Services for Machine Learning applications (part 3 of 3)

Raulian-Ionut Chiorescu IT-CD-PI

Kubeflow Components and Features

Notebooks

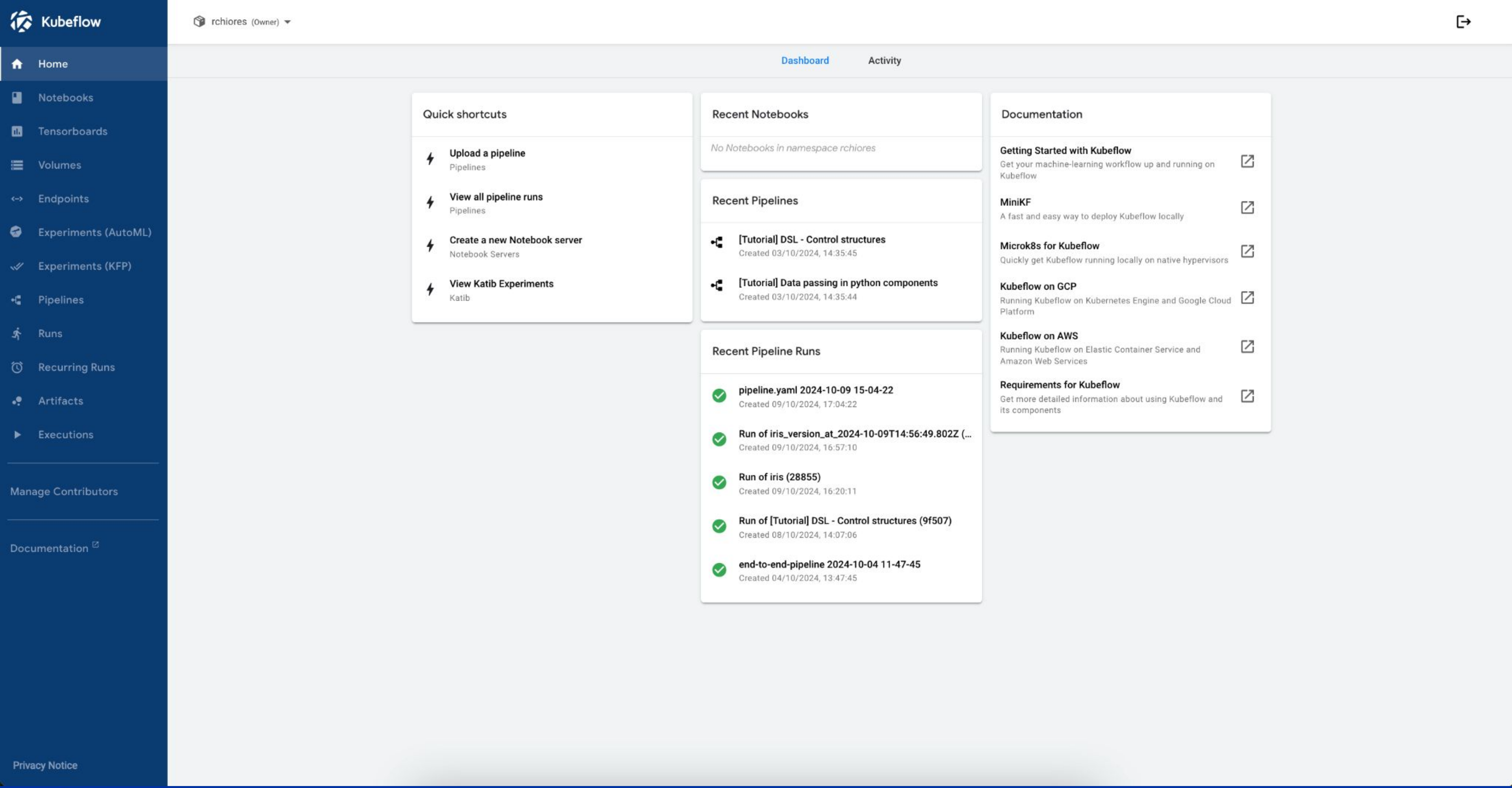
Machine Learning Pipelines

AutoML - Hyperparameter Optimization

Distributed Training

Tensorboards

Model Serving


Kubeflow
rchiores (Owner)
Home
Notebooks
Tensorboards
Volumes
Endpoints
Experiments (AutoML)
Experiments (KFP)
Pipelines
Runs
Recurring Runs
Artifacts
Executions
Manage Contributors
Documentation
Privacy Notice
Dashboard
Activity
Quick shortcuts
Upload a pipeline
Pipelines
View all pipeline runs
Pipelines
Create a new Notebook server
Notebook Servers
View Katib Experiments
Katib
Recent Notebooks
No Notebooks in namespace rchiores
Recent Pipelines
[Tutorial] DSL - Control structures
Created 03/10/2024, 14:35:45
[Tutorial] Data passing in python components
Created 03/10/2024, 14:35:44
Recent Pipeline Runs
pipeline.yaml 2024-10-09 15-04-22
Created 09/10/2024, 17:04:22
Run of iris_version_at_2024-10-09T14:56:49.802Z (...
Created 09/10/2024, 16:57:10
Run of iris (28855)
Created 09/10/2024, 16:20:11
Run of [Tutorial] DSL - Control structures (9f507)
Created 08/10/2024, 14:07:06
end-to-end-pipeline 2024-10-04 11-47-45
Created 04/10/2024, 13:47:45
Documentation
Getting Started with Kubeflow
Get your machine-learning workflow up and running on Kubeflow
MiniKF
A fast and easy way to deploy Kubeflow locally
Microk8s for Kubeflow
Quickly get Kubeflow running locally on native hypervisors
Kubeflow on GCP
Running Kubeflow on Kubernetes Engine and Google Cloud Platform
Kubeflow on AWS
Running Kubeflow on Elastic Container Service and Amazon Web Services
Requirements for Kubeflow
Get more detailed information about using Kubeflow and its components

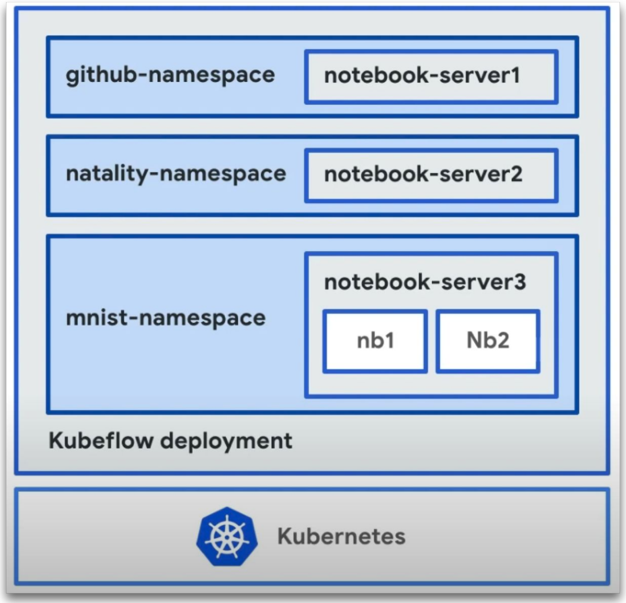
Kubeflow Notebooks

Features:

- Fully customizable environments
- Select resources (CPU,MEM,GPU)
- Selectable GPU flavors
- Integrated EOS storage

Use Cases:

- Quick prototyping
- Exploratory data analysis
- Small Model training



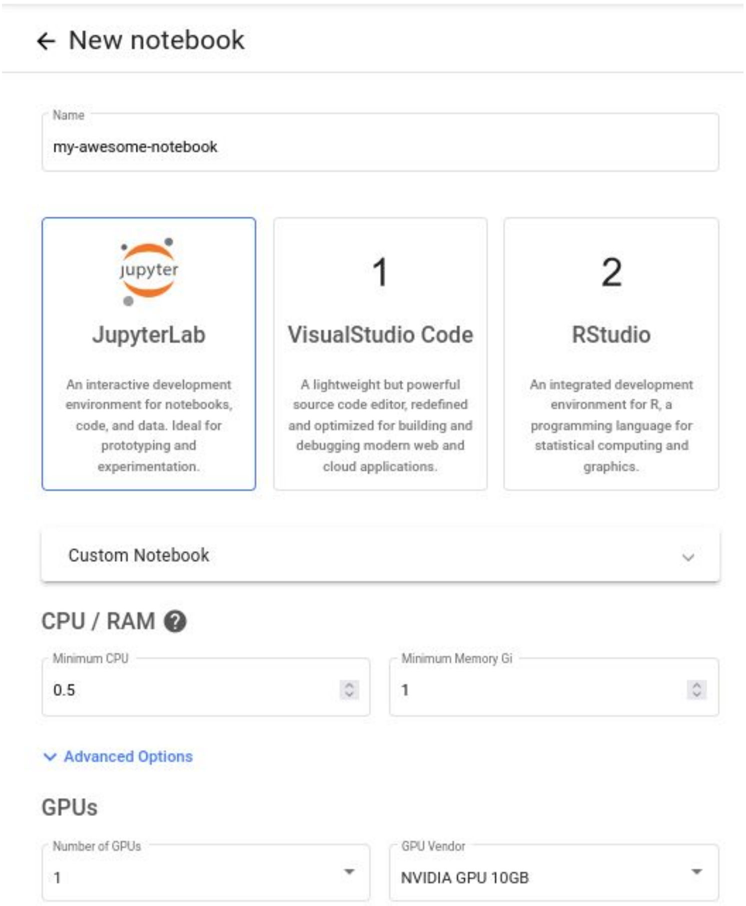
Customizing Notebooks

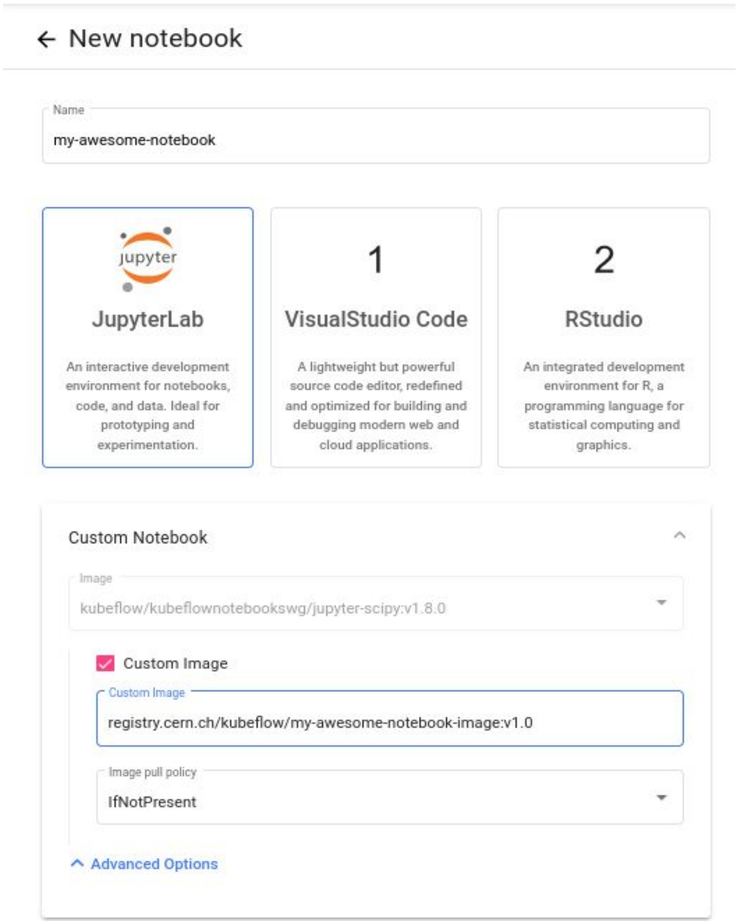
Pre-built Images:

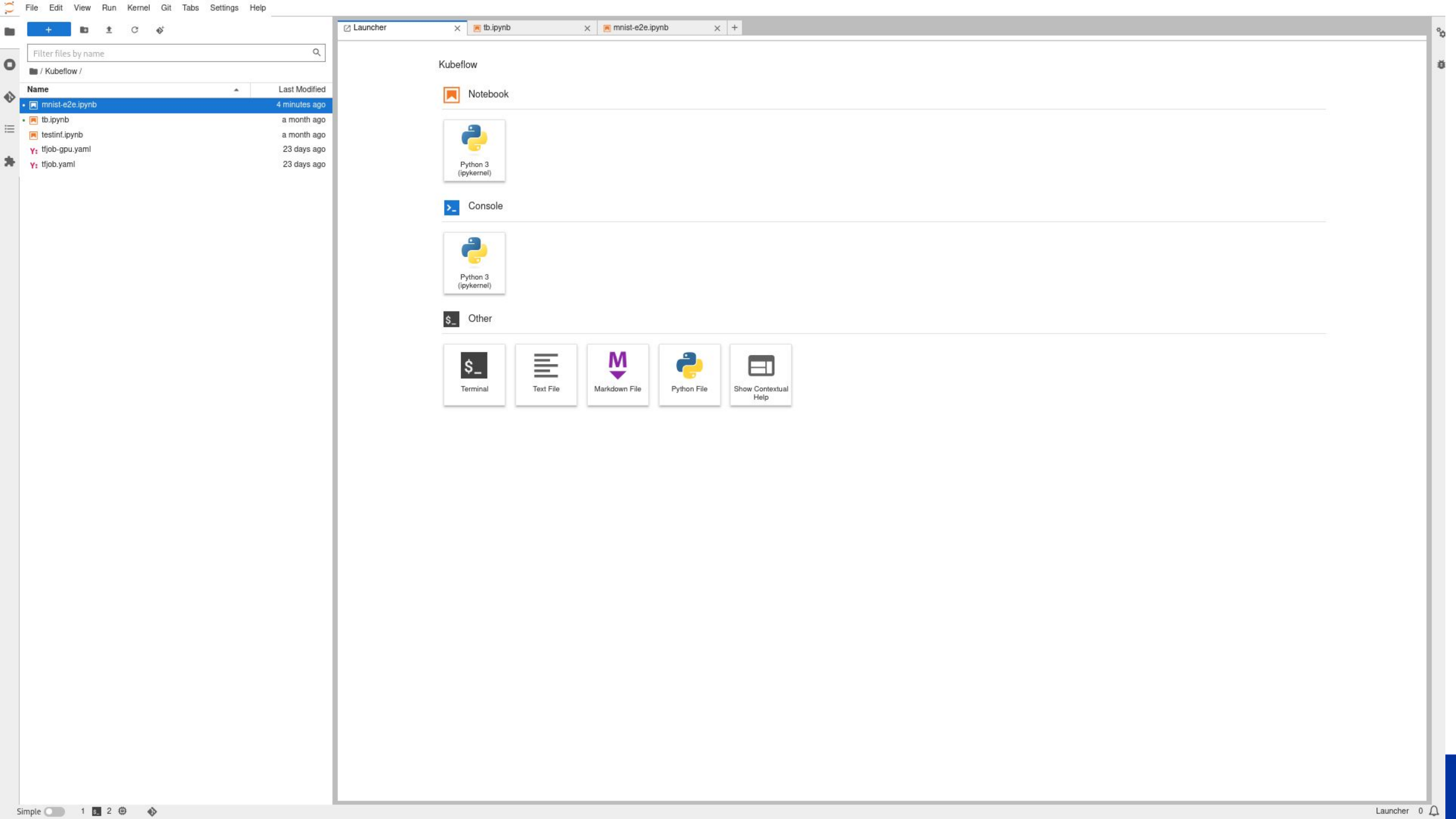
- PyTorch
- TensorFlow
- SciPy

Customization:

- **pip install** additional packages
- Build custom images for specific needs


← New notebook
Name
my-awesome-notebook
jupyter
JupyterLab
An interactive development environment for notebooks, code, and data. Ideal for prototyping and experimentation.
1
VisualStudio Code
A lightweight but powerful source code editor, redefined and optimized for building and debugging modern web and cloud applications.
2
RStudio
An integrated development environment for R, a programming language for statistical computing and graphics.
Custom Notebook
CPU / RAM
Minimum CPU
0.5
Minimum Memory Gi
1
Advanced Options
GPUs
Number of GPUs
1
GPU Vendor
NVIDIA GPU 10GB


← New notebook
Name
my-awesome-notebook
jupyter
JupyterLab
An interactive development environment for notebooks, code, and data. Ideal for prototyping and experimentation.
1
VisualStudio Code
A lightweight but powerful source code editor, redefined and optimized for building and debugging modern web and cloud applications.
2
RStudio
An integrated development environment for R, a programming language for statistical computing and graphics.
Custom Notebook
Image
kubeflow/kubeflownotebookswg/jupyter-scipy:v1.8.0
Custom Image
Custom Image
registry.cern.ch/kubeflow/my-awesome-notebook-image:v1.0
Image pull policy
IfNotPresent
Advanced Options

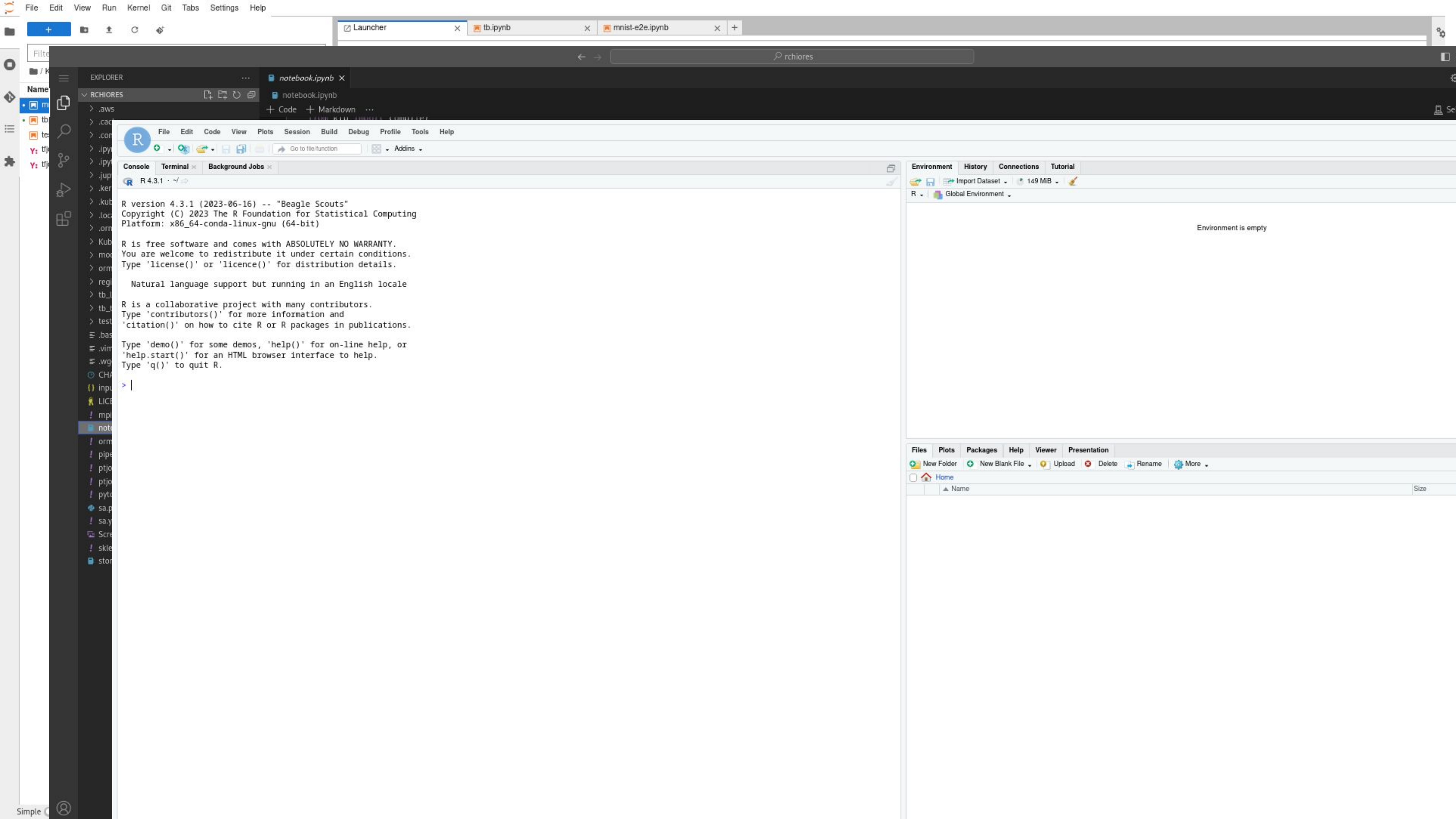

File Edit View Run Kernel Git Tabs Settings Help
Filter files by name
/ Kubeflow /
Name
Last Modified
mnist-e2e.ipynb 4 minutes ago
tb.ipynb a month ago
testinf.ipynb a month ago
tfjob-gpu.yaml 23 days ago
tfjob.yaml 23 days ago
Launcher
tb.ipynb
mnist-e2e.ipynb
Kubeflow
Notebook
Python 3 (ipykernel)
Console
Python 3 (ipykernel)
Other
Terminal
Text File
Markdown File
Python File
Show Contextual Help
Simple
1
2
Launcher
0

```python
from kfp import compiler
from kfp.client import Client
from kfp import client
from kfp import dsl
from kfp.dsl import Dataset
from kfp.dsl import Input
from kfp.dsl import Model
from kfp.dsl import Output
```

[2]

Design the Pipeline

```python
@dsl.component(packages_to_install=['pandas==1.3.5'])
def create_dataset(iris_dataset: Output[Dataset]):
    import pandas as pd

    csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
    col_names = [
        'Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width', 'Labels'
    ]
    df = pd.read_csv(csv_url, names=col_names)

    with open(iris_dataset.path, 'w') as f:
        df.to_csv(f)


@dsl.component(packages_to_install=['pandas==1.3.5', 'scikit-learn==1.0.2'])
def normalize_dataset(
    input_iris_dataset: Input[Dataset],
    normalized_iris_dataset: Output[Dataset],
    standard_scaler: bool,
    min_max_scaler: bool,
):
    if standard_scaler is min_max_scaler:
        raise ValueError(
            'Exactly one of standard_scaler or min_max_scaler must be True.')

    import pandas as pd
    from sklearn.preprocessing import MinMaxScaler
    from sklearn.preprocessing import StandardScaler

    with open(input_iris_dataset.path) as f:
        df = pd.read_csv(f)
    labels = df.pop('Labels')

    if standard_scaler:
        scaler = StandardScaler()
    if min_max_scaler:
        scaler = MinMaxScaler()

    df = pd.DataFrame(scaler.fit_transform(df))
    df['Labels'] = labels
    with open(normalized_iris_dataset.path, 'w') as f:
        df.to_csv(f)


@dsl.component(packages_to_install=['pandas==1.3.5', 'scikit-learn==1.0.2'])
def train_model(
```


