Kale

Kale Overview

Kale (Kubeflow Automated Pipeline Engine) is a tool introduced by kubeflow to seamlessly deploy annotated Jupyter Notebooks to Kubeflow Pipelines. Kale lets you deploy Jupyter Notebooks that run on your laptop or on the cloud to Kubeflow Pipelines, without requiring Kubeflow SDK. All you need to do is to concentrate on the machine learning development of the project and tag each code cell.

The Kubeflow pipeline SDK serves as a great tool for automating the creation of pipelines most especially when dealing with complex workflows and production environments. Some Data scientists who have no software engineering background find it difficult to understand, let alone deploy. This is where Kale comes in.

Kale addresses this problem by providing a tool to simplify the deployment process of a Jupyter Notebook into Kubeflow Pipelines workflows. It works by converting Jupyter Notebooks directly into a KFP pipeline, while ensuring that all the processing building blocks are well organized and leveraging on the experiment tracking and workflows organization provided out-of-the-box by Kubeflow.

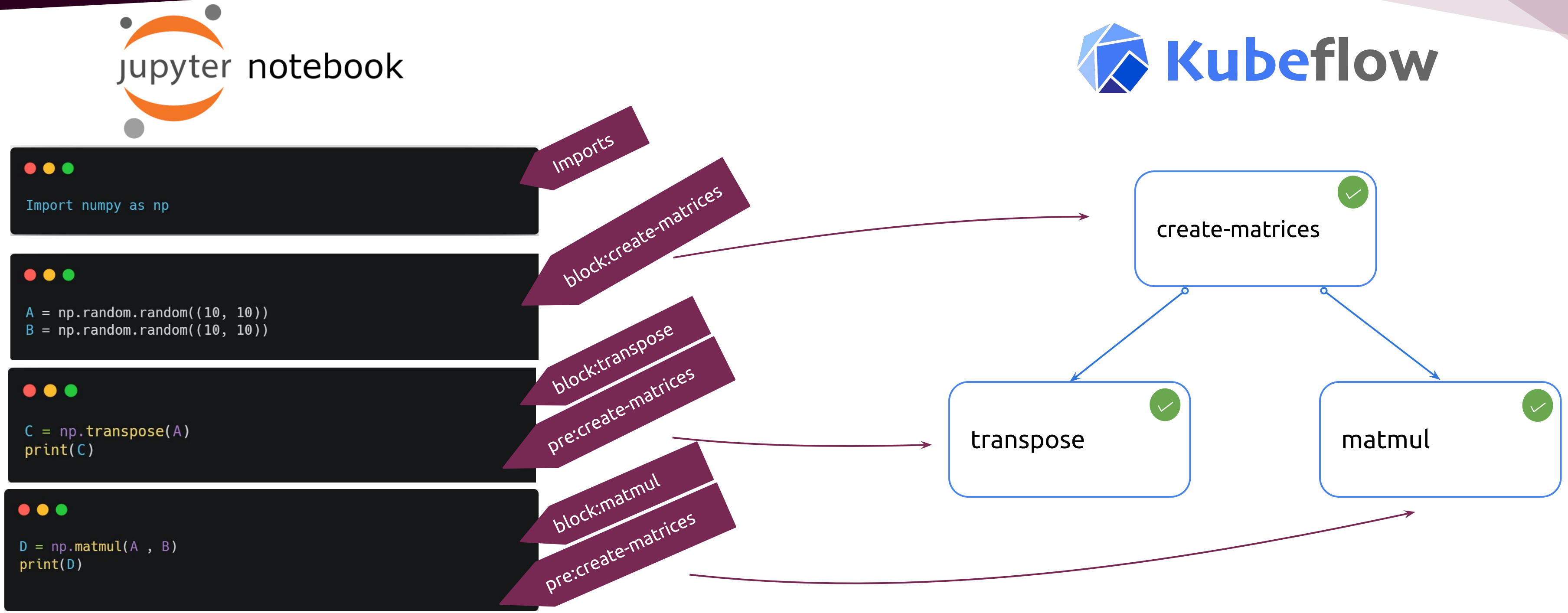
Kale Overview

Kale adopts the Json structure of a notebook through annotations on a notebook level and a single cell level.

