Bighead

Airbnb’s End-to-End Machine Learning Infrastructure

Andrew Hoh and Krishna Puttaswamy
ML Infra @ Airbnb

Q4 2016: Formation of our ML Infra team

In 2016

- Only a few major models in production
- Models took on average <u>8 week to 12 weeks</u> to build
- Everything built in Aerosolve, Spark and Scala
- No support for Tensorflow, PyTorch, SK-Learn or other popular ML packages
- Significant discrepancies between offline and online data

ML Infra was formed with the charter to:

- Enable more users to build ML products
- Reduce time and effort
- Enable easier model evaluation

Before ML Infrastructure

ML has had a massive impact on Airbnb's product

- Search Ranking
- Smart Pricing
- Fraud Detection

After ML Infrastructure

But there were many other areas that had high-potential for ML, but had yet to realize its full potential.

- Paid Growth - Hosts
- Classifying listing
- Room Type Categorizations
- Experience Ranking + Personalization
- Host Availability
- Business Travel Classifier
- Make Listing a Space Easier
- Customer Service Ticket Routing
- … And many more

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*.

(Technology = tools, platforms, knowledge, shared feature data, etc.)

Value of ML Infrastructure

Machine Learning Infrastructure can:

- Remove incidental complexities, by providing generic, reusable solutions
- Simplify the workflow by providing tooling, libraries, and environments that make ML development more efficient

And at the same time:

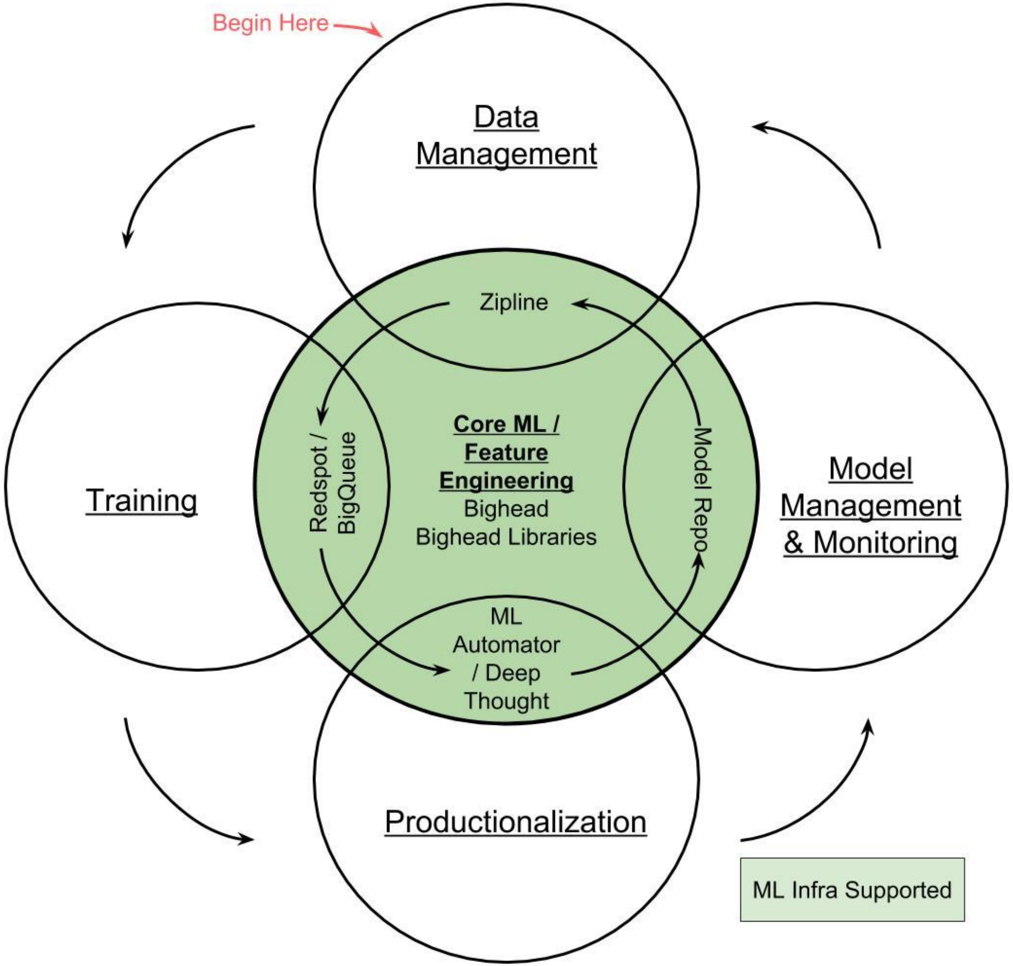
- Establish a standardized platform that enables cross-company sharing of feature data and model components
- "Make it easy to do the right thing" (ex: consistent training/streaming/scoring logic)

Bighead: Motivations

Q1 2017: Figuring out what to build

Learnings:

- No consistency between ML Workflows
- New teams struggle to begin using ML
- Airbnb has a wide variety in ML use cases
- Existing ML workflows are slow, fragmented, and brittle
- Incidental complexity vs. intrinsic complexity
- Build and forget - ML as a linear process


Begin Here
Data Management
Zipline
Training
Redspot / BigQueue
Core ML / Feature Engineering
Bighead
Bighead Libraries
Model Repo
Model Management & Monitoring
ML Automator / Deep Thought
Productionalization
ML Infra Supported

Architecture

Key Design Decisions

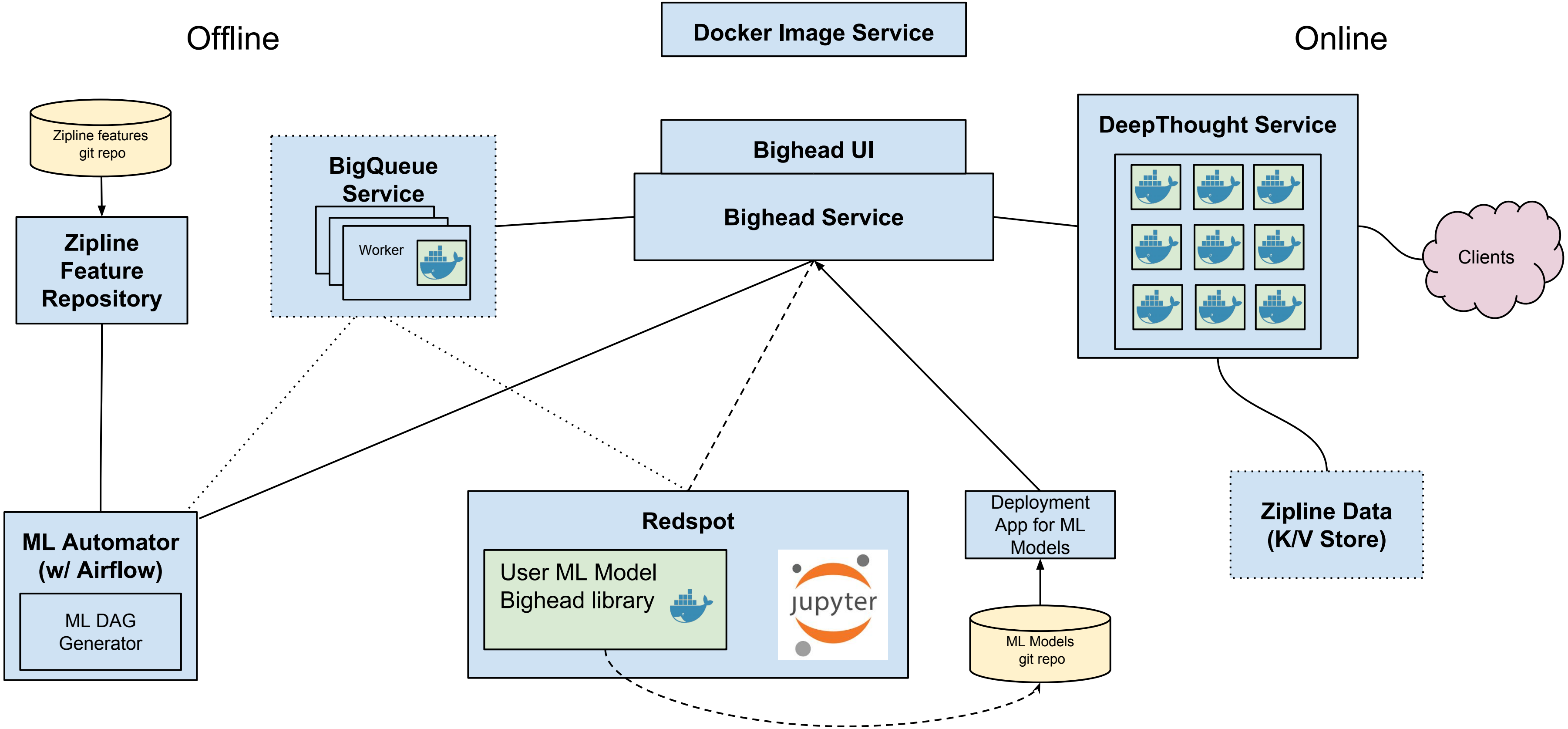
- Consistent environment across the stack with Docker