File Edit View Run Kernel Git Tabs Settings Help
Launcher
tb.ipynb
mnist-e2e.ipynb
rchiores
EXPLORER
RCHIORES
notebook.ipynb
+ Code + Markdown
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Console
Terminal
Background Jobs
R 4.3.1 · ~/
R version 4.3.1 (2023-06-16) -- "Beagle Scouts"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-conda-linux-gnu (64-bit)
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.
Natural language support but running in an English locale
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.
Environment
History
Connections
Tutorial
Import Dataset
149 MiB
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Presentation
New Folder
New Blank File
Upload
Delete
Rename
More
Home
Name
Size
Simple

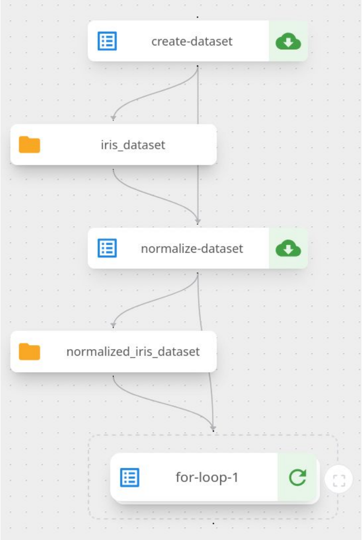
Kubeflow Pipelines

What Are Pipelines?

- Directed acyclic graph (DAG) workflows for ML tasks.
- Flexible dependency management (e.g. parallel training, data streams).

Compared to Notebooks:

- Reproducibility.
- Parallelism for time efficiency.
- Scalability for large datasets and models.



Kubeflow Pipelines

Python script → Compiled to YAML → Submitted for execution.

Concepts:

- Experiments: Group multiple pipeline runs
- Runs: Individual executions of a pipeline
- Pipeline Parameters: Allow dynamic input (e.g., data paths, hyperparameters)

Wrapping your Python Script with Pipeline Components

```python
@dsl.component(packages_to_install=['pandas==1.3.5'])
def create_dataset(iris_dataset: Output[Dataset]):
    import pandas as pd

    csv_url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
    col_names = [
        'Sepal_Length', 'Sepal_Width', 'Petal_Length', 'Petal_Width', 'Labels'
    ]
    df = pd.read_csv(csv_url, names=col_names)

    with open(iris_dataset.path, 'w') as f:
        df.to_csv(f)
```

Wrapping your Python Script with Pipeline Components

```python
@dsl.component(packages_to_install=['pandas==1.3.5', 'scikit-learn==1.0.2'])
def normalize_dataset(
    input_iris_dataset: Input[Dataset],
    normalized_iris_dataset: Output[Dataset],
    standard_scaler: bool,
    min_max_scaler: bool,
):
    if standard_scaler is min_max_scaler:
        raise ValueError(
            'Exactly one of standard_scaler or min_max_scaler must be True.')

    import pandas as pd
    from sklearn.preprocessing import MinMaxScaler
    from sklearn.preprocessing import StandardScaler

    with open(input_iris_dataset.path) as f:
        df = pd.read_csv(f)
    labels = df.pop('Labels')

    if standard_scaler:
        scaler = StandardScaler()
    if min_max_scaler:
        scaler = MinMaxScaler()

    df = pd.DataFrame(scaler.fit_transform(df))
    df['Labels'] = labels
    with open(normalized_iris_dataset.path, 'w') as f:
        df.to_csv(f)
```

Wrapping your Python Script with Pipeline Components

```python
@dsl.component(packages_to_install=['pandas==1.3.5', 'scikit-learn==1.0.2'])
def train_model(
    normalized_iris_dataset: Input[Dataset],
    model: Output[Model],
    n_neighbors: int,
):
    import pickle

    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.neighbors import KNeighborsClassifier

    with open(normalized_iris_dataset.path) as f:
        df = pd.read_csv(f)

    y = df.pop('Labels')
    X = df

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    clf = KNeighborsClassifier(n_neighbors=n_neighbors)
    clf.fit(X_train, y_train)
    with open(model.path, 'wb') as f:
        pickle.dump(clf, f)
```

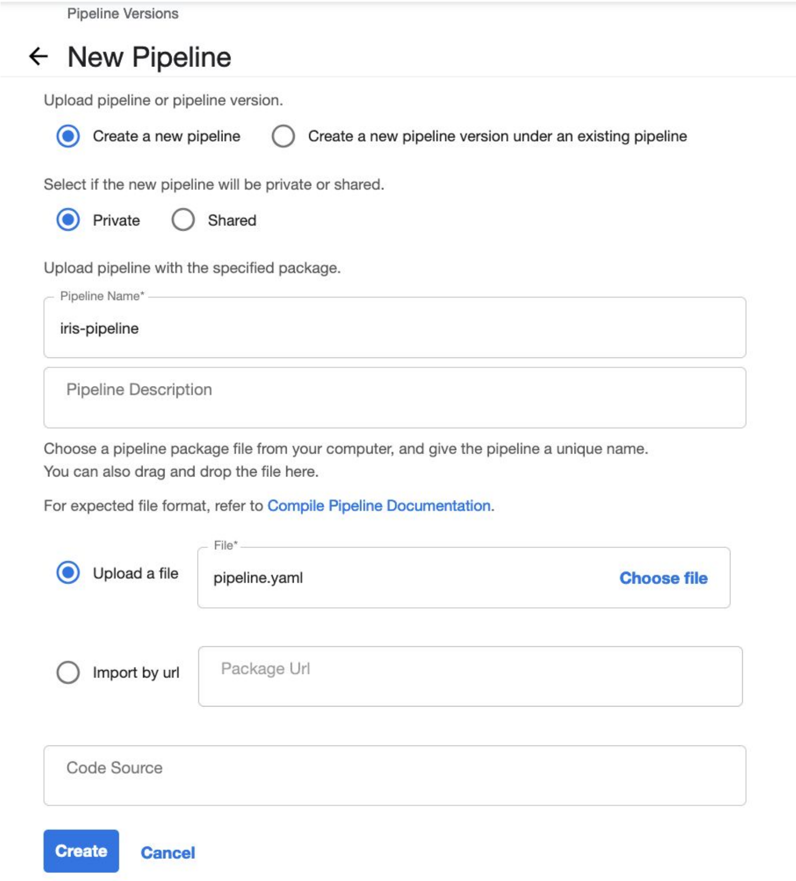
Defining Your Kubeflow Pipeline

```
@dsl.pipeline(name='iris-training-pipeline')
def my_pipeline(
    standard_scaler: bool,
    min_max_scaler: bool,
    neighbors: List[int],
):
    create_dataset_task = create_dataset()

    normalize_dataset_task = normalize_dataset(
        input_iris_dataset=create_dataset_task.outputs['iris_dataset'],
        standard_scaler=standard_scaler,
        min_max_scaler=min_max_scaler)

    with dsl.ParallelFor(neighbors) as n_neighbors:
        train_model(
            normalized_iris_dataset=normalize_dataset_task
            .outputs['normalized_iris_dataset'],
            n_neighbors=n_neighbors)
```

