

Kubeflow Pipelines

Kubeflow Pipeline

The screenshot displays the Kubeflow Pipeline dashboard. On the left is a dark blue sidebar with navigation links: Home, Pipelines (highlighted with a red rectangle), Notebook Servers, Katib, Artifact Store, Manage Contributors, and GitHub. The main content area has a top bar with the user 'kubeflow-sarahmaddox' and tabs for 'Dashboard' and 'Activity'. The dashboard is divided into three columns. The first column, 'Quick shortcuts', lists actions like 'Upload a pipeline', 'View all pipeline runs', 'Create a new Notebook server', 'View Katib Studies', and 'View Metadata Artifacts'. The second column contains 'Recent Notebooks' (showing 'No Notebooks in namespace kubeflow-sarahmaddox') and 'Recent Pipelines' (listing sample and ML pipelines with their creation times). The third column, 'Documentation', provides links to guides such as 'Getting Started with Kubeflow', 'MiniKF', 'Microk8s for Kubeflow', 'Minikube for Kubeflow', 'Kubeflow on GCP', 'Kubeflow on AWS', and 'Requirements for Kubeflow'.

Kubeflow

Home

Pipelines

Notebook Servers

Katib

Artifact Store

Manage Contributors

GitHub

Privacy • Usage Reporting
build version 0.7.0

kubeflow-sarahmaddox (...)

Dashboard Activity

Quick shortcuts

- ⚡ Upload a pipeline
Pipelines
- ⚡ View all pipeline runs
Pipelines
- ⚡ Create a new Notebook server
Notebook Servers
- ⚡ View Katib Studies
Katib
- ⚡ View Metadata Artifacts
Artifact Store

Recent Notebooks

No Notebooks in namespace kubeflow-sarahmaddox

Recent Pipelines

- [Sample] Basic - Exit Handler
Created 22/12/2019, 06:50:18
- [Sample] Basic - Conditional execution
Created 22/12/2019, 06:50:17
- [Sample] Basic - Parallel execution
Created 22/12/2019, 06:50:16
- [Sample] Basic - Sequential execution
Created 22/12/2019, 06:50:15
- [Sample] ML - XGBoost - Training with ...
Created 22/12/2019, 06:50:14

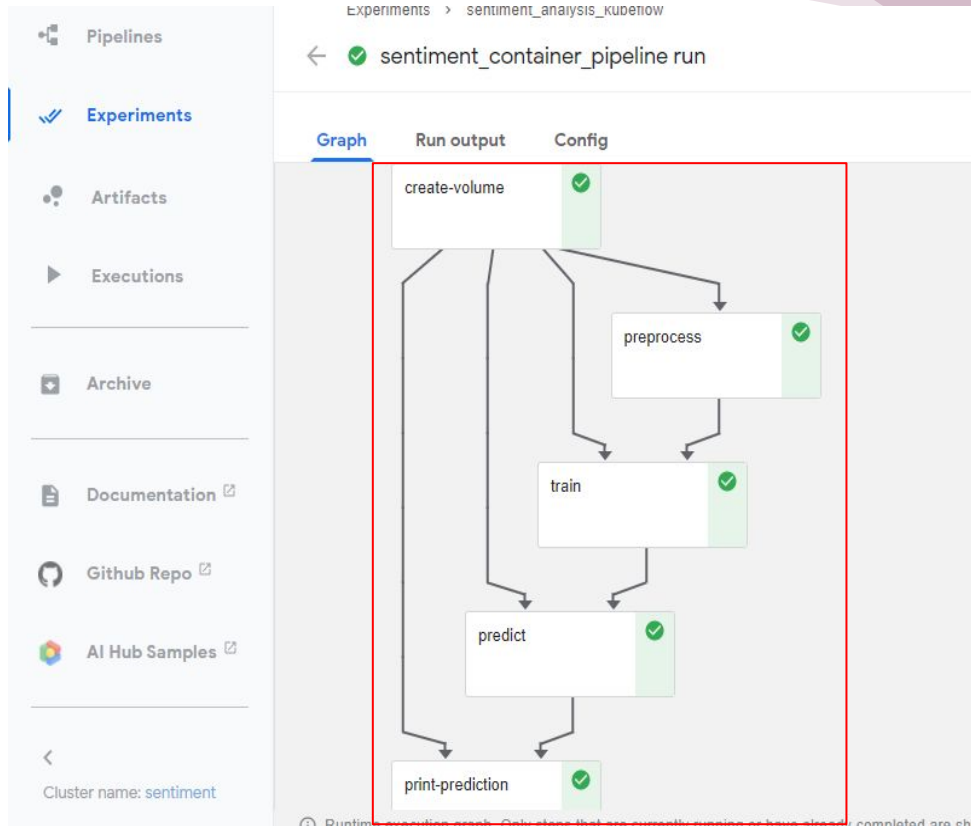
Documentation

- 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
- Minikube for Kubeflow**
Quickly get Kubeflow running locally
- 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**

