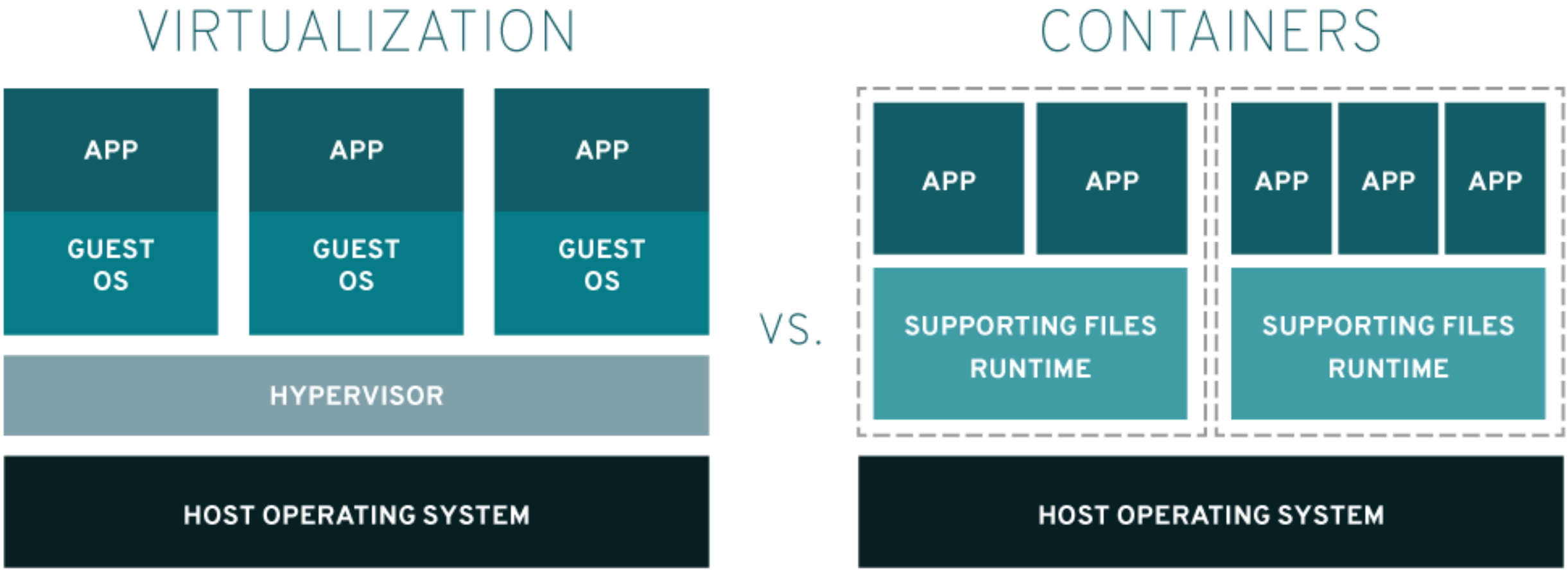
Introduction to, using and use cases of KubeFlow

Jonathan Gershater & Boris Lublinsky

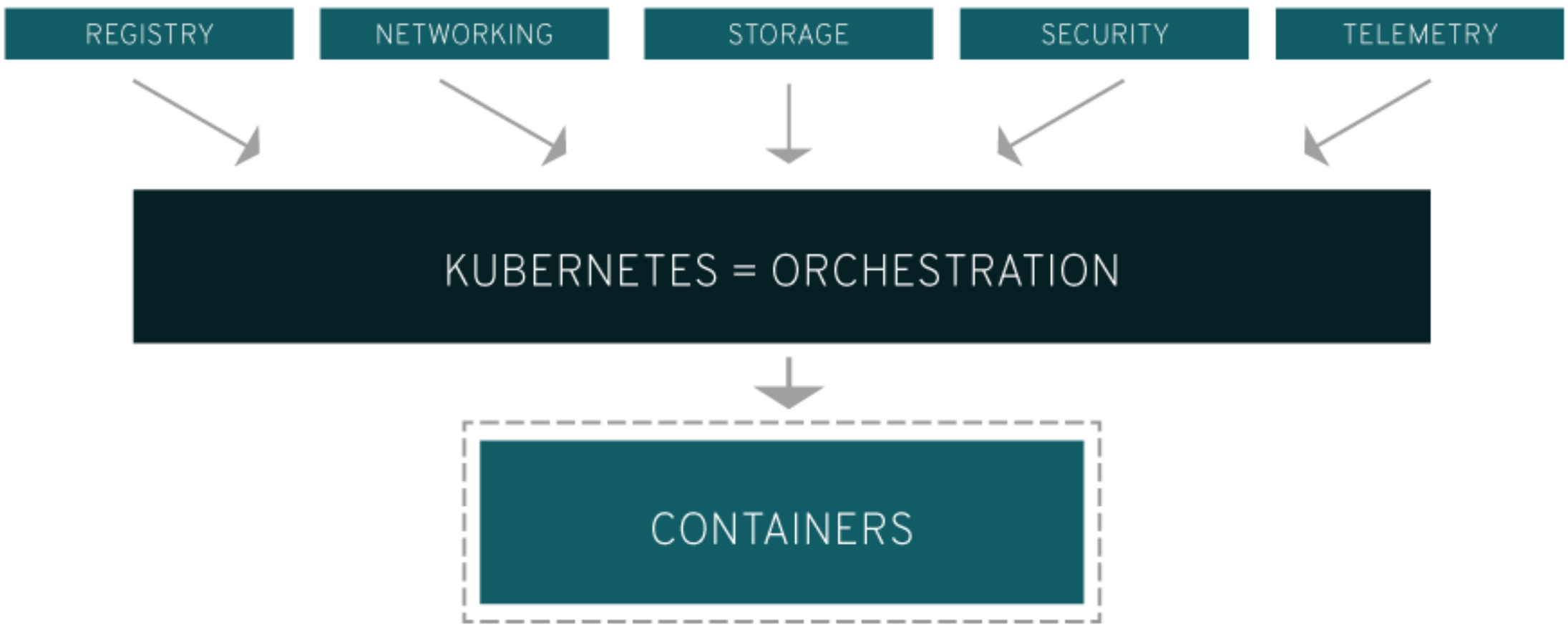
Agenda

- What is Kubernetes
- What is Kubeflow
- Workflow components
- End-to-end example

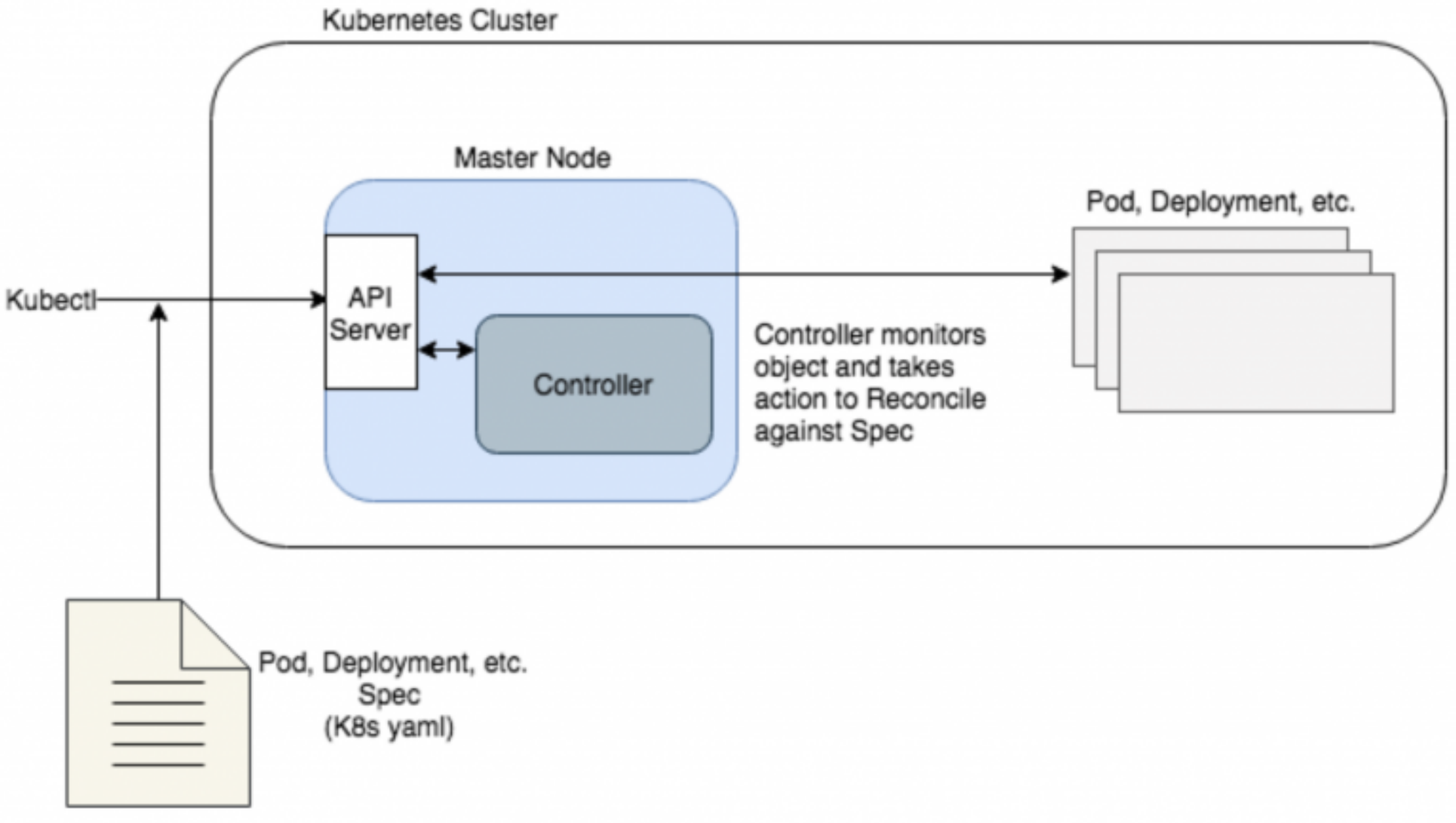
What are containers?



