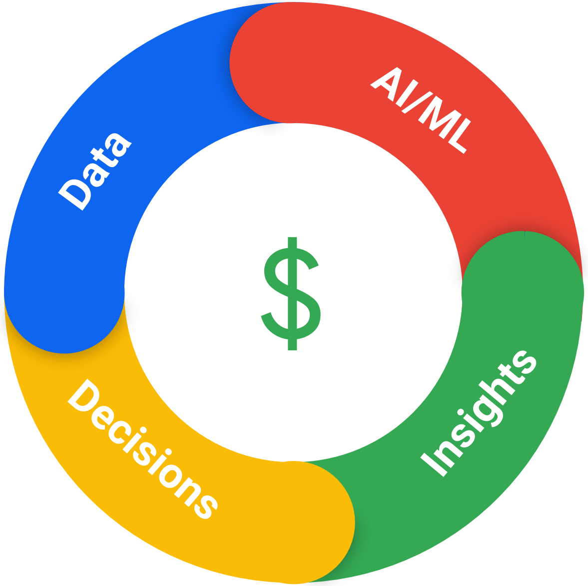
# Economics and best practices of running AI/ML workloads on Kubernetes

**Maulin Patel,** Product Manager, Google
**Yaron Haviv,** CTO and Founder of Iguazio

Google Cloud

# AI/ML driven decisions



Simple

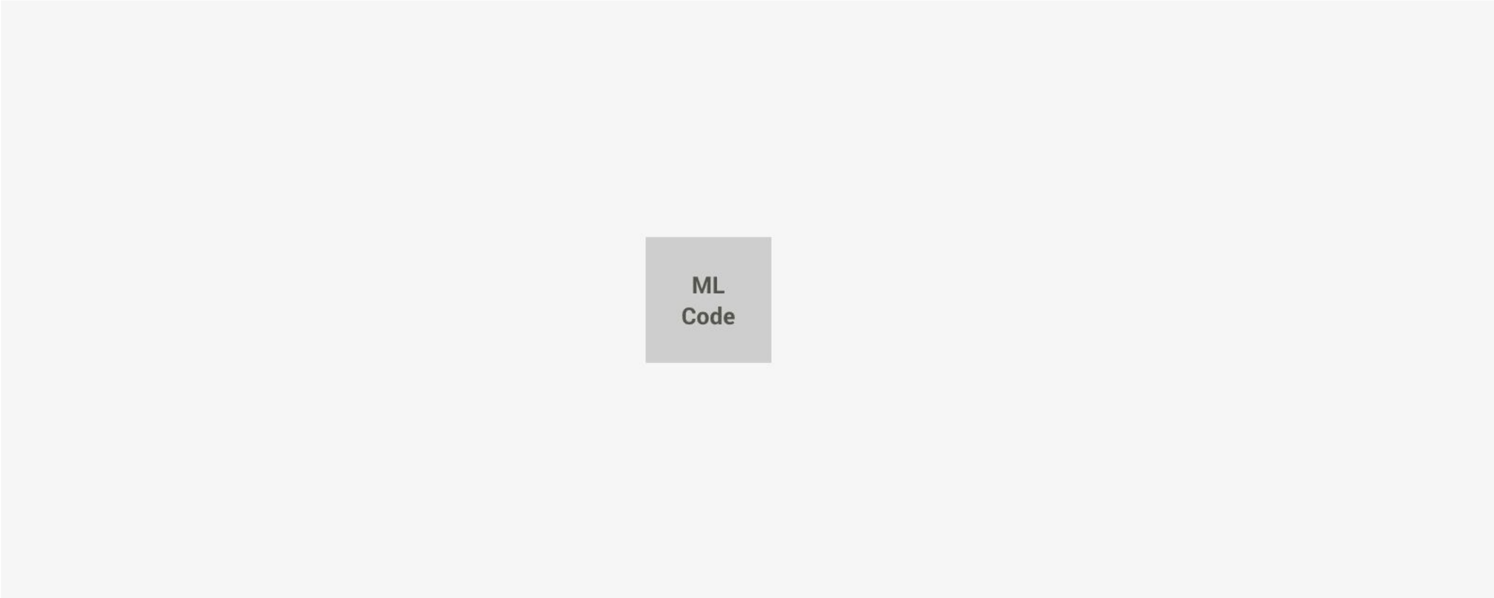
Fast

Cost-effective

# MAKING AI/ML

Google Cloud

# AI/ML is a team sport



How to make AI/ML teams

# 10x

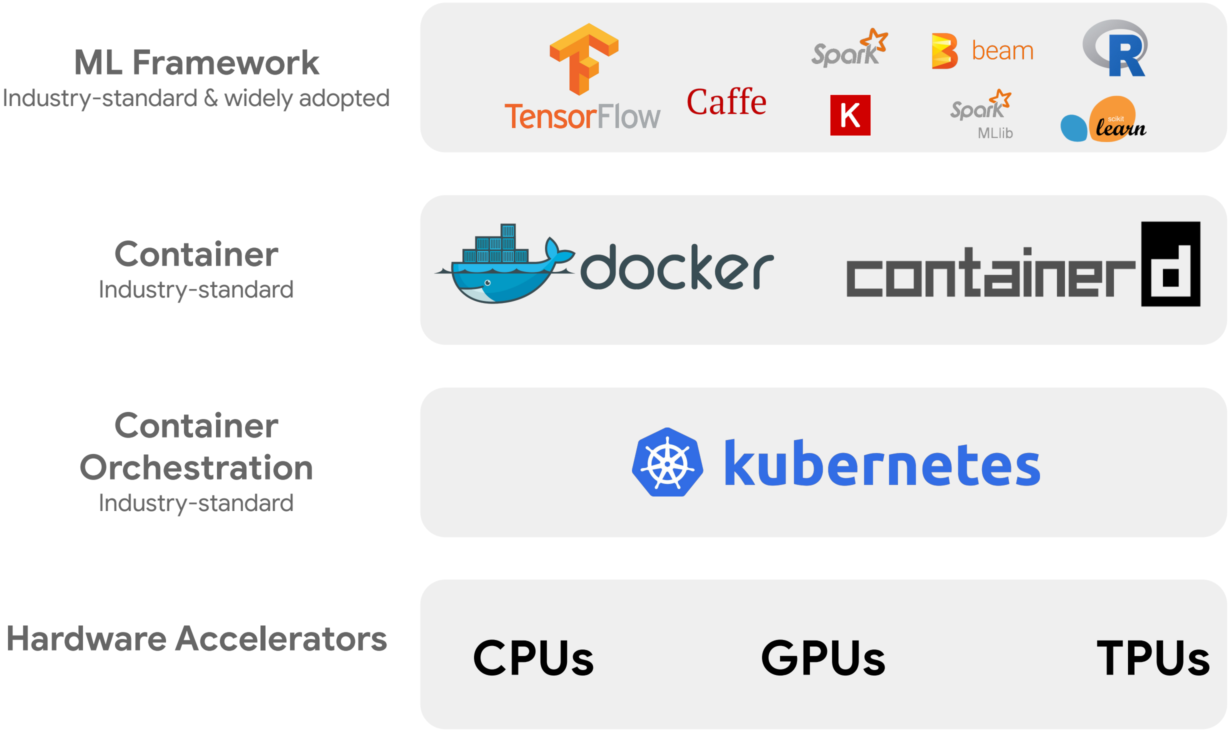
More Productive

Simple/Fast/Cost-effective

# Cloud native AI/ML platform

**ML Framework + Container + Kubernetes + HW Accelerators**

**Choose** your favorite ML Framework, **pack** models up in **Containers**, run on **Kubernetes** at scale

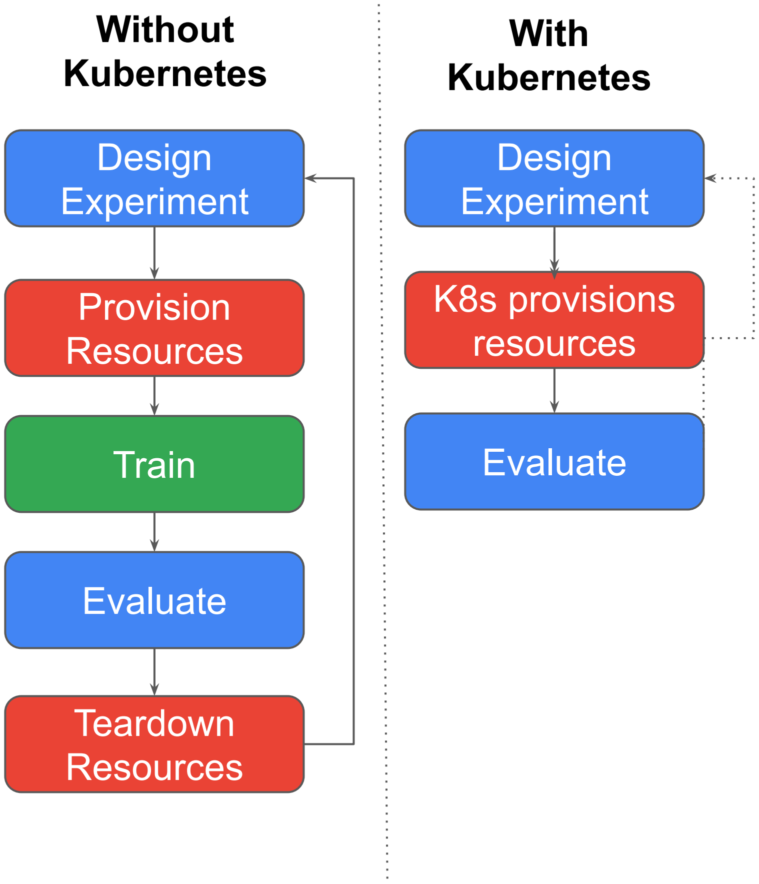



Google Cloud

Simple/Fast/Cost-effective

# Why Kubernetes for AI/ML?

- Portability
  - Cloud native, open, standard APIs
    - Seamlessly port workloads between Laptop/Cloud
- Scalability
  - Kubernetes scales from a single workstation to thousands of nodes
    - Support for GPU/TPU and distributed computing
- Productivity
  - Frees up users from managing their own workstations, servers  and VMs.
