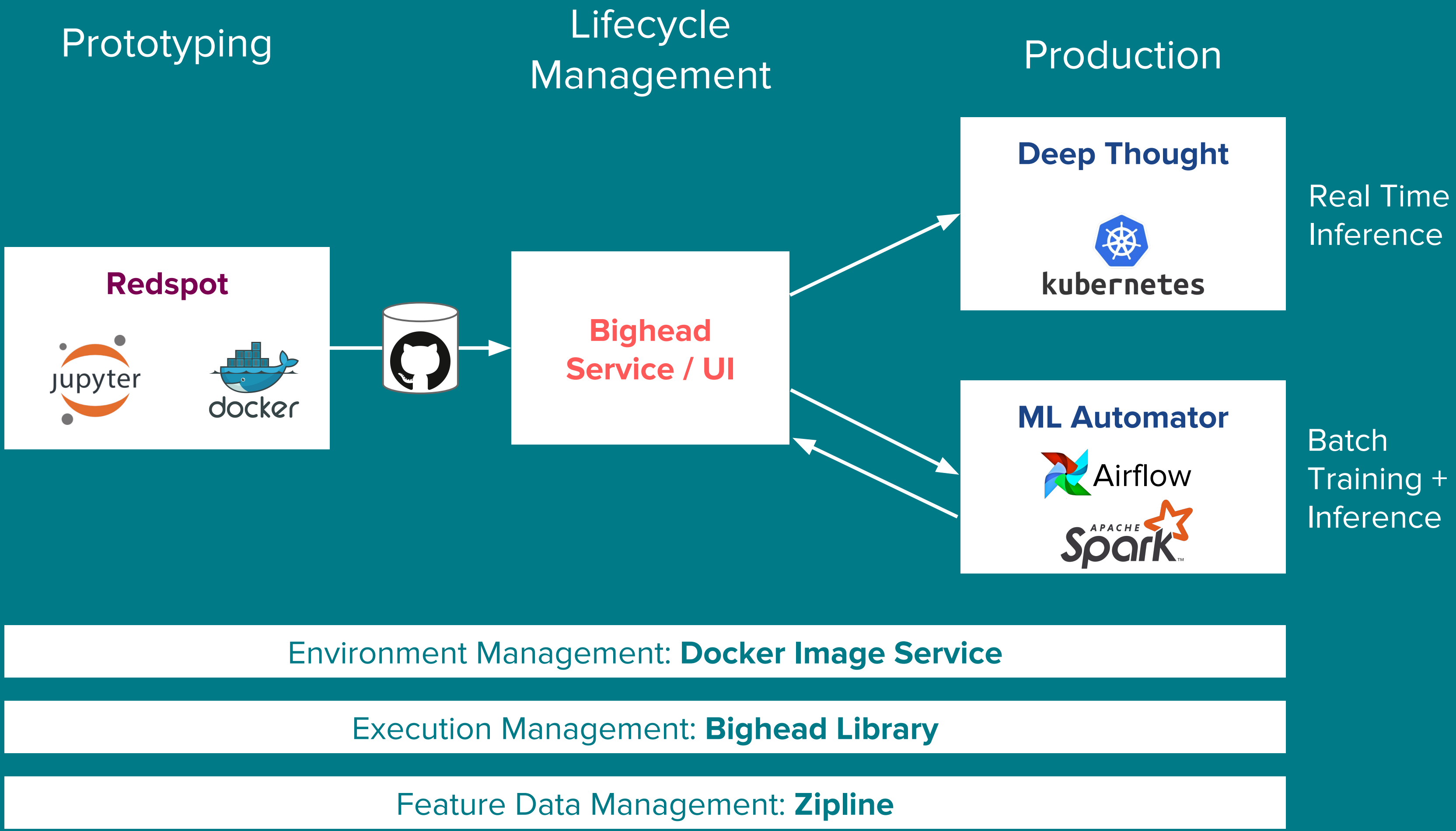
Consistent

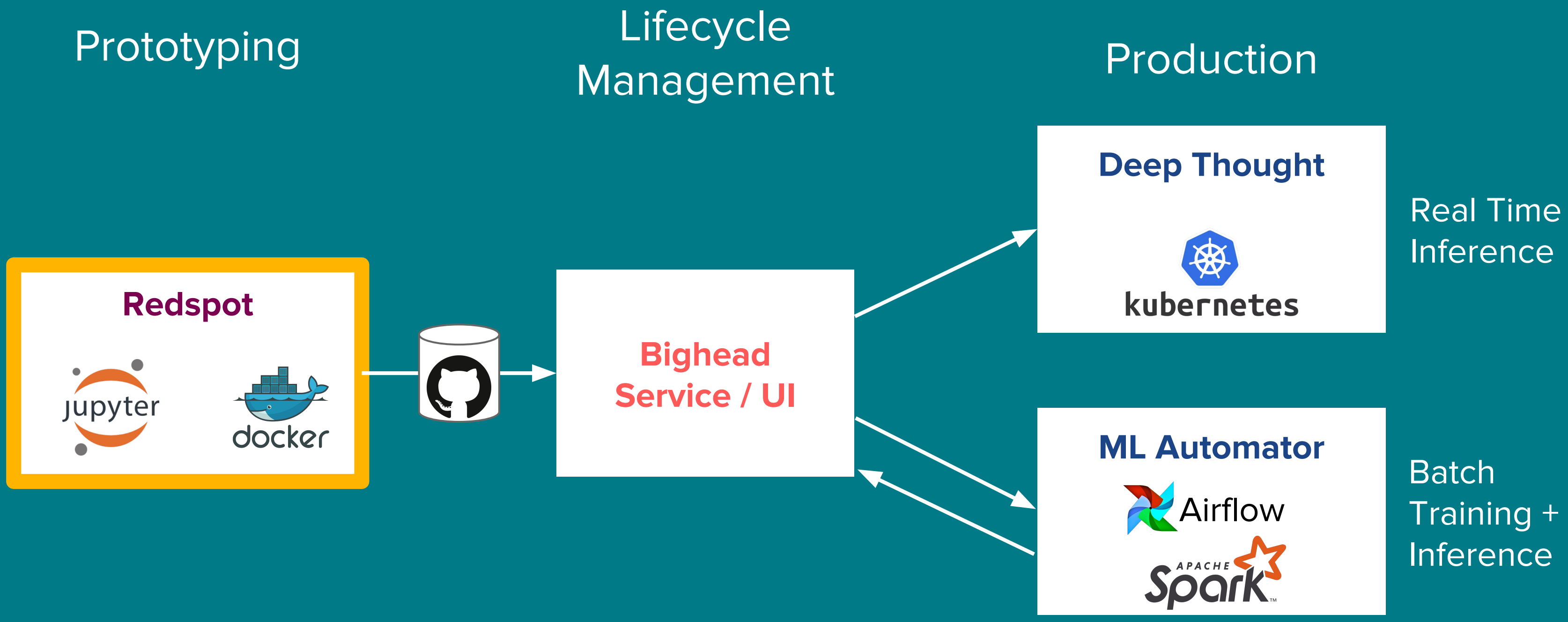
- Consistent environment across the stack
- Consistent data transformation
  - Prototyping and Production
  - Online and Offline

Scalable

- Horizontal
- Elastic

Bighead: Architecture Deep Dive


Prototyping
Lifecycle Management
Production
Redspot
jupyter
docker
Bighead Service / UI
Deep Thought
kubernetes
Real Time Inference
ML Automator
Airflow
APACHE Spark
Batch Training + Inference
Environment Management: Docker Image Service
Execution Management: Bighead Library
Feature Data Management: Zipline