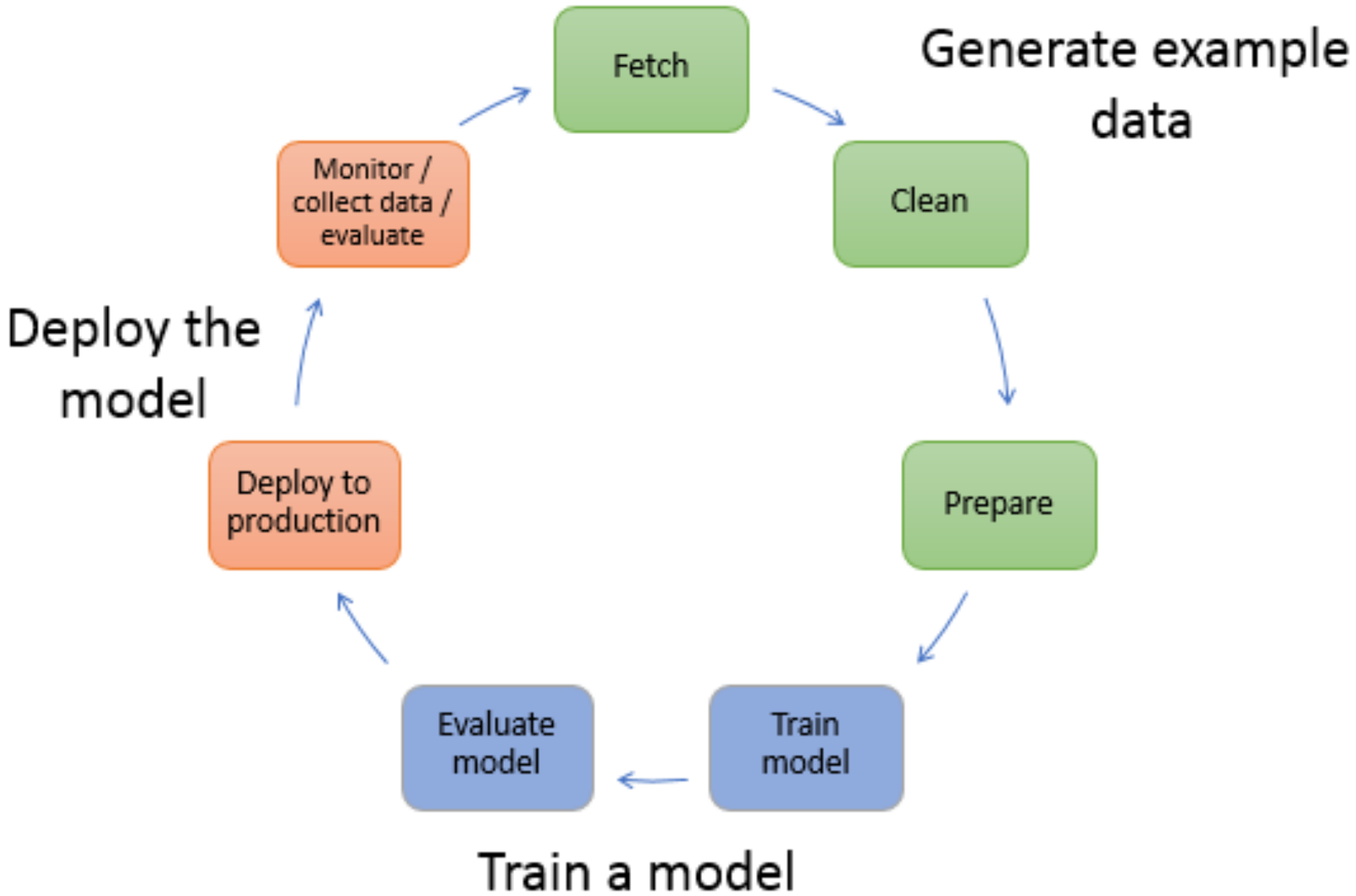
What is kubernetes



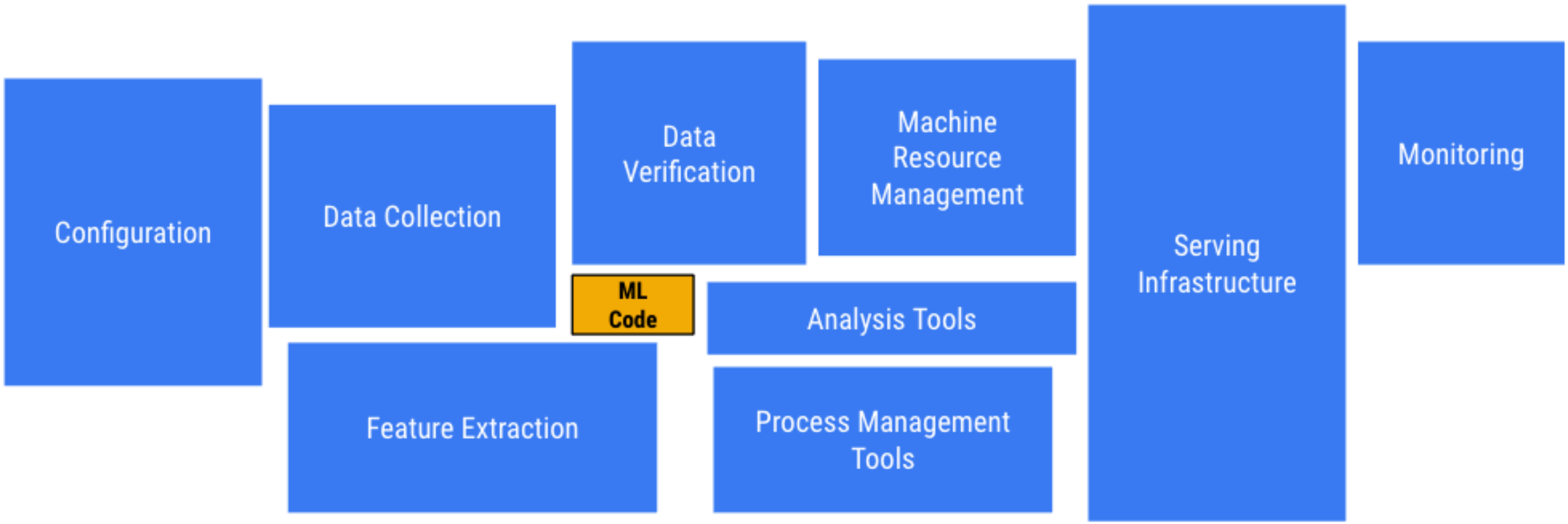
Kubernetes desired state management



What is machine learning?



Challenging of machine learning in production



Source *https://medium.com/kubeflow/why-kubeflow-in-your-infrastructure-56b8fabf1f3e*

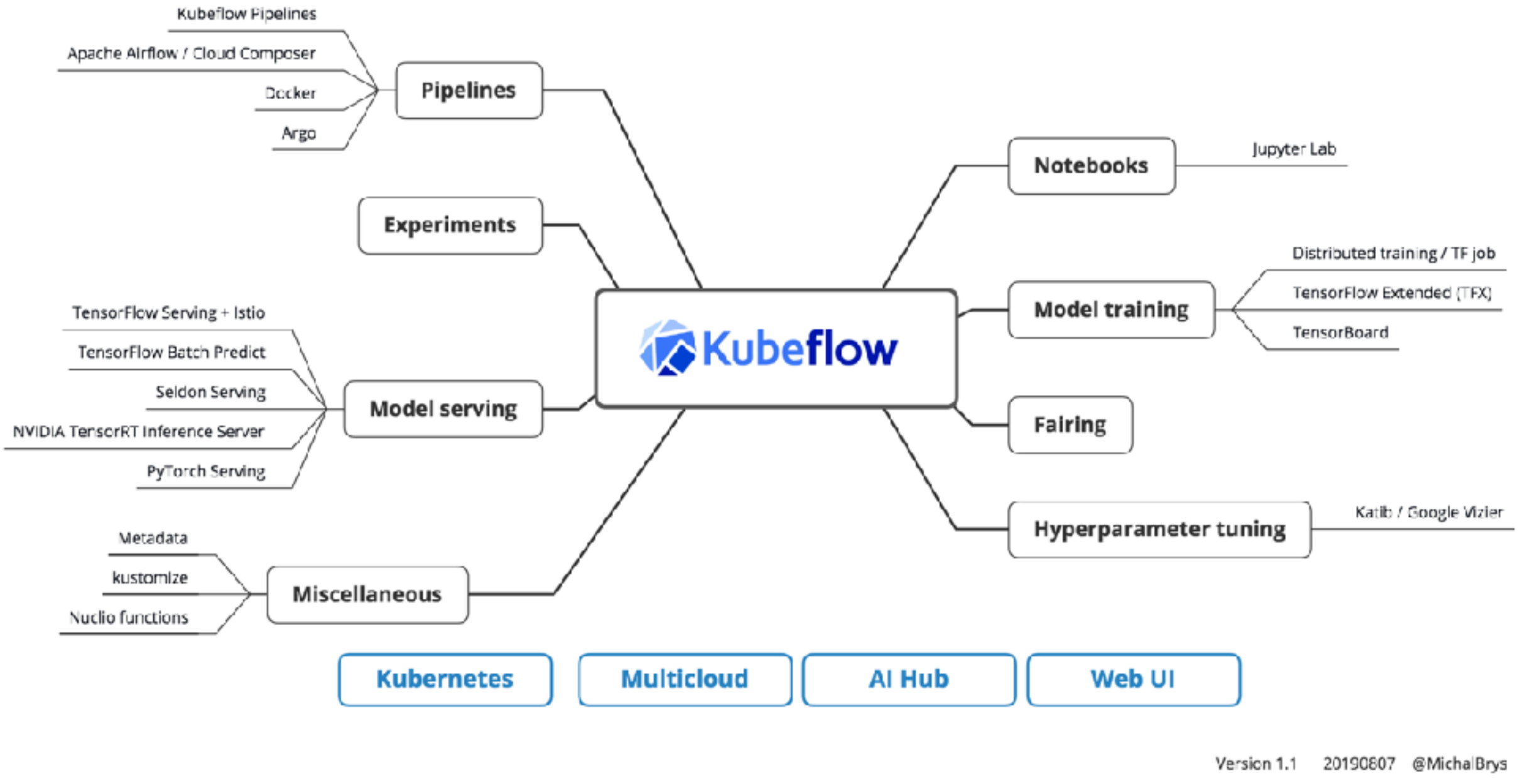
What is kubeflow?

Make deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable.

Provide a straightforward way to deploy open-source systems for ML to diverse infrastructures.

Anywhere you are running Kubernetes, you should be able to run Kubeflow.

Kubeflow components



Source https://medium.com/@michal.brys/kubeflow-a-machine-learning-toolkit-for-kubernetes-d8686f6c91b6

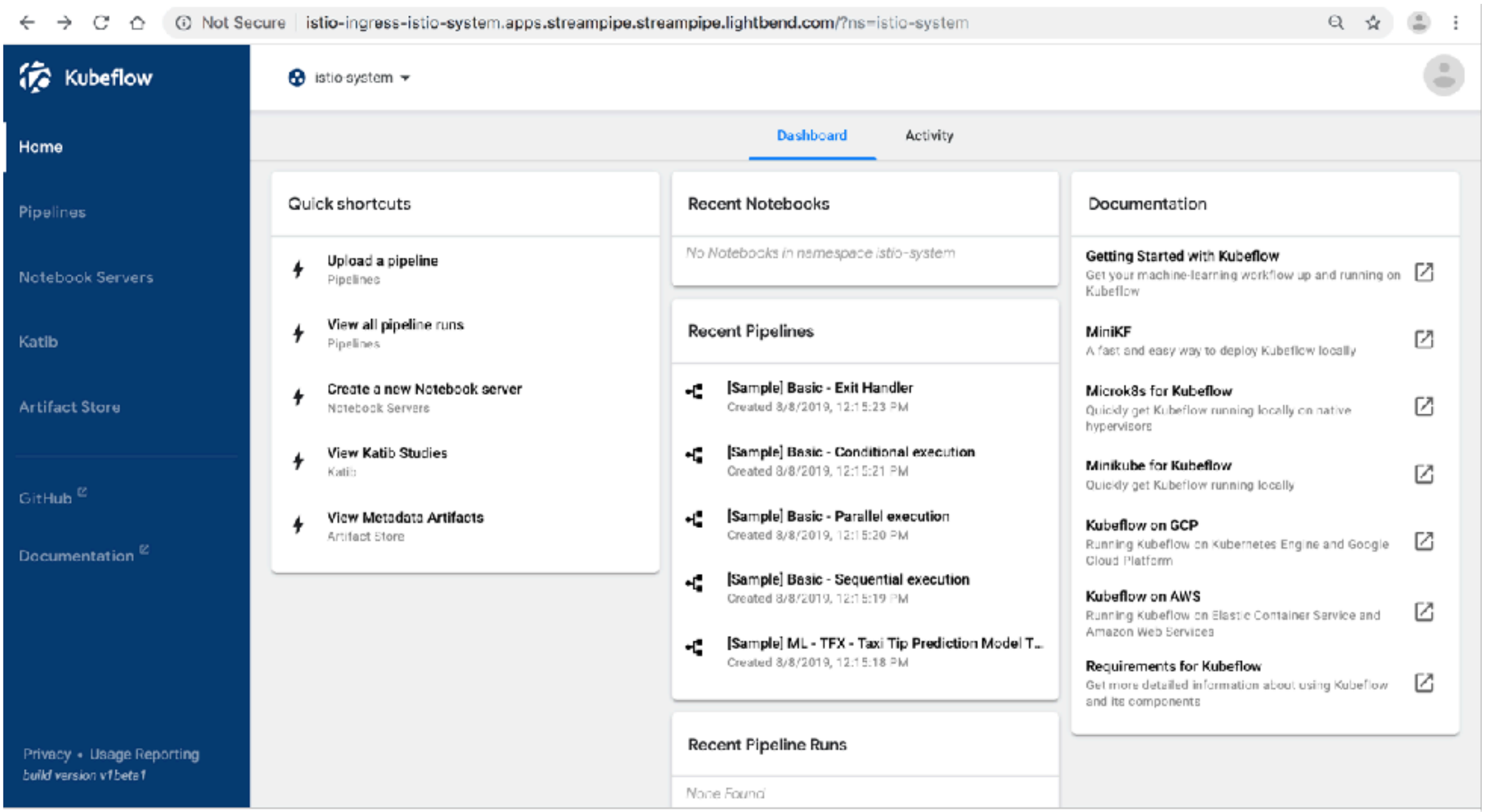
Scalability

- Machine learning (deep learning) today is a result of the larger scale and capacity available in the cloud.
- And various machine types and hardware-specific accelerators (e.g., graphics processing units/Tensor processing units),
- And data locality for improved performance.
- Scale teams through collaboration and simplify the running of a large number of experiments.

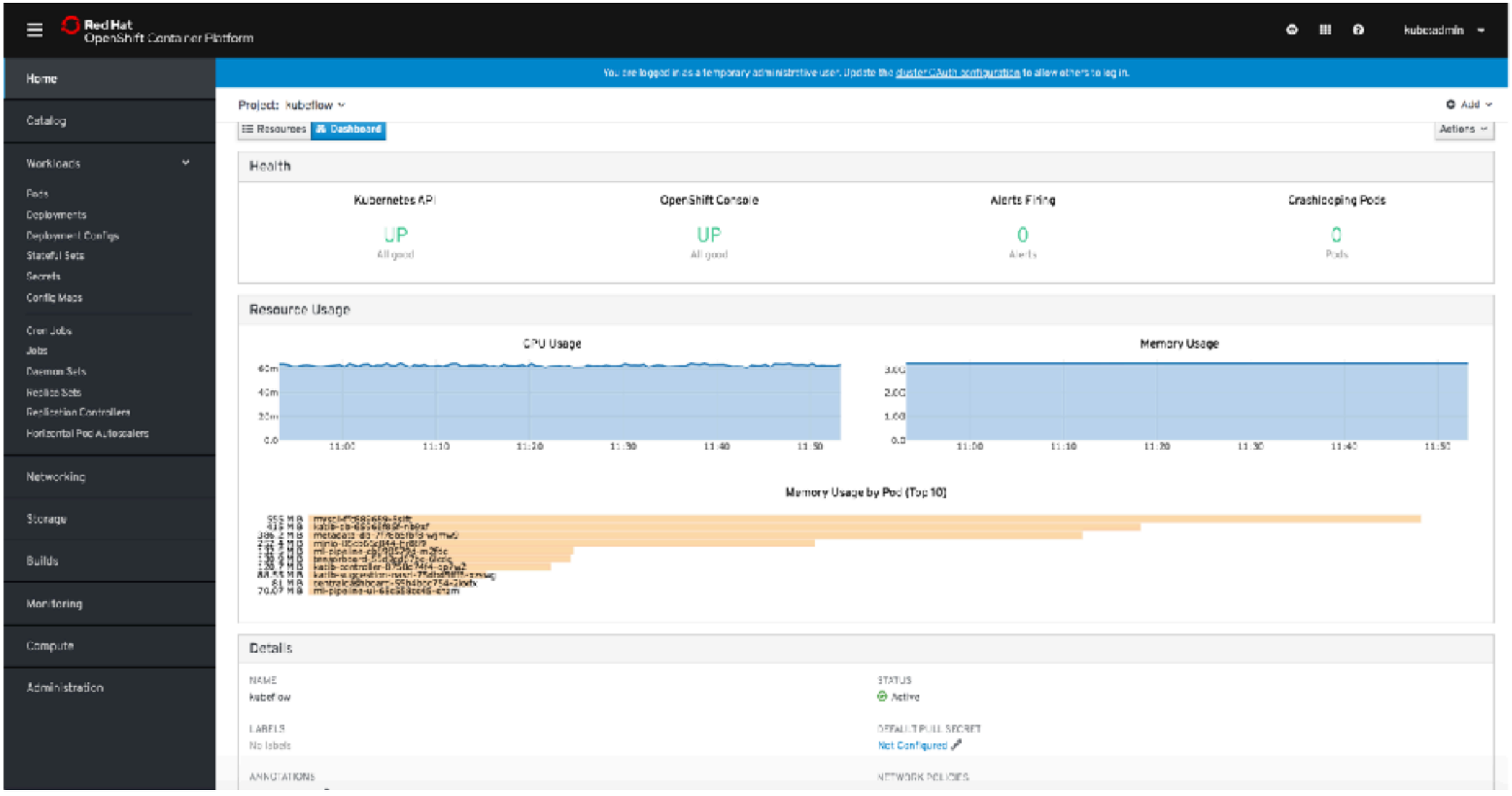
What’s new in 0.6?

