ML Pipeline with Kubeflow on Azure: A Comprehensive Guide

Comprehensive Guide: ML Pipeline with Kubeflow on Azure

Prerequisites

1. System Requirements

- Linux/macOS/Windows with WSL2
- Python 3.8+
- Docker Desktop
- Git

2. Cloud & Tools

```
# Install Azure CLI
curl -sL <https://aka.ms/InstallAzureCLIDeb> | sudo bash

# Install kubectl
curl -LO "<https://dl.k8s.io/release/$(curl> -L -s
<https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl>"
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

# Install Kustomize
curl -s "<https://raw.githubusercontent.com/kubernetes-sigs/kustomize/master/hack/install_kustomize.sh>" |
bash
sudo mv kustomize /usr/local/bin/

# Python packages
pip install azure-cli azure-ml-sdk kubernetes kfp azureml-sdk pandas scikit-learn tensorflow
```

3. Azure Resources

- Active Azure subscription
- Resource group
- Storage account
- Azure Container Registry (ACR)

Setup Infrastructure

1. Create AKS Cluster

```
# Create resource group
az group create --name ml-resources --location eastus

# Create AKS cluster
az aks create \
  --resource-group ml-resources \
  --name ml-cluster \
  --node-count 3 \
  --enable-addons monitoring \
  --generate-ssh-keys \
```

```
  --node-vm-size Standard_DS3_v2

# Get credentials
az aks get-credentials --resource-group ml-resources --name ml-cluster
```

2. Deploy Kubeflow

```
# Clone manifests
git clone <https://github.com/kubeflow/manifests.git>
cd manifests

# Install Kubeflow
while ! kustomize build example | kubectl apply -f -; do
    echo "Retrying to apply resources";
    sleep 30;
done

# Verify deployment
kubectl get pods -n kubeflow
```

Sample ML Pipeline Implementation

1. Dataset Preparation

```
# dataset_prep.py
import pandas as pd
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split

def prepare_data():
    # Load sample diabetes dataset
    diabetes = load_diabetes()
    df = pd.DataFrame(diabetes.data, columns=diabetes.feature_names)
    df['target'] = diabetes.target

    # Split data
    train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

    # Save to files
    train_df.to_csv('train.csv', index=False)
    test_df.to_csv('test.csv', index=False)

    return 'train.csv', 'test.csv'

if __name__ == '__main__':
    prepare_data()
```

2. Model Training

```
# train.py
import pandas as pd
import numpy as np
from sklearn.linear_model import Ridge
import pickle

def train_model(data_path):
    # Load training data
```

```
    train_data = pd.read_csv(data_path)
    X = train_data.drop('target', axis=1)
    y = train_data['target']

    # Train model
    model = Ridge(alpha=1.0)
    model.fit(X, y)

    # Save model
    with open('model.pkl', 'wb') as f:
        pickle.dump(model, f)

    return 'model.pkl'

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=str)
    args = parser.parse_args()
    train_model(args.data)
```

3. Model Evaluation

```
# evaluate.py
import pandas as pd
import pickle
from sklearn.metrics import mean_squared_error, r2_score
import json

def evaluate_model(model_path, test_data_path):
    # Load model and test data
    with open(model_path, 'rb') as f:
        model = pickle.load(f)

    test_data = pd.read_csv(test_data_path)
    X_test = test_data.drop('target', axis=1)
    y_test = test_data['target']

    # Make predictions
    y_pred = model.predict(X_test)

    # Calculate metrics
    metrics = {
        'mse': mean_squared_error(y_test, y_pred),
        'r2': r2_score(y_test, y_pred)
    }

    # Save metrics
    with open('metrics.json', 'w') as f:
        json.dump(metrics, f)

    return 'metrics.json'

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--model', type=str)
    parser.add_argument('--test-data', type=str)
```

```
    args = parser.parse_args()
    evaluate_model(args.model, args.test_data)
```