Kubeflow Pipelines

The Kubeflow Pipeline platform consists of:

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

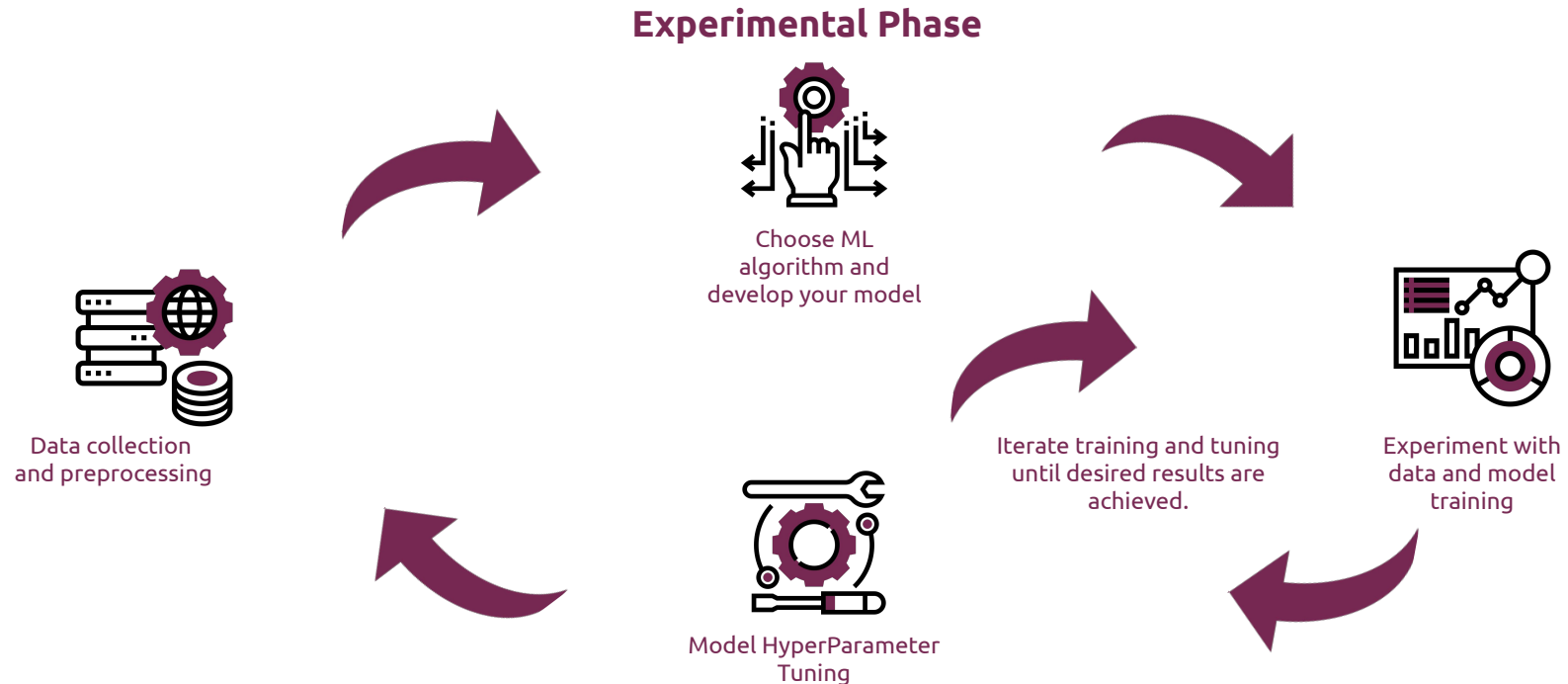


ML Workflow

- An ML workflow defines phases and stages implemented during an ML project.
- It varies with the project, but there are some core stages that must be implemented.
- The stages of an ML workflow helps in defining the components of a pipeline.

Core stages of ML workflow: Experimental Phase

This phase is the development of the workflow on a local jupyter notebook. These stages serve as components in a pipeline.