- Consistent data transformation
  - Multi-row aggregation in the warehouse, single row transformation is part of the model
  - Model transformation code is the same in online and offline

- Common workflow across different ML frameworks
  - Supports Scikit-learn, TF, PyTorch, etc.

- Modular components
  - Easy to customize parts
  - Easy to share data/pipelines

Bighead Architecture



Components

- **Data Management:** *Zipline*
- **Training:** *Redspot / BigQueue*
- **Core ML Library:** *ML Pipeline*
- **Productionisation:** *Deep Thought (online) / ML Automator (offline)*
- **Model Management**: *Bighead service*

Zipline (ML Data Management Framework)

Zipline - Why

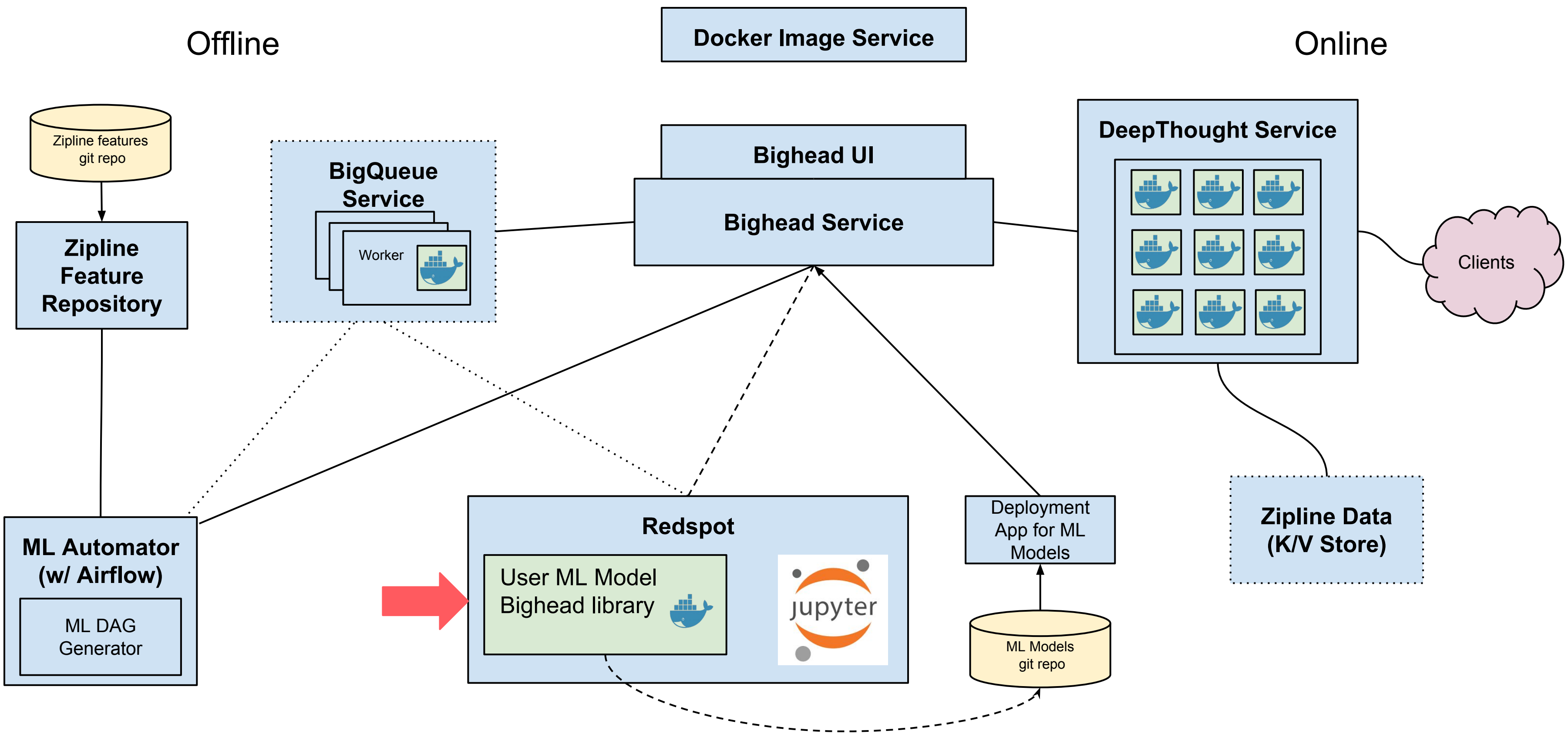
- Defining features (especially windowed) with hive was complicated and error prone
- Backfilling training sets (on inefficient hive queries) was a major bottleneck
- No feature sharing
- Inconsistent offline and online datasets
- Warehouse is built as of end-of-day, lacked point-in-time features
- ML data pipelines lacked data quality checks or monitoring
- Ownership of pipelines was in disarray

For information on Zipline, please watch the recording of our other Spark Summit session:

Zipline: Airbnb's Machine Learning Data Management Platform

Redspot (Hosted Jupyter Notebook Service)

Bighead Architecture



Redspot - Why

- Started with Jupyterhub (open-source project), which manages multiple Jupyter Notebook Servers (prototyping environment)
- But users were installing packages locally, and then creating virtualenv for other parts of our infra
    - Environment was very fragile
- Users wanted to be able to use jupyterhub on larger instances or instances with GPU
- Wanting to share notebooks with other teammates was common too
- Files/content resilient to node failures

Containerized environments

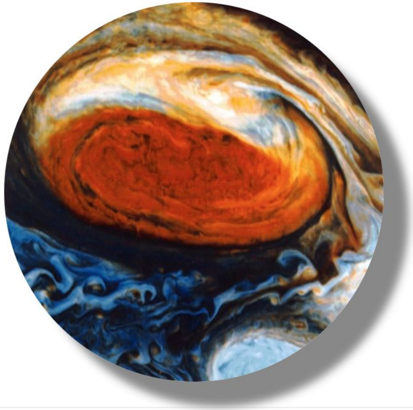
- Every user’s environment is containerized via docker
    - Allows customizing the notebook environment without affecting other users
        - e.g. install system/python packages
    - Easier to restore state therefore helps with reproducibility
- Support using custom docker images
    - Base images based on user’s needs
        - e.g. GPU access, pre-installed ML packages

Redspot

jupyter

Logout

Welcome to Redspot



RedSpot is a multitenant version of **Jupyter** (aka **iPython Notebook**)

To get access to Redspot, ask your manager to grant you ssh access to the "redspot-*" role. For more information, see the Getting Started Documentation.