    - Lets you focus on model building and training

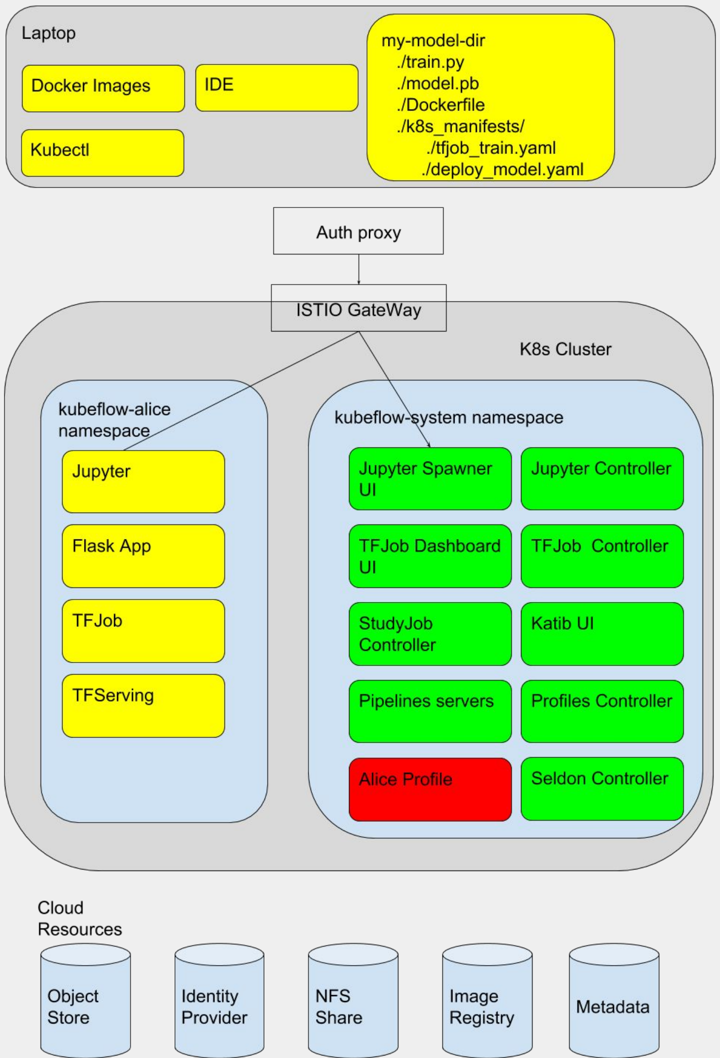



Google Cloud

# Kubeflow

A Kubernetes-native OSS Platform to Develop, Deploy and Manage, Scalable and End-to-End ML Workloads

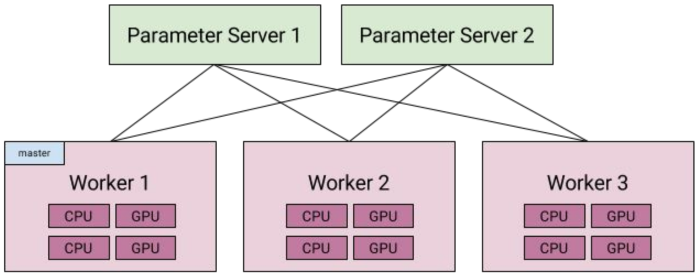
https://kubeflow.org



# TensorFlow training (TFJob)

- Integrates TensorFlow **distributed training** and estimator API with Kubernetes
- Uses Kubernetes to **scale training** and **leverage** hardware accelerators
- Users benefit from **Kubernetes toolchain**
  - kubectl for CLI
  - Kubernetes dashboard for monitoring

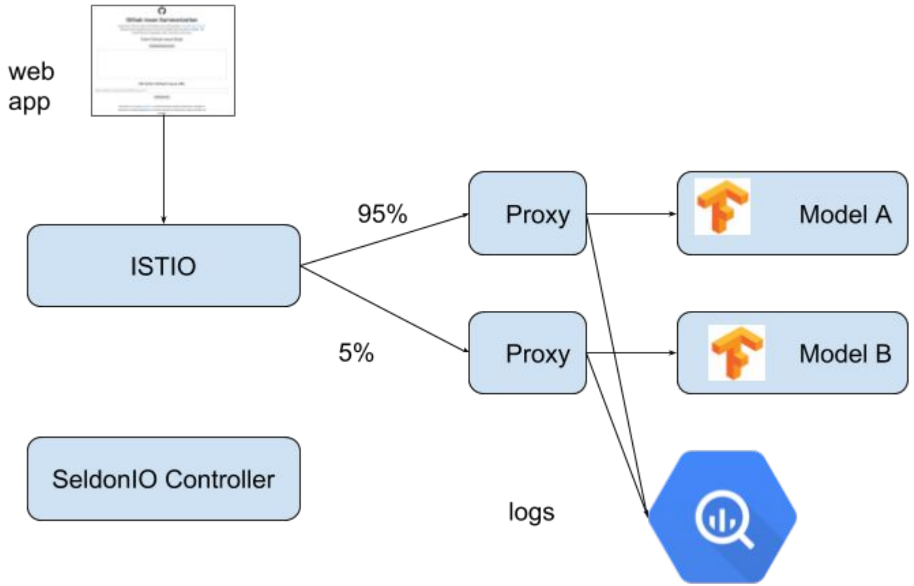
```
apiVersion: kubeflow.org/v1alpha2
kind: TFJob
metadata:
  name: tf-job-simple
  namespace: kubeflow
spec:
 tfReplicaSpecs:
   Workers:
     replicas: 3
     template:
      spec:
        containers:
        - image: acme/myjob
```



# TensorFlow serving

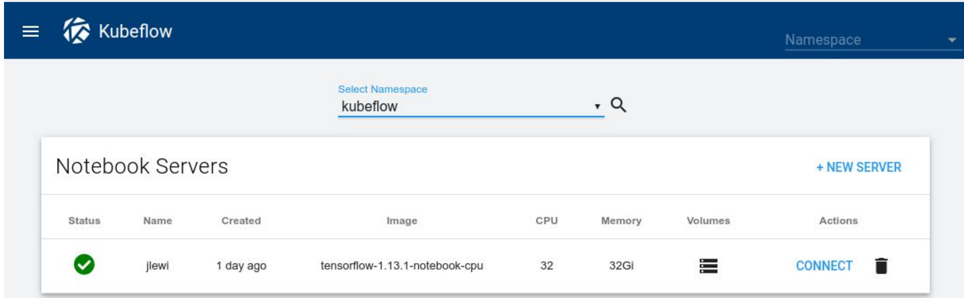
- Kubernetes **native TFServing**
- Leveraging Kubernetes to **simplify model rollouts**
- **Prometheus** exporter for **metrics**
- **ISTIO** for **telemetry** and traffic splitting

