Parallelizing Your ETL with Dask on Kubeflow

dask

Typical Machine Learning Workflow

Kubeflow

jupyter NumPy Katib dmlc XGBoost PyTorch KServe pandas TensorFlow TRITON INFERENCE SERVER

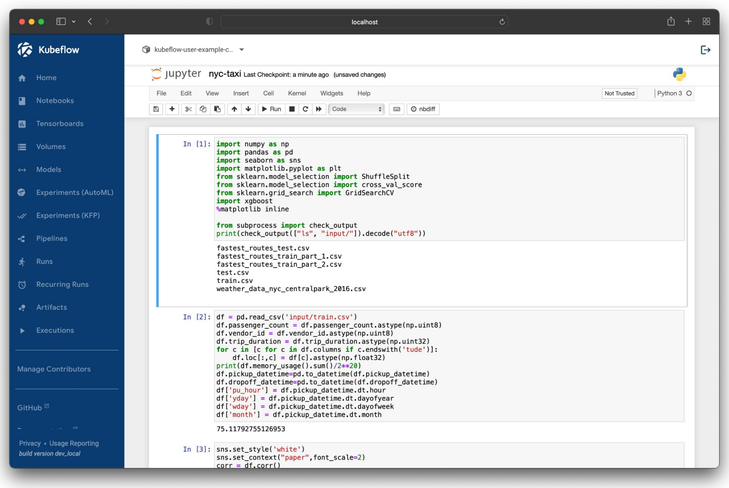
ETL is commonly done in notebooks

The extract, transform, load parts of a typical machine learning workflow are typically done interactively in Jupyter.

This means that tools like NumPy and Pandas are restricted to the resources of the single Pod that Jupyter is running in.

Later stages of the workflow tend to use accelerated and distributed computing to improve performance.

But what if we could accelerate our ETL too?



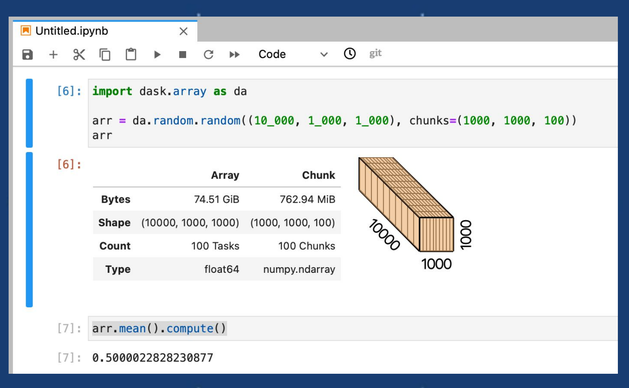
dask

Scale the Python libraries that you know and love with Dask

Python has grown to become the dominant language both in data analytics and general programming.

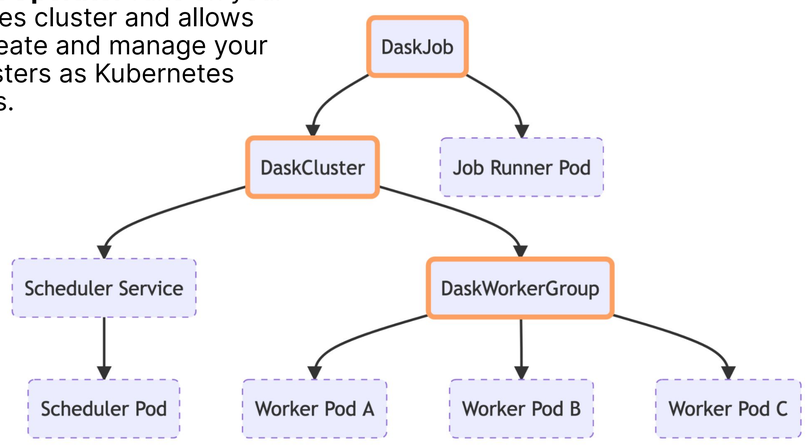
This growth has been fueled by computational libraries like NumPy, pandas, and scikit-learn. However, these packages weren't designed to scale beyond a single machine.