- Secure architecture for multi-user support by leveraging a new integration with Istio
- Extensions to the Kubeflow Pipelines’ Domain Specific Language (DSL) by adding new primitives that enable data and execution context versioning in Kubeflow Pipelines
- New metadata component, along with a metadata API and initial corresponding clients. These allow users to track their artifacts and execution contexts through an end-to-end ML workflow.
- Several documentation updates and valuable new operational and configuration capabilities, most notably the introduction of Kustomize as a complete replacement of ksonnet.

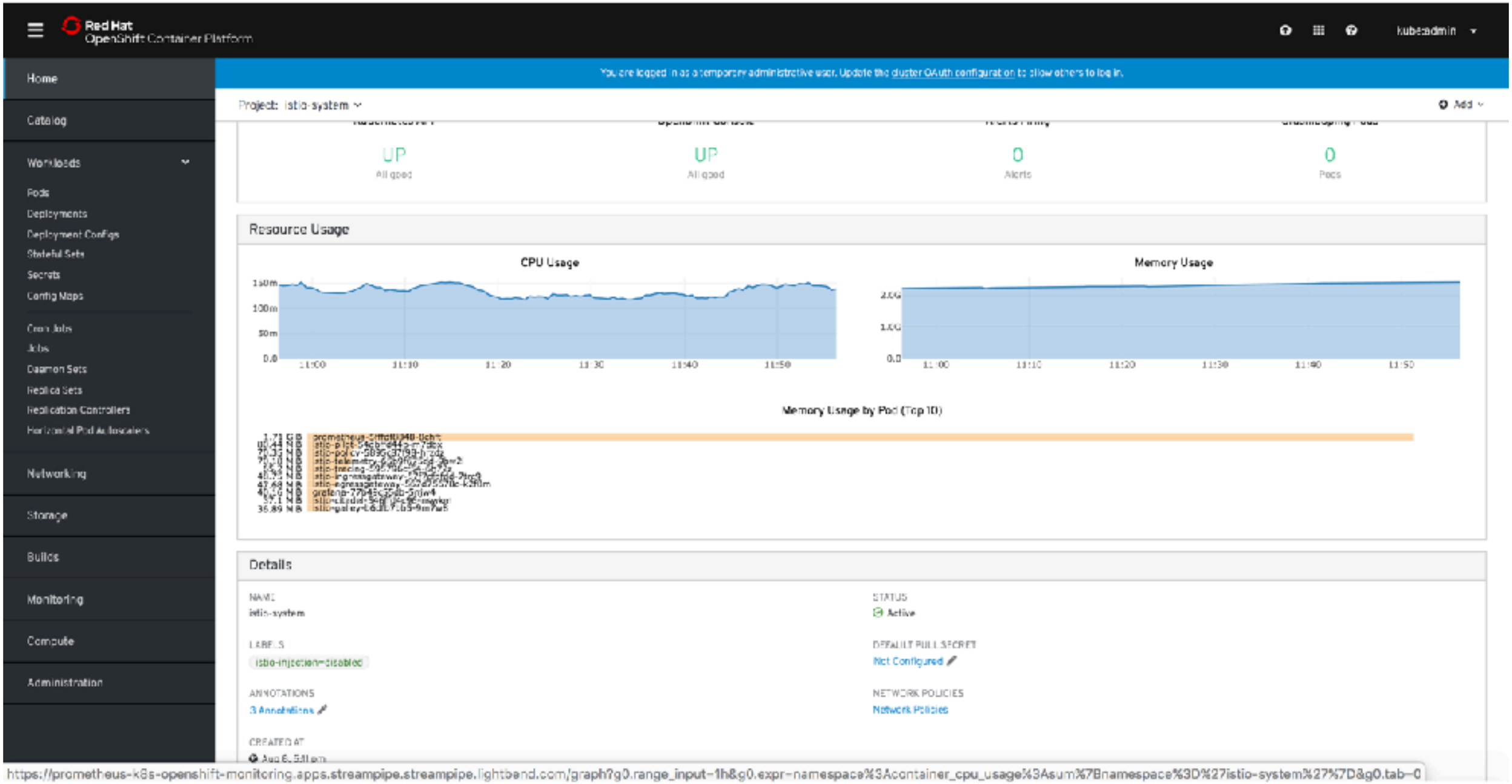
Meet Kubeflow



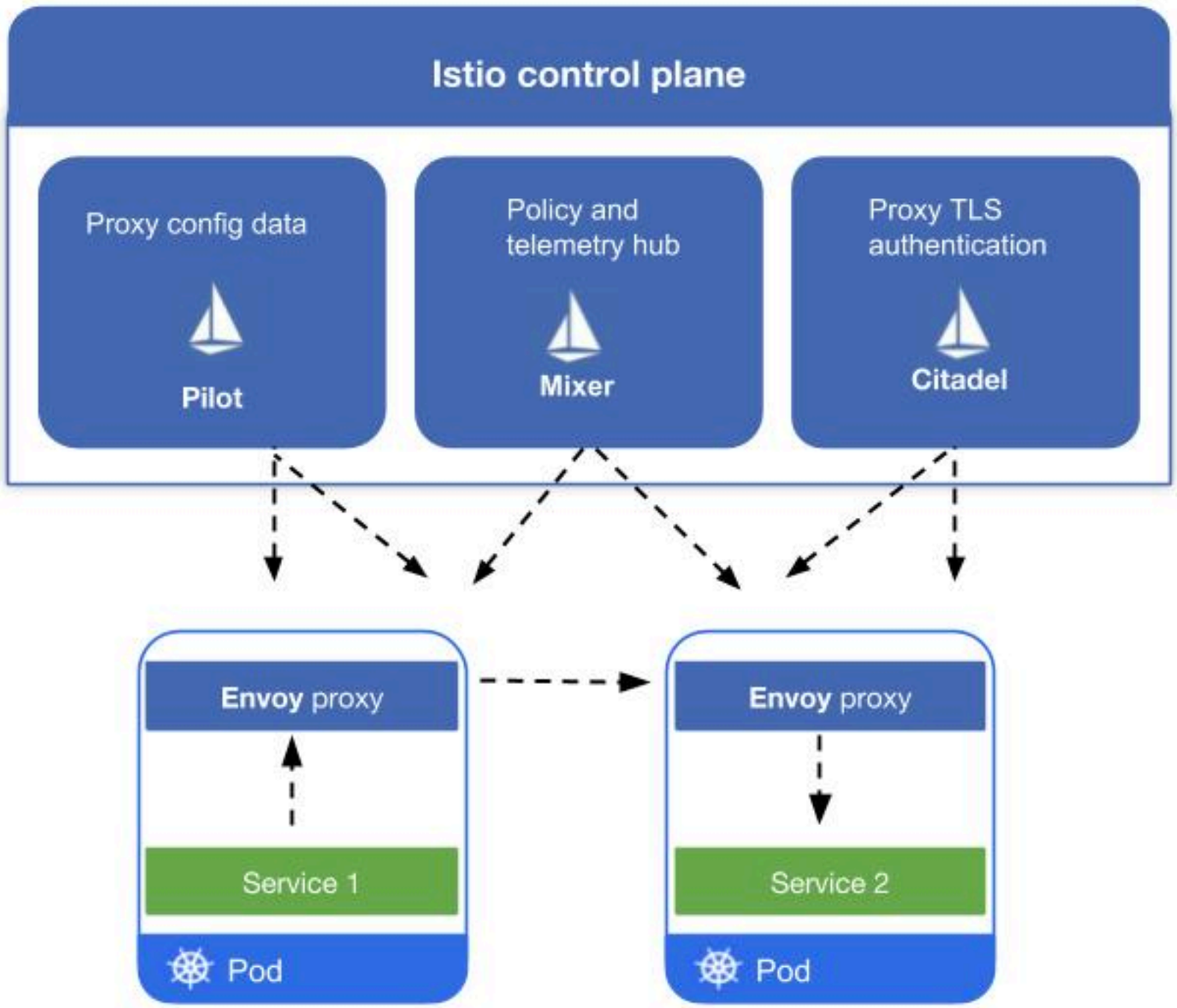
Cluster view



Cluster view

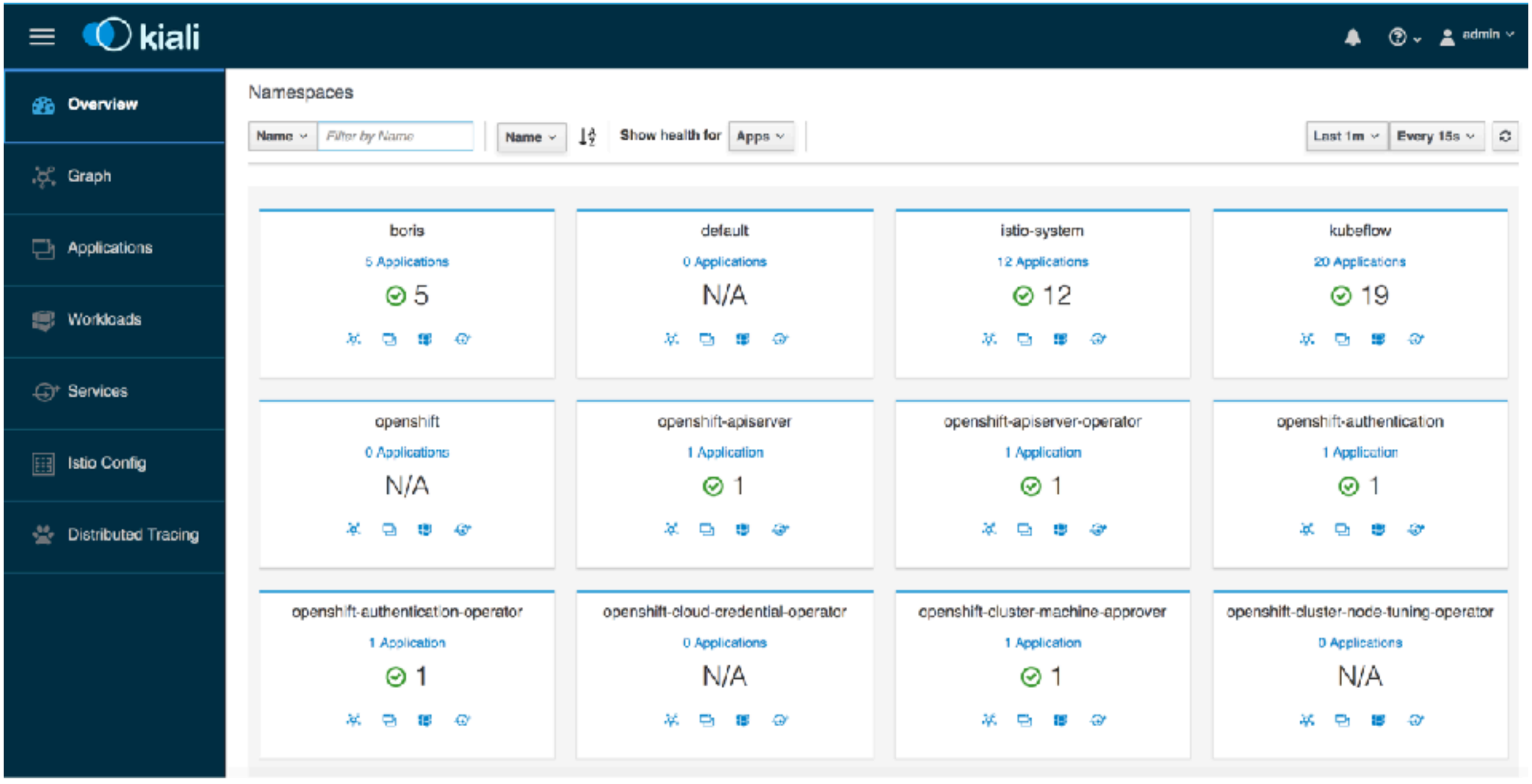


Istio

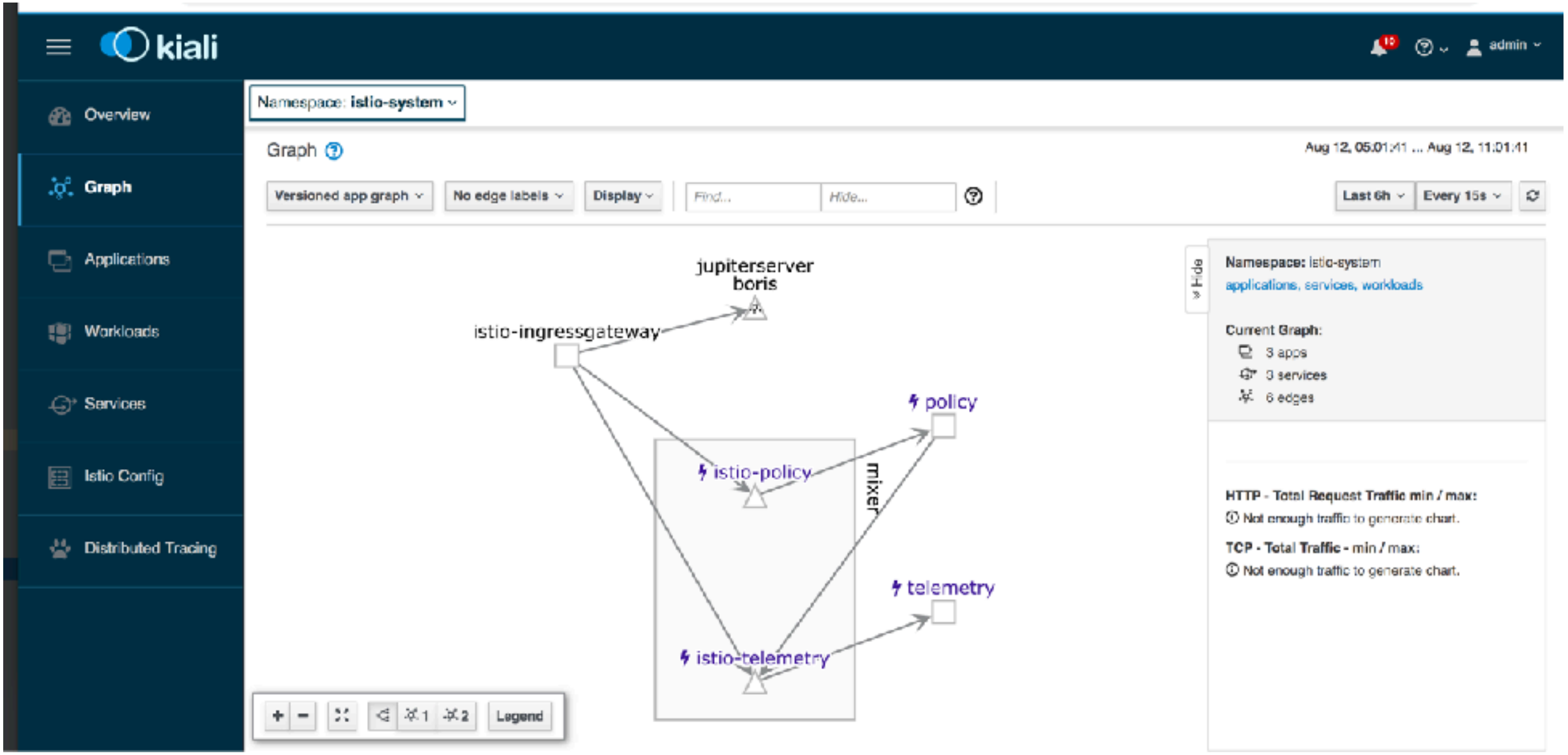