With the annotations you can simply:

1. Assign code cells to specific pipeline components
2. Merge together multiple cells into a single pipeline component
3. Define the (execution) dependencies between them

Kale Overview



Kale Features

1. Metadata Management - With Kale you can store and retrieve metadata from the Notebook Metadata.

2. Data Parsing - A persistent volume is automatically provisioned by Kale to save marshal data

3. Reusability - Each step from Notebook to deployment can be programmatically called as an API

4. Pipeline Parameters - Kale has tag parameters for each cell. Assigning the pipeline-parameters tag on any cell that contains some variables will instruct Kale to transform them to pipeline parameters which are passed to the pipeline steps that make use of them

5. Data Version and Snapshots - With the integration of Rok client, Kale can recover Notebook data by taking snapshot volumes at the beginning of each step run and at the end of the pipeline run. Kale can also identify existing workspaces in the Notebook servers, snapshot them and mount them into the pipeline steps. This helps preserve user Notebook workpaces (data uses, installed libraries, files, code)

Setting Up Kale

Kale is very easy to set up, you can set this up on your local environment with the following steps:

- Install kale
  `pip install kubeflow-kale`

- Install jupyter lab
  `pip install "jupyterlab>=2.0.0,<3.0.0"`

- Install the extension
  `jupyter labextension install kubeflow-kale-labextension`

- Verify extension status
  `jupyter labextension list`

- Run
  `jupyter lab`

Setting up Kale with Kubeflow

If you are already running Kubeflow you can create a new Notebook Server using a kale custom docker image `gcr.io/arrikto/jupyter-kale:v0.5.0`

Create a new server with the custom kale image and connect to it to get a jupyter lab notebook with kale installed

Dataset for Kale walkthrough

We will be using the titanic dataset from kaggle, It contains data collated from the ship's manifest on its passengers. There are 1309 rows of data segmented in train and test files

- **Features**: The dataset contains features on the ticket class, gender, age, number of siblings, children, spouses or parents and other information about the participants and whether or not they survived the ship's sinking.

- **Goal**: Predict who survived the sinking ship and who didn't,t. In the process we would also build a pipeline and deploy it to Kubeflow.

Running an example with Kale

Using a notebook on the Titanic dataset, let's take a look at Kale

imports

Cell type
Imports

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from matplotlib import style

from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

step: load_data

```
test_df = pd.read_csv("https://raw.githubusercontent.com/kubeflow-kale/kale/master/examples/titanic-ml-dataset/data/test.csv")
train_df = pd.read_csv("https://raw.githubusercontent.com/kubeflow-kale/kale/master/examples/titanic-ml-dataset/data/train.csv")
```

skip

```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 14 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   PassengerId    418 non-null    int64
 1   Pclass         418 non-null    int64
 2   Sex            418 non-null    int64
 3   Age            418 non-null    int64
 4   SibSp          418 non-null    int64
```

- Labels can be easily set to dictate the pipeline's behaviour.
- Import for package dependencies
- The Pipeline step named load_data signifying the imputation process.

- Any aspects of the notebook note needed in the pipeline can be skipped during pipeline compilation using the skip annotation.

Running an example with Kale

step: preprocessing *depends on:*

Cell type: Pipeline Step | Step name: preprocessing | Depends on: load_data | GPU

```
data = [train_df, test_df]
for dataset in data:
    dataset['relatives'] = dataset['SibSp'] + dataset['Parch']
    dataset.loc[dataset['relatives'] > 0, 'not_alone'] = 0
    dataset.loc[dataset['relatives'] == 0, 'not_alone'] = 1
    dataset['not_alone'] = dataset['not_alone'].astype(int)
```

```
import re
deck = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6, "G": 7, "U": 8}
data = [train_df, test_df]
```

step: modelling *depends on:* *GPU request: nvidia.com/gpu - 1*

Cell type: Pipeline Step | Step name: modelling | Depends on: preprocessing | GPU

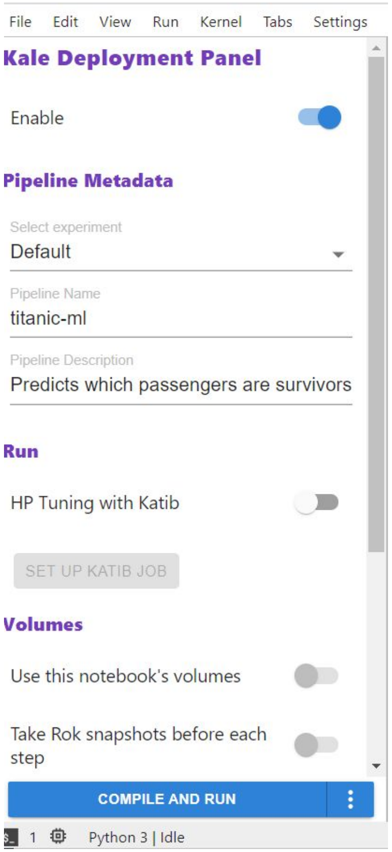
```
sgd = linear_model.SGDClassifier(max_iter=5, tol=None)
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)

sgd.score(X_train, Y_train)

acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
```

```
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)

Y_prediction = random_forest.predict(X_test)

random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
```

Kale allows for hierarchies in pipeline steps, grouping of numerous notebook cells under a label.

Also, each pipeline step can have its own GPU needs set from the notebook.

Running an example with Kale



A pipeline of the notebook can be created by clicking the "Compile and Run" button which compiles and uploads the pipeline to Kubeflow

The pipeline is created with as many steps and components as indicated in the notebook, giving you the freedom to create complex and simple model pipelines.