Start My Server

Admin

Redspot

Choose your Jupyter environment

Select a job profile:

Containerized environment
Creates notebook server container on a shared instance. The server runs with limited CPU and memory resources. (CPU: 20, memory : 300G) Currently does not support GPU.
Remote Docker
Creates notebook server container on a dedicated instance. Takes about 40 minutes to start a new instance. Supports connecting to GPU if you spin up instances with NVIDIA GPU (p2, p3).

Containerized environment

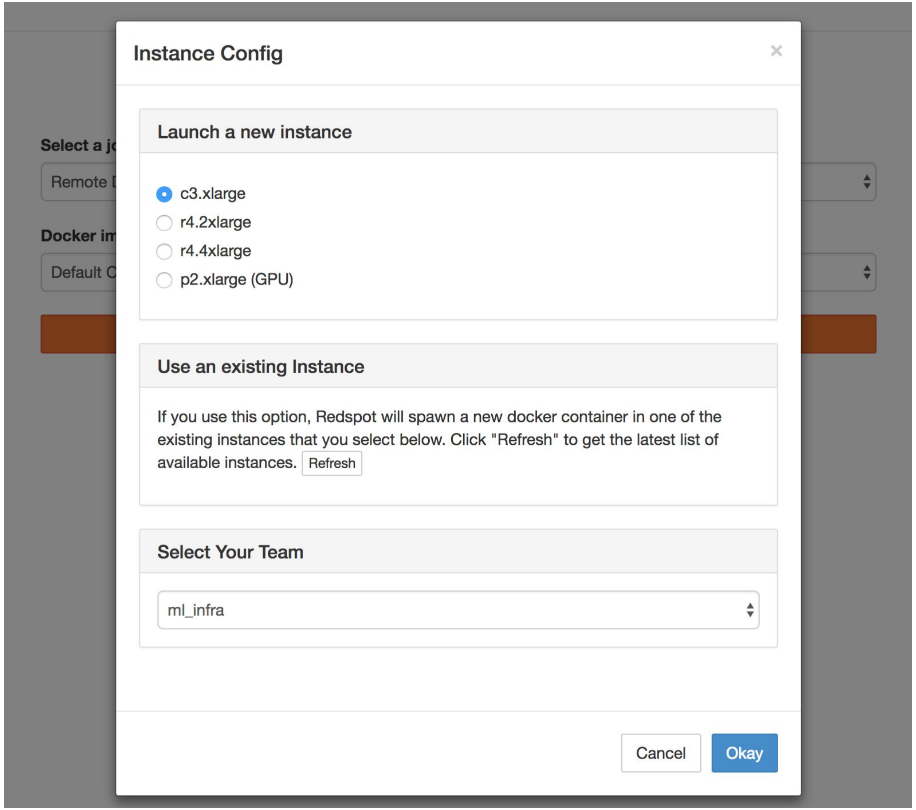
Docker image configuration:

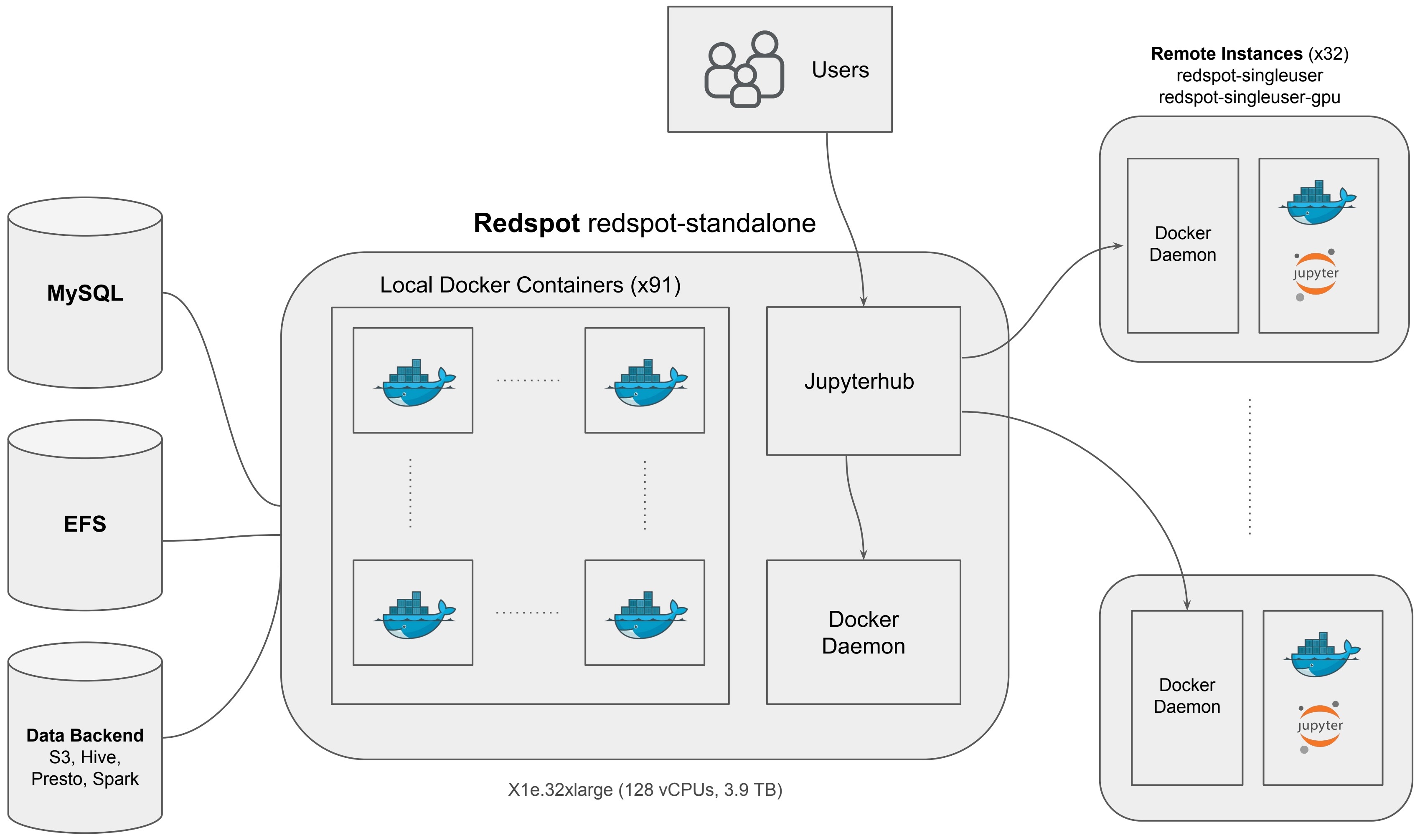
Ubuntu 14.04 image with python 2.7 (for CPU)

Launch

Remote Instance Spawner

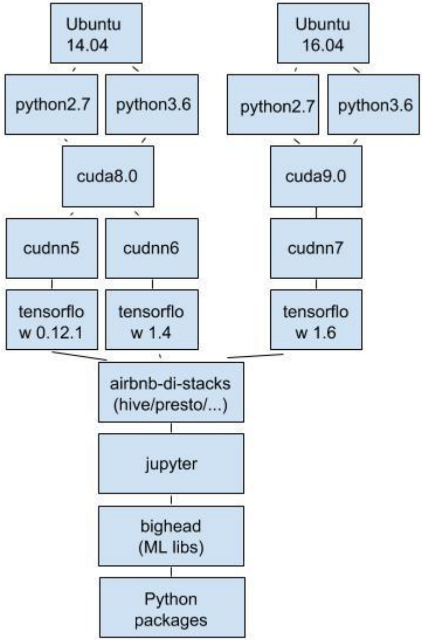
- For bigger jobs and total isolation, Redspot allows launching a dedicated instance
- Hardware resources not shared with other users
- Automatically terminates idle instances periodically