Source https://blog.newrelic.com/engineering/istio-service-mesh/

Istio in Kubeflow



Istio in Kubeflow

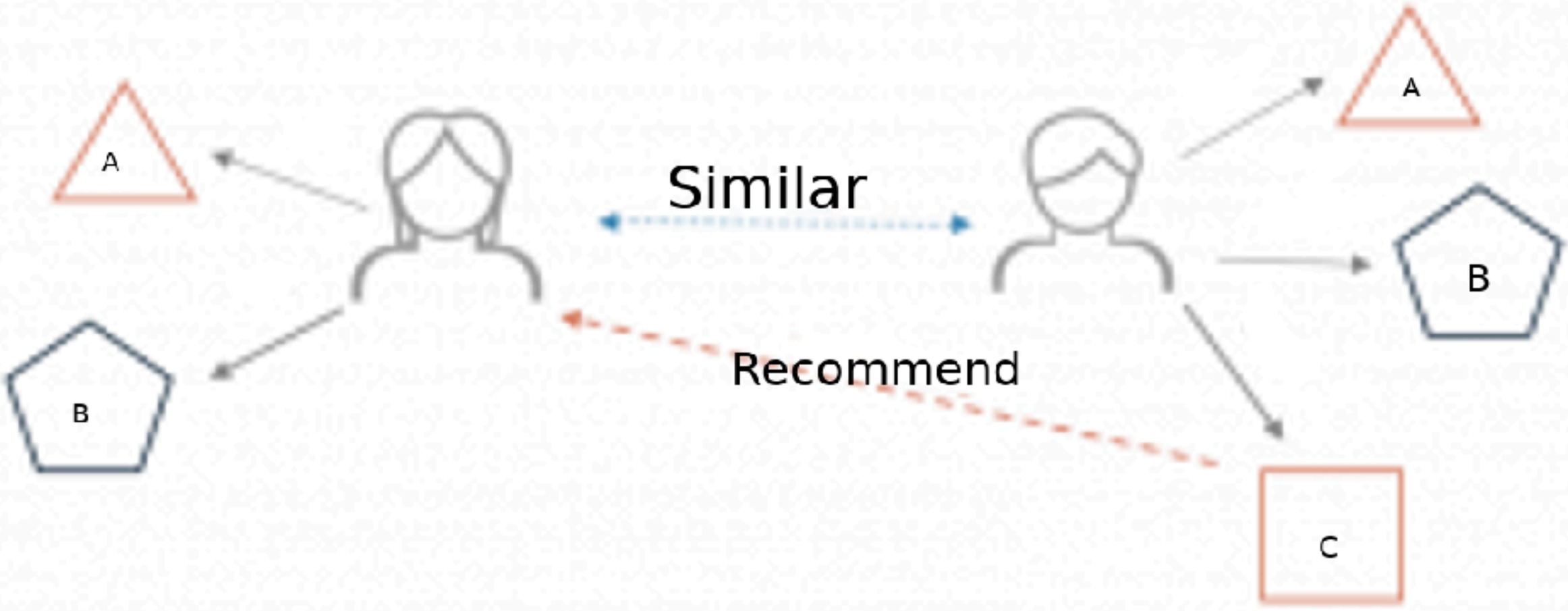


Motivating example

For our example we will use implementation of a product recommender.

Source https://towardsdatascience.com/recommender-system-a1e4595fc0f0

Collaborative filtering



Source *https://towardsdatascience.com/how-to-build-a-simple-recommender-system-in-python-375093c3fb7d*

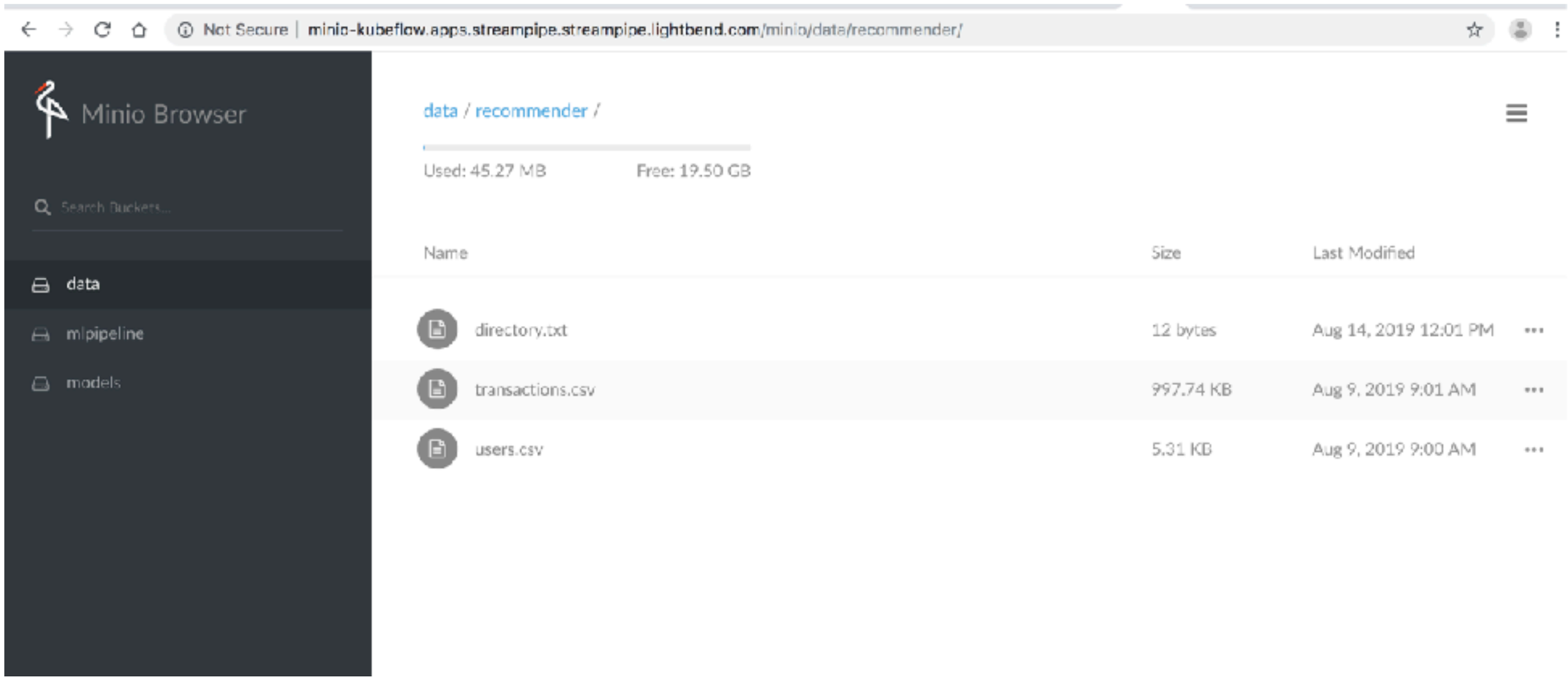
Rating Matrix

Users \ Items	1	2	3	4	...	n
1	4	3			5	
2	5		4		4	
3	4		5	3	4	
4		3				5
...		4				4
m			2	4		5

Source https://www.researchgate.net/figure/Users-rating-matrix-on-items-Ratings-1-5-express-users-preference-on-items-Empty_fig1_323135959

Using Minio as a centralized storage

Minio, a high-performance distributed object storage server, designed for large-scale private cloud infrastructure.