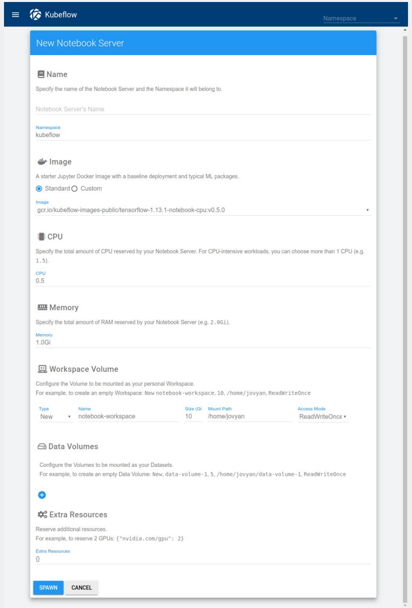
model push ≠ binary push



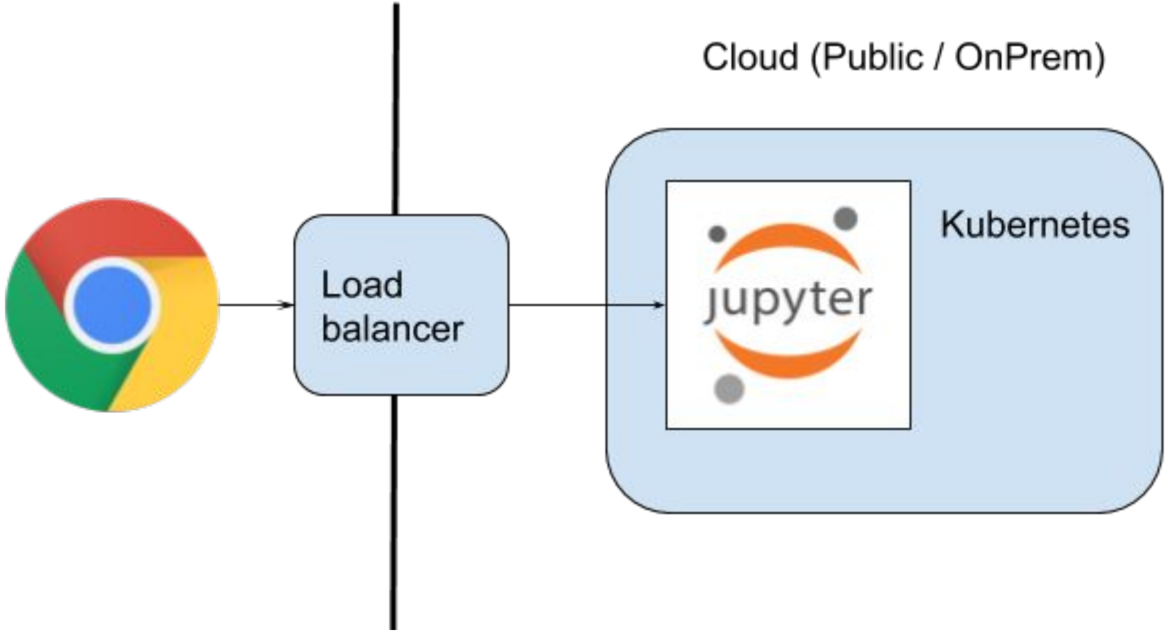
# Get started right

- Day 0 start with the infrastructure (Notebook, Kubernetes, ISTIO, etc...)
- Day 0 focus on model development
  - Use UIs to launch notebooks
  - Python SDK (fairing) for training / deploying models
- Day N leverage K8s to scale
  - Use the same infrastructure as non-ML applications
  - Build a single infrastructure team