4. Kubeflow Pipeline Definition

```
# pipeline.py
from kfp import dsl
from kfp import components

def prepare_data_op():
    return dsl.ContainerOp(
        name='Prepare Data',
        image='python:3.8',
        command=['python', 'dataset_prep.py'],
        file_outputs={
            'train': '/train.csv',
            'test': '/test.csv'
        }
    )

def train_model_op(train_data):
    return dsl.ContainerOp(
        name='Train Model',
        image='python:3.8',
        command=['python', 'train.py'],
        arguments=['--data', train_data],
        file_outputs={'model': '/model.pkl'}
    )

def evaluate_model_op(model, test_data):
    return dsl.ContainerOp(
        name='Evaluate Model',
        image='python:3.8',
        command=['python', 'evaluate.py'],
        arguments=[
            '--model', model,
            '--test-data', test_data
        ],
        file_outputs={'metrics': '/metrics.json'}
    )

@dsl.pipeline(
    name='Diabetes Prediction Pipeline',
    description='Train and evaluate a diabetes prediction model'
)
def diabetes_pipeline():
    data_prep = prepare_data_op()
    train = train_model_op(data_prep.outputs['train'])
    evaluate = evaluate_model_op(
        train.outputs['model'],
        data_prep.outputs['test']
    )

# Compile pipeline
from kfp.compiler import Compiler
Compiler().compile(diabetes_pipeline, 'diabetes_pipeline.yaml')
```

5. Deploy Model to Azure ML

```python
# deploy.py
from azureml.core import Workspace, Model, Environment
from azureml.core.webservice import AciWebservice
from azureml.core.model import InferenceConfig

def deploy_model(model_path):
    # Initialize workspace
    ws = Workspace.from_config()

    # Register model
    model = Model.register(
        workspace=ws,
        model_path=model_path,
        model_name='diabetes-predictor'
    )

    # Create environment
    env = Environment.from_conda_specification(
        name='diabetes-env',
        file_path='environment.yml'
    )

    # Define inference config
    inference_config = InferenceConfig(
        entry_script='score.py',
        environment=env
    )

    # Define deployment config
    deployment_config = AciWebservice.deploy_configuration(
        cpu_cores=1,
        memory_gb=1,
        auth_enabled=True
    )

    # Deploy model
    service = Model.deploy(
        workspace=ws,
        name='diabetes-service',
        models=[model],
        inference_config=inference_config,
        deployment_config=deployment_config
    )

    service.wait_for_deployment(show_output=True)
    return service.scoring_uri

if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('--model', type=str)
    args = parser.parse_args()
    deploy_model(args.model)
```

6. Scoring Script

```python
# score.py
import json
```

```python
import numpy as np
import pandas as pd
from azureml.core.model import Model
import pickle

def init():
    global model
    model_path = Model.get_model_path('diabetes-predictor')
    with open(model_path, 'rb') as f:
        model = pickle.load(f)

def run(raw_data):
    try:
        data = json.loads(raw_data)
        data = pd.DataFrame(data)
        prediction = model.predict(data)
        return json.dumps({'prediction': prediction.tolist()})
    except Exception as e:
        return json.dumps({"error": str(e)})
```

Environment Configuration

Conda Environment

```yaml
# environment.yml
name: diabetes-env
channels:
  - conda-forge
  - defaults
dependencies:
  - python=3.8
  - scikit-learn
  - pandas
  - numpy
  - pip
  - pip:
    - azureml-defaults
    - azureml-core
```

Usage Instructions

1. Set up infrastructure:

```
# Follow the setup steps above to create AKS cluster and deploy Kubeflow
```

2. Prepare environment:

```
# Create conda environment
conda env create -f environment.yml
conda activate diabetes-env
```

3. Run pipeline:

```
# Upload pipeline to Kubeflow
python pipeline.py

# Access Kubeflow UI
kubectl port-forward svc/istio-ingressgateway -n istio-system 8080:80
```

```
# Open browser at <http://localhost:8080>
```

4. Monitor deployment:

```
# Check pod status
kubectl get pods -n kubeflow

# Check service status
kubectl get services -n kubeflow
```

Monitoring

1. Azure Monitor Integration:

```python
# Enable monitoring
from azureml.core import Workspace
from azureml.core.webservice import AciWebservice

ws = Workspace.from_config()
service = AciWebservice(ws, 'diabetes-service')

# Get logs
print(service.get_logs())

# Get metrics
print(service.get_metrics())
```

2. Kubeflow Dashboard:

- Access metrics and logs through the Kubeflow UI
- Monitor pipeline runs and component status
- Track model versions and deployments

This implementation provides a complete ML pipeline that:

- Uses the diabetes dataset as an example
- Implements data preparation, model training, and evaluation
- Deploys the model to Azure ML
- Includes monitoring capabilities

The code is production-ready but may need adjustments based on your specific requirements and scale.