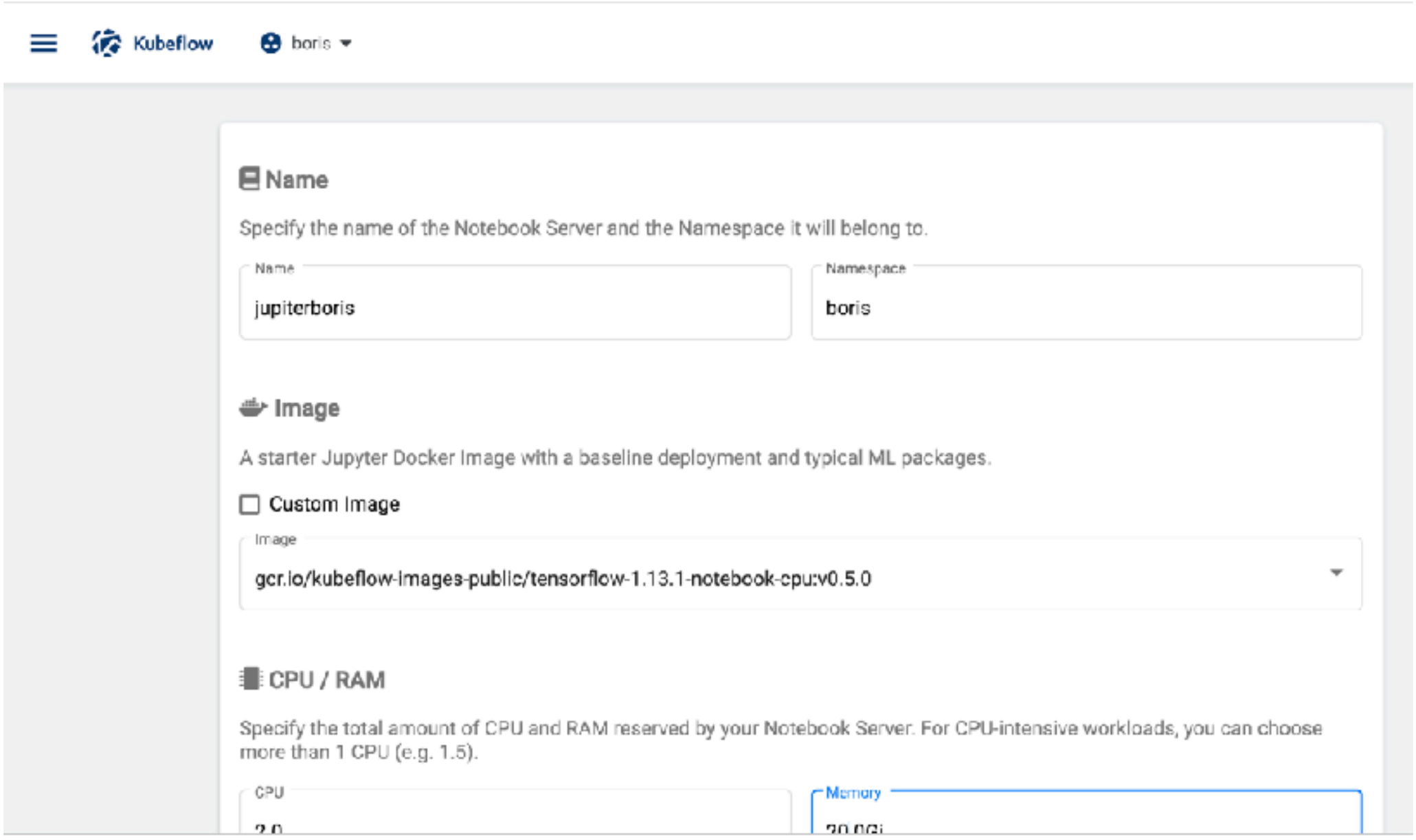


Kubeflow options for machine learning and model serving

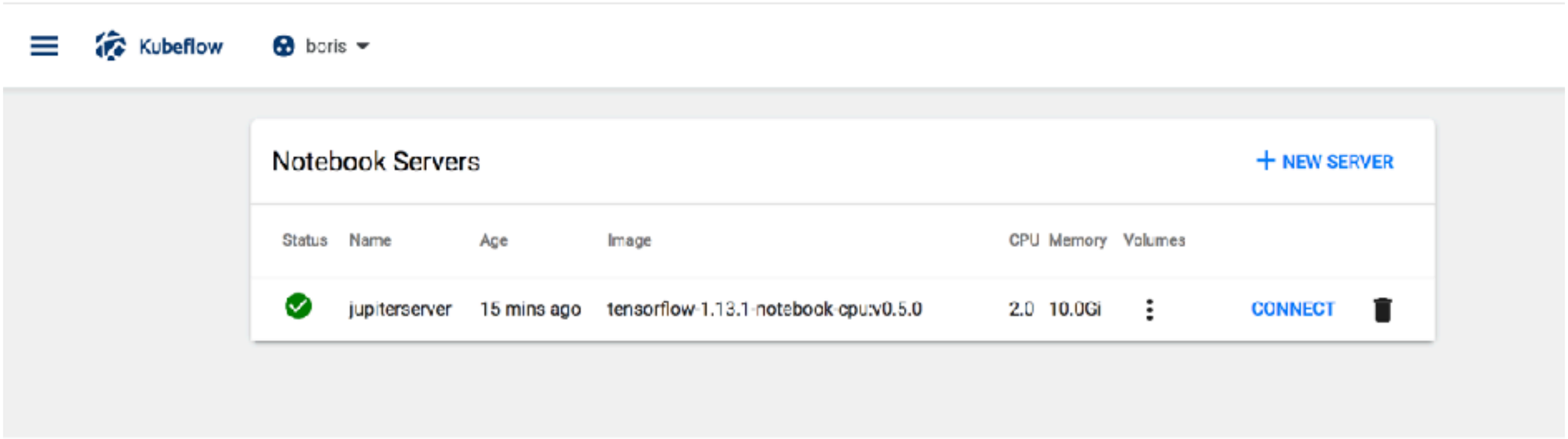
Learning

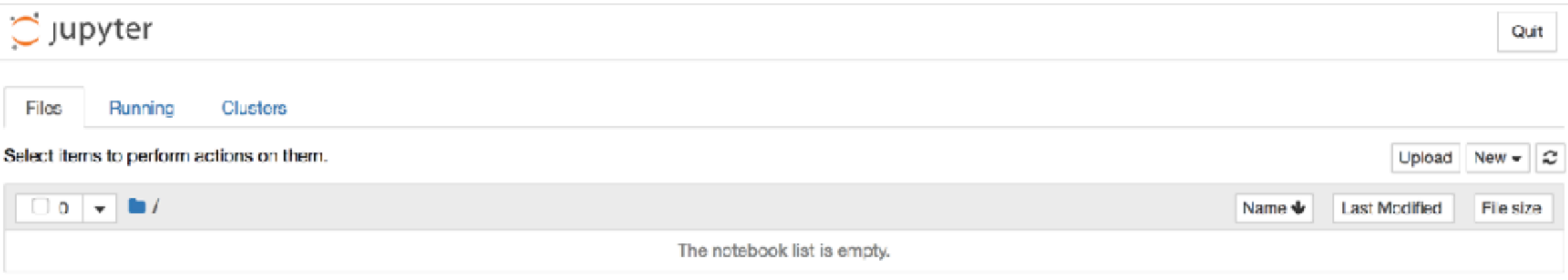
Serving

Using Jupiter for creating implementation



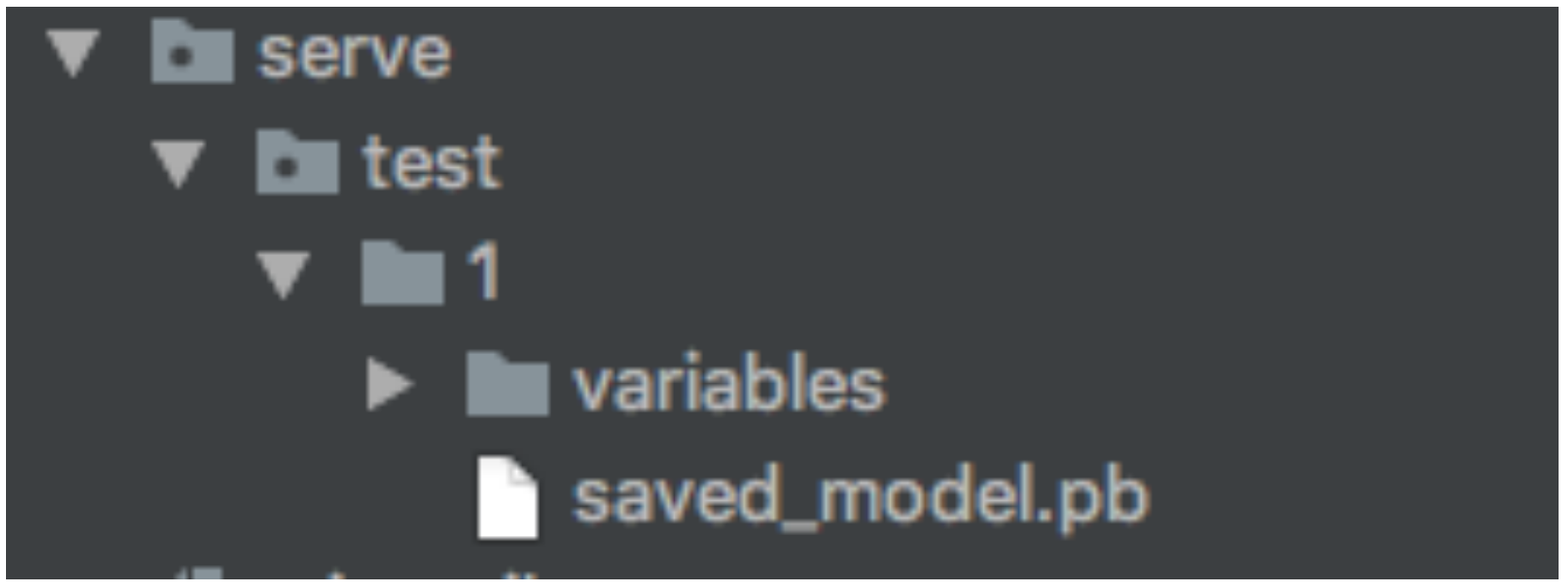
Using Jupiter for creating implementation





Jupiter - exporting model

Export model in a standard tensorflow saved model format, which defines both the layout of the content of the saved model on disk and the information stored in the model. The layout on disk looks as follows:



Converting implementation to TFJob

A distributed TensorFlow job typically contains some of the following processes:

- Chief - responsible for orchestrating training and performing tasks like checkpointing the model.
- Ps - parameter servers; these servers provide a distributed data store for the model parameters.
- Worker - who does the actual work of training the model. In some cases, worker 0 might also act as the chief.
- Evaluator - can be used to compute evaluation metrics as the model is trained.

Converting implementation to TFJob

- Export notebook as Python code.

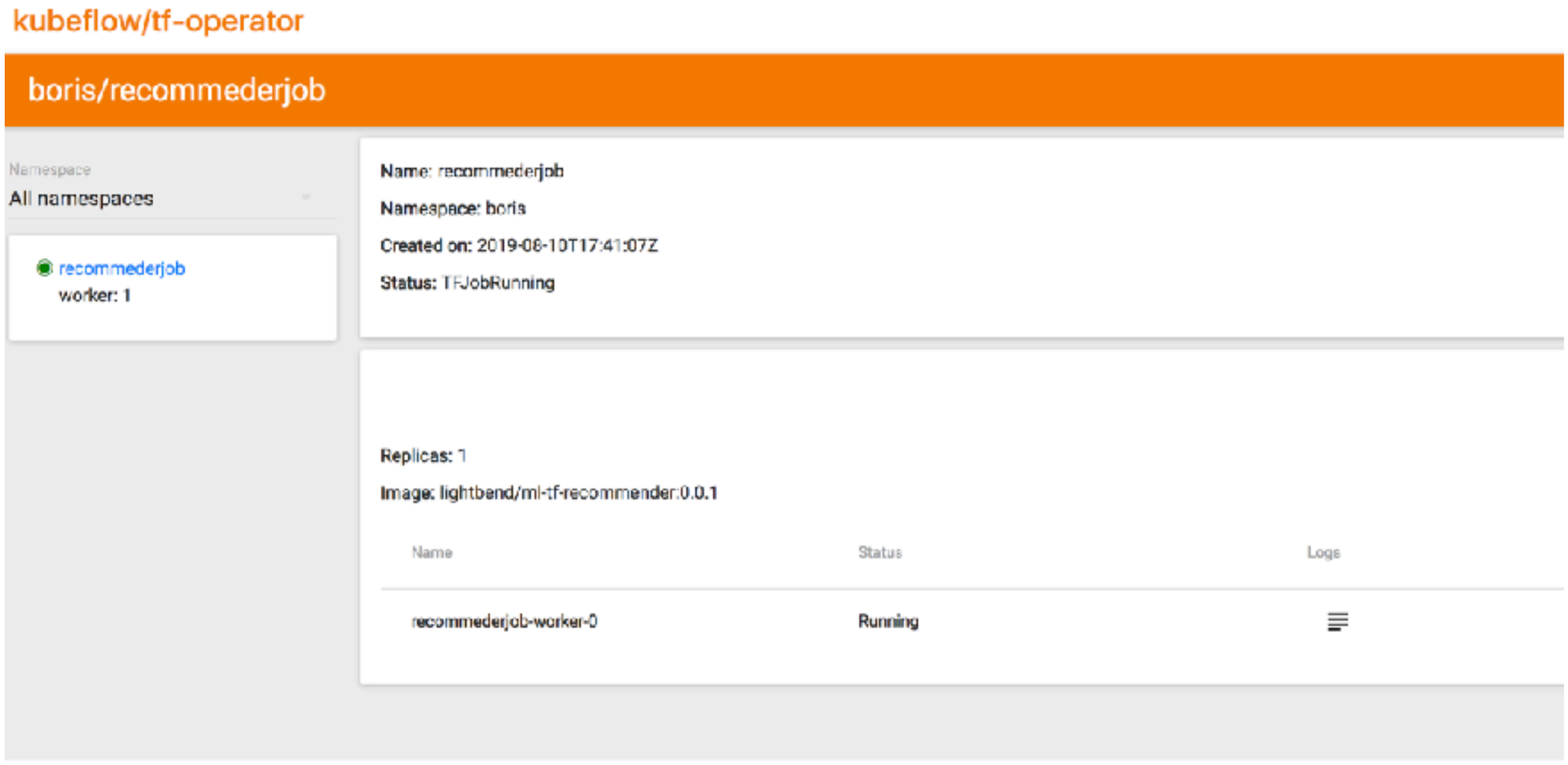
- Build docker image

```
FROM  tensorflow/tensorflow:1.12.0-devel-py3
RUN pip3 install --upgrade pip
RUN pip3 install pandas --upgrade
RUN pip3 install keras --upgrade
RUN pip3 install minio --upgrade
RUN mkdir -p /opt/kubeflow
COPY Recommender_Kubeflow.py /opt/kubeflow/
ENTRYPOINT ["python3", "/opt/kubeflow/
Recommender_Kubeflow.py"]
```

Converting implementation to TFJob

```yaml
apiVersion: "kubeflow.org/v1"
kind: "TFJob"
metadata:
  name: "recommederjob"
  namespace: boris
spec:
  cleanPodPolicy: None
  tfReplicaSpecs:
    Worker:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: tensorflow
              image: lightbend/ml-tf-recommender:0.0.1
              imagePullPolicy: "Always"
              env:
                - name: "MINIO_URL"
                  value: "minio-service.kubeflow.svc.cluster.local:9000"
                - name: "MINIO_KEY"
                  valueFrom: { secretKeyRef: { name: "minioaccess", key:
"AWS_ACCESS_KEY_ID" } }
                - name: "MINIO_SECRET"
```