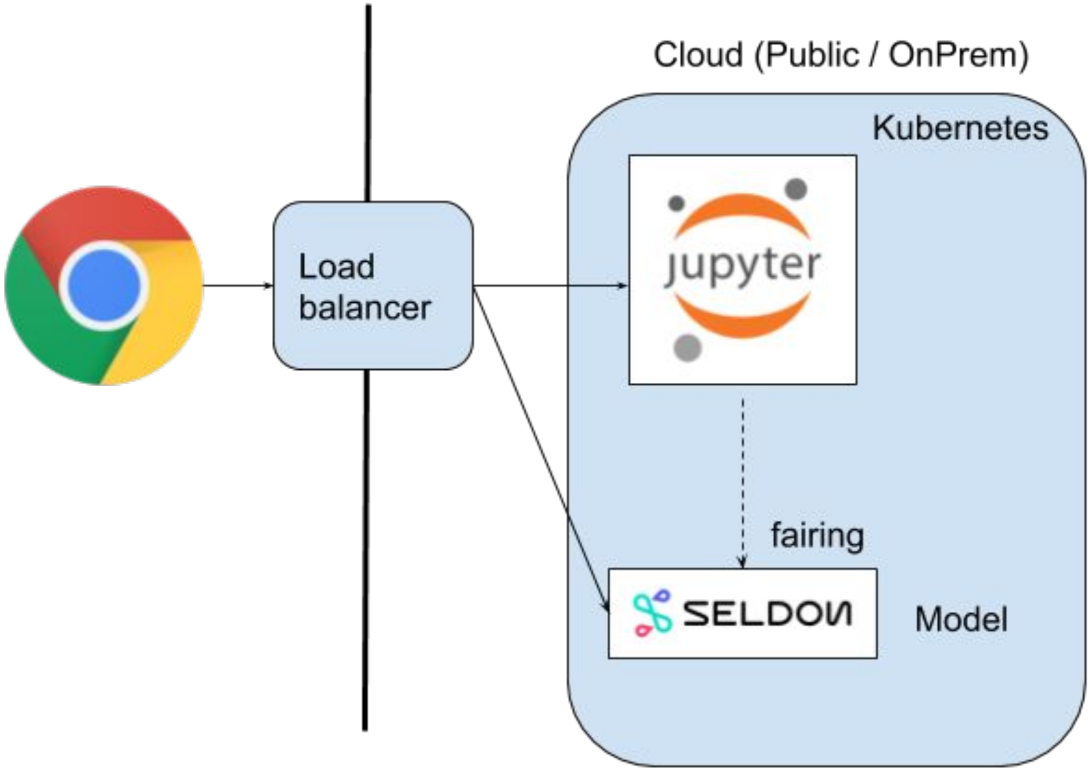



# Day 0: Data scientist friendly Notebooks

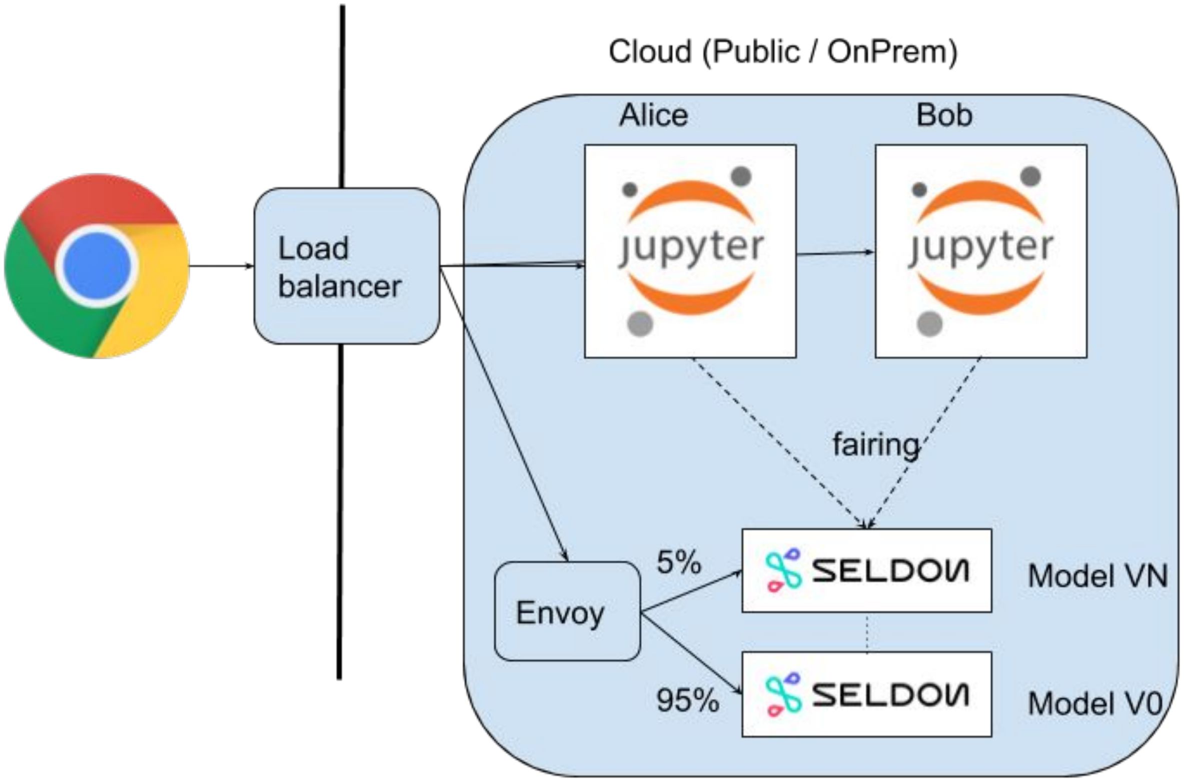



1. Connect to data, machines
2. Build models
3. Train
4. Deploy

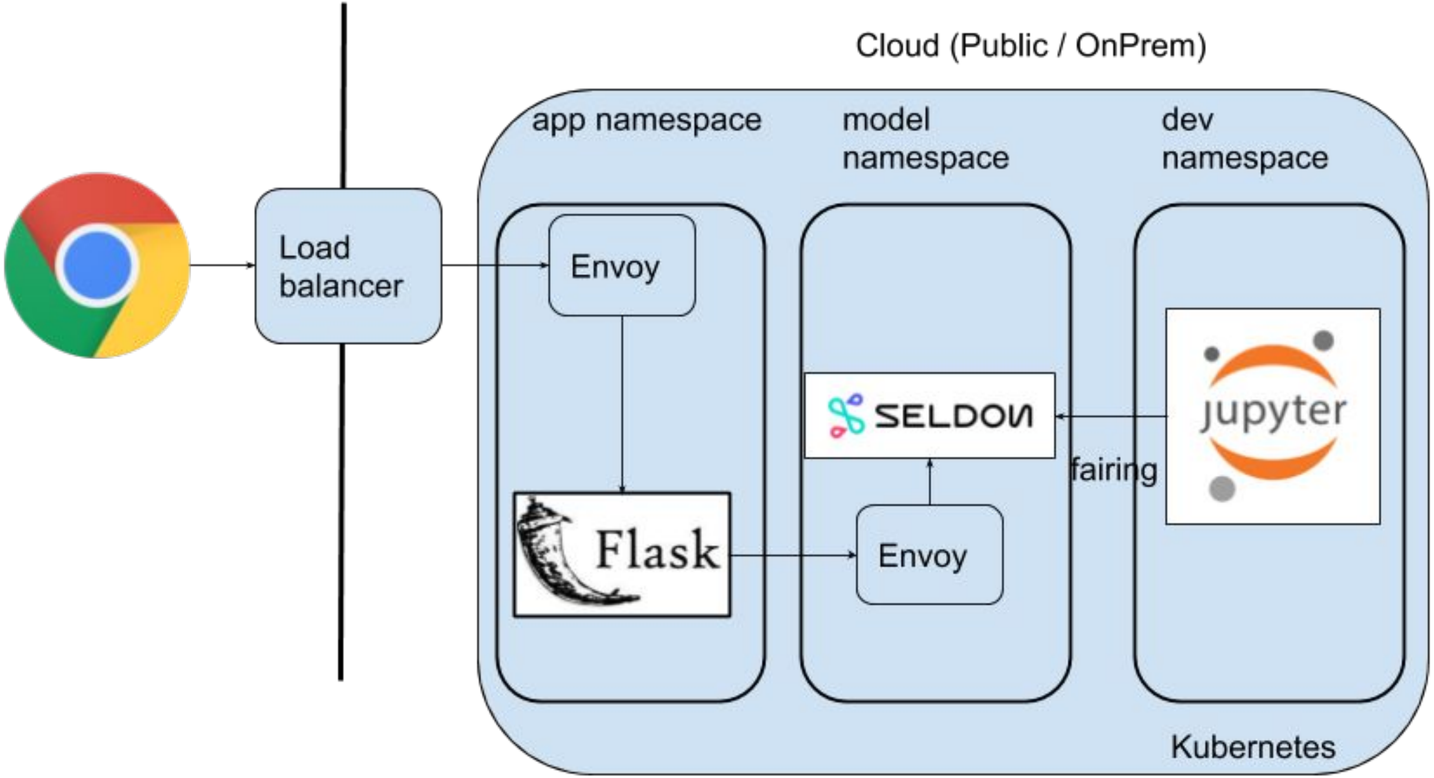
# Deploy Model



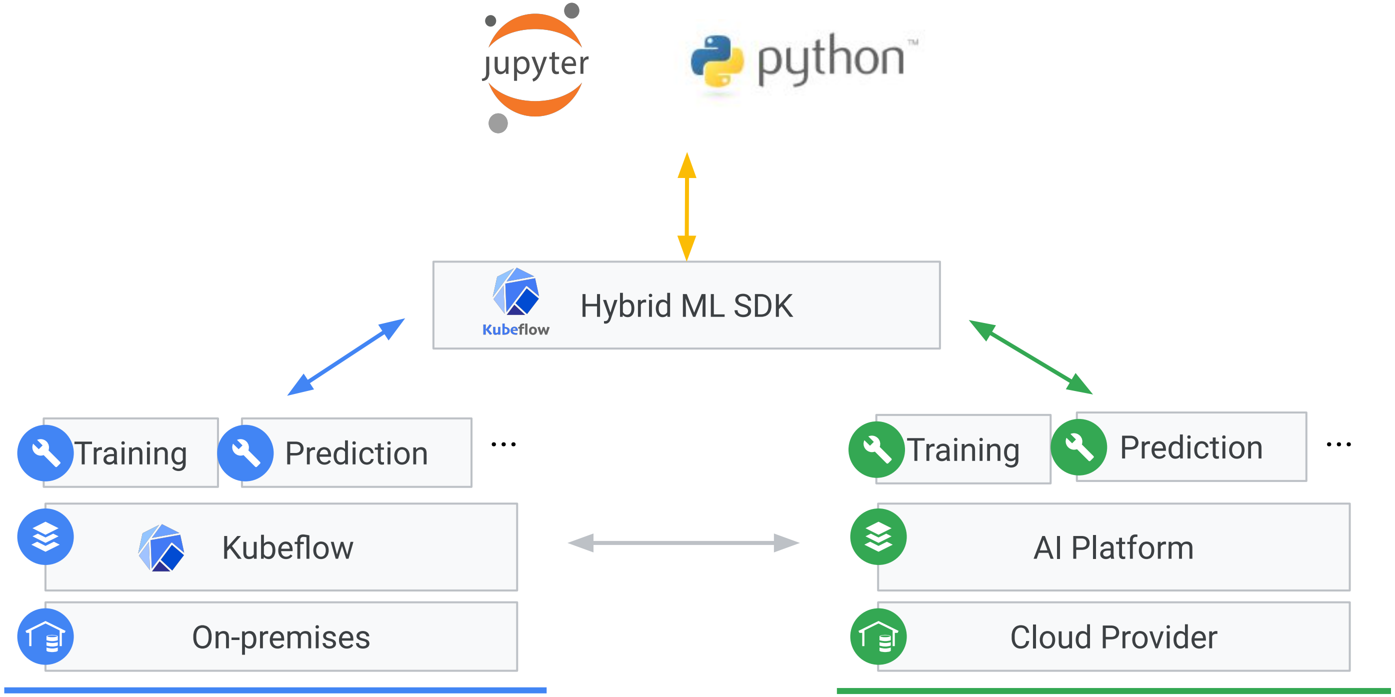
# Experimentation by multiple data scientists



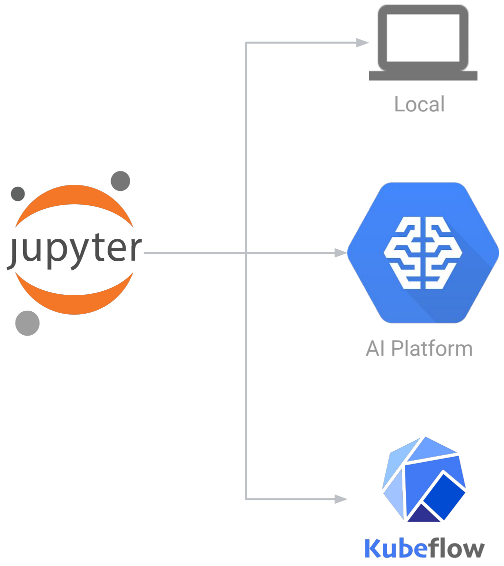
# Kubernetes can handle the complete stack



**Simple**/Fast/Cost-effective



Kubeflow Fairing is an open source Hybrid ML SDK for data scientists to 'write ML code once and run anywhere'.