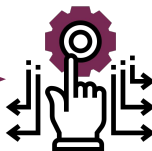
Kubeflow Components in the ML workflow

Experimental Phase

Kubeflow



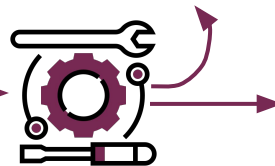
Identify problem and collect data for Analysis



Choose ML algorithm and develop your model



Experiment with data and model training



Iterate Tuning/Training

Model hyperparameter Tuning

Pytorch

Scikit-Learn

Tensorflow

XGBoost

Jupyter Notebook

Fairing

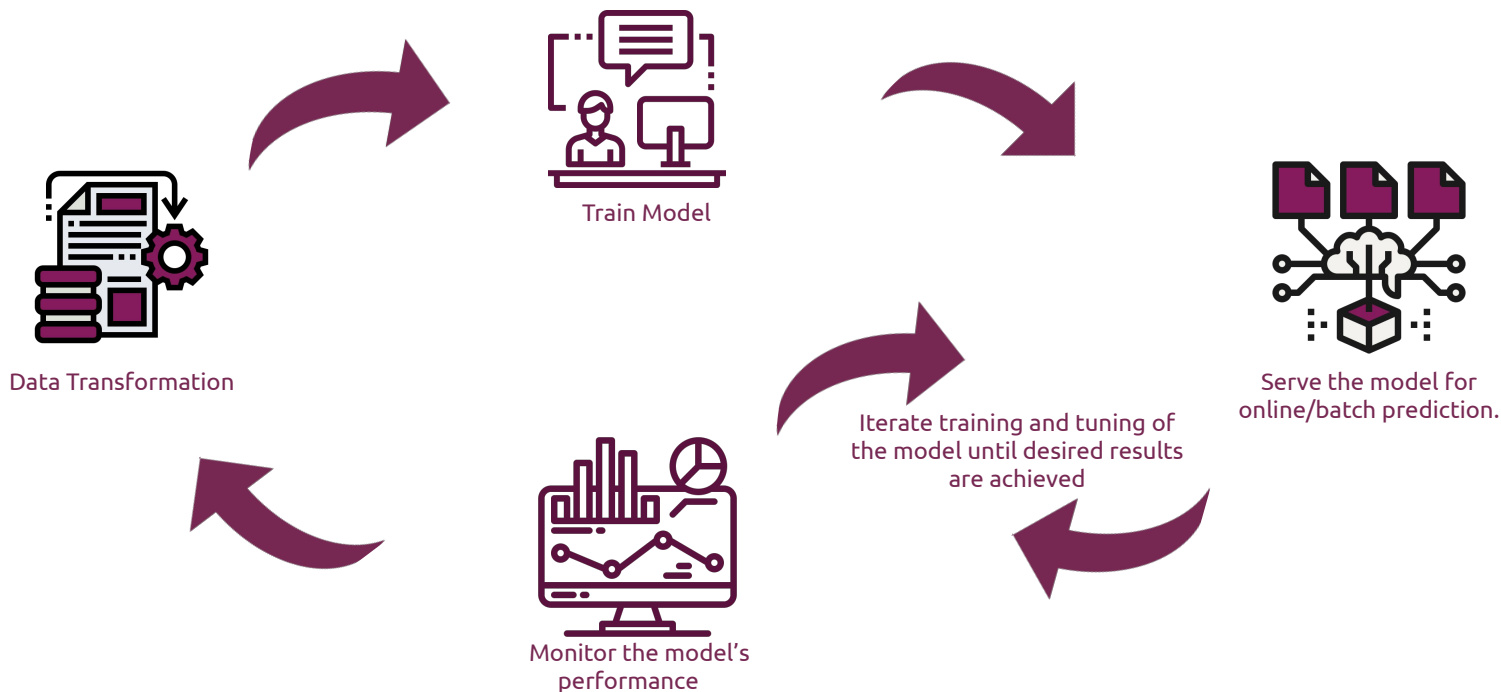
Pipelines

Katib

Core stages of ML workflow: Production Phase

This production phase involves using kubeflow to build scalable and reusable models for deployment.