Environment Management: **Docker Image Service**

Execution Management: **Bighead Library**

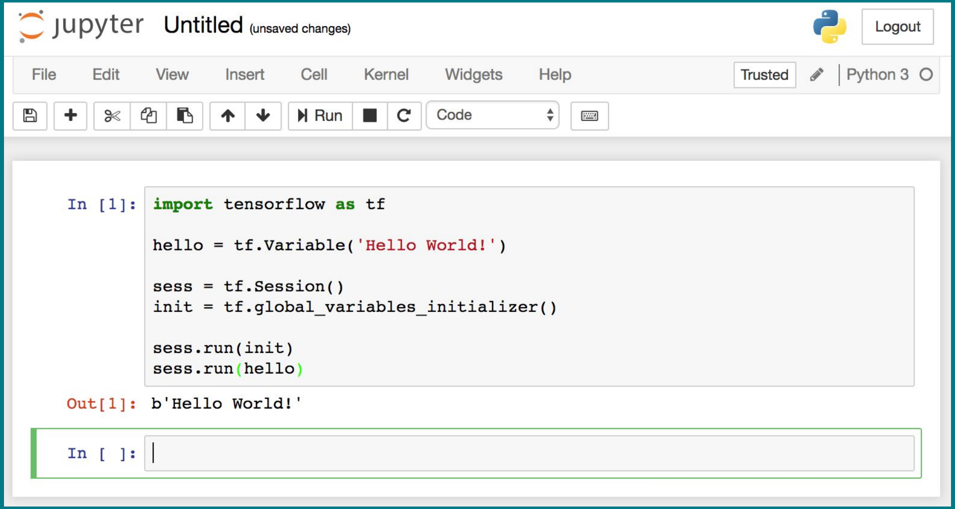
Feature Data Management: **Zipline**

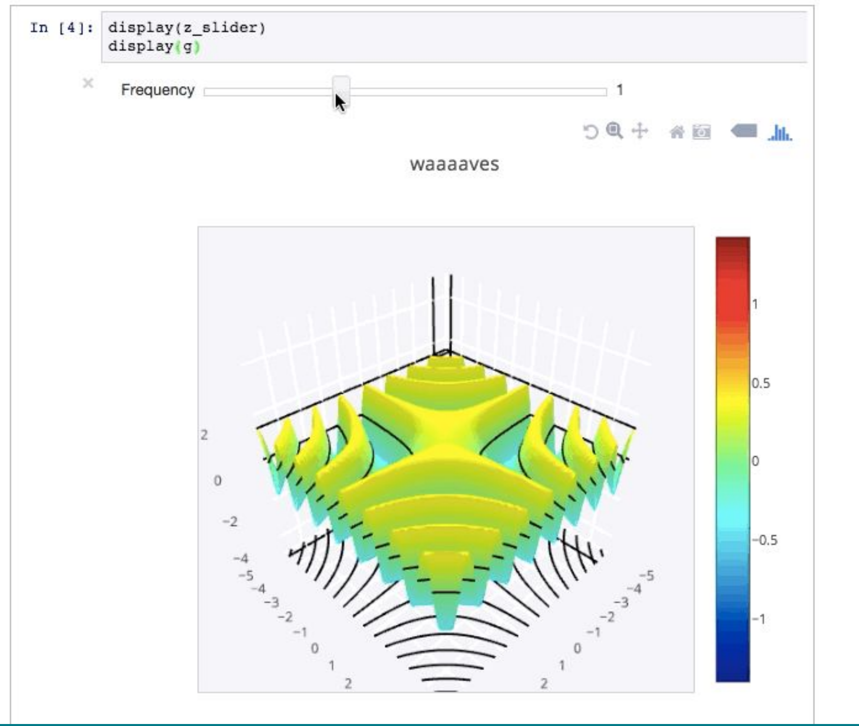
Redspot

Prototyping with Jupyter Notebooks

Jupyter Notebooks?

What are those?





"Creators need an immediate connection to what they are creating."
- Bret Victor

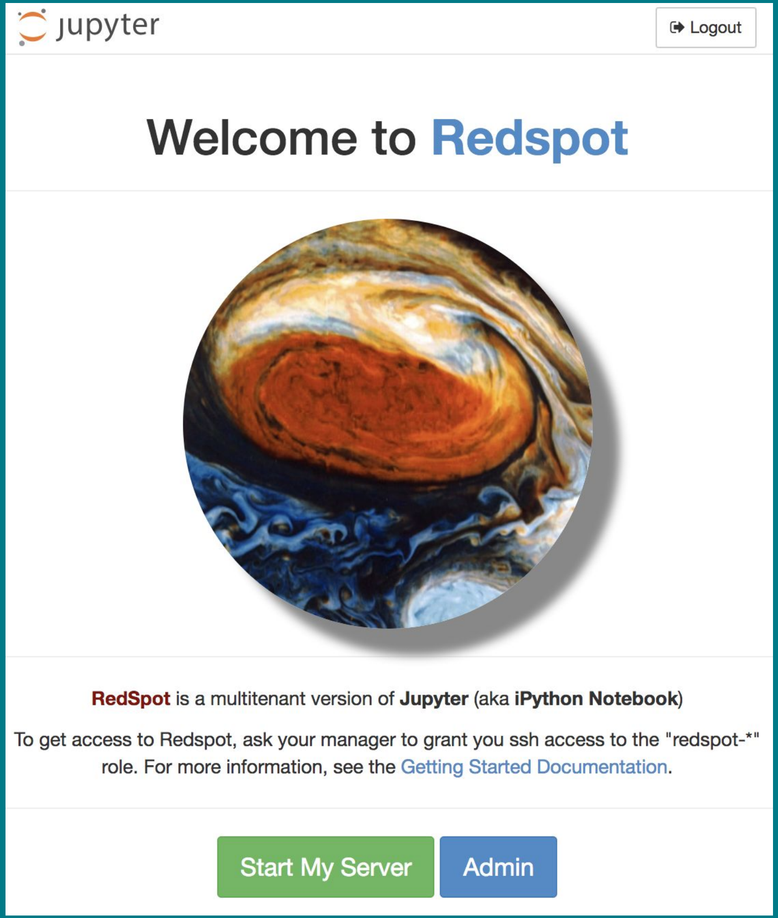
The ideal Machine Learning development environment?

- Interactivity and Feedback
- Access to Powerful Hardware
- Access to **Data**

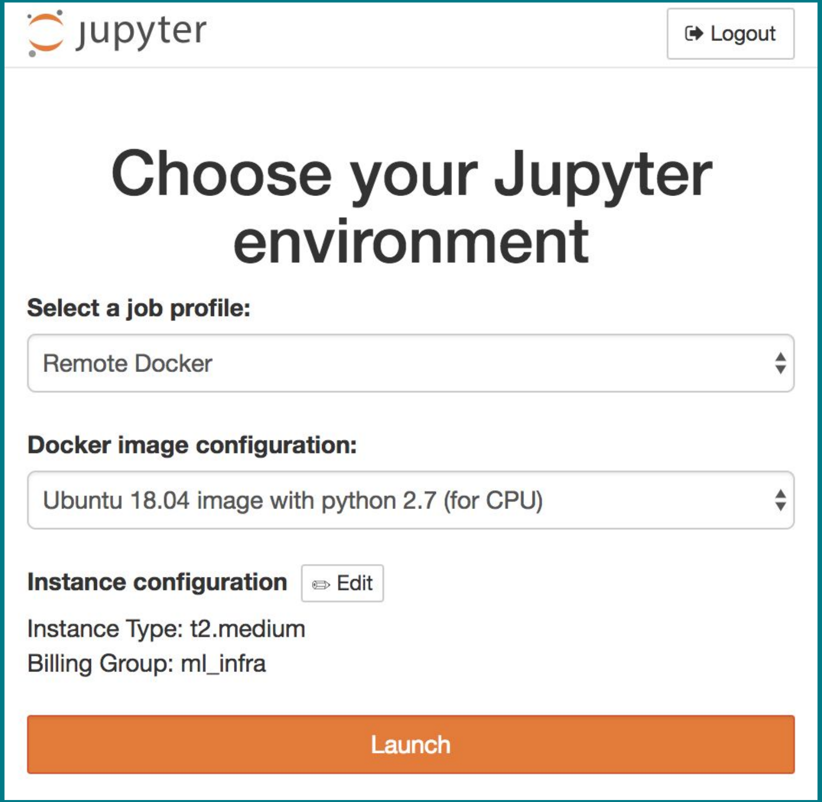
Redspot

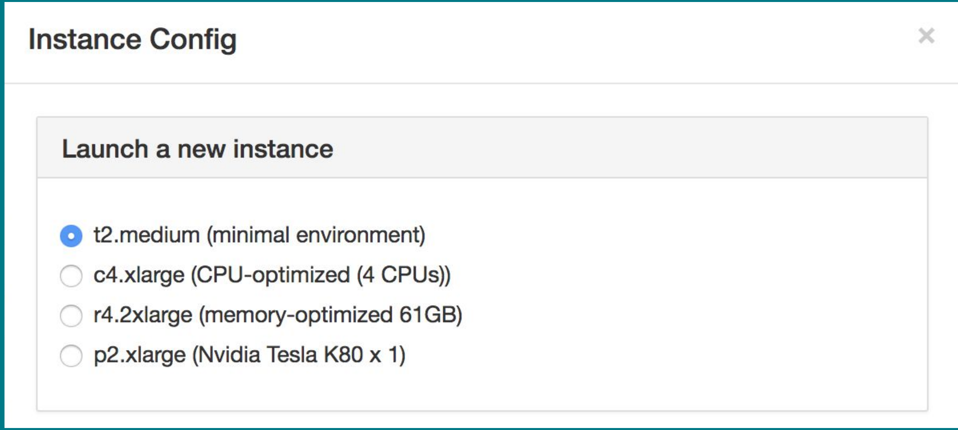
a Supercharged Jupyter Notebook Service

- A fork of the JupyterHub project
- Integrated with our Data Warehouse
- Access to specialized hardware (e.g. GPUs)
- File sharing between users via AWS EFS
- Packaged in a familiar Jupyterhub UI



Redspot


jupyter
Logout
Choose your Jupyter environment
Select a job profile:
Remote Docker
Docker image configuration:
Ubuntu 18.04 image with python 2.7 (for CPU)
Instance configuration
Edit
Instance Type: t2.medium
Billing Group: ml_infra
Launch


Instance Config
Launch a new instance
t2.medium (minimal environment)
c4.xlarge (CPU-optimized (4 CPUs))
r4.2xlarge (memory-optimized 61GB)
p2.xlarge (Nvidia Tesla K80 x 1)