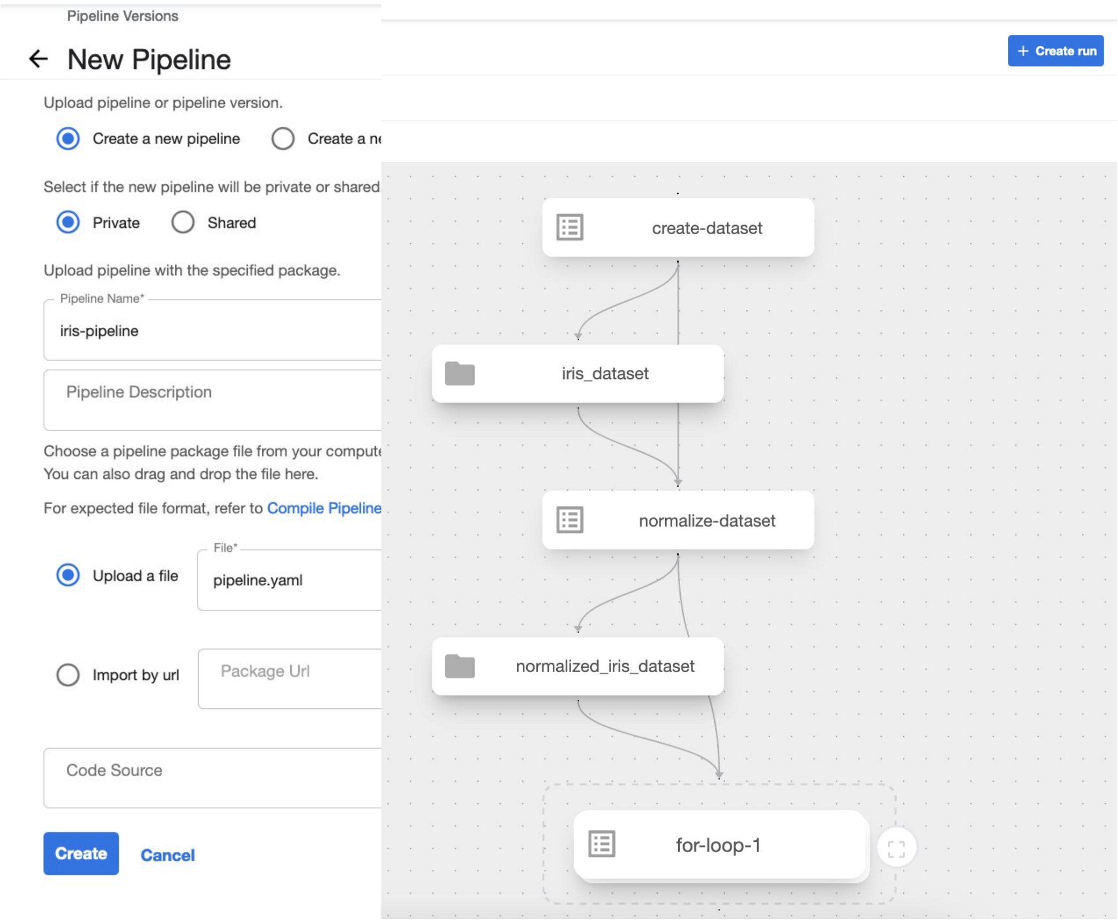
Compiling your Python Script into YAML

```python
if __name__ == '__main__':
  import kfp.compiler as compiler
  compiler.Compiler().compile(my_pipeline, __file__ + '.yaml')
```