# Code: Today

## Local

```python
import xgboost

class MyModel(object):
    def train(self):
       # load data
       # do feature engineering
       # train a model

    def predict():
       # prediction logic

if __name__ == '__main__':
    model = MyModel()
    model.train()
```

## Build & Deploy to AI Platform

### Training

```
gcloud ml-engine jobs submit training my_job \
          --module-name trainer.task \
          --staging-bucket gs://my-bucket \
          --package-path /my/code/path/trainer \
          --packages additional-dep1.tar.gz,dep2.whl
```

### Prediction

```
gcloud alpha ml-engine versions create
{VERSION_NAME} --model {MODEL_NAME} \ --origin
gs://{BUCKET}/{MODEL_DIR}/ \ --runtime-version
{RUNTIME_VERSION} \ --package-uris
gs://{BUCKET}/{PACKAGES_DIR}/my_package-0.2.tar.gz \
--model-class=my_model.ModelExample
```

## Build & Deploy to Kubeflow

```yaml
apiVersion: kubeflow.org/v1alpha2
kind: TFJob
metadata:
  labels:
    experiment: experiment10
  name: tfjob
  namespace: kubeflow
spec:
  tfReplicaSpecs:
    Ps:
      replicas: 1
      template:
        metadata:
          creationTimestamp: null
        spec:
          containers:
          - args:
            - python
            - tf_cnn_benchmarks.py
            image:
.
.
.
.
```

# Code: With Kubeflow Fairing

### Local

```
import xgboost

class MyModel(object):
    def train(self):
       # load data
       # train a model

    def predict():
       # prediction logic

from fairing import TrainJob
from fairing.backends import Backend

job = TrainJob(MyModel,
    backend=Backend("Local",
      "fairing.config"))
job.submit()

endpoint = PredictionEndpoint(MyModel,
    backend=Backend("Local",
      "fairing.config"))
endpoint.create()
```

### Build & Deploy to AI Platform

```
import xgboost

class MyModel(object):
    def train(self):
       # load data
       # train a model

    def predict():
       # prediction logic

from fairing import TrainJob
from fairing.backends import Backend

job = TrainJob(MyModel,
    backend=Backend("ai_platform",
      "fairing.config"))
job.submit()

endpoint = PredictionEndpoint(MyModel,
    backend=Backend("ai_platform",
      "fairing.config"))
endpoint.create()
```

### Build & Deploy to Kubeflow

```
import xgboost

class MyModel(object):
    def train(self):
       # load data
       # train a model

    def predict():
       # prediction logic

from fairing import TrainJob
from fairing.backends import Backend

job = TrainJob(MyModel,
    backend=Backend("Kubeflow",
      "fairing.config"))
job.submit()

endpoint = PredictionEndpoint(MyModel,
    backend=Backend("Kubeflow",
      "fairing.config"))
endpoint.create()
```

# Code: With Kubeflow Fairing

### Local

```
import xgboost

class MyModel(object):
    def train(self):
        # load data
        # train a model

    def predict():
        # prediction logic

from fairing import TrainJob
from fairing.backends import Backend

job = TrainJob(MyModel,
    backend=Backend("Local",
      "fairing.config"))
job.submit()

endpoint = PredictionEndpoint(MyModel,
    backend=Backend("Local",
      "fairing.config"))
endpoint.create()
```

### Build & Deploy to AI Platform

```
import xgboost

class MyModel(object):
    def train(self):
        # load data
        # train a model

    def predict():
        # prediction logic

from fairing import TrainJob
from fairing.backends import Backend

job = TrainJob(MyModel,
    backend=Backend("ai_platform",
      "fairing.config"))
job.submit()

endpoint = PredictionEndpoint(MyModel,
    backend=Backend("ai_platform",
      "fairing.config"))
endpoint.create()
```

### Build & Deploy to Kubeflow

```
import xgboost

class MyModel(object):
    def train(self):
        # load data
        # train a model

    def predict():
        # prediction logic

from fairing import TrainJob
from fairing.backends import Backend

job = TrainJob(MyModel,
    backend=Backend("Kubeflow",
      "fairing.config"))
job.submit()

endpoint = PredictionEndpoint(MyModel,
    backend=Backend("Kubeflow",
      "fairing.config"))
endpoint.create()
```

# Kubeflow Fairing

An open source Hybrid ML SDK for data scientists to ‘write ML code once and run anywhere’

**Data Scientist Focused:** Simple and uses language familiar to Data Scientists

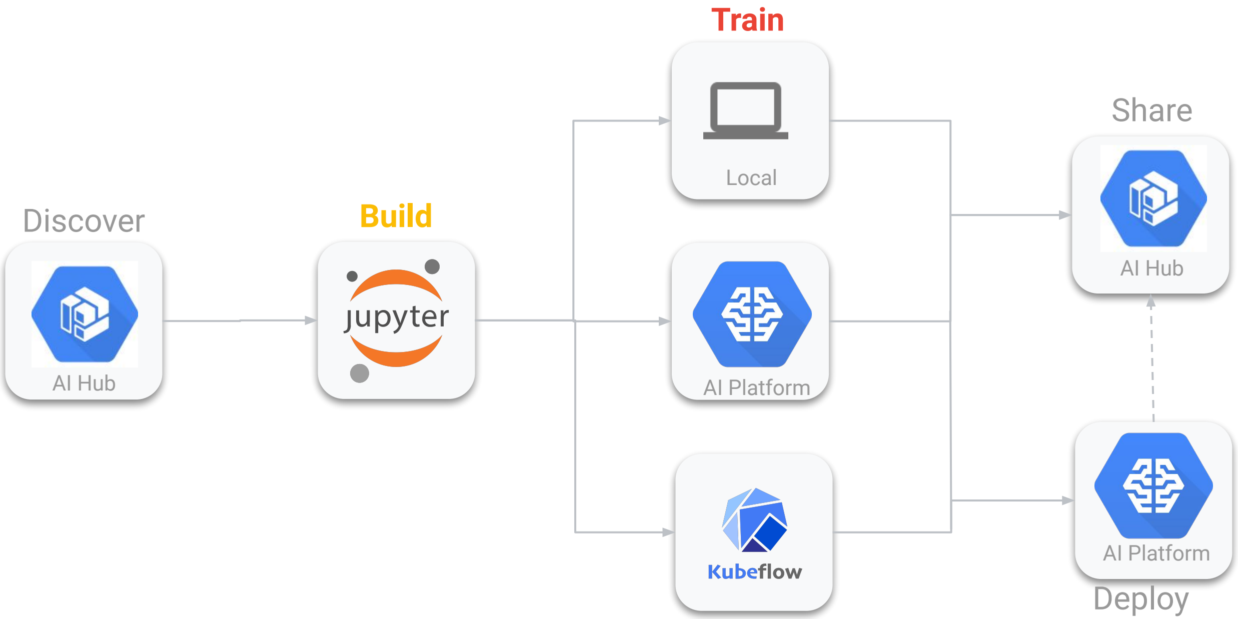
**Multi-Platform:** Supports AI Platform and Kubleflow, making it easy for users to switch between on-prem and GCP.

**Scalable and Cost Effective:** Data Scientists can easily burst onto GCP when they need more resources (i.e. more machines, GPUs, or TPUs).