Redspot

a Supercharged Jupyter Notebook Service

Consistent

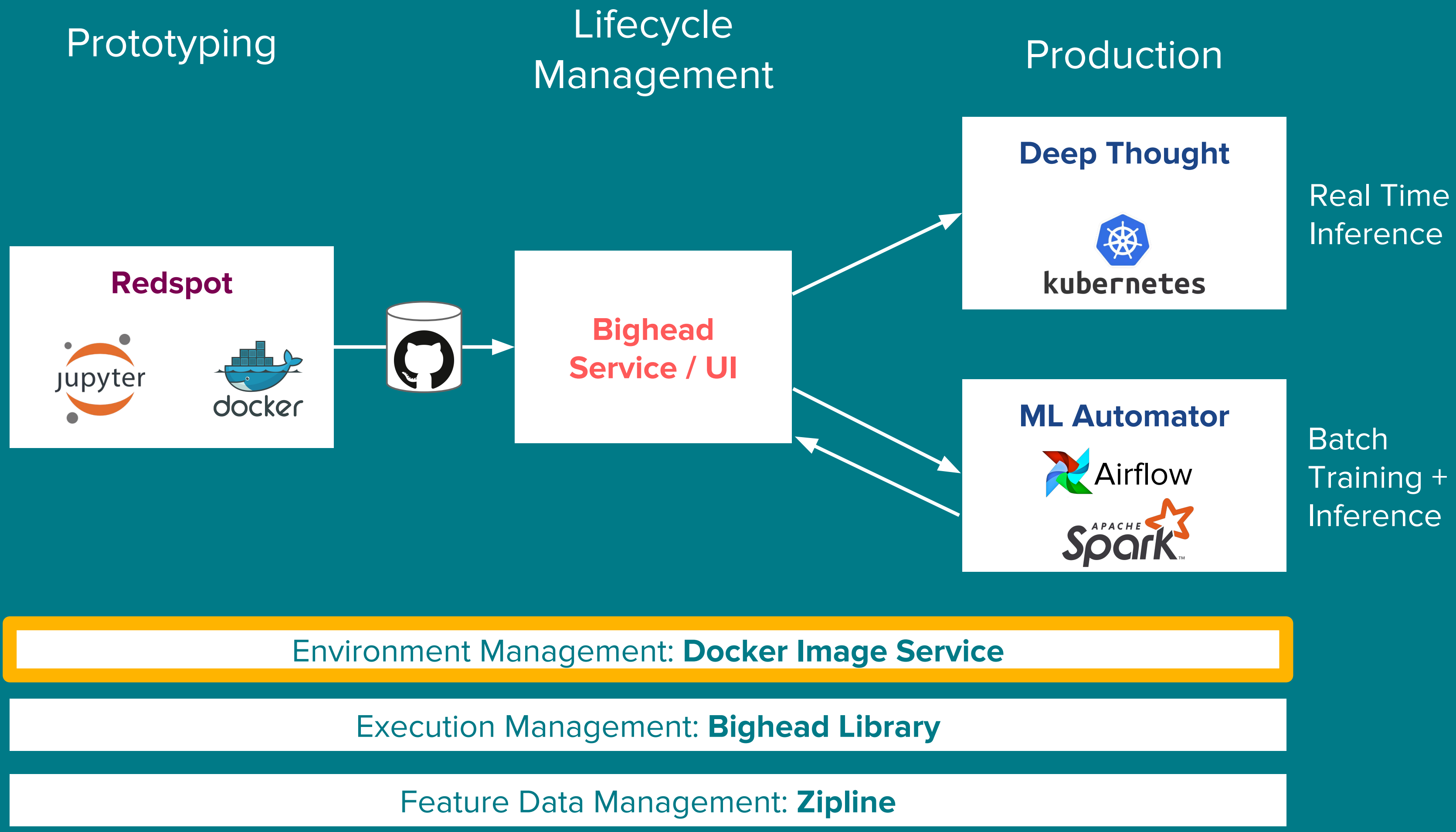
- Promotes prototyping in the exact environment that your model will use in production

Versatile

- Customized Hardware: AWS EC2 Instance Types e.g. P3, X1
- Customized Dependencies: Docker Images e.g. Py2.7, Py3.6+Tensorflow

Seamless

- Integrated with Bighead Service & Docker Image Service via APIs & UI widgets


Prototyping
Lifecycle Management
Production
Redspot
jupyter
docker
Bighead Service / UI
Deep Thought
kubernetes
Real Time Inference
ML Automator
Airflow
APACHE Spark
Batch Training + Inference
Environment Management: Docker Image Service
Execution Management: Bighead Library
Feature Data Management: Zipline

Docker Image Service

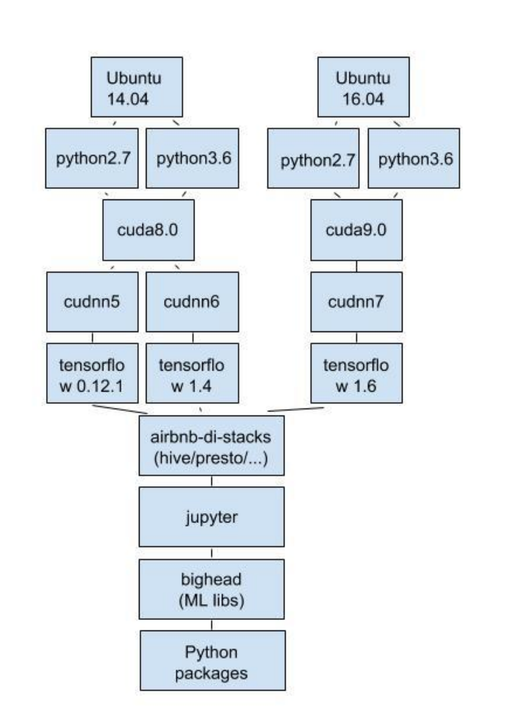
Environment Customization

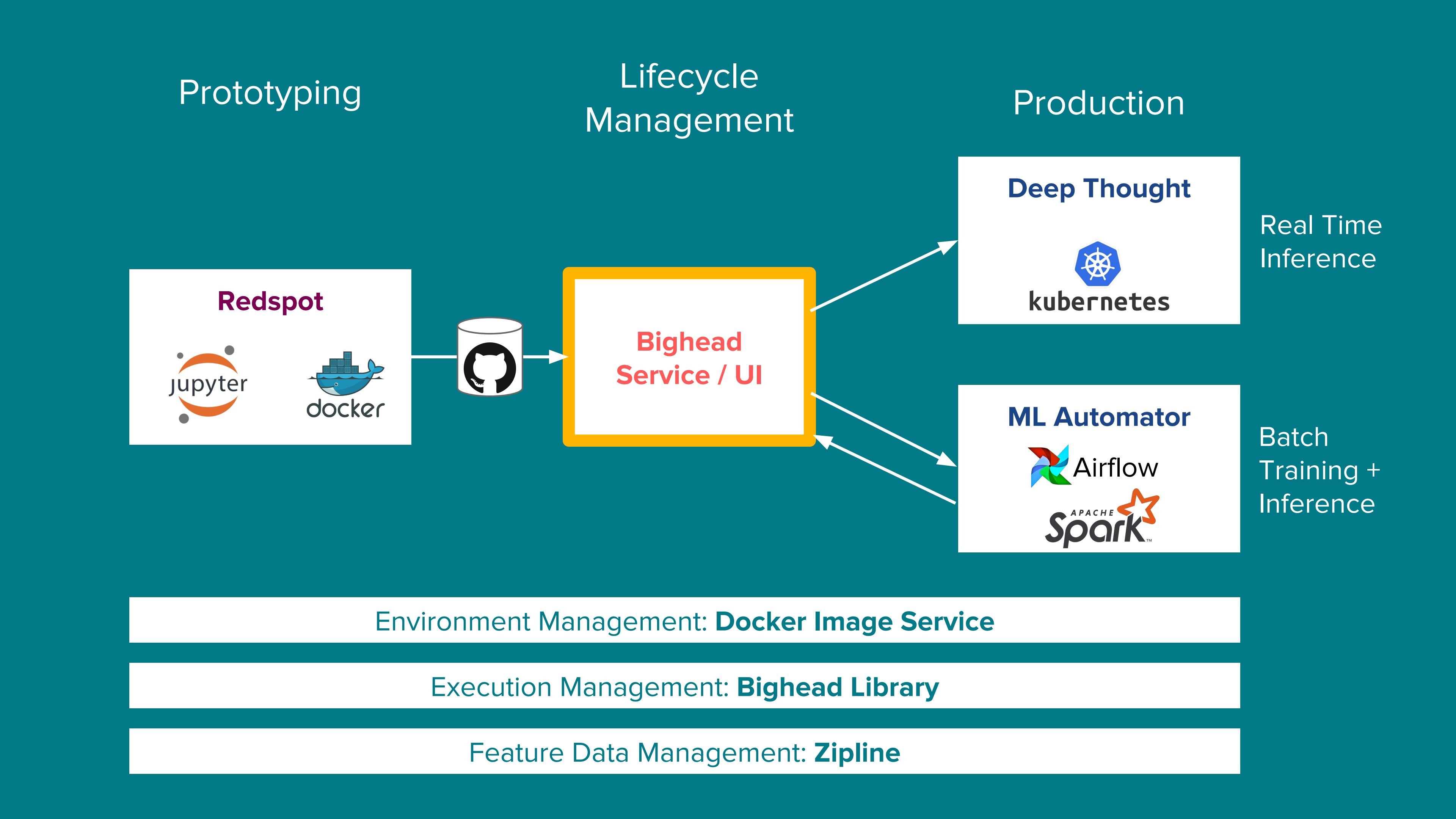
Docker Image Service - Why

- ML Users have a diverse, heterogeneous set of dependencies
- Need an easy way to bootstrap their own runtime environments
- Need to be consistent with the rest of Airbnb’s infrastructure

Docker Image Service - Dependency Customization

- Our configuration management solution
- A composition layer on top of Docker
- Includes a customization service that faces our users
- Promotes **Consistency** and **Versatility**





Prototyping
Lifecycle Management
Production
Redspot
jupyter
docker
Bighead Service / UI
Deep Thought
kubernetes
Real Time Inference
ML Automator
Airflow
APACHE Spark
Batch Training + Inference
Environment Management: Docker Image Service
Execution Management: Bighead Library
Feature Data Management: Zipline

Bighead Service

Model Lifecycle Management

Model Lifecycle Management - why?