Running TF Job



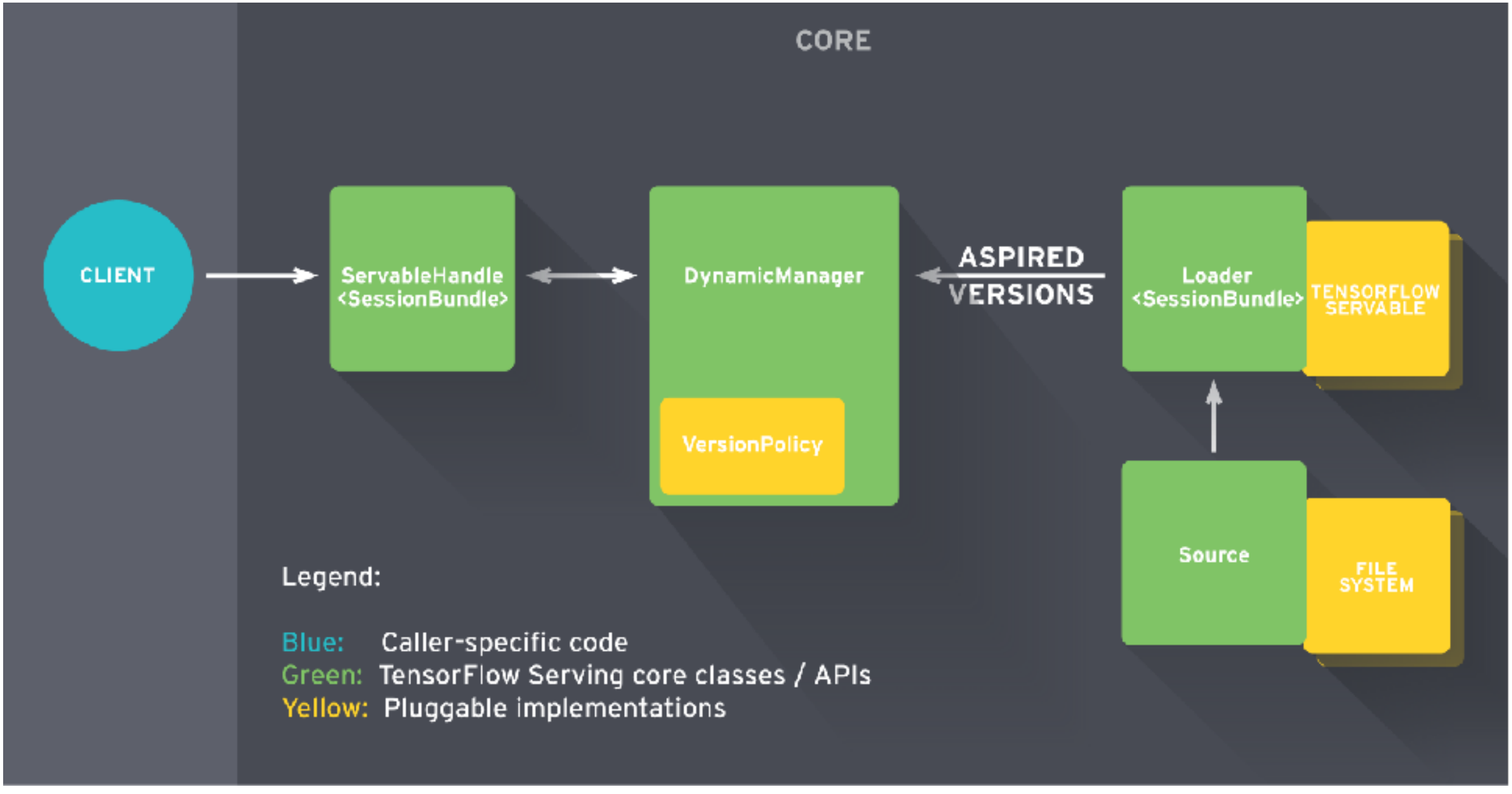
Running TF Job

Logs

Layer (type)	Output Shape	Param #	Connected to
user (InputLayer)	(None, 1)	0	
product (InputLayer)	(None, 1)	0	
embedding_1 (Embedding)	(None, 1, 50)	1430300	user[0][0]
embedding_2 (Embedding)	(None, 1, 50)	15000	product[0][0]
reshape_1 (Reshape)	(None, 50)	0	embedding_1[0][0]
reshape_2 (Reshape)	(None, 50)	0	embedding_2[0][0]
concatenate_1 (Concatenate)	(None, 100)	0	reshape_1[0][0]

CLOSE

TF - serving



Source https://medium.freecodecamp.org/how-to-deploy-tensorflow-models-to-production-using-tf-serving-4b4b78d41700

Deploying TF-serving

Create and deploy secret to access Minio

```
apiVersion: v1
kind: Secret
metadata:
  name: minioaccess
data:
  AWS_ACCESS_KEY_ID: xxxxxxxxxx
  AWS_SECRET_ACCESS_KEY: xxxxxxxxxxxxxxxxxxxxxx
```

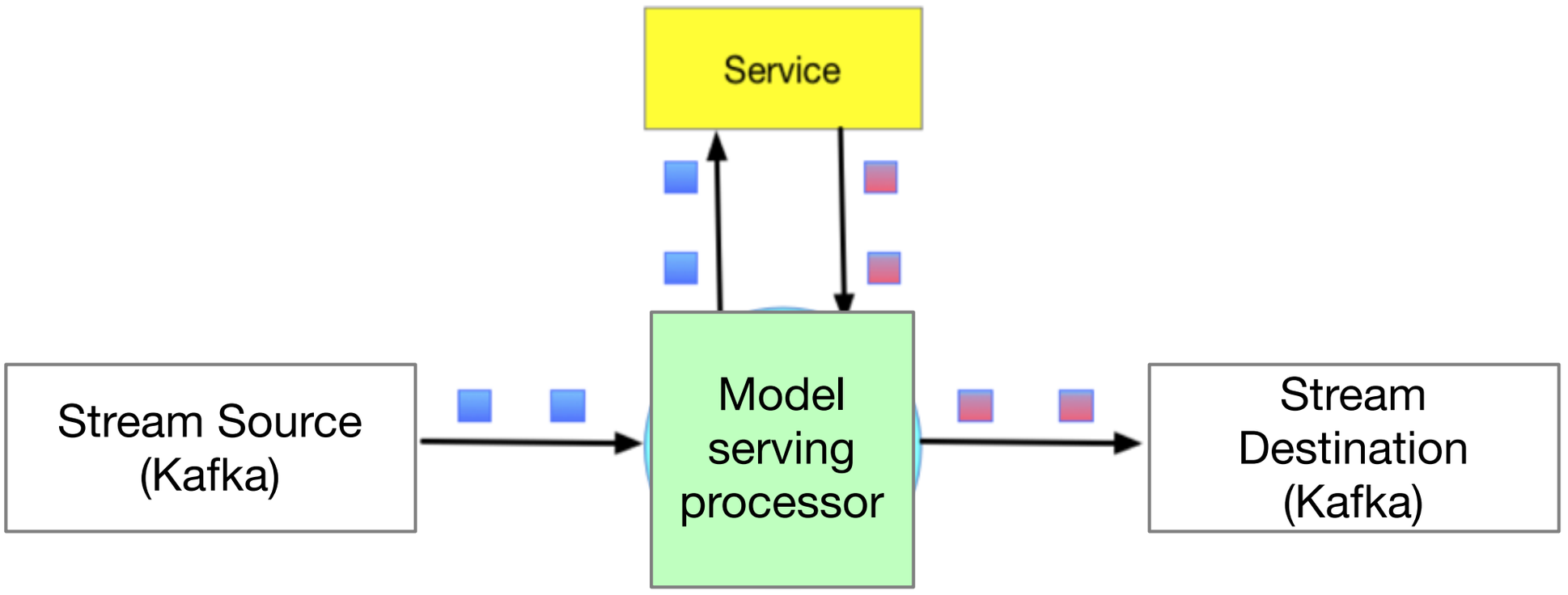
Deploying TF-serving

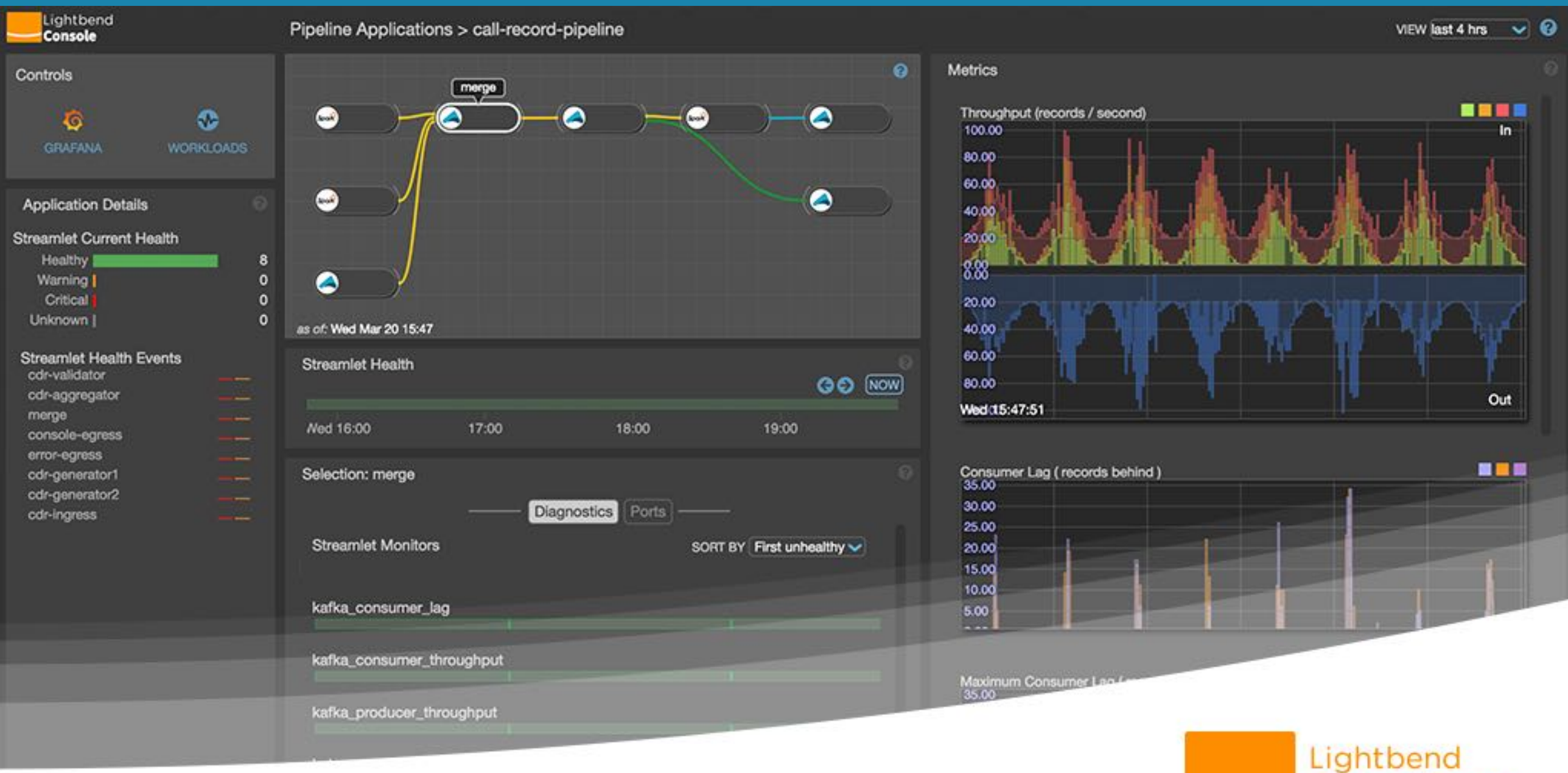
TF-serving is not yet ported into Kubeflow 0.6. As a result I used custom Helm chart

Validating TF-Serving

```
curl -X POST http://recommender-service-
kubeflow.lightshift.lightbend.com/v1/models/recommender/
versions/1:predict -d
'{"signature_name":"serving_default","inputs": {"products":
[[1],[2]],"users" : [[25], [3]]}}'
{
    "outputs": {
        "model-version": [
            "1"
        ],
        "recommendations": [
            [
                0.185666
            ],
            [
                0.0364329
```

Using TF Serving in streaming applications





What we're up to at Lightbend…

lightbend.com/lightbend-pipelines-demo

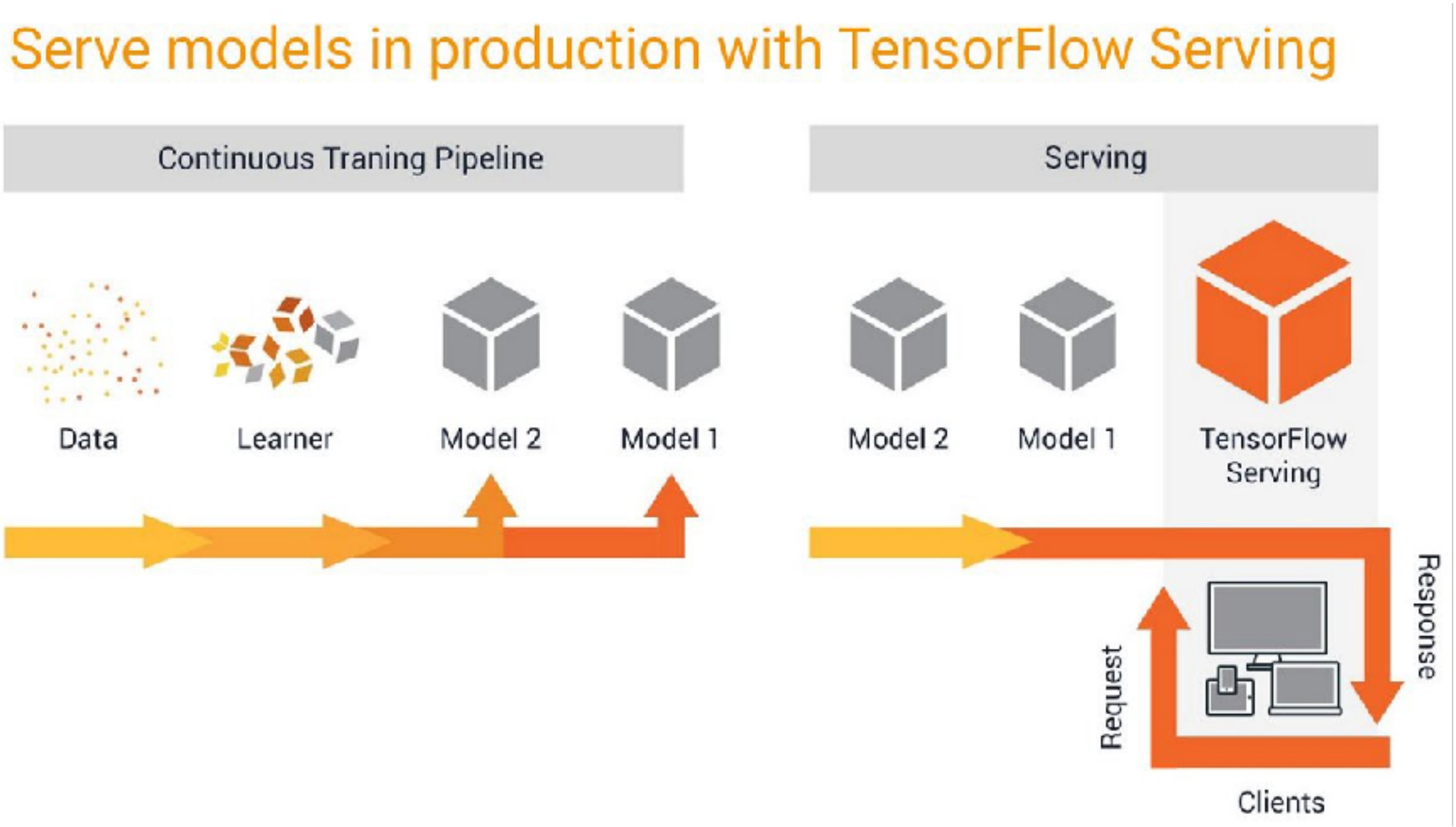
Lightbend Pipelines

Concept drift

In most challenging data analysis applications, data evolve over time and must be analyzed in near real time. Patterns and relations in such data often evolve over time, thus, models built for analyzing such data quickly become obsolete over time. In machine learning and data mining this phenomenon is referred to as concept drift.