**Easily Train, Tune and Deploy models:** Supports the full ML lifecycle.

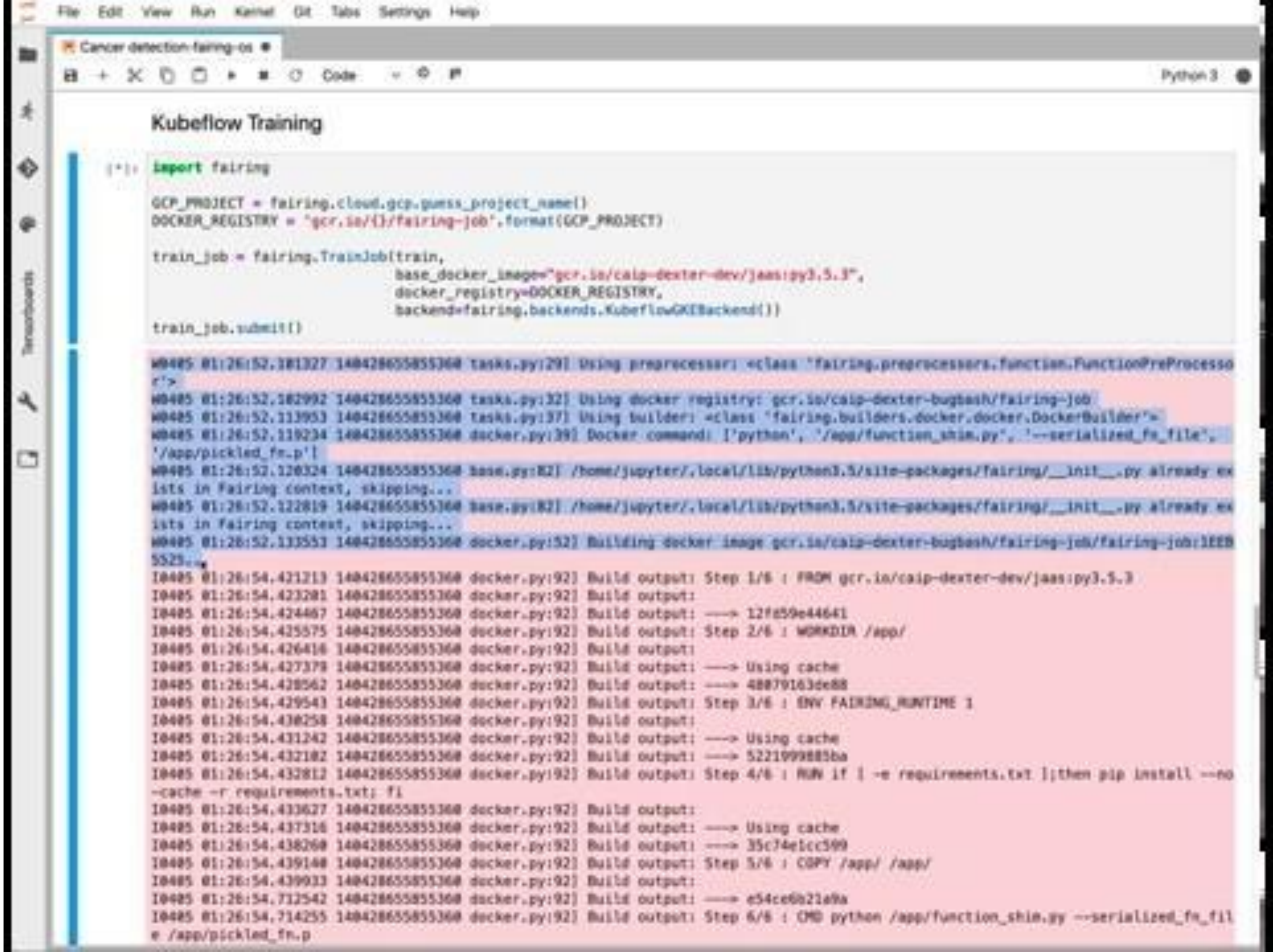
**Multi-Framework:** Supports XGBoost, TensorFlow (single node), and Pytorch (single node).

# Demo: Hybrid E2E ML with Kubeflow Fairing




Google Cloud

# DEMO



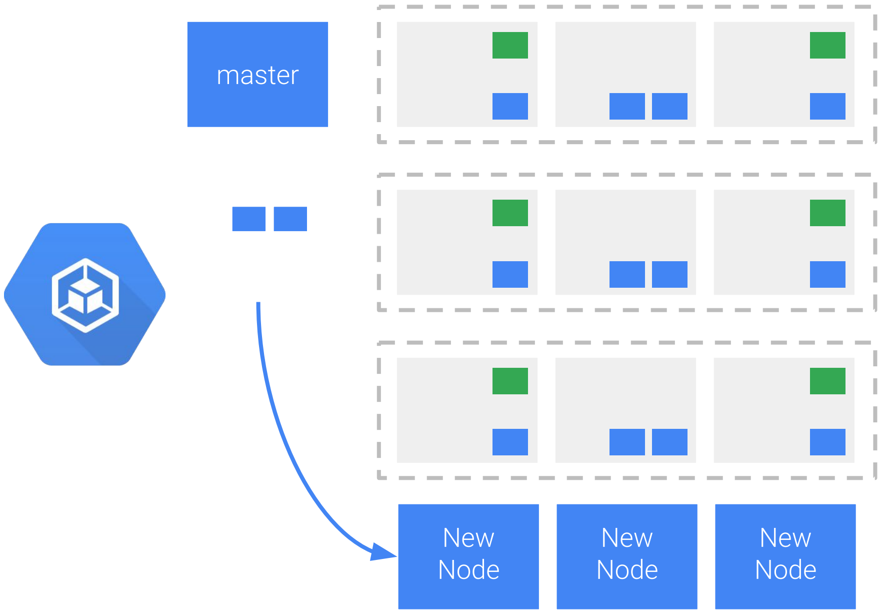

Google Cloud

# Kubeflow Fairing: Key Benefits for ML Ops Teams

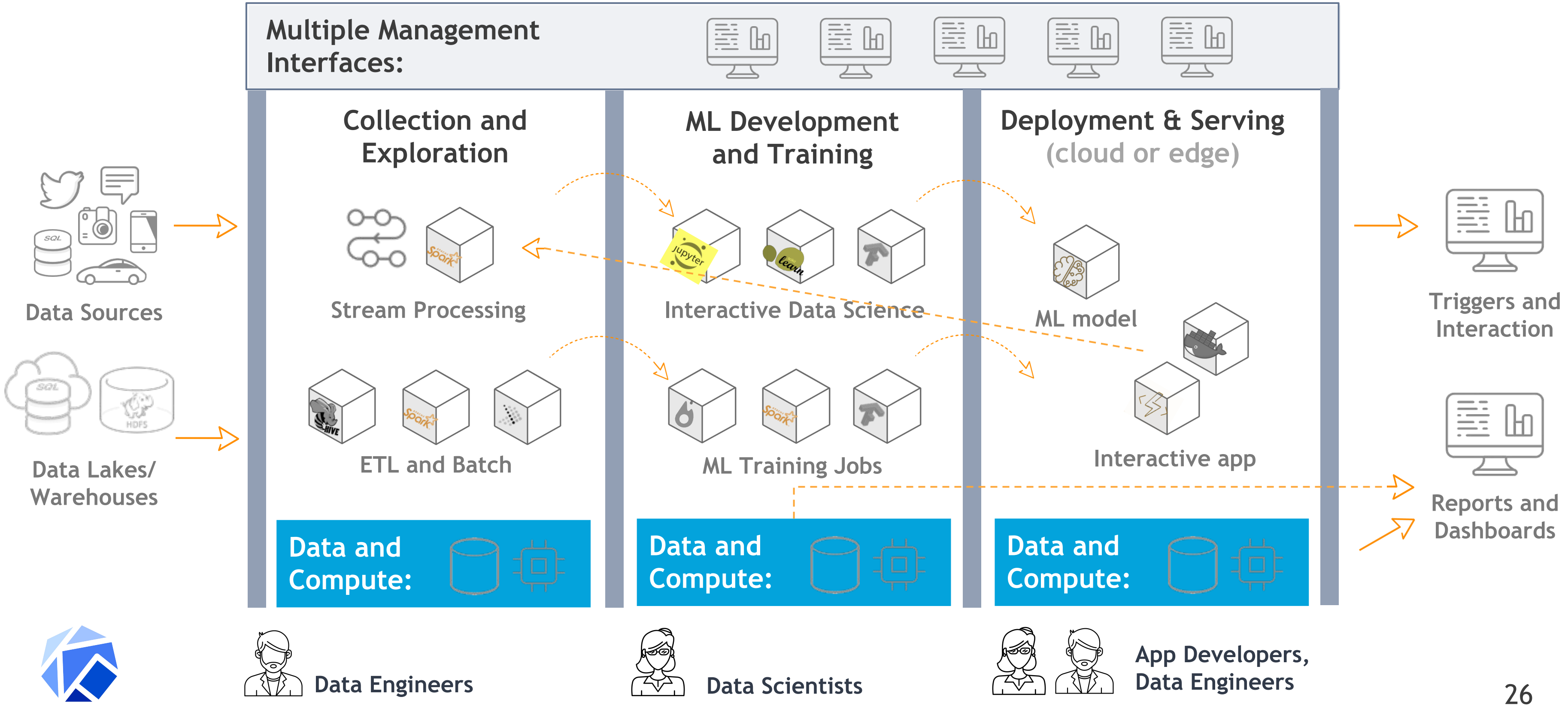
1. Standardized API Enforces Best Practices
2. Open Source SDK --> No Lock-in
3. Easy ‘Remoting’ & Bursting to the Cloud