- Tracking ML model changes is just as important as tracking code changes
- ML model work needs to be **reproducible** to be **sustainable**
- Comparing experiments before you launch models into production is critical

Bighead

Send Feedback Documentation

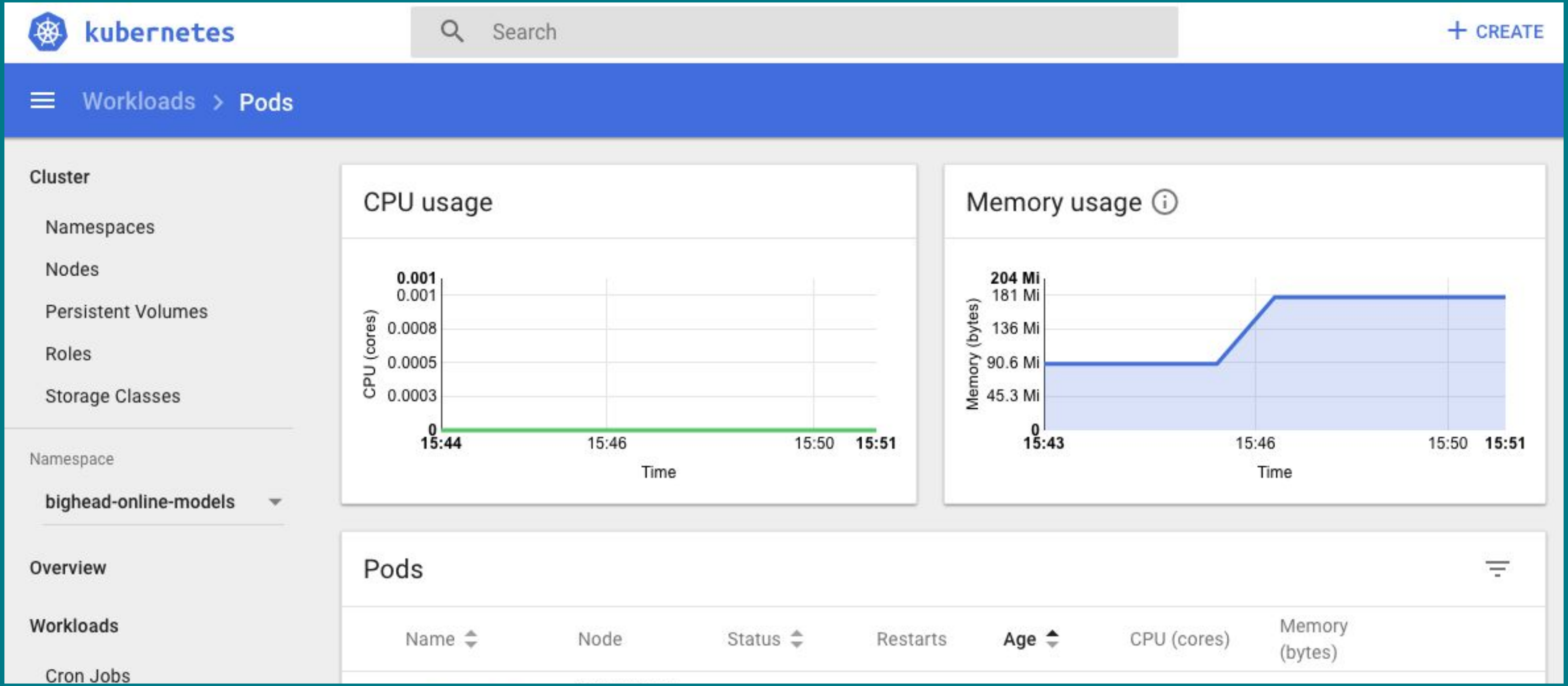
Introducing Bighead

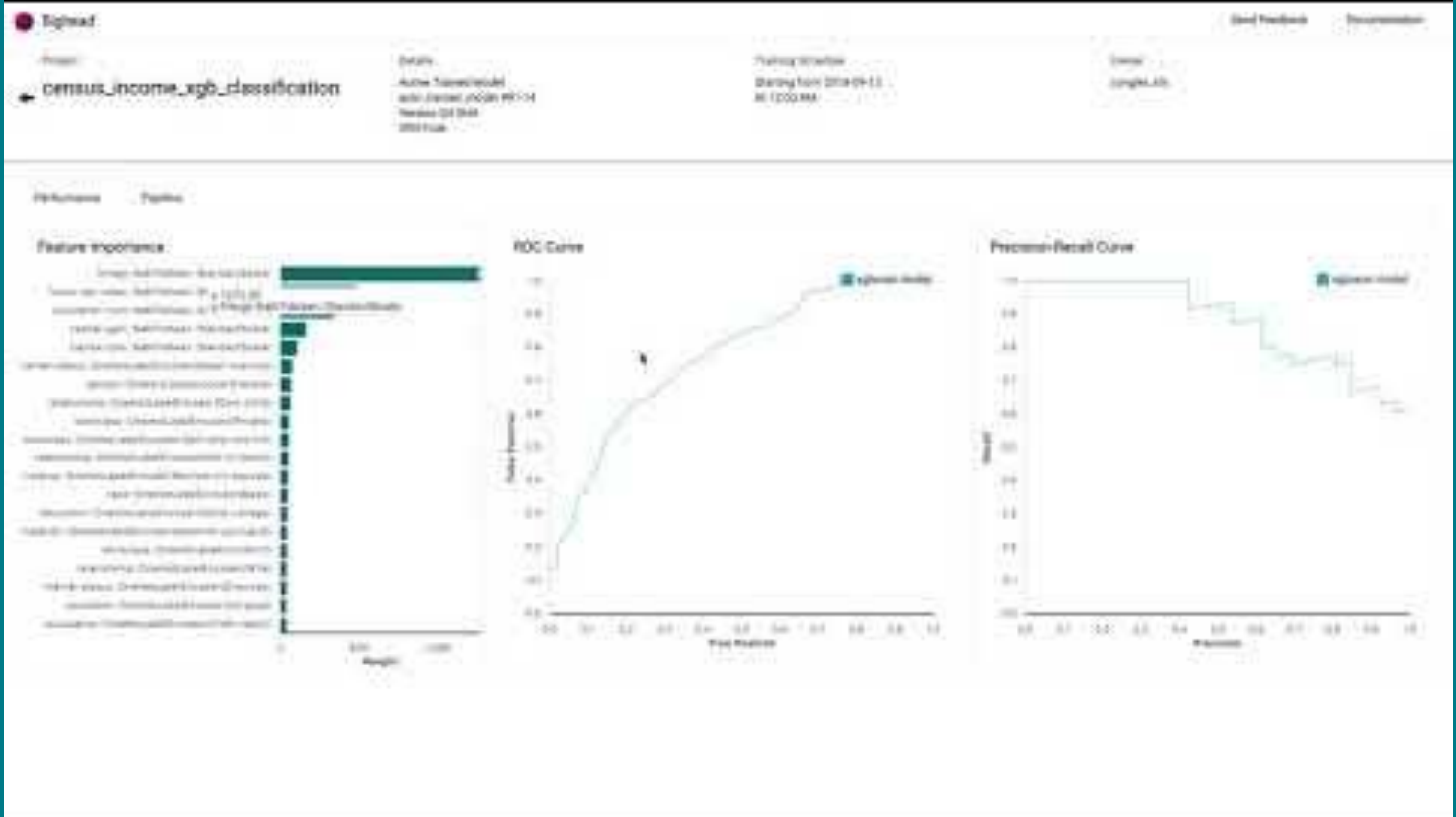
Explore, deploy, and monitor machine learning models.

learn more

Showing 15 machine learning projects

Name	Owner	Active Artifact Name
census_income_xgb_classification	conglei_shi	auto_trained_model #0
price_prediction_keras	conglei_shi	auto_trained_model #0-0
census_income_pytorch_classification	conglei_shi	auto_trained_model #18-1
image_classification_tensorflow	conglei_shi	auto_trained_model #0-2
image_classification_pytorch_CNN	conglei_shi	N/A
speech_recognition_spanish	conglei_shi	auto_trained_model #0-19
census_income_mxnet_classification	conglei_shi	auto_trained_model #0-20
speech_recognition_english	conglei_shi	auto_trained_model #0-21
translation_model_eng_esp	conglei_shi	auto_trained_model #0-22
object_detection_image_tensorflow	conglei_shi	auto_trained_model #0-23
vr_listing_spark	conglei_shi	auto_trained_model #0-24
translation_model_eng_chn	conglei_shi	auto_trained_model #0-25


kubernetes
Search
+ CREATE
Workloads > Pods
Cluster
Namespaces
Nodes
Persistent Volumes
Roles
Storage Classes
Namespace
bighead-online-models
Overview
Workloads
Cron Jobs
CPU usage
CPU (cores)
0.001
0.001
0.0008
0.0005
0.0003
0
15:44
15:46
15:50
15:51
Time
Memory usage
Memory (bytes)
204 Mi
181 Mi
136 Mi
90.6 Mi
45.3 Mi
0
15:43
15:46
15:50
15:51
Time
Pods
Name
Node
Status
Restarts
Age
CPU (cores)
Memory (bytes)


census_income_xgb_classification
Feature Importance
ROC Curve
Precision-Recall Curve