Users
Remote Instances (x32)
redspot-singleuser
redspot-singleuser-gpu
Redspot redspot-standalone
MySQL
Local Docker Containers (x91)
Jupyterhub
Docker
Daemon
jupyter
EFS
Data Backend
S3, Hive,
Presto, Spark
Docker
Daemon
X1e.32xlarge (128 vCPUs, 3.9 TB)

Docker Image Repo/Service

- Native dockerfile enforces strict single inheritance.
  - Prevents composition of base images
  - Might lead to copy/pasting Dockerfile snippets around.
- A git repo of Dockerfiles for each stage and yml file expressing:
  - Pre-build/post-build commands.
  - Build time/runtime dependencies. (mounting directories, docker runtime)
- Image builder:
  - Build flow tool for chaining stages to produce a single image.
  - Build independent images in parallel.




```
ubuntu16.04-py3.6-cuda9-cudnn7:
  base: ubuntu14.04-py3.6
  description: "A base Ubuntu 16.04 image with
python 3.6, CUDA 9, CUDNN 7"
  stages:
    - cuda/9.0
    - cudnn/7
  args:
    python_version: '3.6'
    cuda_version: '9.0'
    cudnn_version: '7.0'
```

Redspot Summary

- A multi-tenant jupyter notebook environment
- Makes it easy to iterate and prototype ML models, share work
  - Integrated with the rest of our infra - so one can deploy a notebook to prod
- Improved upon open source Jupyterhub
  - Containerized; can bring custom Docker env
  - Remote notebook spawner for dedicated instances (P3 and X1 machines on AWS)
  - Persist notebooks in EFS and share with teams
  - Reverting to prior checkpoint
- Support 200+ Weekly Active Users

Bighead Library

Bighead Library - Why

- Transformations (NLP, images) are often re-written by different users
- No clear abstraction for data transformation in the model
  - Every user can do data processing in a different way, leading to confusion
  - Users can easily write inefficient code
  - Loss of feature metadata during transformation (can’t plot feature importance)
  - Need special handling of CPU/GPU
  - No visualization of transformations
- Visualizing and understanding input data is key
  - But few good libraries to do so

Bighead Library

- Library of transformations; holds more than 100+ different transformations including automated preprocessing for common input formats (NLP, images, etc.)
- Pipeline abstraction to build a DAG of transformation on input data
  - Propagate feature metadata so we can plot feature importance at the end and connect it to feature names
  - Pipelines for data processing are reusable in other pipelines
  - Feature parallel and data parallel transformations
  - CPU and GPU support
  - Supports Scikit APIs
- Wrappers for model frameworks (XGB, TF, etc.) so they can be easily serialized/deserialized (robust to minor version changes)
- Provides training data visualization to help identify data issues

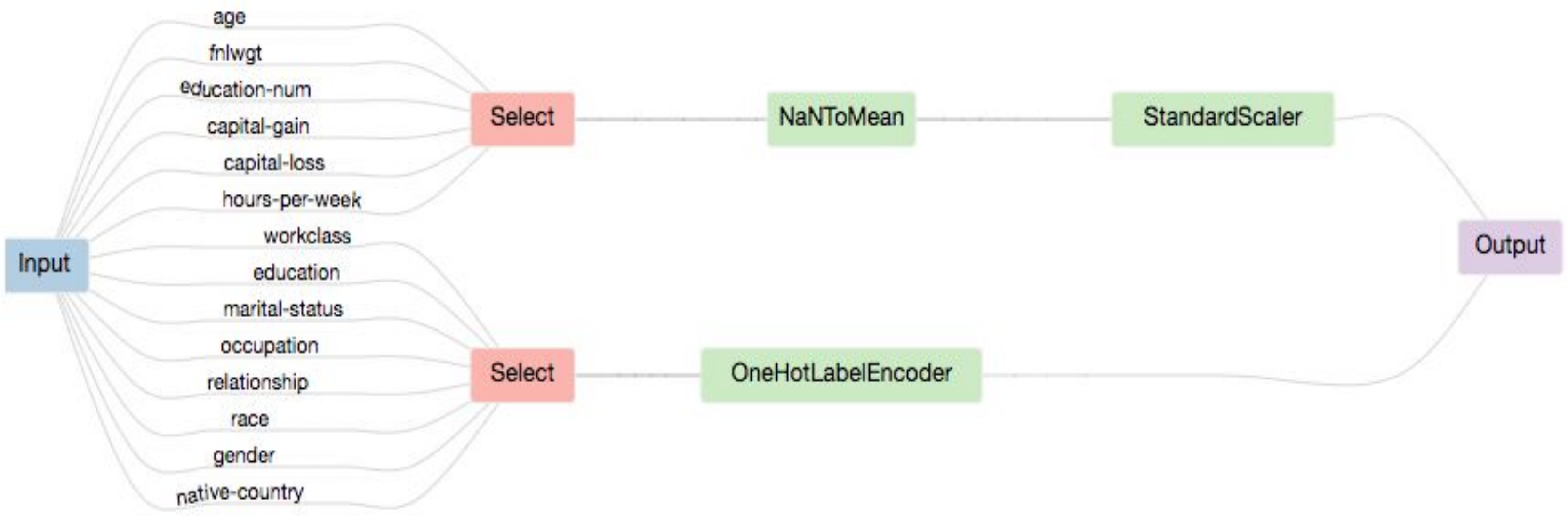
Bighead Library: ML Pipeline

```python
def create_pipeline():
    # create a list of features and organize by type
    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('ClassifyCensusIncomeSerial')
    p = p[numeric] >> [NaNToMean(dtype=np.float32), StandardScaler()]
    p = p[categorical] >> OneHotLabelEncoder()
    p >>= XGBClassifier(objective='binary:logistic',
                        n_estimators=100,
                        learning_rate=0.1,
                        max_depth=5
                        )

    return p
```

ML Pipeline Serial Transformation Visualization

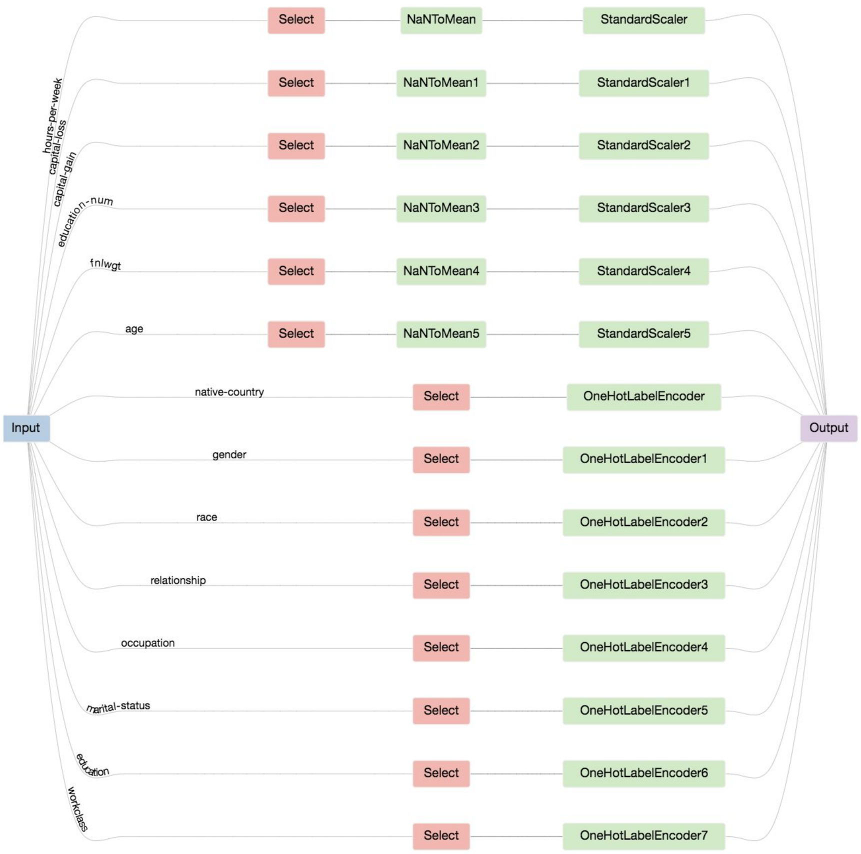
In [3]:
```
# serial transformation
p = Pipeline('ClassifyCensusIncomeSerial')
p = p[numeric] >> [NaNToMean(dtype=np.float32), StandardScaler()]
p = p[categorical] >> OneHotLabelEncoder()
p
```