# Cluster autoscaler with GPUs and TPUs

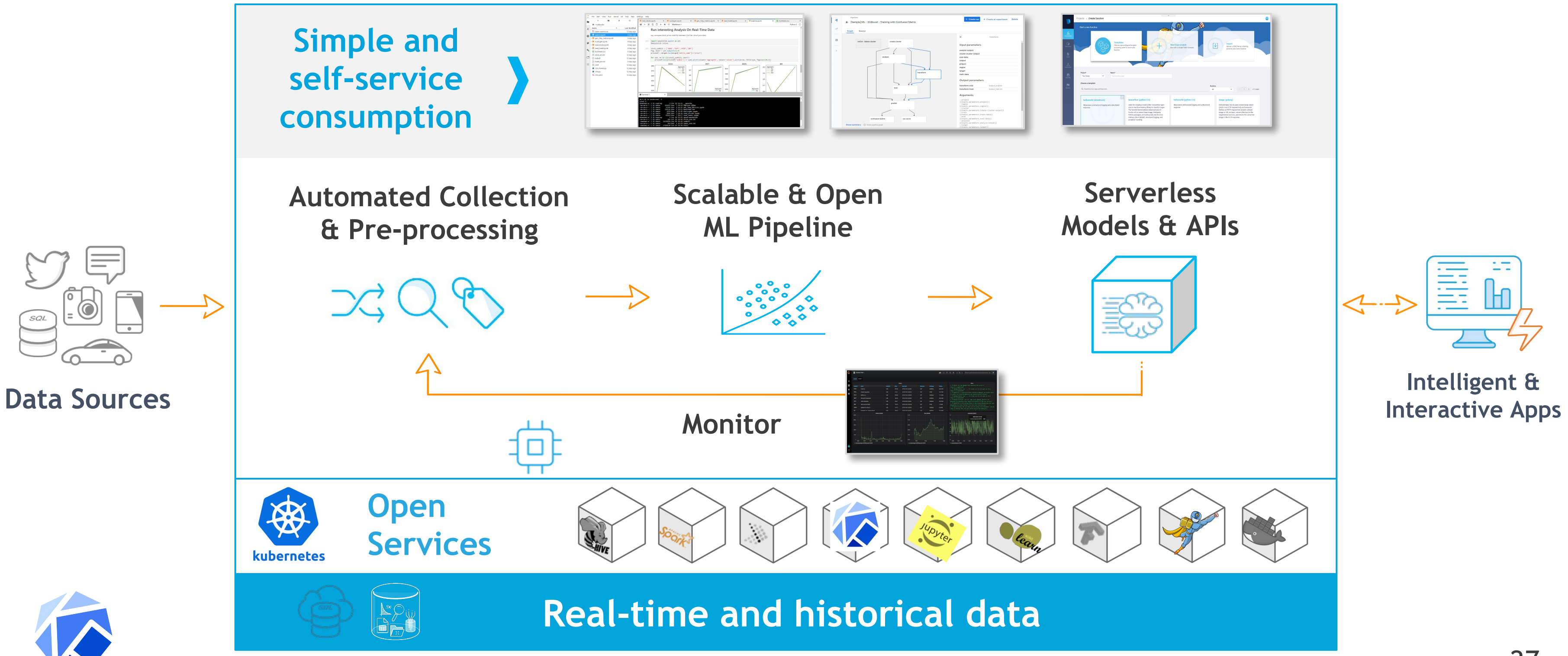
- Automatically **scale up/down** the cluster for the **best performance over cost**
- Nodes with **GPUs/TPUs** get **created** when a cluster needs **more capacity**
- Nodes with **GPUs/TPUs** get **deleted** when they're **idle**