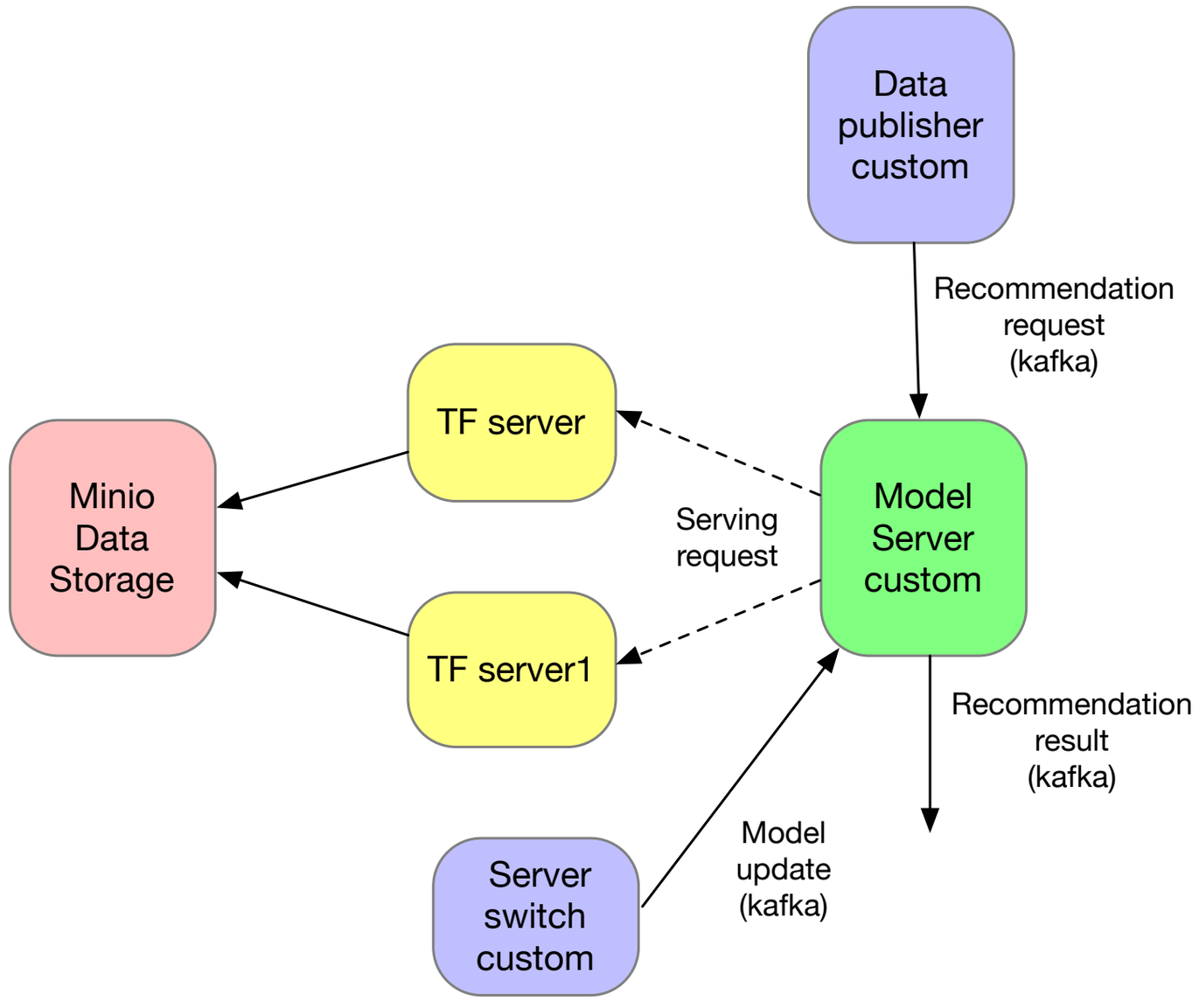
Source https://www.win.tue.nl/~mpechen/publications/pubs/CD_applications15.pdf

Continuous model updates



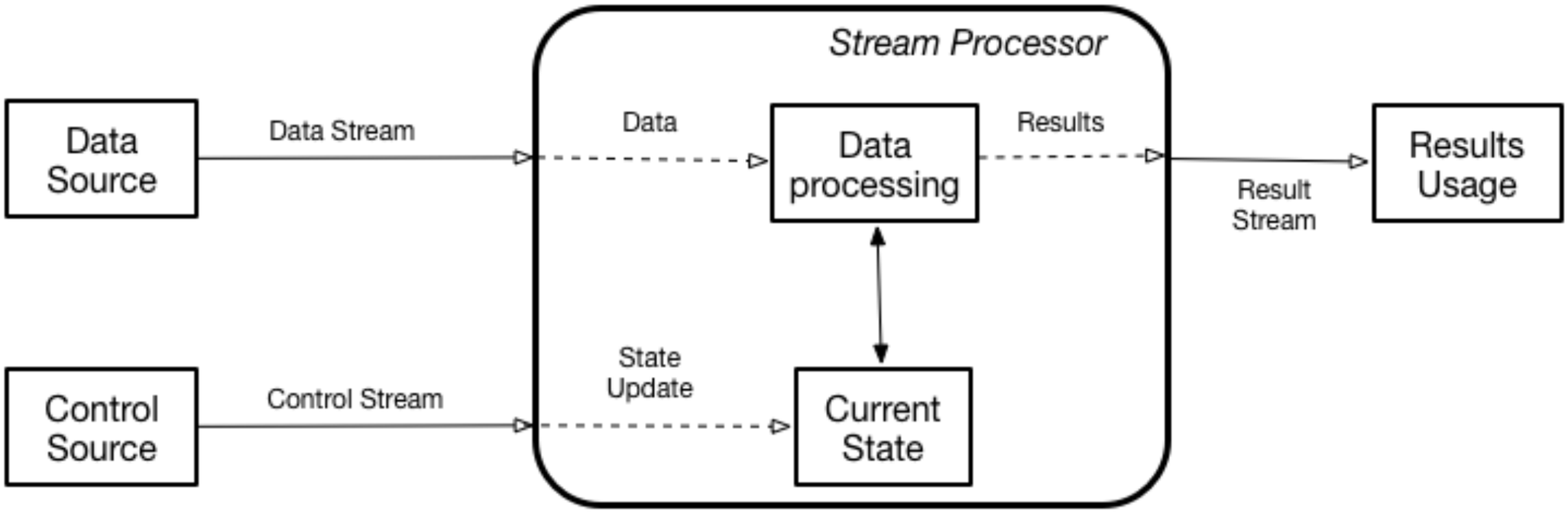
Source https://towardsdatascience.com/using-tensorflow-serving-grpc-38a722451064

Continuous model updates implementation



Source https://towardsdatascience.com/using-tensorflow-serving-grpc-38a722451064

TF Serving in streaming applications



Additional custom components

- Model serving data provider - generator to produce recommendation requests

- Data updater for model serving - component responsible for publishing of new data for machine learning

- Model updater for model serving - component responsible for updating Tensorflow server information for model serving

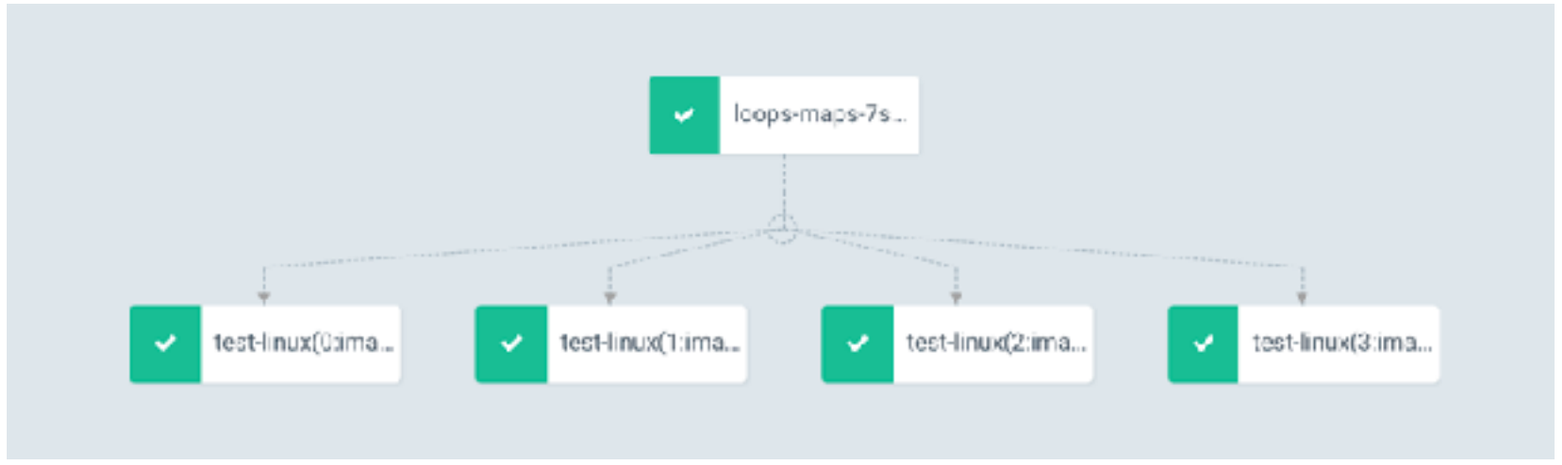
Kubeflow pipelines

Kubeflow Pipelines is a platform for building and deploying portable, scalable machine learning (ML) workflows based on Docker containers. The Kubeflow Pipelines platform consists of:

- A user interface (UI) for managing and tracking experiments, jobs, and runs.
- An engine for scheduling multi-step ML workflows
- An SDK for defining and manipulating pipelines and components.
- Notebooks for interacting with the system using the SDK.

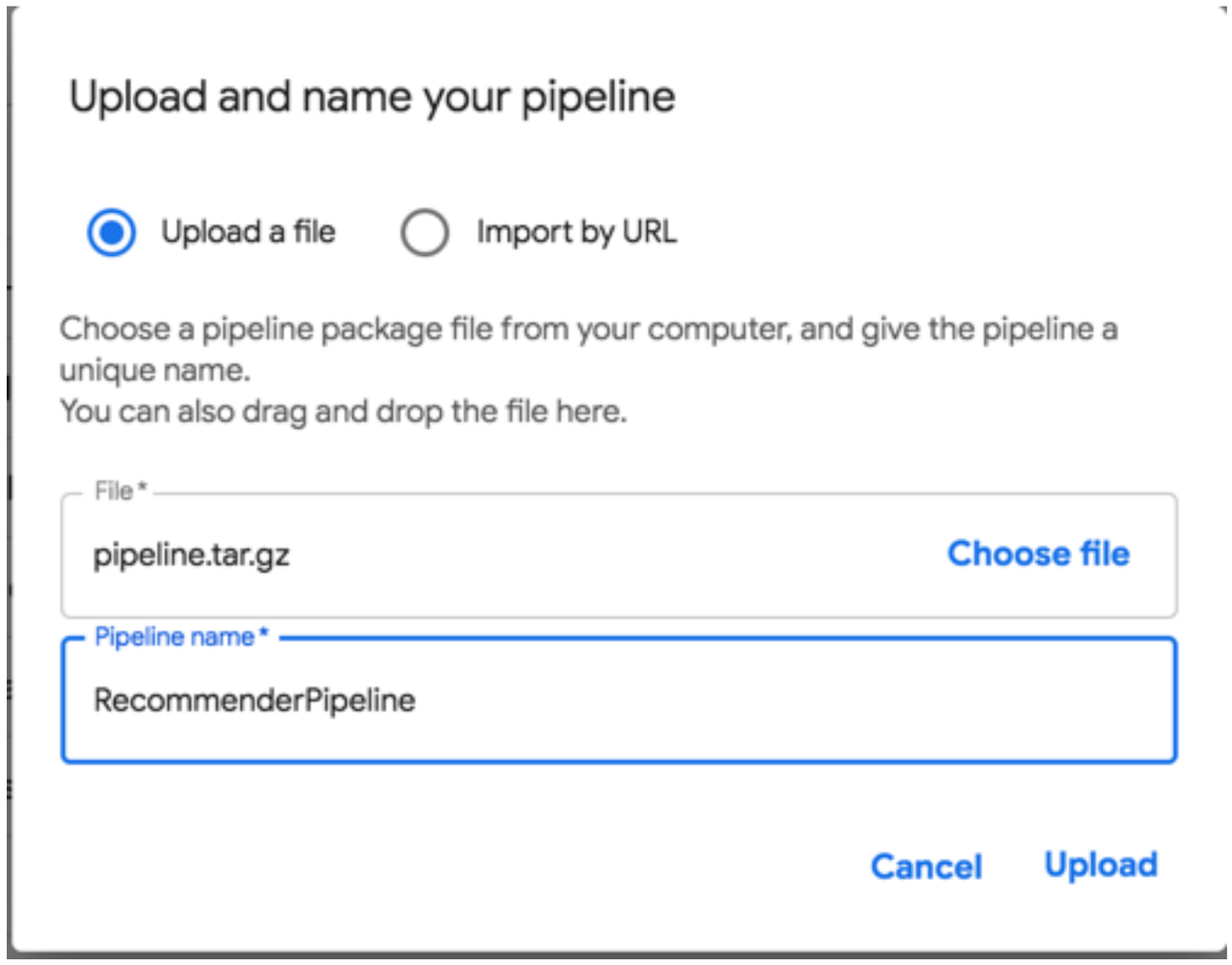
Source *https://www.kubeflow.org/docs/pipelines/pipelines-overview/*