Dask was developed to natively scale these packages and the surrounding ecosystem to multi-core machines and distributed clusters when datasets exceed memory.



Dask Operator for Kubernetes

The **Dask Operator** runs on your Kubernetes cluster and allows you to create and manage your Dask clusters as Kubernetes resources.



Scan for Dask Operator documentation

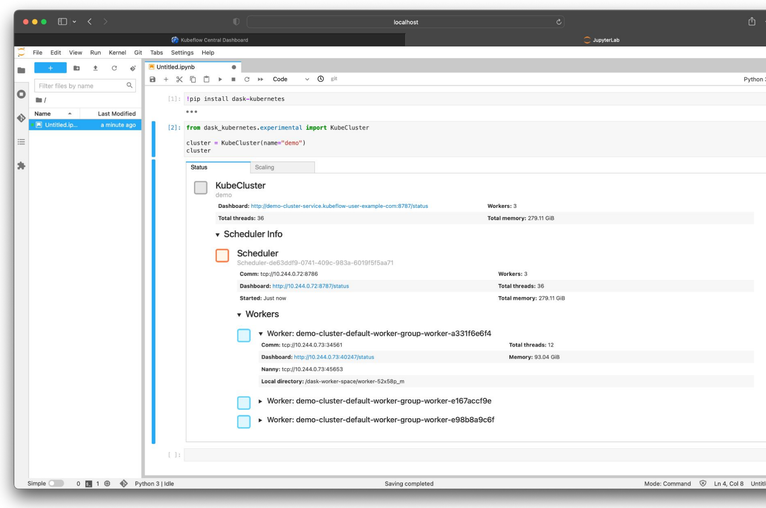
https://kubernetes.dask.org

Installing the operator

```
# Install the Custom Resource Definitions and Operator Deployment
$ kubectl apply -f  <manifest urls...>

# Patch KubeFlow permissions to allow users to create Dask clusters
$ kubectl patch clusterrole kubeflow-kubernetes-edit --patch '{"rules": [{"apiGroups":
["kubernetes.dask.org"],"resources": ["*"],"verbs": ["*"]}, ...]}'

# Now we can list DaskCluster resources
$ kubectl get daskclusters
No resources found in default namespace.

# We can check the operator pod is running
$ kubectl get pods -A -l application=dask-kubernetes-operator
NAMESPACE       NAME                                          READY   STATUS
dask-operator   dask-kubernetes-operator-775b8bbbd5-zdrf7     1/1     Running

# 🚀 done!
```

Creating Dask clusters in Python

```
# Install dask-kubernetes
$ pip install dask-kubernetes

# Launch a cluster
>>> from dask_kubernetes.experimental import KubeCluster
>>> cluster = KubeCluster(name="demo")

# List the DaskCluster custom resource that was created for us under the hood
$ kubectl get daskclusters
NAME           AGE
demo-cluster   6m3s

# Connect a client to use the cluster
>>> from dask.distributed import Client
>>> client = Client(cluster)

# Scale the workers up
>>> cluster.scale(5)

# Delete the cluster
>>> cluster.close()
```



Creating Dask clusters with YAML

```
# Apply a DaskCluster manifest directly
$ cat <<EOF | kubectl apply -f -
apiVersion: kubernetes.dask.org/v1
kind: DaskCluster
metadata:
  name: simple-cluster
spec:
  worker:
    replicas: 2
    spec:
    containers:
      ...
  scheduler:
    spec:
      containers:
        ...
    service:
      ...
EOF

# List the DaskCluster custom resource
$ kubectl get daskclusters
NAME             AGE
simple-cluster   6m3s

# Scale our default DaskWorkerGroup
$ kubectl scale --replicas=5 daskworkergroup \
           simple-cluster-default-worker-group

# Delete the cluster
$ kubectl delete daskcluster simple-cluster
```


Jacob Tomlinson
Dask Core Maintainer
Senior Software Engineer, NVIDIA

SciPy 2022
Austin, TX, USA

dask