# Today: ML Pipeline is Complex and Siloed



# Kubernetes: One Platform, Complete ML Lifecycle



# Open-Source ML Pipeline Components By Category

**Dev Tools**
<..>

- Istio
- Katib
- jupyter
- Kubeflow Pipelines

**Data Ingest & Prep**

- nuclio
- Apache Spark
- Pachyderm
- DASK

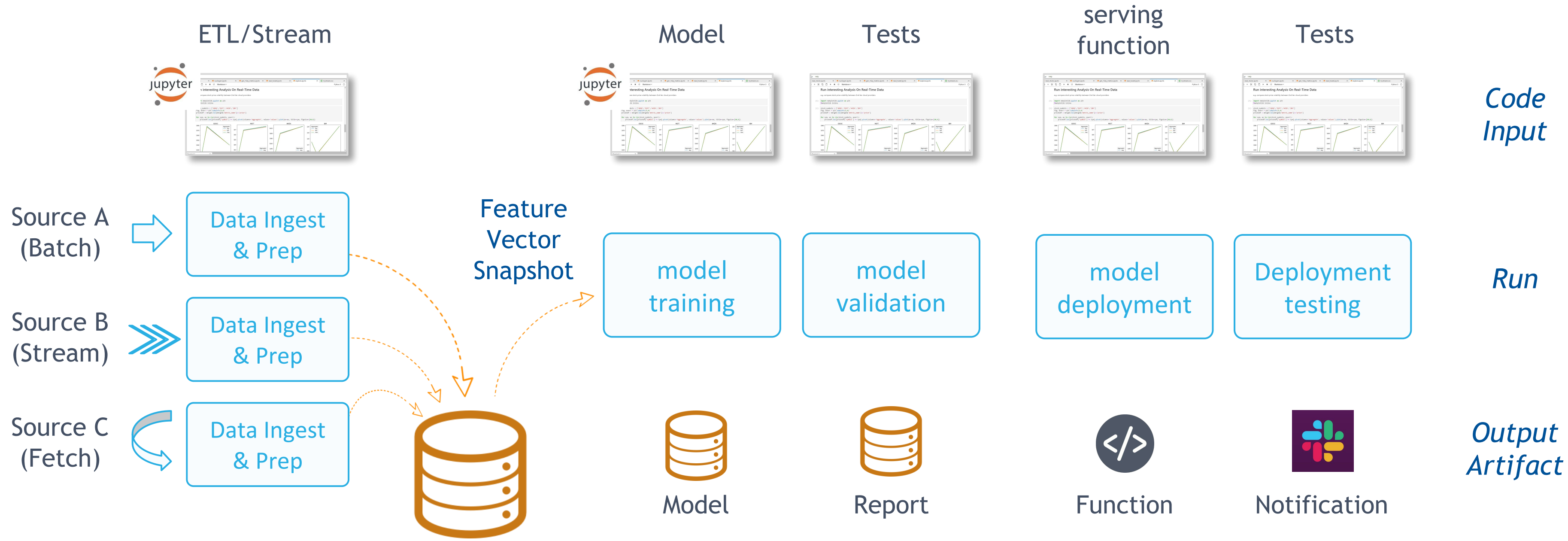
**Training & Validation**

- MXNet
- PyTorch
- TensorFlow
- Chainer
- DASK
- scikit learn contrib
- HOROVOD

**model serving**

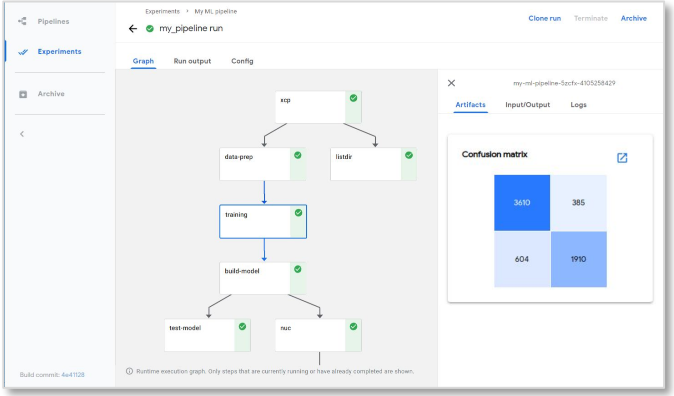
- SELDON
- nuclio
- NVIDIA TensorRT

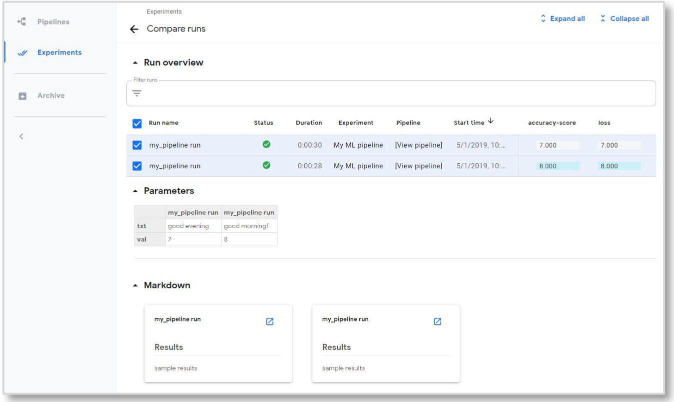
# Typical Data Science Pipeline




**Pipeline must be automated !**

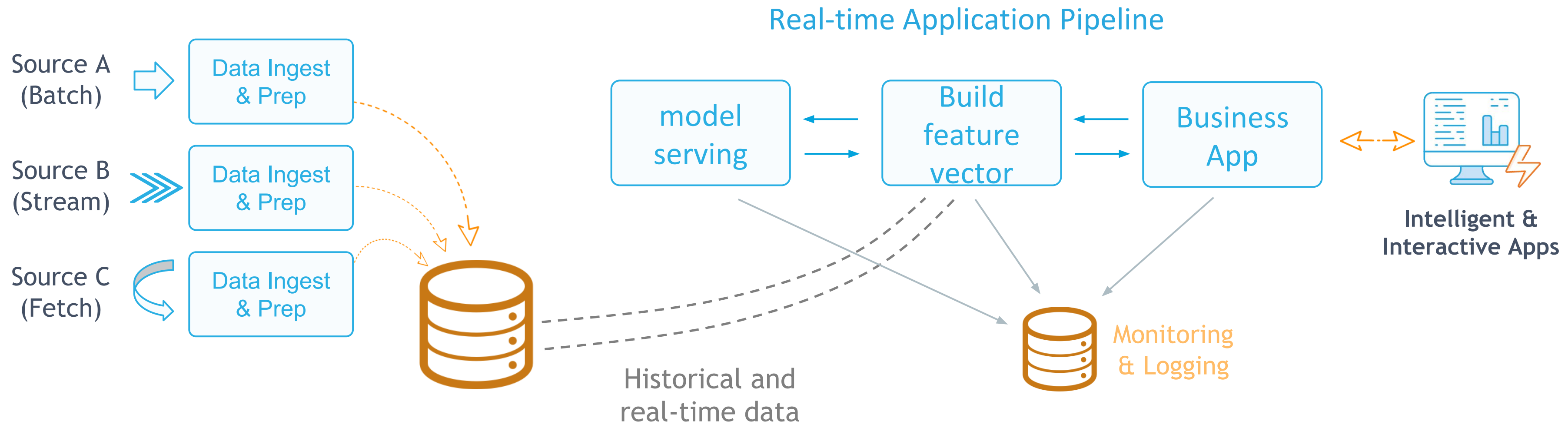
# KubeFlow Pipeline




- Advanced workflow engine and experiment management in one tool
- Convert python code to workflows
- Reusable component library
- Managing multiple runs, compare artifacts and results between runs
- Steps can be containers, code scripts, CRDs (e.g. TFJob), and now functions

# Application Serving Environment, More Challenges




**Scale, performance, online updates, monitoring, security…**

# Serverless A Way To Simplify Data Science

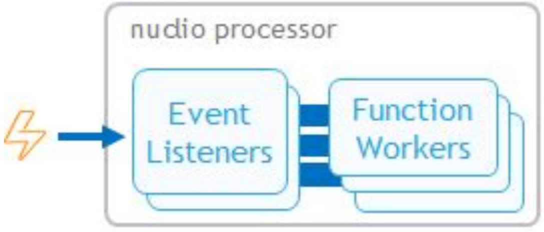
- Automate process from code to container and assigned cluster resources
- Add instrumentation with minimal developer overhead
- Auto scaling, rolling upgrades, …

## Sounds Ideal So Why Not?

- Slow performance, lack of concurrency, no GPUs
- Stateless, limit application patterns
- No stream processing support (mostly HTTP)
- Hard to debug and diagnose and build dependencies

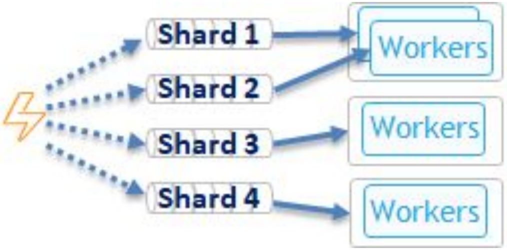
# Nuclio: Taking Serverless to Data Intensive Apps

## Extreme Performance




- Non-blocking, parallel
- Zero copy, buffer reuse
- Up to 400K events/sec/proc
- GPU optimizations

## Advanced Data & AI Features

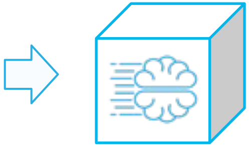



- Auto-rebalance, checkpoints
- Any source: Kafka, NATS, Kinesis, event-hub, iguazio, pub/sub, RabbitMQ, Cron, ..
- Jupyter, NVIDIA Rapids integration

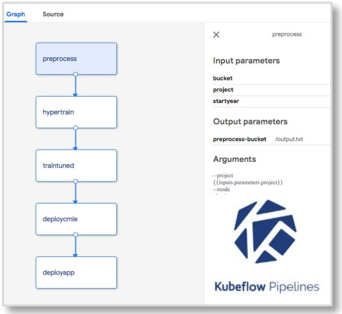
## Statefulness


- Data bindings
- Shared volumes
- Context cache

# Nuclio Automating & Accelerating Data Science



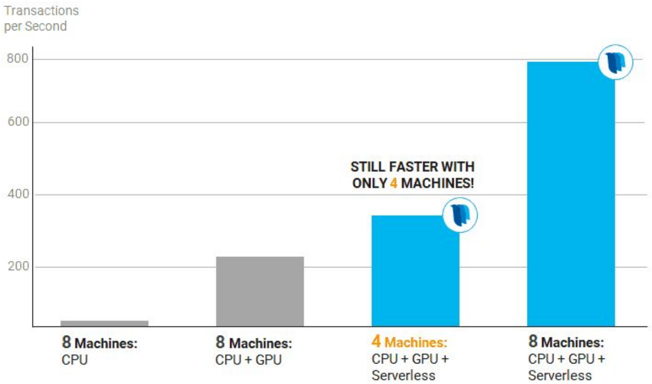
One magic command from notebook to function




Extending Pipelines from batch:

1. Parallel processing steps
2. Code build/deployment steps
3. Stream processing

GPU resource optimization for ETL, DL and ML

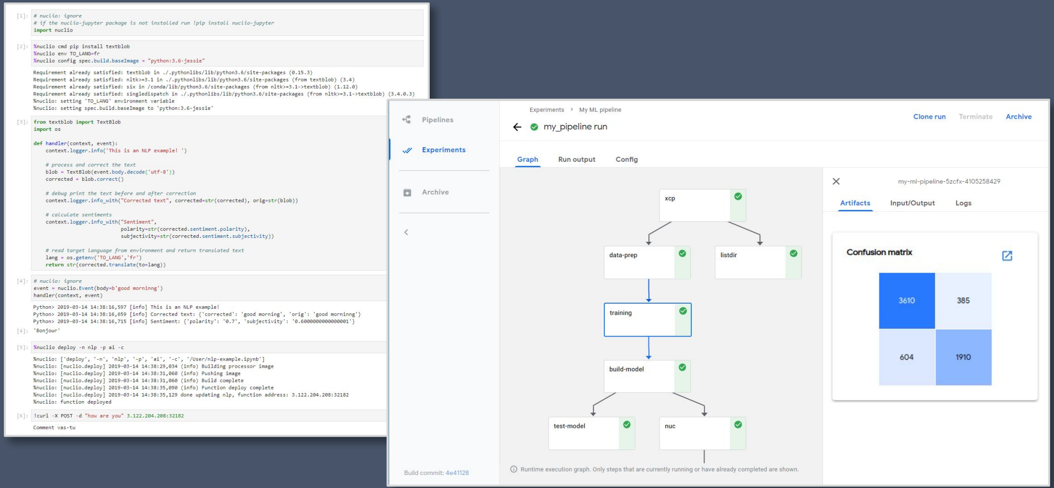




Automation:

1. Auto-scaling (to zero)
2. Automated logging & monitoring
3. Security hardening
4. Auto-build and CI/CD
5. Workload mobility (cloud/edge/..)

# Demo: Building an end to end ML pipeline in minutes with KubeFlow



# Empowering your teams to drive innovation

**Simple**

- Data Scientist friendly notebooks
- Freedom from managing infrastructure
- TFJob, TFServing, ...

**Fast**

- On-demand scale up and down
- GPUs and TPUs

**Cost-effective**

- Making AI/ML teams more productive
- Avoid vendor lock-in with open platform
- Write once run anywhere
- Preemptible GPUs/TPUs