Argo workflow

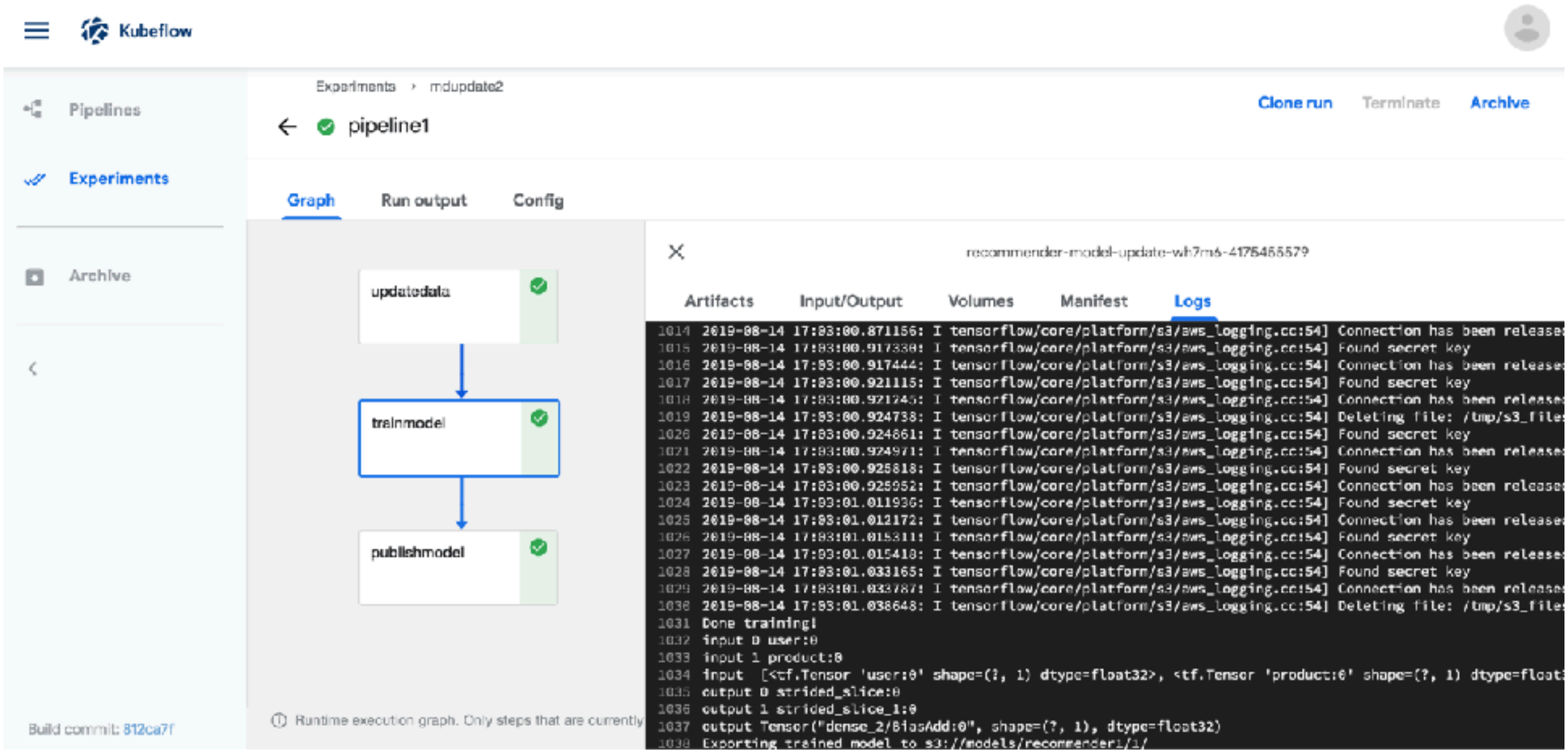


Implementing Kubeflow pipelines

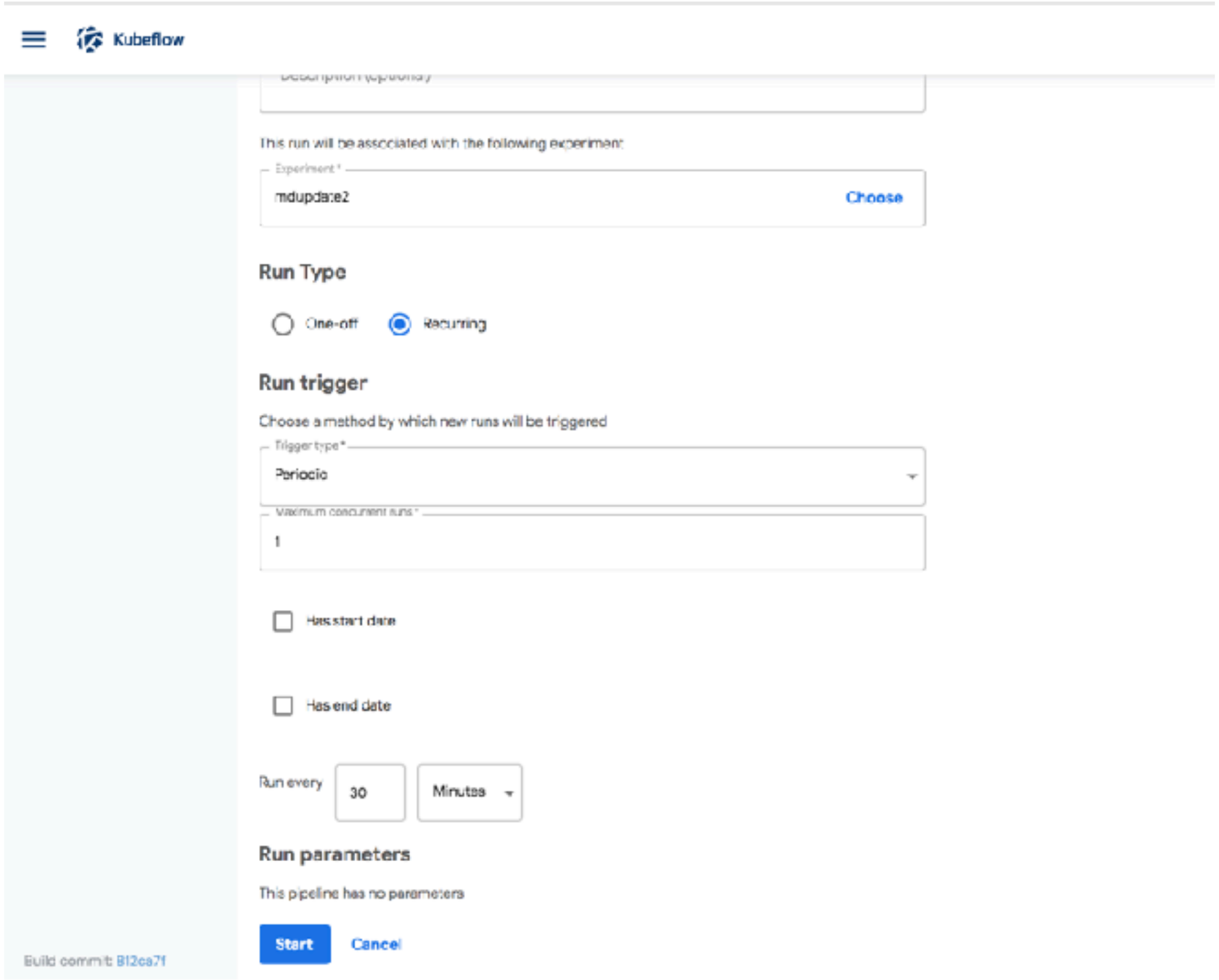
- Based on the Python library
- Fully supported in Jupiter
- Jupiter implementation can be exported to Python
- Python code creates pipeline.tar.gz file, which now can be uploaded to the Kubeflow pipelines UI



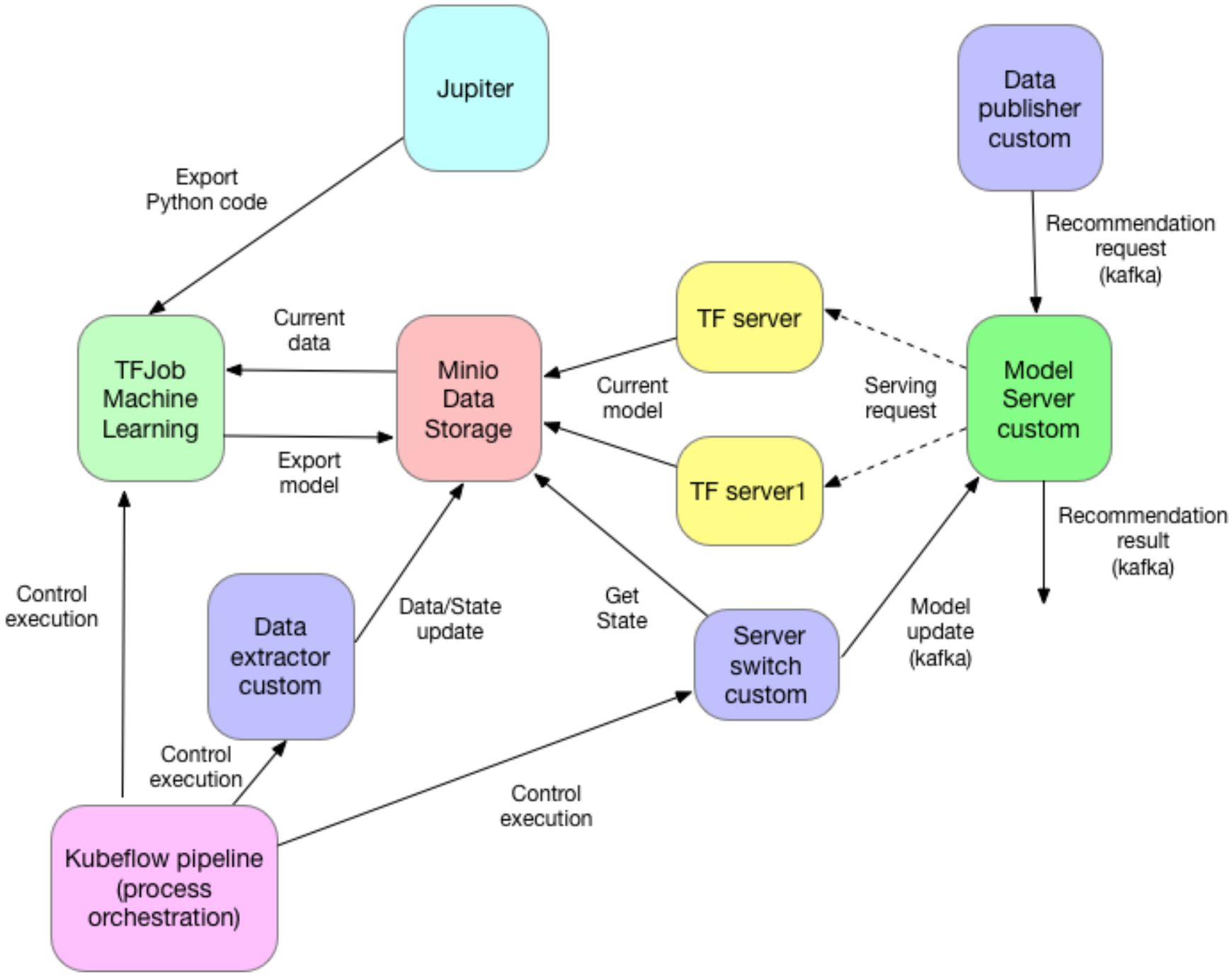
Pipeline execution



Execution of pipelines on schedule



Bringing it all together



Try this yourself

- Recommender implementation is heavily based on this blog post

★https://medium.com/datadriveninvestor/how-to-build-a-recommendation-system-for-purchase-data-step-by-step-d6d7a78800b6

- We used purchases data provided here to test our implementation.

★https://github.com/moorissa/medium/blob/master/items-recommender/data/trx_data.csv

- The full code, including Kubeflow setup information, is available here

★https://github.com/lightbend/kubeflow-recommender

Thank you

Any questions?