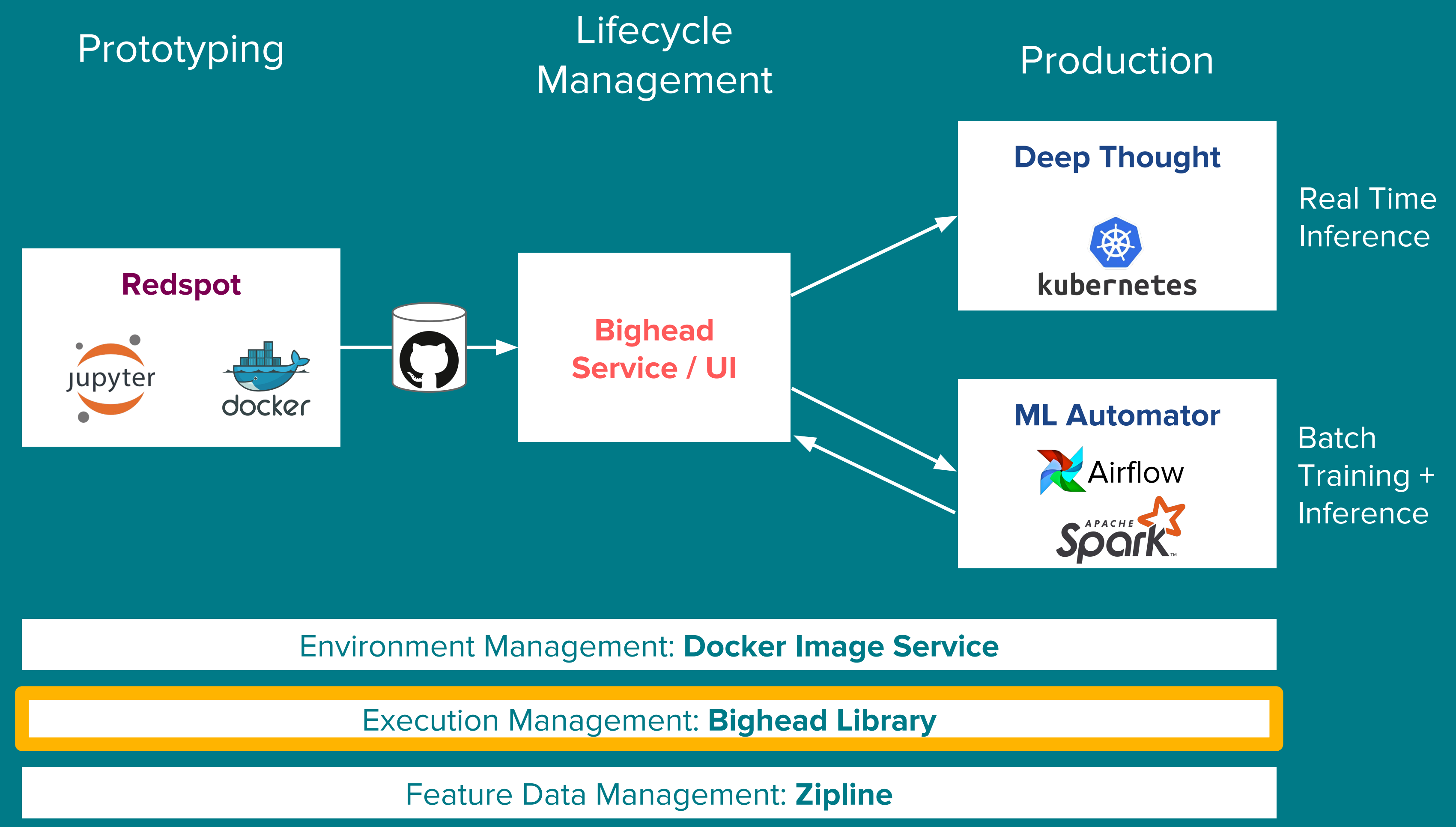
Bighead Service

Consistent

- Central model management service
- Single source of truth about the state of a model, it's model, it's dependencies, and what's deployed

Seamless

- Context-aware visualizations that carry over from the prototyping experience


Prototyping
Lifecycle Management
Production
Redspot
jupyter
docker
Bighead Service / UI
Deep Thought
kubernetes
Real Time Inference
ML Automator
Airflow
APACHE Spark
Batch Training + Inference
Environment Management: Docker Image Service
Execution Management: Bighead Library
Feature Data Management: Zipline

Bighead Library

ML Models are highly heterogeneous in

Frameworks

Training data

- Data quality
- Structured vs Unstructured (image, text)

Environment

- GPU vs CPU
- Dependencies

ML Models are hard to keep consistent

- Data in *production* is different from data in *training*
- *Offline* pipeline is different from *online* pipeline
- Everyone does everything in a *different* way

Bighead Library

Versatile

- Pipeline on steroids - compute graph for preprocessing / inference / training / evaluation / visualization
- Composable, Reusable, Shareable
- Support popular frameworks
- Fast primitives for preprocessing
- Metadata for trained models

Consistent

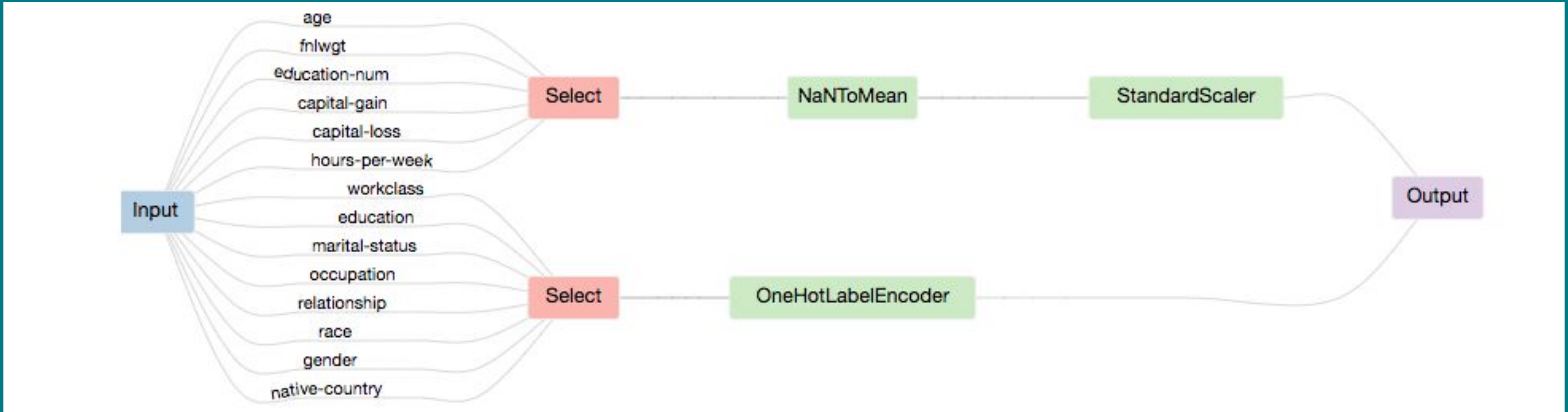
- Uniform API
- Serializable - same pipeline used in training, offline inference, online inference

Bighead Library: ML Pipeline

```
categorical = [
    'workclass',
    'education',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'gender',
    'native-country',
]
numeric = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    'hours-per-week',
]

p = Pipeline('ClassifyCensusIncome')
p[numeric] >>= [NaNToMean(dtype=np.float32), StandardScaler()]
p[categorical] >>= OneHotLabelEncoder()
p >>= XGBClassifier(objective='binary:logistic',
                    n_estimators=100,
                    learning_rate=0.1,
                    max_depth=5)
```

Visualization - Pipeline



Easy to Serialize/Deserialize

```
In [ ]: p.serialize('test.tar.xz')
```

```
In [ ]: p2 = Pipeline.deserialize('test.tar.xz')
```

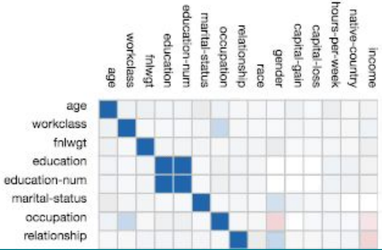
Visualization - Training Data

In [2]:
```python
train, test = load_census_income()
categorical = [
    'workclass',
    'education',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'gender',
    'native-country',
]
numeric = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    'hours-per-week',
]
labels = np.array([0 if x== ' <=50K' else 1 for x in train['income'].values])
```

In [3]:
```python
from bighead.core.visualization import show_stats
show_stats(train)
```