Out[3]:



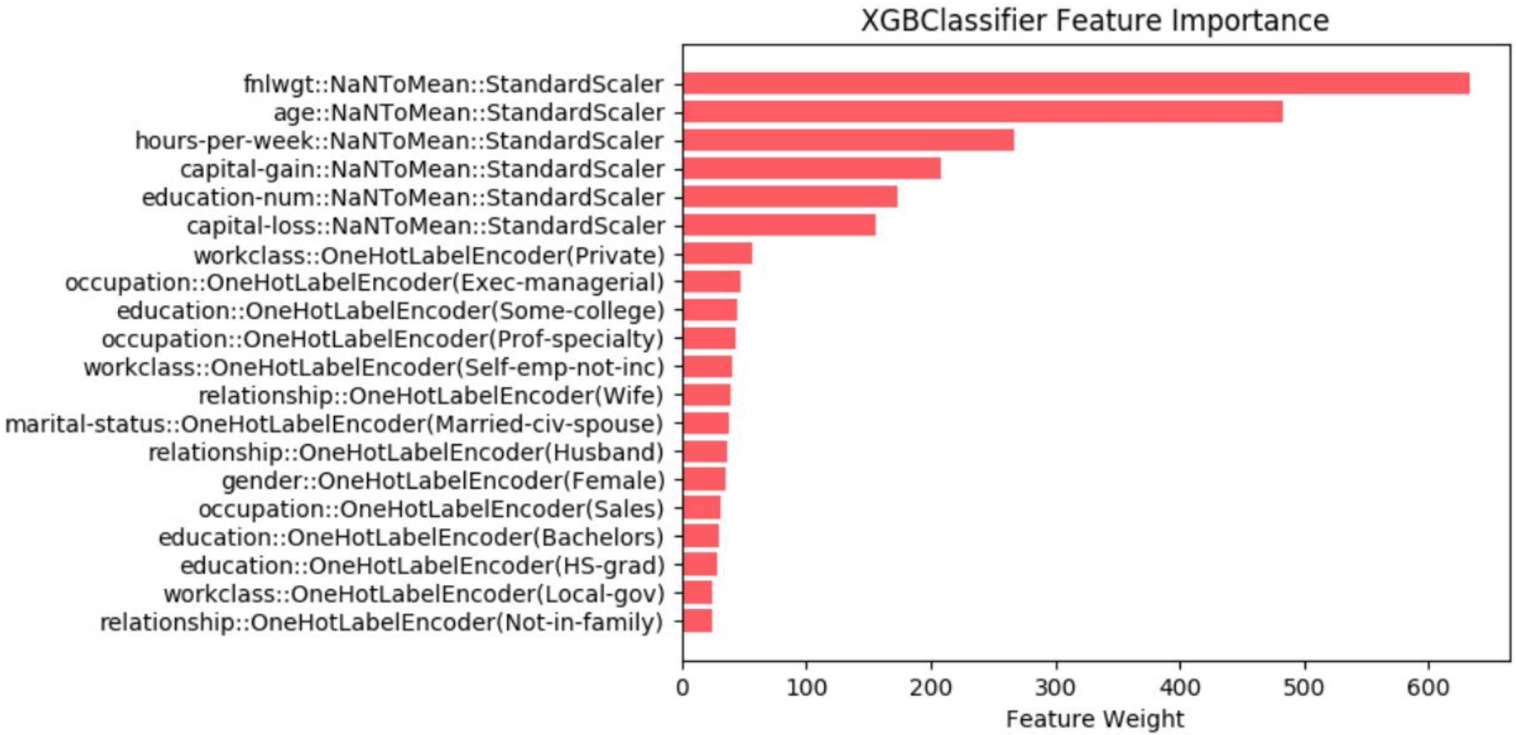
ML Parallel Visualization

In [5]:
```
# parallel transformation
p = Pipeline('ClassifyCensusIncome')
p = p.parallel()[numeric] >> [NaNToMean(dtype=np.float32),StandardScaler()]
p = p.parallel()[categorical] >> OneHotLabelEncoder()
p
```



Feature Importance: Metadata Preserved Through Transformations

In [10]:
```
p.get_transformer('XGBClassifier').plot_feature_importance()
```



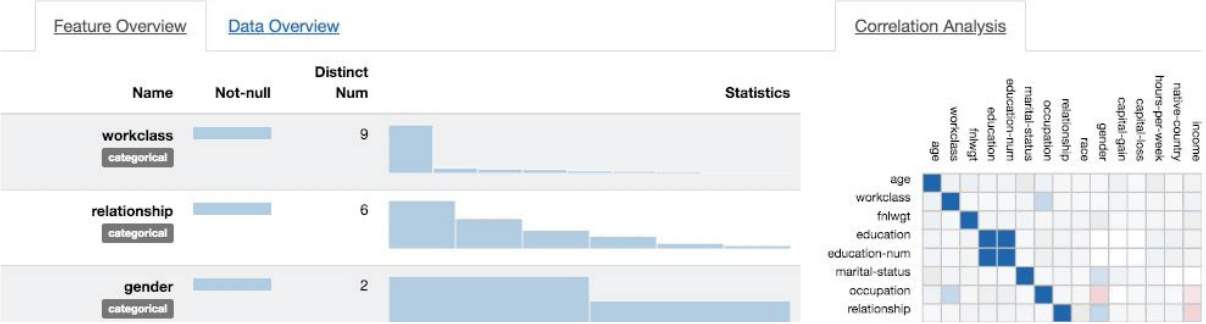
Easy to Serialize/Deserialize

```
In [11]: p.serialize('test.tar.xz')

In [12]: p2 = Pipeline.deserialize('test.tar.xz')
```