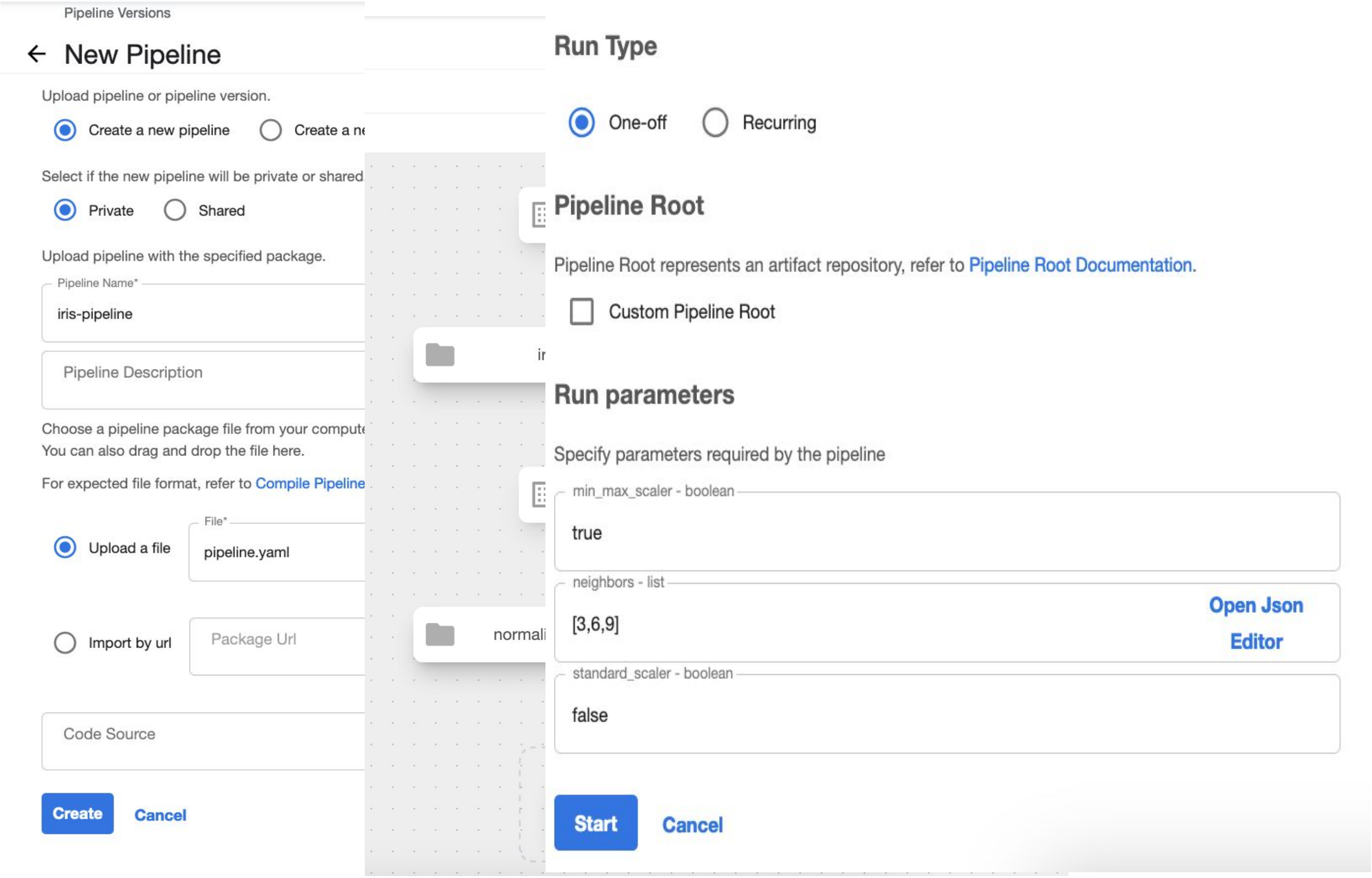
Running your Pipeline



Running your Pipeline



Running your Pipeline



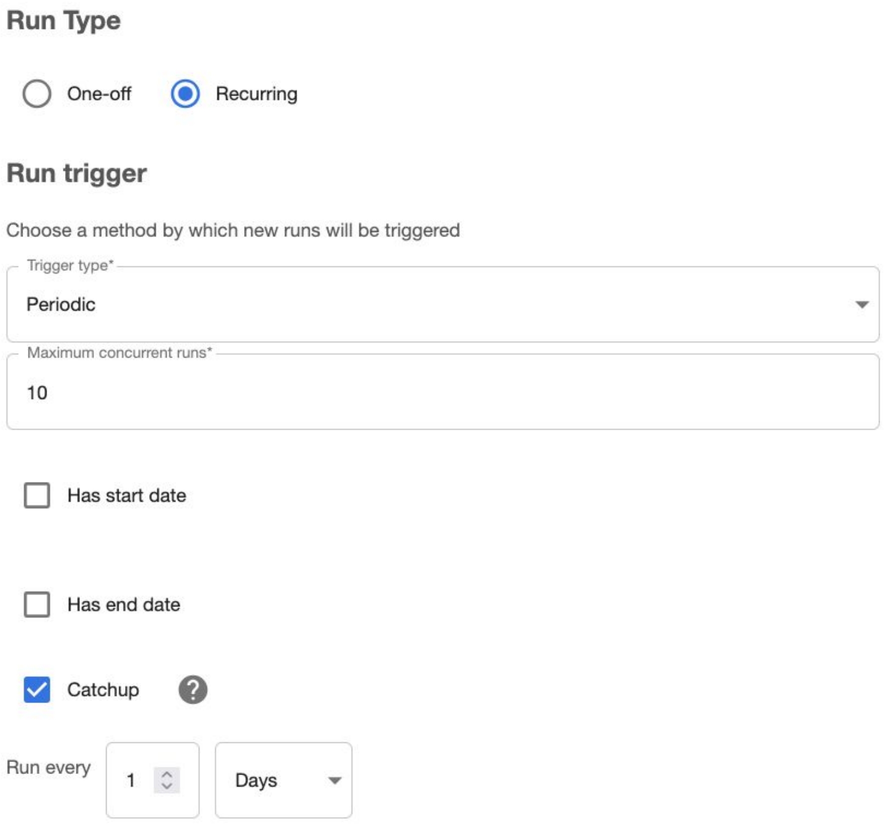
Recurring Pipelines

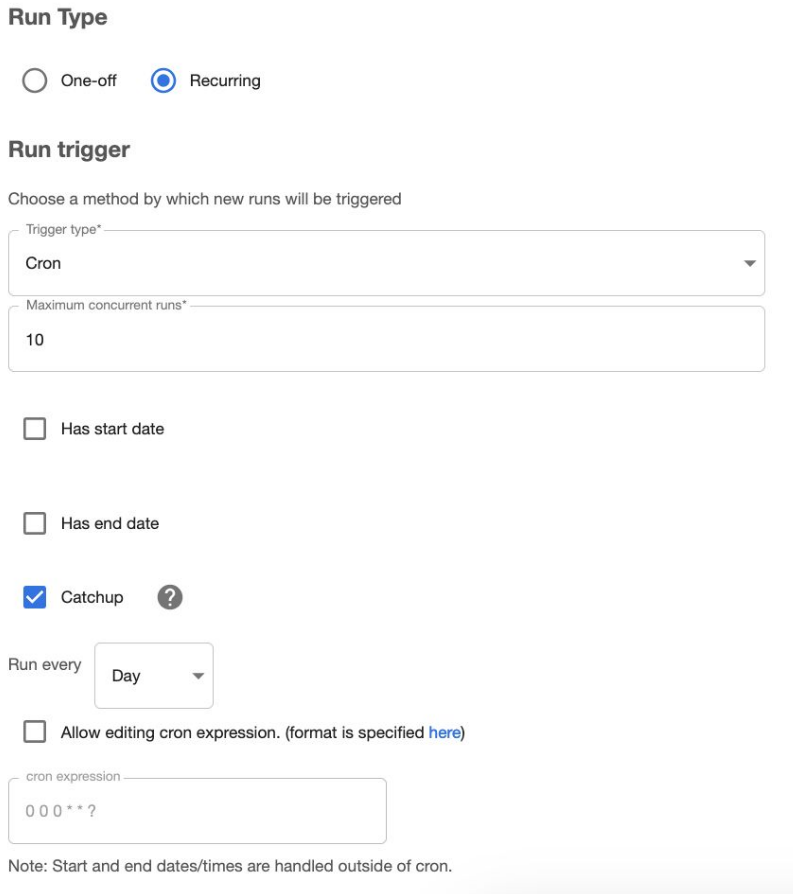
Automate repetitive tasks:

- Daily model training
- Weekly data refreshes

Triggering Options:

Set intervals, start/end times, or use cron syntax


Run Type
One-off
Recurring
Run trigger
Choose a method by which new runs will be triggered
Trigger type*
Periodic
Maximum concurrent runs*
10
Has start date
Has end date
Catchup
Run every
1
Days


Run Type
One-off
Recurring
Run trigger
Choose a method by which new runs will be triggered
Trigger type*
Cron
Maximum concurrent runs*
10
Has start date
Has end date
Catchup
Run every
Day
Allow editing cron expression. (format is specified here)
cron expression
0 0 0 * * ?
Note: Start and end dates/times are handled outside of cron.

Model Training

Two Approaches to Model Training:

- Classical (Single-Node) Training
- Distributed Training

Why the Distinction Matters:

Classical: Simpler, but limited by single machine resources
Distributed: Scales across multiple nodes for large datasets and models

Classical Training

Overview:

- Training runs on a single node
- Suitable for smaller datasets and models

Limitations:

- Memory and computation constrained by a single machine
- Slower for large models or datasets

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 20, 5, 1)
        self.conv2 = nn.Conv2d(20, 50, 5, 1)
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2, 2)
        x = x.view(-1, 4*4*50)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

def train(args, model, device, train_loader, optimizer, epoch, writer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tloss={:.4f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
            niter = epoch * len(train_loader) + batch_idx
            writer.add_scalar('loss', loss.item(), niter)

def test(args, model, device, test_loader, writer, epoch):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item() #
sum up batch loss
            pred = output.max(1, keepdim=True)[1] # get the index of the max
log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    print('\naccuracy={:.4f}\n'.format(float(correct) /
len(test_loader.dataset)))
    writer.add_scalar('accuracy', float(correct) / len(test_loader.dataset),
epoch)
```

```
FROM registry.cern.ch/kubeflow/kubeflownotebookswg/jupyter-pytorch-cuda-
full:v1.8.0

USER root

ENV NB_PREFIX /

RUN apt-get -qq update
RUN DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends
apt-utils

ENV SHELL /bin/bash

COPY requirements.txt /requirements.txt
RUN pip3 install -r /requirements.txt

COPY mnist.py /

RUN echo "jovyan ALL=(ALL:ALL) NOPASSWD:ALL" > /etc/sudoers.d/jovyan
WORKDIR /home/jovyan
USER jovyan
```

```yaml
apiVersion: "kubeflow.org/v1"
kind: "PyTorchJob"
metadata:
  name: "pytorch-dist-mnist-nccl"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        metadata:
          annotations:
            sidecar.istio.io/inject: "false"
        spec:
          containers:
            - name: pytorch
              image: registry.cern.ch/kubeflow/custom-pytorchjob:v1.0
              args: ["--backend", "nccl"]
              resources:
                limits:
                  nvidia.com/gpu: 1
```

How can we make this better?

Distributed Training

Training jobs run across multiple CPUs/GPUs, either on the same machine or across a cluster

Speeds up training and allows handling of larger datasets/models

- Data Parallelism:
  - Data split across workers
  - Each worker trains on a different subset of the data
- Model Parallelism:
  - Model split across workers
  - Useful for very large models

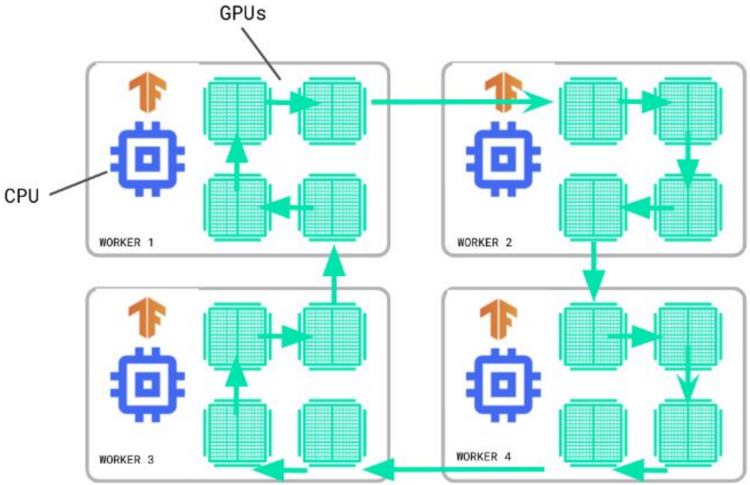
Distributed Training

Major ML frameworks support **distributed training**

- Training jobs split across **multiple** local GPUs

Kubeflow offers distributed training in Kubernetes

- TFJob, PytorchJob, MXNetJob, MPIJob, XGBoostJob
- Jobs split across **multiple** cluster GPUs