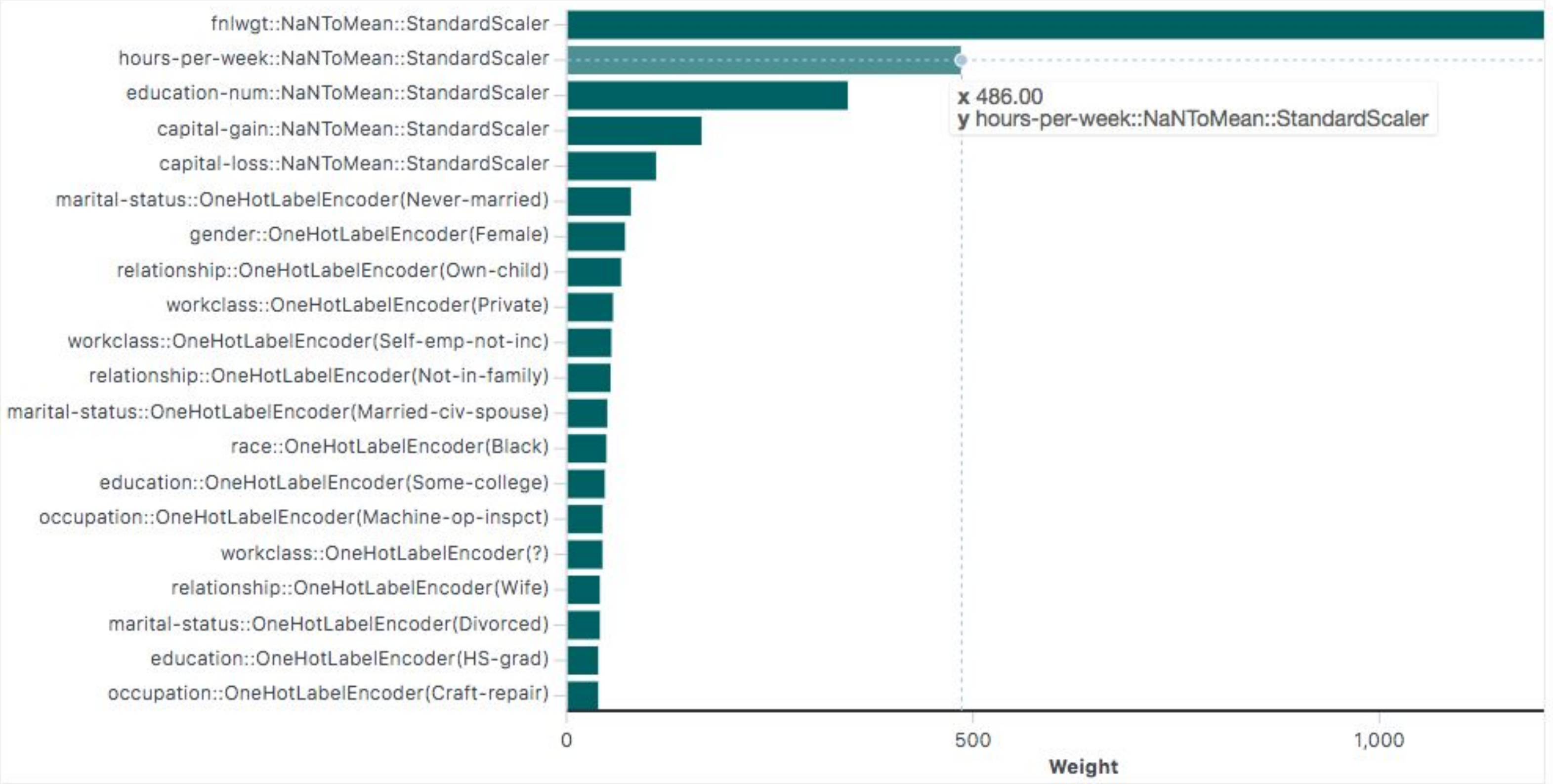
Feature Overview | Data Overview | Correlation Analysis

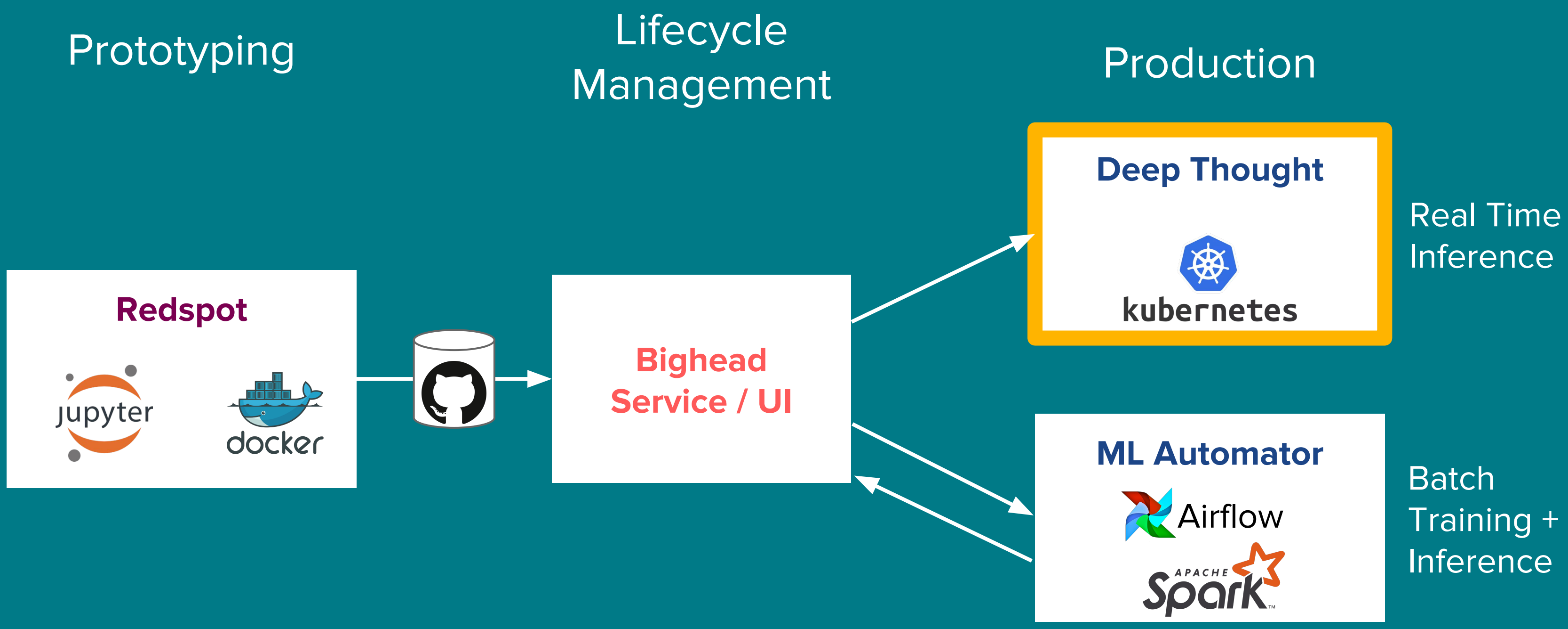
Name	Not-null	Distinct Num	Statistics
workclass categorical		9	
relationship categorical		6	
gender categorical		2	



Visualization - Transformer

In [8]:
```
from bighead.core.visualization import visualize
visualize(xgboost.get_feature_importance_metadata()[0])
```





Environment Management: **Docker Image Service**

Execution Management: **Bighead Library**

Feature Data Management: **Zipline**

Deep Thought

Online Inference

Hard to make online model serving...

Consistent with training

- Different data
- Different pipeline
- Different dependencies

Easy to do

- Data scientists can’t launch models without engineer team
- Engineers often need to rebuild models

Scalable

- Resource requirements varies across models
- Throughput fluctuates across time

Deep Thought

Consistent

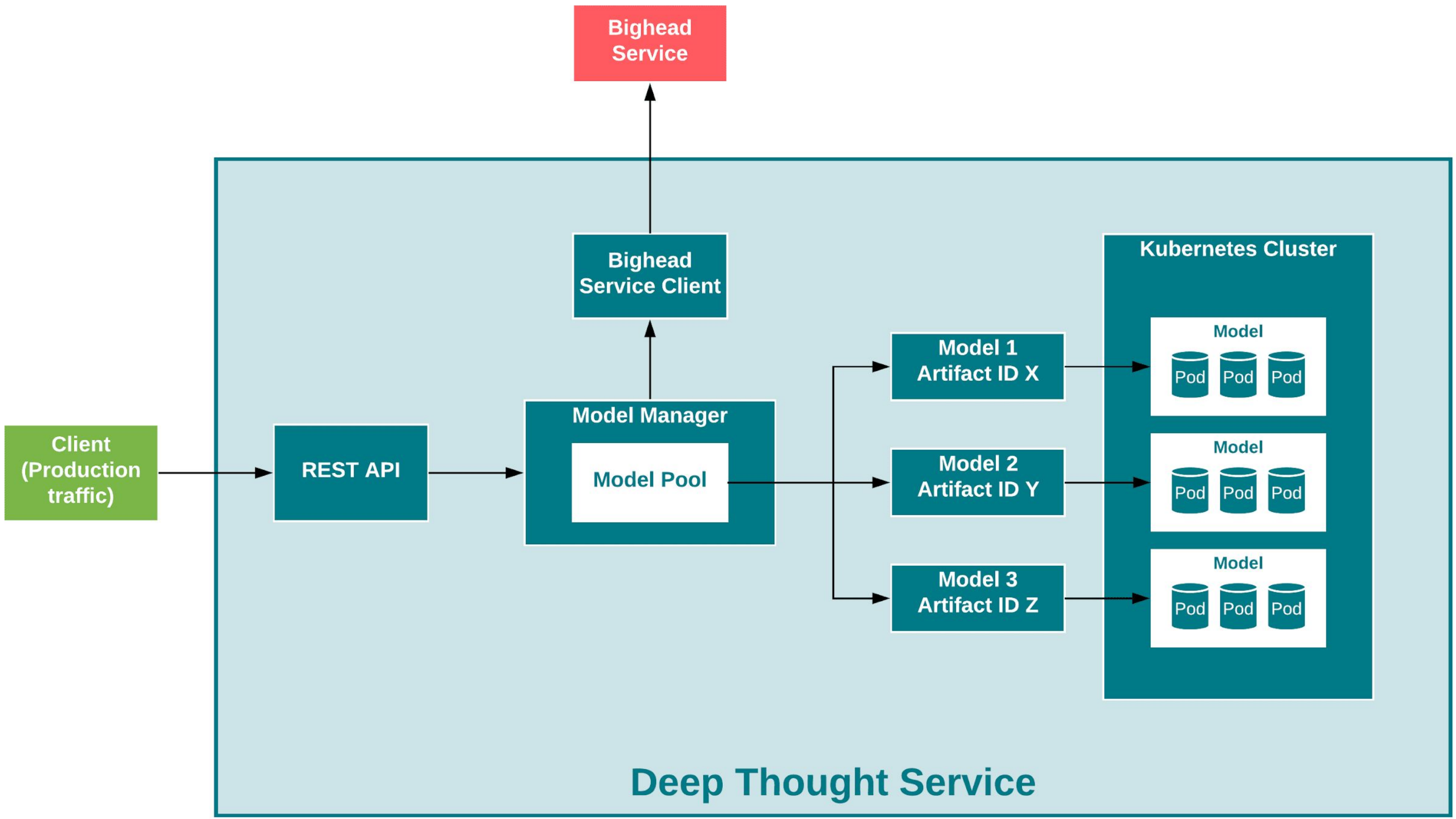
- Docker + Bighead Library: Same data source, pipeline, environment from training