Training Data - Visualization

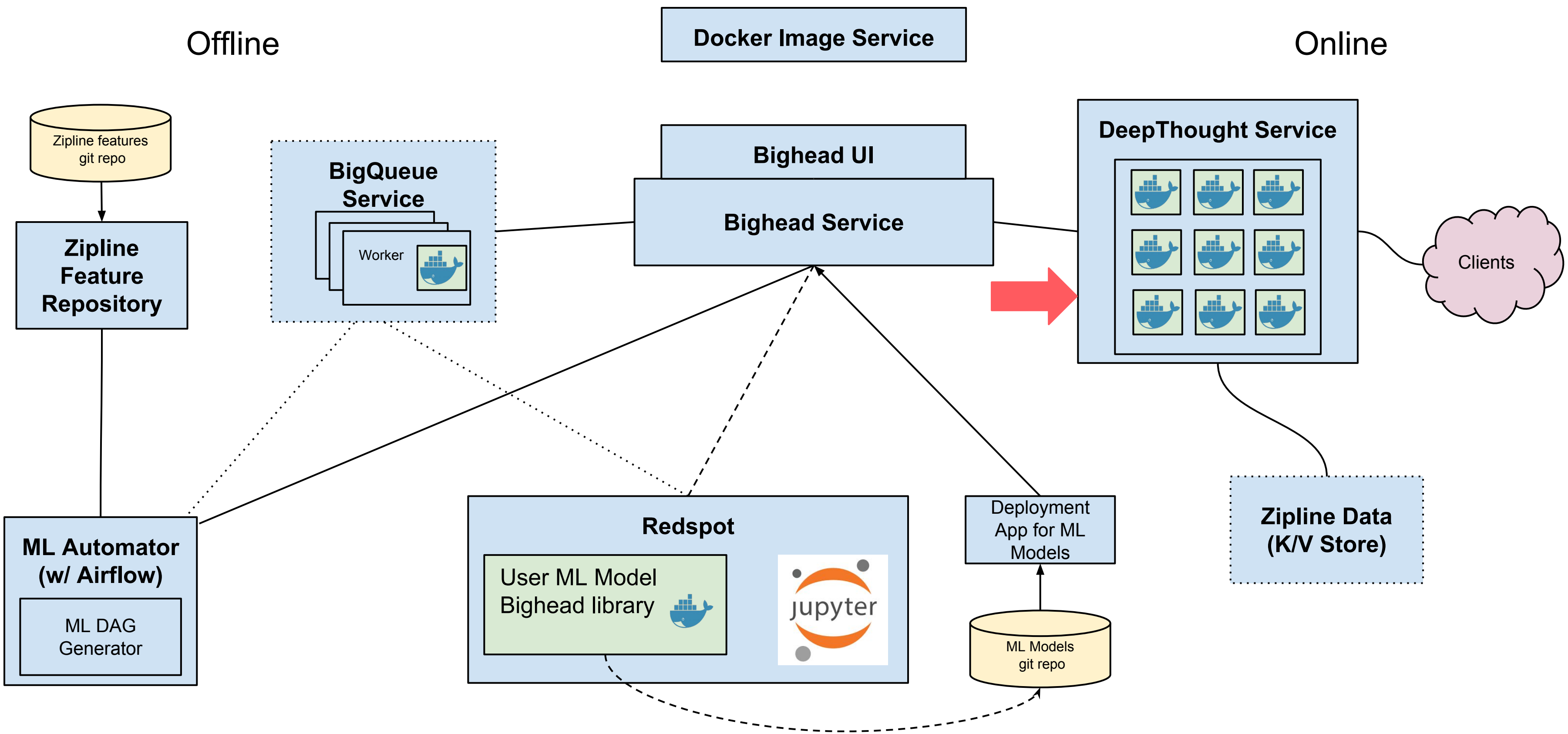
In [2]:
```python
train, test = load_census_income()
categorical = [
    'workclass',
    'education',
    'marital-status',
    'occupation',
    'relationship',
    'race',
    'gender',
    'native-country',
]
numeric = [
    'age',
    'fnlwgt',
    'education-num',
    'capital-gain',
    'capital-loss',
    '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)
```



Productionisation:
Deep Thought (Online Inference Service)

Bighead Architecture

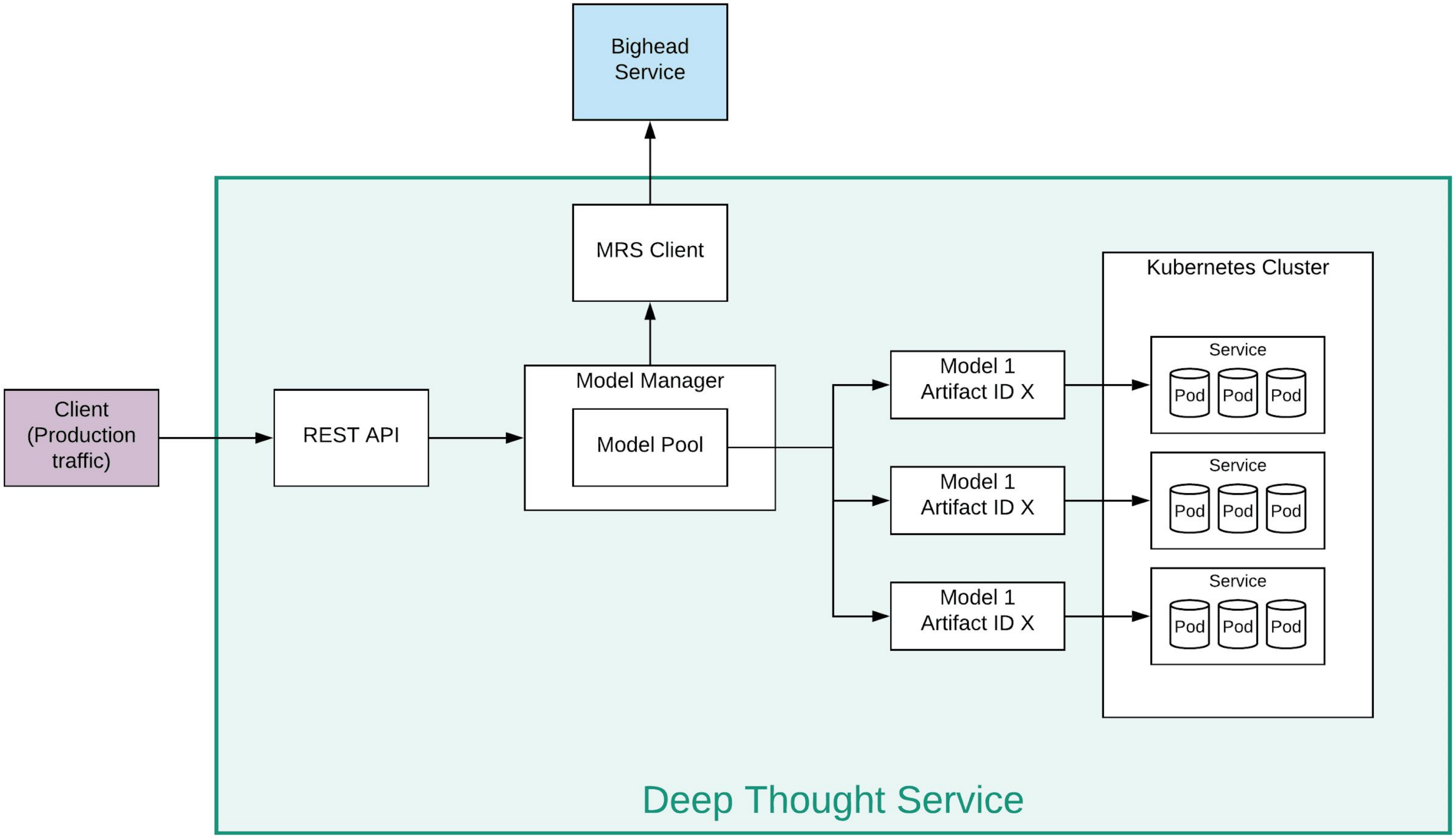


Deep Thought - Why

- **Performant, scalable execution of model inference in production is hard**
  - Engineers shouldn't build one off solutions for every model.
  - Data scientists should be able to launch new models in production with minimal eng involvement.
- **Debugging differences between online inference and training are difficult**
  - We should support the exact serialized version of the model the data scientist built
  - We should be able to run the same python transformations data scientists write for training.
  - We should be able to load data computed in the warehouse or streaming easily into online scoring.