```yaml
apiVersion: kubeflow.org/v1
kind: PyTorchJob
metadata:
  name: "ptjob-dist"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: Never
      template:
        spec:
          containers:
            - name: pytorch
              resources:
                limits:
                  nvidia.com/gpu: 1
              image: registry.cern.ch/kubeflow/custom-pytorchjob:v1.0
              args: ["--backend", "nccl"]
              command:
                - "python3"
                - "/opt/pytorch-mnist/mnist.py"
                - "--epochs=10"
                - "--batch-size=512"
    Worker:
      replicas: 1
      restartPolicy: OnFailure
      template:
        spec:
          containers:
            - name: pytorch
              resources:
                limits:
                  nvidia.com/gpu: 1
              image: registry.cern.ch/kubeflow/custom-pytorchjob:v1.0
              args: ["--backend", "nccl"]
              command:
                - "python3"
                - "/opt/pytorch-mnist/mnist.py"
                - "--epochs=10"
                - "--batch-size=512"
```

Classical vs Distributed Training

Feature	Classical Training	Distributed Training
Resources	Limited to the resources of a single node	Scales across multiple nodes
Scalability	Limited	High
Dataset Size	Small/Moderate	Large
Training Speed	Slower	Faster
Complexity	Simple	Requires orchestration

When to Use Which?

Classical: Prototyping, small models/datasets
Distributed: Large-scale models, big datasets, or time-sensitive tasks

Katib: Hyperparameter Optimization

Parameters that define model structure and training process:

- Learning rate
- Number of layers/nodes
- Activation functions
- They are not learned during training but must be optimized

Why is HPO Important?

- Improves model **accuracy** and **performance**
- Reduces training time by finding optimal values efficiently

Katib: Hyperparameter Optimization

Katib is Kubeflow's automated machine learning (AutoML) tool.

- Hyperparameter Tuning (**HPO**)
- Neural Architecture Search (**NAS**)
- **Early Stopping** for experiments

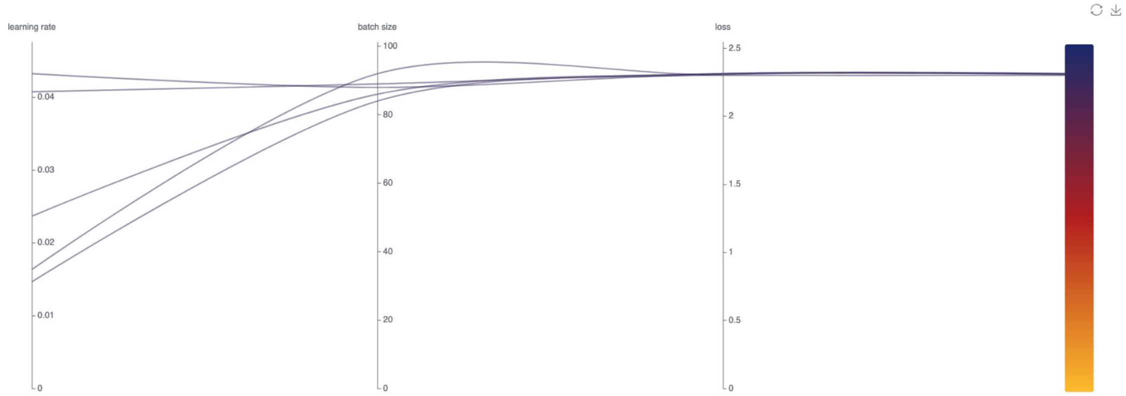
Standardized **development process**

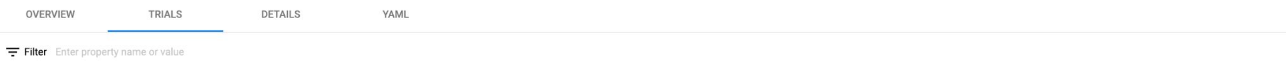
- Create a training script
- Build a Docker image
- Run with various sets of inputs

Improved **hardware efficiency**

- Run each trial on a separate GPU

Visualization of **results and metrics**



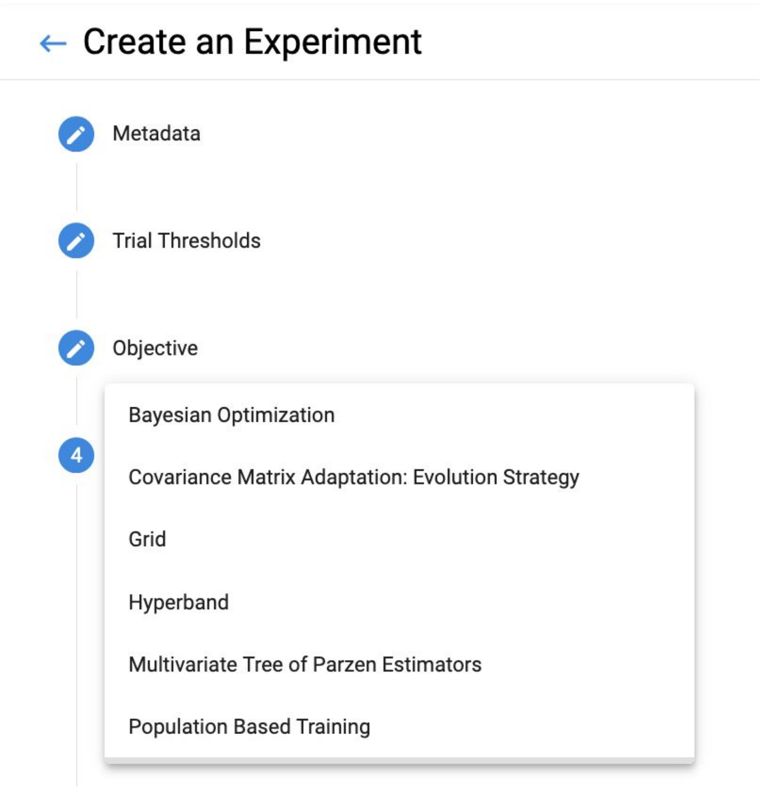


Status	Trial name ↑	Loss	Learning rate	Batch size
✓	mnist-e2e-8brx44lf	2.31258	2.36452e-2	86
✓	mnist-e2e-flvbsjxq	2.30078	1.63211e-2	92

Katib Hyperparameter Optimization

Algorithms

- Random Search
- Bayesian Optimization
- Tree of Parzen Estimators
- Hyperband
- .. and more



Neural Architecture Search (NAS)

Automates the design of neural network architectures.

Optimizes:

- Number of layers
- Types of operations (e.g., convolutions, pooling)
- Connections between layers

Why Use NAS?

Manual architecture design is **time-consuming**
NAS helps discover architectures that **balance performance** and **resource efficiency** (e.g. accuracy, inference time)

NAS in Katib