Seamless

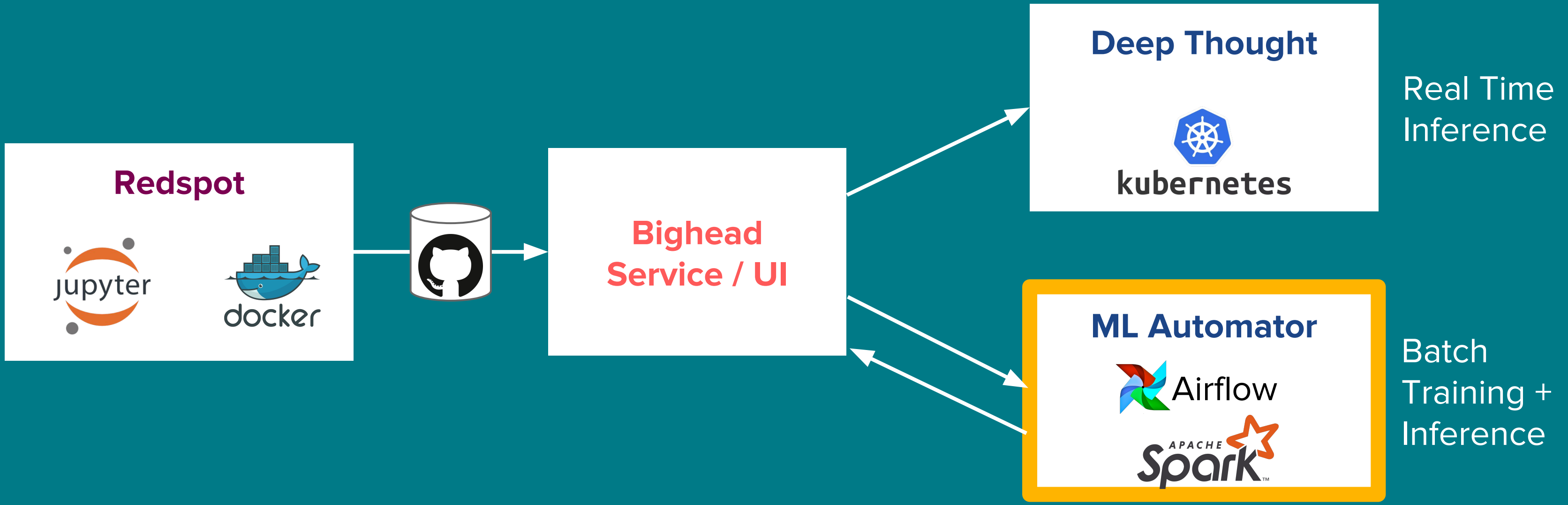
- Integration with event logging, dashboard
- Integration with Zipline

Scalable

- Kubernetes: Model pods can easily scale
- Resource segregation across models


Bighead Service
Bighead Service Client
Kubernetes Cluster
Model 1 Artifact ID X
Model
Pod
Pod
Pod
Model Manager
Client (Production traffic)
REST API
Model Pool
Model 2 Artifact ID Y
Model
Pod
Pod
Pod
Model 3 Artifact ID Z
Model
Pod
Pod
Pod
Deep Thought Service

Prototyping | Lifecycle Management | Production



Environment Management: **Docker Image Service**

Execution Management: **Bighead Library**

Feature Data Management: **Zipline**

ML Automator

Offline Training and Batch Inference

ML Automator - Why

Automated training, inference, and evaluation are necessary

- Scheduling
- Resource allocation
- Saving results
- Dashboards and alerts
- Orchestration

ML Automator

Consistent

- Docker + Bighead Library: Same data source, pipeline, environment across the stack

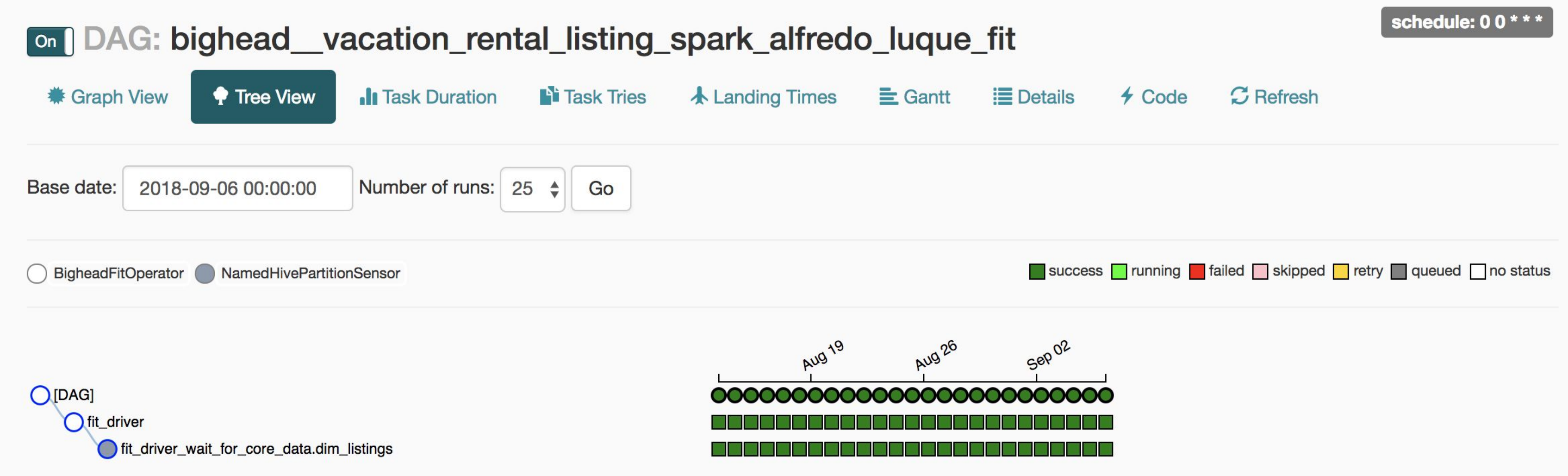
Seamless

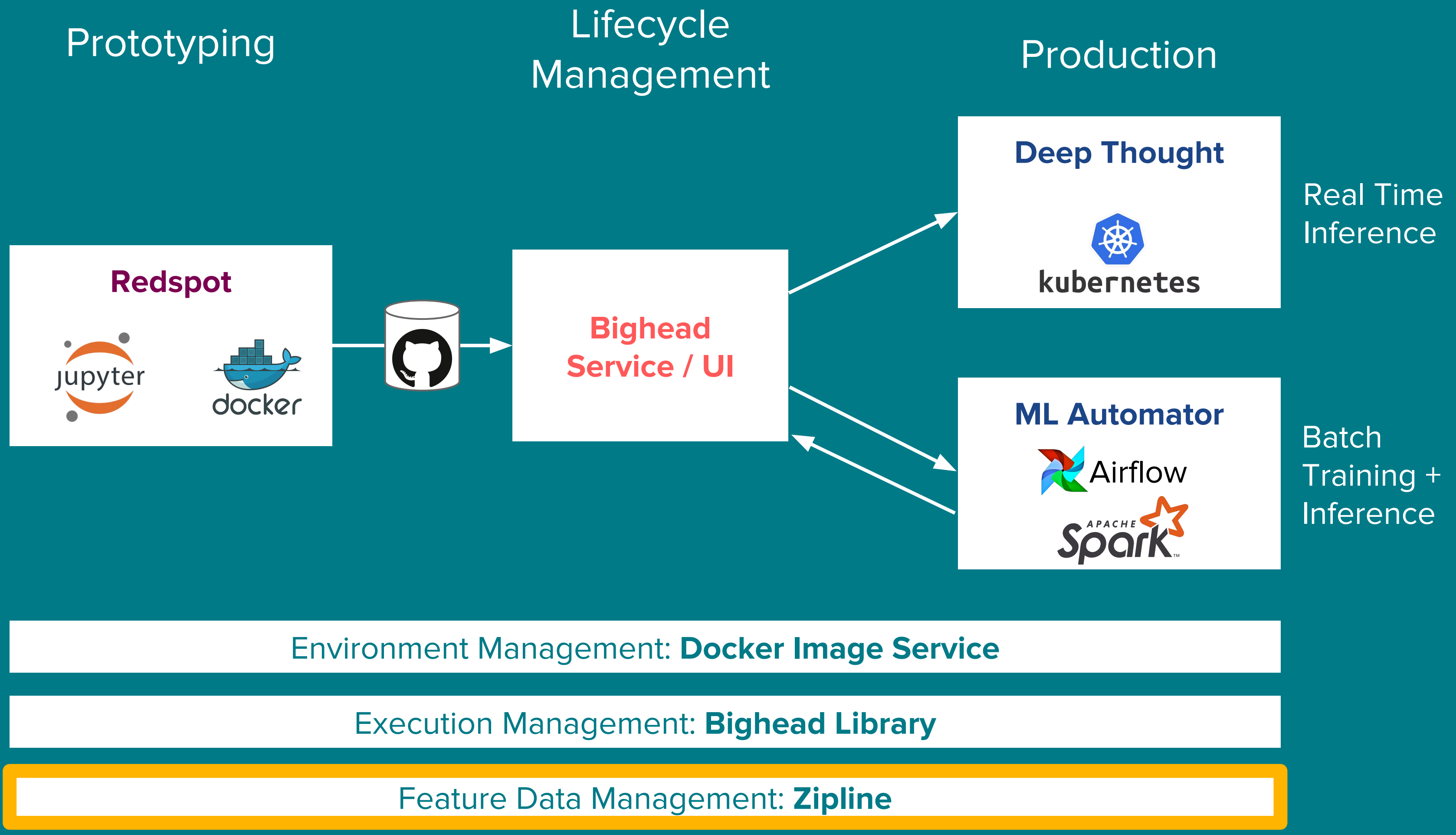
- Automate tasks via Airflow: Generate DAGs for training, inference, etc. with appropriate resources
- Integration with Zipline for training and scoring data