Production Phase



Kubeflow Components in the ML workflow

Production Phase

Kubeflow



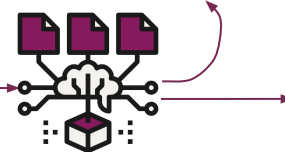
Transform Data



Train Model



Serve the model for
online/batch prediction



Monitor the model's
performance

Chainer

MPI

MXNet

PyTorch

TFJob

KFServing

NVIDIA TensorRT

PyTorch

TfServing

Seldon

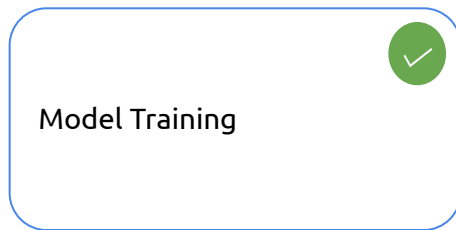
Metadata

TensorBoard

Kubeflow Pipelines

Kubeflow Pipeline Components

- Self-contained set of codes that perform one of the stages in the ML workflow
- Each component takes in one or more inputs and produces a specified output(s).
- The code for each component must include:
 - **Client code:** This code talks to the endpoints to submit jobs. For example, a code that retrieves the data to be worked with.
 - **Runtime code:** This code does the actual job and usually runs in the cluster. For example, a code that trains the model on the pre-processed data.



Kubeflow Pipeline Component type (Light Weight)

There are two ways to develop Kubeflow Pipeline components:

- **Lightweight Component:**
 - For rapid development
 - Fast and easy
 - Downside : it is not reusable
- Uses a stand-alone python function and called with **kfp.components.func_to_container_op(func)** to convert it to a kubeflow component in your pipeline.

```
def predict(data_path):  
  
    import pickle  
    import sys, subprocess;  
    subprocess.run([sys.executable, '-m', 'pip', 'install', 'keras==1.2.2'])  
    import numpy as np  
    import tensorflow  
    import keras  
  
    # Load the saved Keras model  
    classifier = tensorflow.keras.models.load_model(f'{data_path}/sentiment_model.h5')  
    # Load and unpack the test_data  
    with open(f'{data_path}/test_data', 'rb') as f:  
        test_data = pickle.load(f)  
    # Separate the X_test from y_test.  
    X_test, y_test = test_data  
    # make predictions.  
    y_pred = classifier.predict(X_test)  
    # create a threshold  
    y_pred=(y_pred>0.5)  
  
    with open(f'{data_path}/result.txt', 'w') as result:  
        result.write(" Prediction: {}, Actual: {}".format(y_pred,y_test.astype("int64")))  
    print('Prediction has be saved successfully!')
```

```
#creating lightweight predict component  
predict_op = comp.func_to_container_op(predict , base_image = "tensorflow/tensorflow:latest-gpu-py3")
```

Kubeflow Pipeline Component Type (Reusable)

- **Reusable Component:**
 - Requires more time to implement because container images are built
 - Reusable

```
import argparse
import numpy as np
import joblib
import tensorflow as tf
import keras

def predict(X_test,y_test,model):
    #loading test data
    X_test = np.load(X_test)
    y_test = np.load(y_test)
    # Load the saved Keras model
    classifier = tensorflow.keras.models.load_model(model)
    # make predictions.
    y_pred = classifier.predict(X_test)
    # create a threshold
    y_pred=(y_pred>0.5)
    with open('results.txt', 'w') as result:
        result.write(" Prediction: {}, Actual: {} ".format(y_pred,y_test.astype("int64")))

if __name__ == '__main__':
    print('Predicting data ...')
    parser = argparse.ArgumentParser()
    parser.add_argument('--X_test')
    parser.add_argument('--y_test')
    parser.add_argument('--sentiment_model')
    args = parser.parse_args()
    predict(args.X_test, args.y_test, args.sentiment_model)
```

Kubeflow Pipeline Component Type (Reusable)

DockerFile: create a dockerfile to build your image for the python script in the previous slide. This image is pushed to Docker Hub to ensure it is accessible by the kubernetes cluster.

Components: The component is defined as a function that returns an object of type ContainerOp. The ContainerOp represents a pipeline task implemented by a container image.

```
#DockerFile
FROM python:3.8.4
WORKDIR /data_predict
RUN pip install -U tensorflow numpy pandas
COPY predict.py /data_predict
ENTRYPOINT ["python", "predict.py"]
```

```
#predict component definition
def predict_op(X_test, y_test, model):
    return dsl.ContainerOp(
        name='Predict Model',
        image='jadesola/predict-component:v.0.1',
        arguments = [
            '--X_test', x_test,
            'y_test', y_test,
            '--model', model
        ],
        file_outputs={
            'results': '/data_predict/results.txt'
        }
    )
```

Collaborative ML Pipeline Development