Concepts:

Search Space: Possible architectures to explore
Optimization Objective: maximizing **accuracy** or minimizing **loss**

Algorithms:
Efficient Neural Architecture Search (**ENAS**)
Differentiable Architecture Search (**DARTS**)

Name	darts-cpu
Status	Experiment has succeeded because max trial count has reached
Best trial	darts-cpu-xlj4n5q7
Best trial's params	algorithm-settings: {'num_epochs': '1', 'w_lr': 0.025, 'w_lr_min': 0.001, 'w_momentum': 0.9, 'w_weight_decay': 0.0003, 'w_grad_clip': 5.0, 'alpha_lr': 0.0003, 'alpha_weight_decay': 0.001, 'batch_size': 128, 'num_workers': 4, 'init_channels': '1', 'print_step': 50, 'num_nodes': '1', 'stem_multiplier': '1'} search-space: ['max_pooling_3x3'] num-layers: 1
Best trial performance	Best-Genotype: Genotype(normal=[[('max_pooling_3x3',1),('max_pooling_3x3',0)]],normal_concat=range(2,3),reduce=[],reduce_concat=range(2,3))

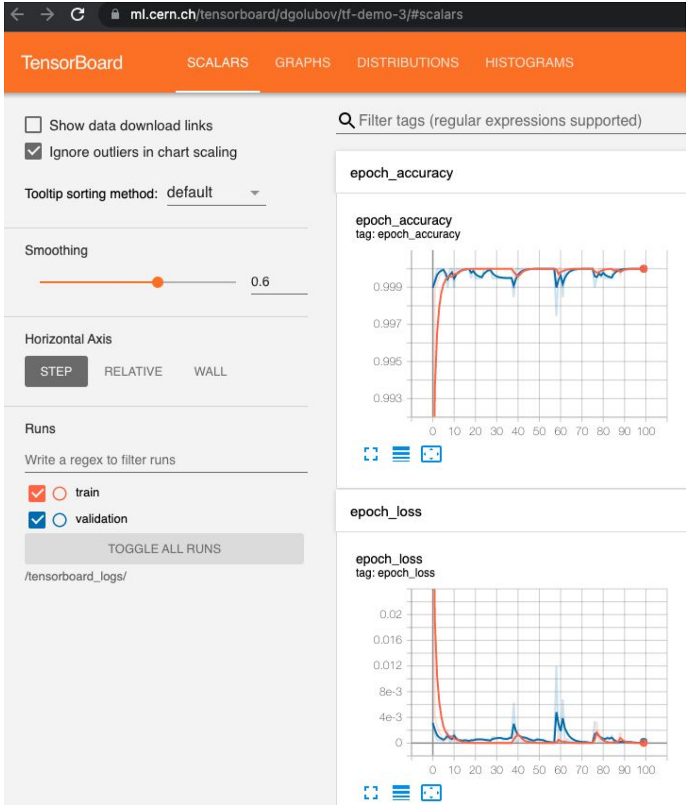
Tensorboards

Measurements and visualizations for ML workloads

- Track **loss and accuracy**
- Visualize **model graph**
- View custom metrics

Kubeflow allows creation of **Tensorboard servers**

- Monitor model training real-time
- Training from any Kubeflow component



Model Serving

Deploy a server to **run inference via http requests**

curl -v -H “Host: host” “http://host_ip/v1/models/mnist:predict" -d @./input.json

Serverless architecture

Automatic scaling per number of requests

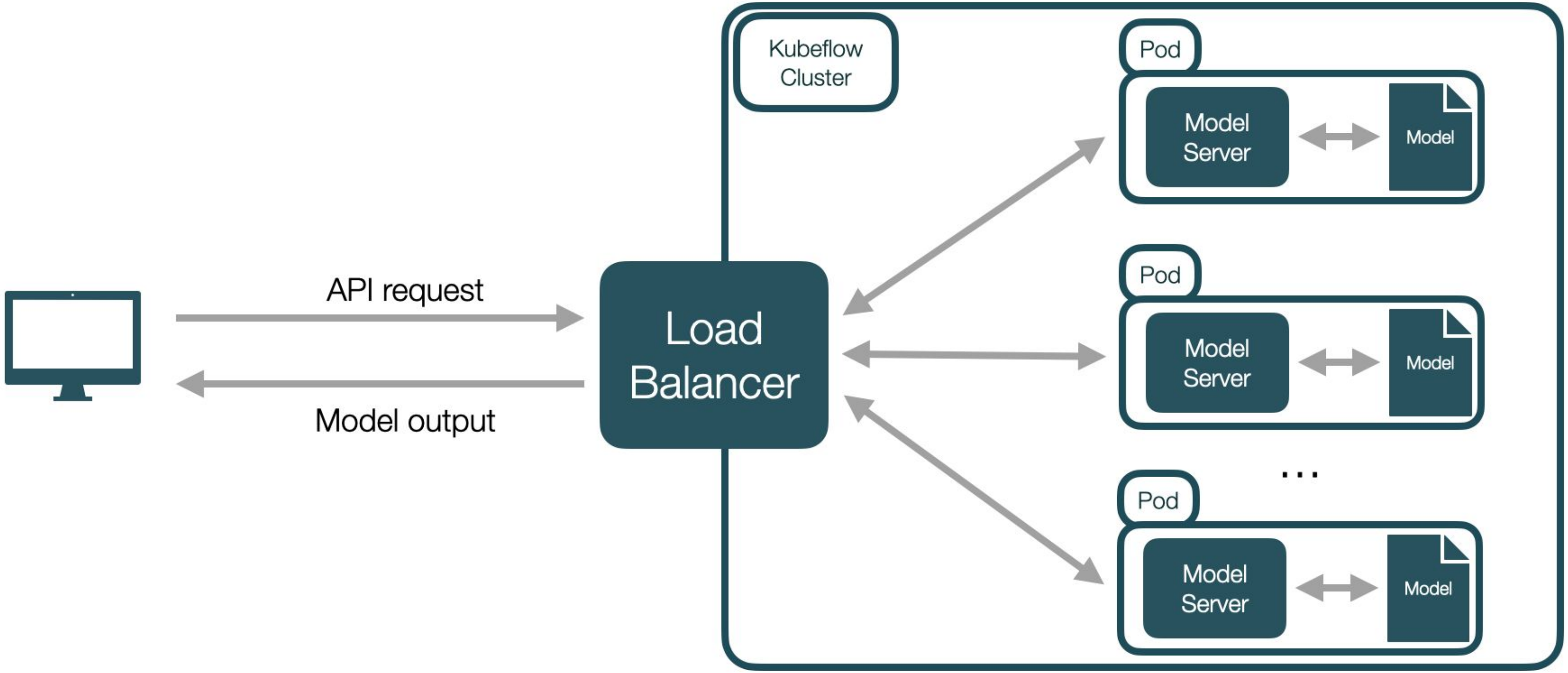
Provided via **KServe** component

Supports major ML frameworks: TensorFlow, PyTorch, SKLearn, ONNX, Triton, etc.

Enables multi-model serving in the **same service**

Ideal for use cases requiring access to **various models simultaneously**

Model Serving



Model Autoscaling

Why Autoscaling?

- Efficient resource usage
- Automatically scales based on traffic

Easy to Configure :

- Add resource limits in YAML:

```yaml
limits:
  cpu: "250m"
  memory: "2Gi"
  nvidia.com/gpu: 1
```

Conclusions

- Kubeflow Streamlines the Entire ML Lifecycle
  - From prototyping in notebooks to deploying models as scalable APIs, Kubeflow simplifies and integrates every stage of the machine learning process.
- Pipelines Enable Reproducible and Automated Workflows
  - By defining ML tasks as pipelines, workflows are reproducible, automated, and easy to manage.
- Distributed Training Unlocks Scalability
  - Kubeflow's support for distributed training with PyTorch and TensorFlow training large models efficiently by leveraging multiple nodes and GPUs.
- Katib Automates Hyperparameter and Architecture Optimization
  - Katib reduces the manual effort of tuning models by automating hyperparameter search and neural architecture design, leading to better-performing models.
- Serving Provides Scalable and Efficient Model Deployment
  - Models are deployed as REST APIs with built-in support for autoscaling, multi-model serving, and GPU acceleration, ensuring reliable and fast inference.

Where to find us

- https://ml.docs.cern.ch/
- https://ml.cern.ch/
- Mattermost

Thank You!

Demo Time

Demo Materials

- MNIST-End-to-End Pipeline
- PytorchJob - Distributed
- Flower - InferenceService