Intel® Cloud Optimization Module for Microsoft Azure*:
XGBoost* Pipeline on Kubernetes*

Building and deploying high-performance AI applications can be a challenging task that requires a significant amount of computing resources and expertise. Fortunately, modern technologies such as Kubernetes*, Docker*, Intel® Optimization for XGBoost*, and Intel® oneDAL make it easier to develop and deploy AI applications optimized for performance and scalability. By using cloud services like Microsoft Azure*, developers can further streamline the process and take advantage of the flexible and scalable infrastructure provided by the cloud. This sheet outlines the key components of running an XGBoost pipeline on Kubernetes—more detail can be found on the GitHub* repo.

Set up Azure Resources

I. Sign in with the Azure command-line interface:

```
az login
```

II. Create an Azure resource group:

```
export RG=intel-sgx-loan-default-app
export LOC=westus
az group create -n $RG -l $LOC
```

III. Create an Azure file share:

```
export STORAGE_NAME=loanappstorage
az storage account create --resource-group $RG --name $STORAGE_NAME --
kind StorageV2 \
--sku Standard_LRS --enable-large-file-share --allow-blob-public-access
false

az storage share-rm create --resource-group $RG --storage-account
$STORAGE_NAME \
--name loan-app-file-share --quota 1024
```

IV. Create an Azure Container Registry:

```
export ACR=loandefaultapp
az acr create --resource-group $RG --name $ACR --sku Standard
```

V. Create an Azure Kubernetes Service (AKS) Cluster with Intel® Software Guard Extensions (Intel® SGX) confidential computing nodes:

```
export AKS=aks-intel-sgx-loan-app

az aks create --resource-group $RG --name $AKS --node-count 1
--node-vm-size Standard_D4_v5 --kubernetes-version 1.25.5 \
--enable-managed-identity --generate-ssh-keys -l $LOC \
--load-balancer-sku standard --attach-acr $ACR \
--enable-addons confcom

az aks nodepool add --resource-group $RG --name intelsgx \
--cluster-name $AKS --node-count 1 --node-vm-size Standard_DC4s_v3 \
--enable-cluster-autoscaler --min-count 1 --max-count 5
```

VI. Provide access credentials to the managed Kubernetes cluster:

```
az aks get-credentials -n $AKS -g $RG
```

Upload Docker Image to Azure Container Registry

I. Log in to the Azure Container Registry:

```
az acr login -n $ACR
```

II. Upload the application image to the Azure Container Registry:

```
az acr build --image loan-default-app:latest --registry $ACR -g $RG --
file Dockerfile .
```

III. Verify the image was successfully pushed to the repository:

```
az acr repository show -n $ACR --repository loan-default-app -o table
```

Next Steps:
All Cloud Modules | GitHub* Repo | DevMesh on Discord*

Intel® Cloud Optimization Module for Microsoft Azure*:
XGBoost* Pipeline on Kubernetes*

Set up the Kubernetes Resources

I. Create a Kubernetes namespace:

```
export NS=intel-sgx-loan-app
kubectl create namespace $NS
```

II. Create a Kubernetes Secret object for Azure storage account:

```
export STORAGE_KEY=$(az storage account keys list -g $RG -n $STORAGE_NAME
--query [0].value -o tsv)
kubectl create secret generic azure-secret \
--from-literal azurestorageaccountname=$STORAGE_NAME \
--from-literal azurestorageaccountkey=$STORAGE_KEY \
--type=Opaque
```

III. Create a Kubernetes persistent volume:

```
kubectl create -f kubernetes/pv-azure.yaml -n $NS
kubectl create -f kubernetes/pvc-azure.yaml -n $NS
```

IV. Create a Kubernetes load balancer:

```
kubectl create -f kubernetes/loadbalancer.yaml -n $NS
```

V. Create a Kubernetes deployment:

```
kubectl create -f kubernetes/deployment.yaml -n $NS
```

VI. Create a Kubernetes horizontal pod autoscaler:

```
kubectl create -f kubernetes/hpa.yaml -n $NS
```

VII. Check that the Kubernetes resources were created and save the external IP address of the load balancer:

```
kubectl get all -n $NS
```

Deploy the Application

I. Process the data:

```
curl <external-IP>:8080/data_processing -H "Content-Type: multipart/form-
data" \
-F az_file_path=/loan_app/azure-fileshare \
-F data_directory=data \
-F file=@credit_risk_dataset.csv \
-F size=4000000 | jq
```

II. Train the XGBoost model:

```
curl <external-IP>:8080/train -H "Content-Type: multipart/form-data" \
-F az_file_path=/loan_app/azure-fileshare \
-F data_directory=data \
-F model_directory=models \
-F model_name=XGBoost \
-F continue_training=False \
-F size=4000000 | jq
```

III. Process new data:

```
curl <external-IP>:8080/data_processing -H "Content-Type: multipart/form-
data" \
-F az_file_path=/loan_app/azure-fileshare \
-F data_directory=data \
-F file=@credit_risk_dataset.csv \
-F size=1000000 | jq
```

IV. Perform model inference:

```
curl <external-IP>:8080/predict -H "Content-Type: multipart/form-data" \
-F file=@sample.csv \
-F az_file_path=/loan_app/azure-fileshare \
-F data_directory=data \
-F model_directory=models \
-F model_name=XGBoost \
-F sample_directory=samples| jq
```

Next Steps:

All Cloud Modules | GitHub* Repo | DevMesh on Discord*

Intel® Cloud Optimization Module for Microsoft Azure*:
XGBoost* Pipeline on Kubernetes*

Clean up Resources

I. Delete the Kubernetes Namespace:

```
kubectl delete namespace $NS
```

II. Turn off, or stop, the AKS Cluster:

```
az aks stop -n $AKS -g $RG
```

III. Delete all resources in the Resource Group:

```
az group delete -n $RG --yes --no-wait
```

XGBoost* dmlc XGBoost v1.x+

Optimizations for training and prediction on CPU are **upstreamed.**

Install the latest XGBoost with PyPi* or Anaconda* – newer versions have the most optimizations.

```
pip install xgboost
```

```
conda install xgboost -c conda-forge
```

Put data in an XGBoost DMatrix:

```
DMatrix = xgb.DMatrix(
X_train.values, y_train.values)
```

Train the XGBoost model:

```
model = xgb.train(params, Dmatrix,
num_boost_round=500)
```

Cheat Sheet | Docs | Medium Example

daal4py*

daal4py from the Intel® oneAPI Data Analytics Library (oneDAL) can be used to speed up inference of the XGBoost model. Install the latest daal4py:

```
pip install daal4py
```

```
conda install daal4py –c conda-forge
```

Convert a model to daal4py format from XGBoost:

```
d4p_model =
d4p.get_gbt_model_from_xgboost(model)
```

For optimized inference:

```
prediction = (d4p.
gbt_classification_prediction(
nClasses, resultsToEvaluate)
.compute(data, model)
.probabilities[:,1]
```

GitHub Repo | Docs | Medium Example

Next Steps:

All Cloud Modules | GitHub* Repo | DevMesh on Discord*