Scalable

- Spark: Distributed computing for large datasets

ML Automator




Prototyping
Lifecycle Management
Production
Redspot
jupyter
docker
Bighead Service / UI
Deep Thought
kubernetes
Real Time Inference
ML Automator
Airflow
APACHE Spark
Batch Training + Inference
Environment Management: Docker Image Service
Execution Management: Bighead Library
Feature Data Management: Zipline

Zipline

ML Data Management Framework

Feature management is hard

- **Inconsistent** offline and online datasets
- **Tricky** to generate training sets that depend on time correctly
- **Slow** training sets backfill
- **Inadequate** data quality checks or monitoring
- **Unclear** feature ownership and sharing

Zipline

Consistent

- Consistent data across training/scoring
- Consistent data across development/production
- Point-in-time correctness across features to prevent label leakage

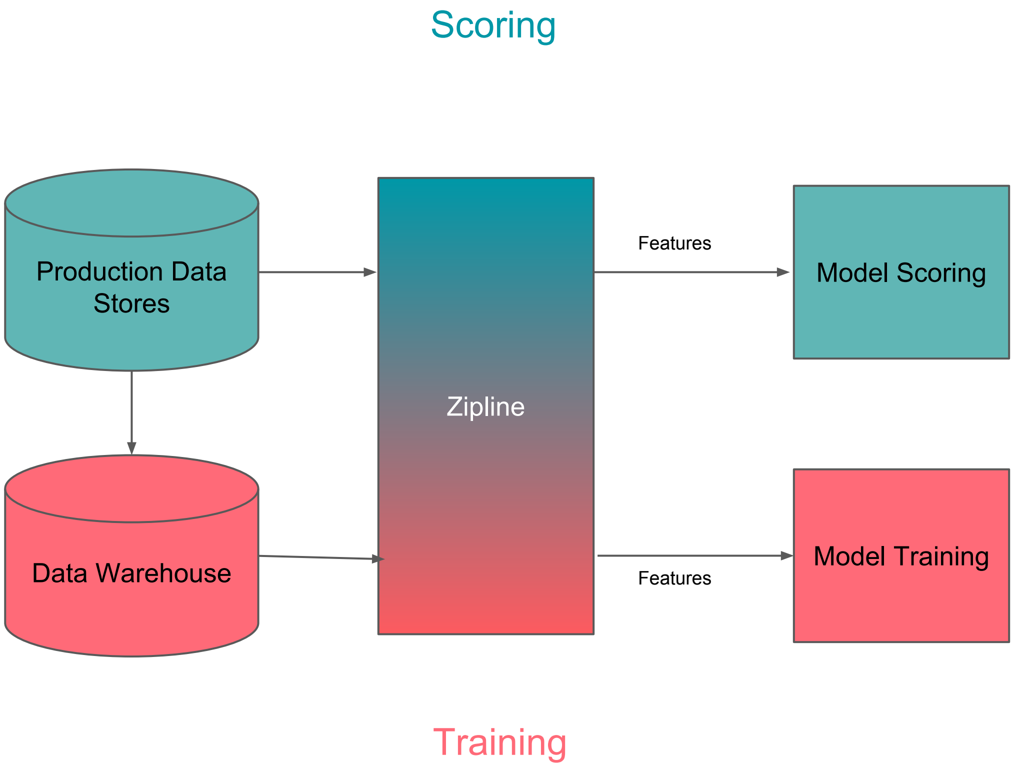
Seamless

- Integration with Deep Thought and ML Automator

Scalable

- Leverages Spark and Flink to scale Batch and Streaming workloads

Zipline Addresses the Consistency Challenge Between Training and Scoring



Big Summary

End-to-End platform to build and deploy ML models to production that is **seamless**, **versatile**, **consistent**, and **scalable**

- Model lifecycle management
- Feature generation & management
- Online & offline inference
- Pipeline library supporting major frameworks
- Docker image customization service
- Multi-tenant training environment

Built on **open source** technology

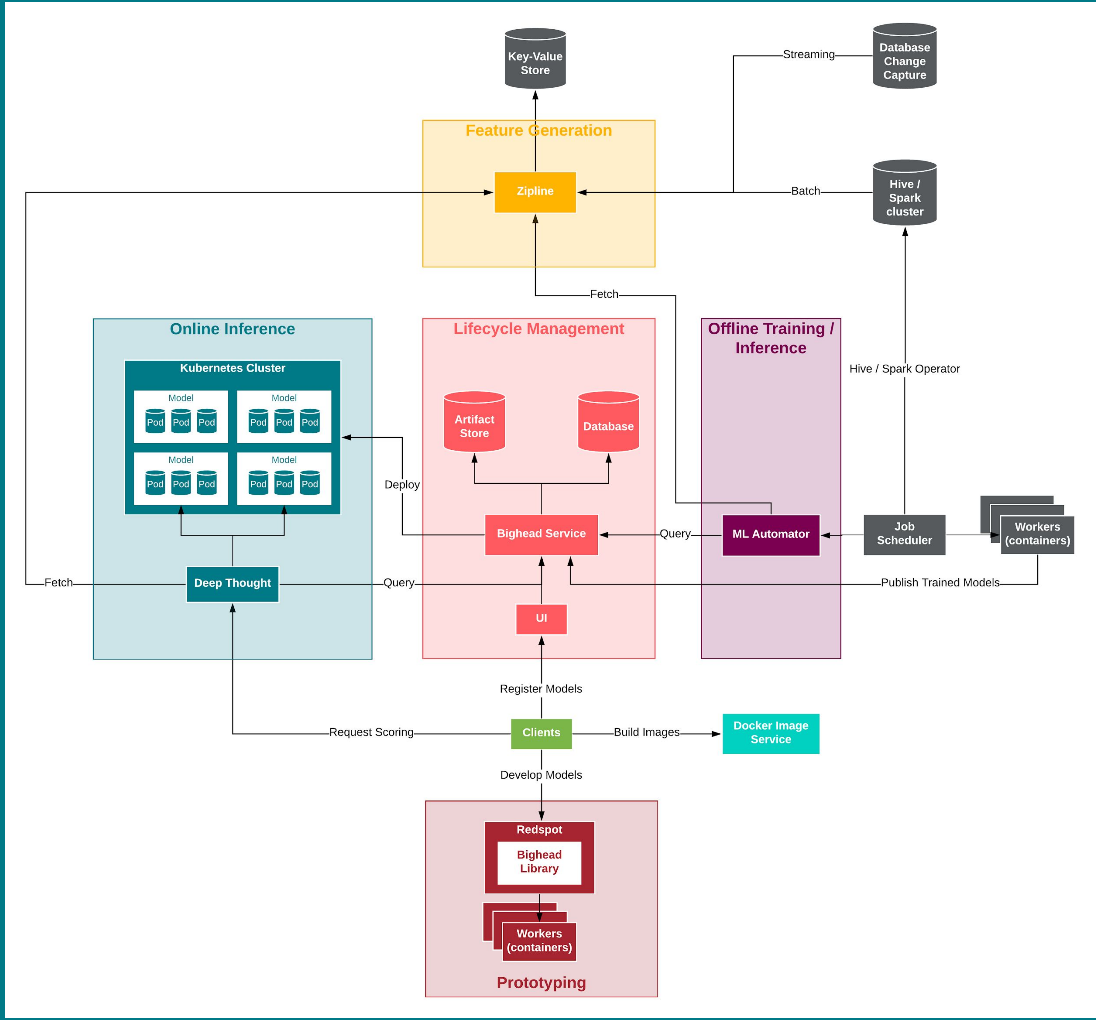
- TensorFlow, PyTorch, Keras, MXNet, Scikit-learn, XGBoost
- Spark, Jupyter, Kubernetes, Docker, Airflow

To be Open Sourced

We are selecting our first couple private collaborators. If you are interested, please email me at **andrew.hoh@airbnb.com**

Questions?

Appendix


Key-Value Store
Database Change Capture
Streaming
Feature Generation
Zipline
Batch
Hive / Spark cluster
Fetch
Online Inference
Kubernetes Cluster
Model
Pod Pod Pod
Model
Pod Pod Pod
Model
Pod Pod Pod
Model
Pod Pod Pod
Lifecycle Management
Artifact Store
Database
Offline Training / Inference
Hive / Spark Operator
Deploy
Bighead Service
Query
ML Automator
Job Scheduler
Workers (containers)
Fetch
Deep Thought
Query
Publish Trained Models
UI
Register Models
Request Scoring
Clients
Build Images
Docker Image Service
Develop Models
Redspot
Bighead Library
Workers (containers)
Prototyping