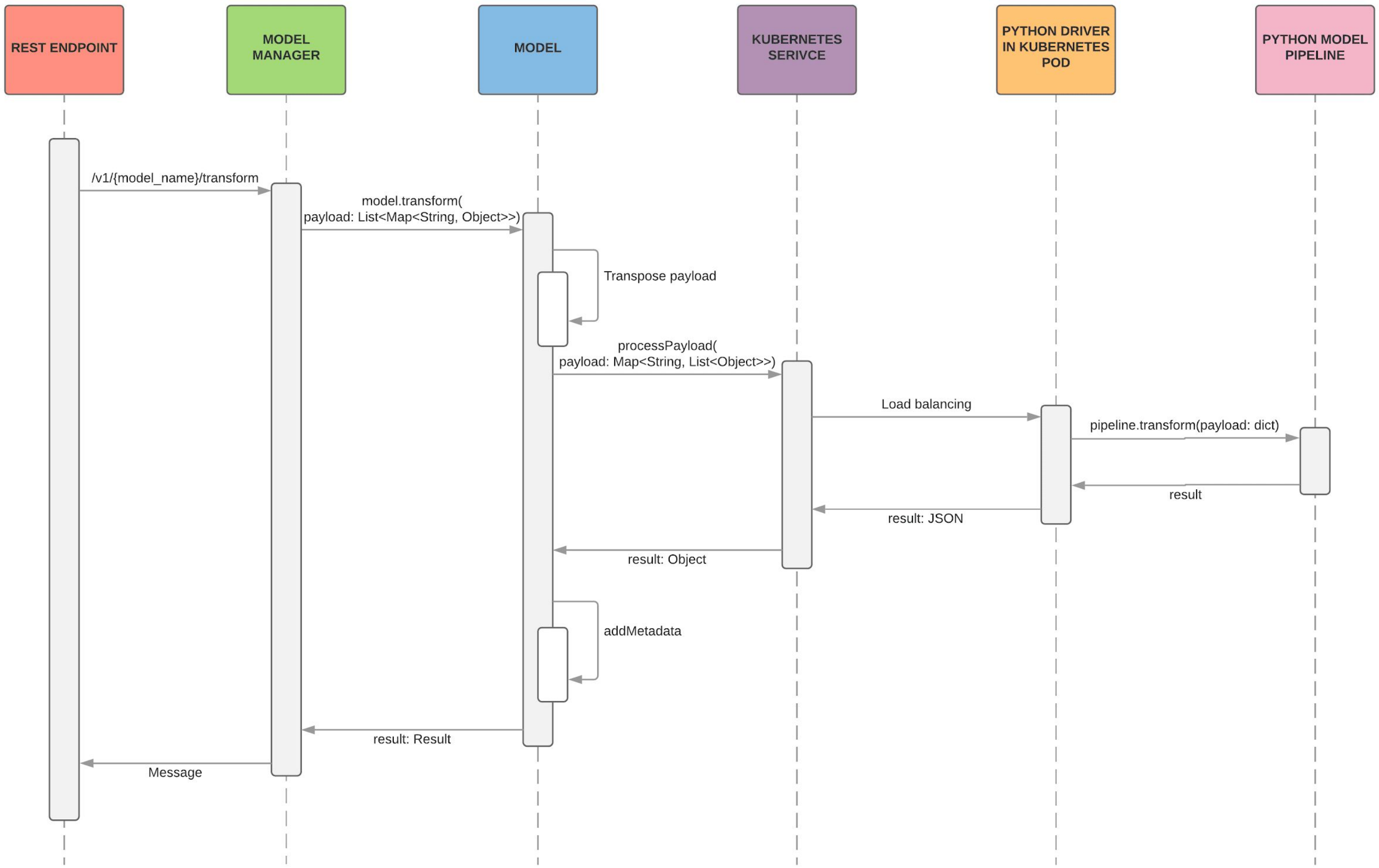
Deep Thought - How

- **Deep Thought is a shared service for online inference**
  - Supports all frameworks integrated in ML Pipeline
  - Deployment is completely config driven so data scientists don't have to involve engineers to launch new models.
  - Engineers can then connect to a REST API from other services to get scores.
  - Support for loading data from K/V stores
  - Standardized logging, alerting and dashboarding for monitoring and offline analysis of model performance
  - Isolation to enable multi-tenancy
  - Scalable and Reliable: 100+ models. Highest QPS service at Airbnb. Median response time: 4ms. p95: 13ms.


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

PYTHON MODEL WITH DOCKER CONTAINER

April 21, 2018



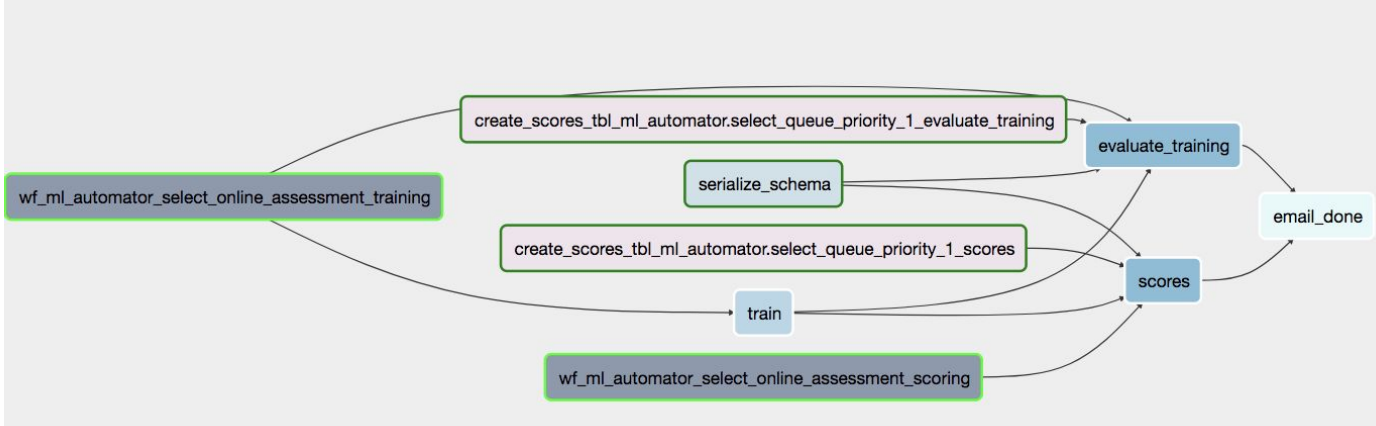
Productionisation: ML Automator (Offline Training and Inference Service)

ML Automator - Why

- **Tools and services to automate common (mundane) tasks**
  - Periodic training, evaluation and offline scoring of model
    - Get specified amount of resources (GPU/memory) for these tasks
  - Uploading scores to K/V stores
  - Dashboards on scores, alert on score changes
  - Orchestration of these tasks (via Airflow in our case),
  - Scoring on large tables is tricky to scale

ML Automator

- Automate such tasks via configuration
  - We generate Airflow DAGs automatically to train, score, evaluate, upload scores, etc. with appropriate resources
- Built custom tools to train/score on Spark for large datasets
  - Tools to get training data to the training machine quickly
  - Tool to generate virtualenv (that's equivalent to a specified docker image) and run executors in it as (our version of) Yarn doesn't run executors in a docker image



Bighead Service

Bighead Service - Why

- **Model management is hard**
  - Need a single source of truth to track the model history
- **Models reproducibility is hard**
  - Models are trained on developer laptops and put into production
  - Model artifact isn't tagged with model code git sha
- **Model monitoring is done in many places**
  - Evaluation metrics are stored in ipython notebooks
  - Online scores and model features are monitored separately
  - Online/offline scoring may not be consistent version

Bighead Service

Overview

Bighead's model management service

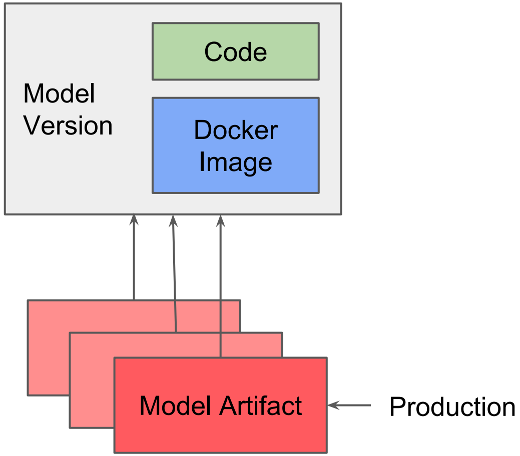
- Contains prototype and production models
- Can serve models "raw" or trained
- The source of truth on which trained models are in production
- Stores model health data

Bighead Service

Internals

We decompose Models into two components:

- Model Version - raw model code + docker image
- Model Artifact - parameters learned via training



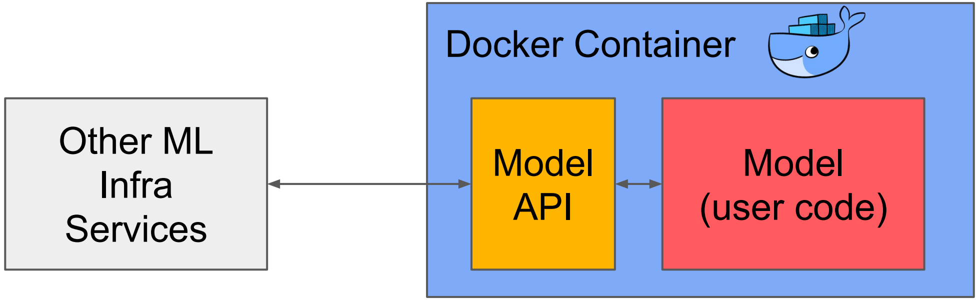
A trained model consists of:

Model Version
+
Model Artifact

Dockerized Models

ML models have diverse dependency sets (tensorflow, xgboost, etc.). We allow users to provide a docker image within which model code *always* runs.

ML models don't run in isolation however, so we've built a lightweight API to interact with the "dockerized model"

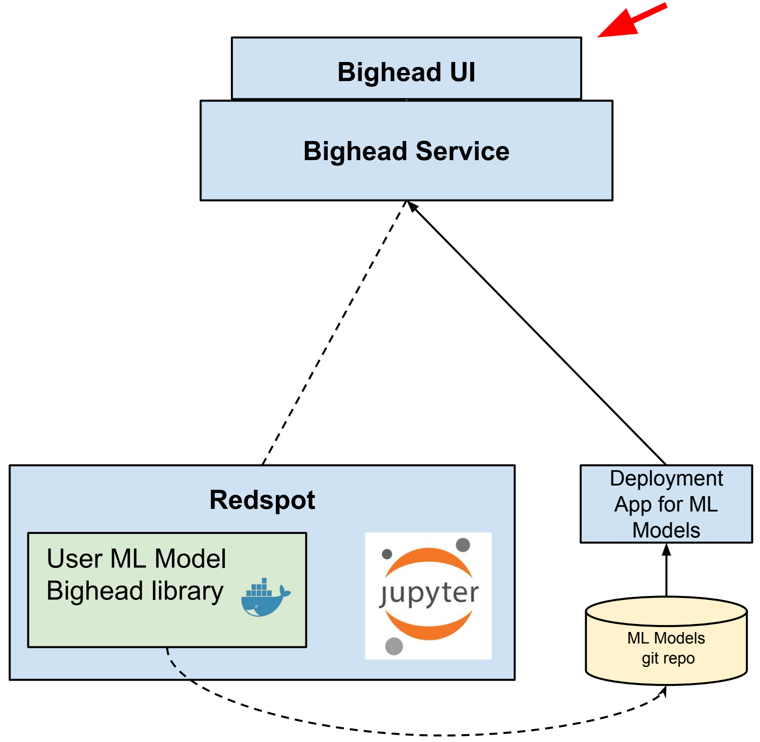


Bighead: UI

Our built-in UI provides:

- Deployment - review changes, deploy, and rollback trained models
- Model Health - metrics, visualizations, alerting, central dashboard
- Experimentation - Ability to setup model experiments - e.g. split traffic between two or more models

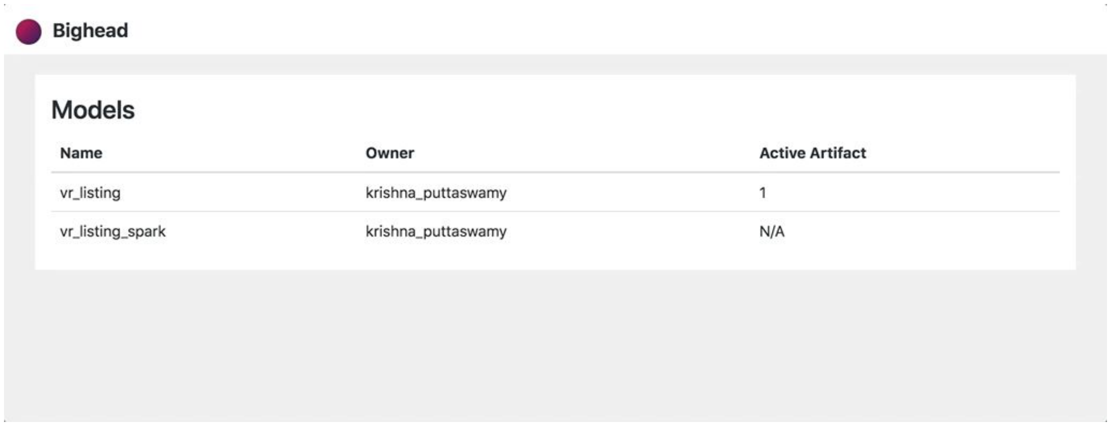
Bighead Architecture

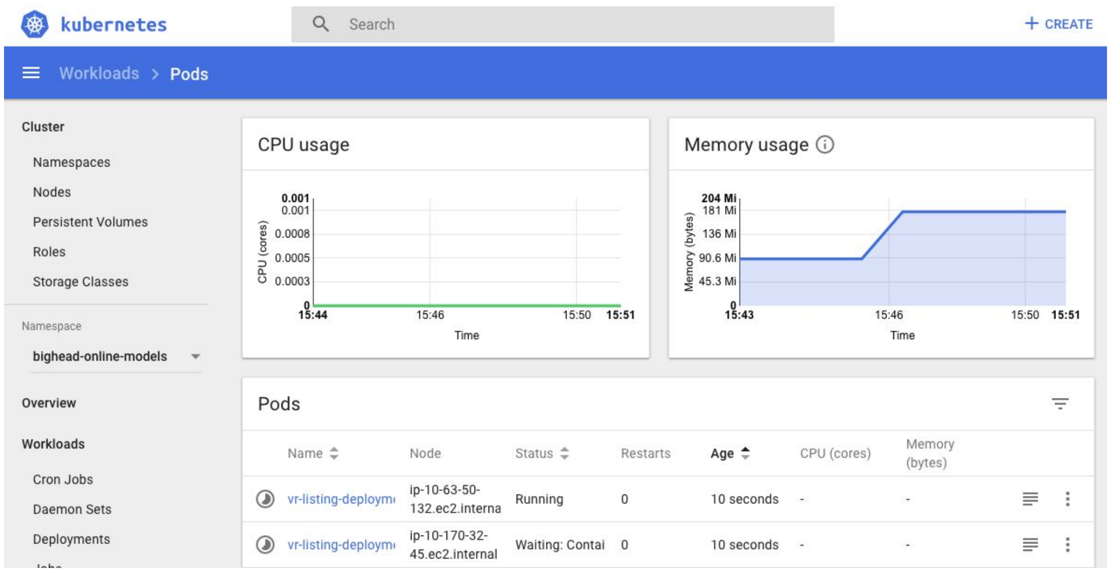


Bighead Service/UI

- Bighead's central model management service

- The "Deployboard" for *trained* ML models - i.e. the sole source of truth about what model is deployed





Bighead Summary

- End-to-End platform to build and deploy ML models to production
- Built on open source technology
  - Spark, Jupyter, Kubernetes, Scikit-learn, TF, XGBoost, etc.
- But had to fix various gaps in the path to productionisation
  - Generic online and offline inference service (that supports different frameworks)
  - Feature generation and management framework
  - Model data transformation library
  - Model and data visualization libraries
  - Docker image customization service
  - Multi-tenant training environment

Plan to Open Source Soon

If you want to collaborate come talk to

andrew.hoh@airbnb.com