Solution Brief
Cross-Industry Architecture
Intel® Cloud Optimization Modules

intel software

Intel® Cloud Optimization Modules for AWS*: XGBoost* on Kubernetes*

Author: Eduardo Alvarez

Date: August 9, 2023

Build an accelerated Kubernetes cluster with Intel optimizations for XGBoost on an AWS computing cluster.

The **Intel Cloud Optimization Modules for AWS: XGBoost on Kubernetes** is designed to facilitate building and deploying accelerated AI applications on the AWS cloud with Elastic Kubernetes Service* (EKS) and the Intel® AI Analytics Toolkit. This reference solution focuses on an end-to-end loan default prediction application. We use the daal4py library (Python* API) as part of the Intel® oneAPI Data Analytics Library to accelerate inference with an XGBoost* classifier. By the end of the article, readers will gain a basic understanding of building and deploying performant AI applications on AWS, along with a practical example of leveraging these technologies for loan default prediction.

Use it as a reference solution for:

- Data Processing with Scikit-Learn* Extension
- XGBoost Training
- Accelerated XGBoost Inference
- Kubernetes applications on AWS with Elastic Kubernetes Service (EKS)
- Building API Endpoints with FastAPI*

Who needs it?

- Developers looking to leverage tools like Kubernetes and the Intel AI Analytics Toolkit to develop and deploy AI applications optimized for performance and scalability.
- Developers building AI applications on Intel® Xeon® CPU platforms in the AWS Cloud with M5, M6i, and M7i instances.
- Organizations building on AWS services and seeking to leverage AI application deployment processes, such as Amazon Elastic Kubernetes Service (EKS), Amazon Elastic Container Registry (ECR), EC2, and Elastic Load Balancer (ELB).

What it does

The module brings together an efficient, scalable, and unified cloud-native reference architecture ready to implement for enterprise AI workloads running on AWS.

XGBoost-Daal4py Kubernetes

- Provides a complete set of installation instructions for infrastructure deployment:
  - **Configures AWS cloud resources,** including an Elastic Kubernetes Cluster, Elastic Compute Cloud (EC2), Elastic Load Balancer (ELB), and more.
  - **Instructions for Configuring Compute Infrastructure** instance types, regions, scaling conditions, and more.
  - **Set up Kubernetes** services, deployment, pod auto-scaler, and more.
- **Implements a complete, end-to-end machine learning pipeline to predict credit risk using an XGBoost model,** from data preprocessing to model inference with **Intel oneDAL** optimizations.
- **Leverages oneDAL** which uses the **Intel® Advanced Vector Extensions 512 (Intel® AVX-512) instruction** set to maximize gradient boosting performance on **Intel® Xeon® processors**.

Cloud Solution Architecture

The architecture uses Docker for application containerization and Elastic Container (ECR) Storage on AWS. The application image is then deployed on a cluster managed by Elastic Kubernetes Service (EKS). Our clusters are made up of EC2 instances. We use S3 for storing data and model objects, which are retrieved during various steps of our ML pipeline. The client interacts with our infrastructure through our Elastic Load Balancer, which gets provisioned by our Kubernetes service.

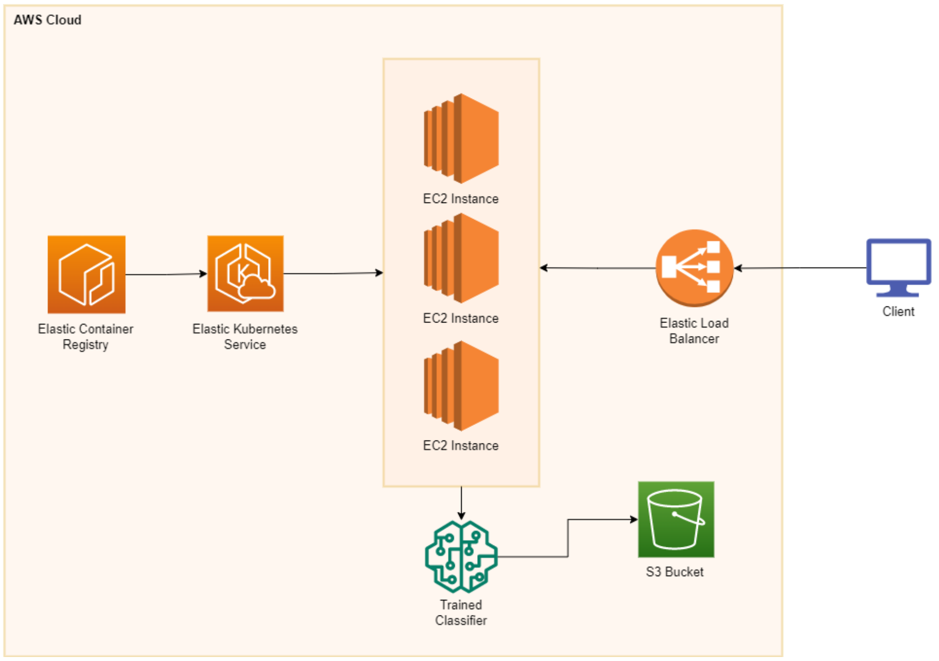

Figure 1 AWS Cloud Optimization Module Solution Architecture. Image by author.

Solution Component Overview

This solution is based on the **Loan Default Risk Prediction AI Reference Kit**. The code has been refactored to be more modular in support of our three main APIs:

- **Data processing:** This endpoint preprocesses and stores data in a data lake or another structured format. This codebase also handles the expansion of the dataset for benchmarking purposes.
- **Model Training:** This endpoint trains an XGBoost Classifier and converts it to an inference-optimized daal4py format.
- **Inference:** This endpoint receives a payload with raw data and returns the loan default classification of each sample.

The solution includes preconfigured docker files and FastAPI scripts to launch the FastAPI server and endpoints into Kubernetes pods within the EKS infrastructure. Instructions for achieving this are included in the optimization module's documentation.

Code Highlights

Enable the Intel optimizations for XGBoost

XGBoost optimizations for training and inference on CPUs are upstreamed. Ensure you are using the latest version for the most Intel optimizations. The following code is implemented in the **model.py** python module.

```
# Create the XGBoost DMatrix, which is optimized for memory efficiency and training speed
self.DMatrix = xgb.DMatrix(self.X_train.values, self.y_train.values)

# Define model parameters
params = {"objective": "binary:logistic",
          "eval_metric": "logloss",
          "nthread": 4,  # num_cpu
          "tree_method": "hist",
          "learning_rate": 0.02,
          "max_depth": 10,
          "min_child_weight": 6,
          "n_jobs": 4,  # num_cpu,
          "verbosity": 1}

# Train initial XGBoost model
self.clf = xgb.train(params, self.DMatrix, num_boost_round=500)
```

Convert the trained XGBoost model to daal4py

Daal4py is the Python API of the oneAPI Data Analytics Library, oneDAL. Daal4py helps to further optimize model prediction, or inference, on CPUs. The following code is implemented in the **model.py** and **predict.py** python modules.

```
# Convert XGBoost model to daal4py
self.clf = d4p.get_gbt_model_from_xgboost(self.clf)

# Compute both class labels and probabilities
daal_prediction = d4p.gbt_classification_prediction(
        nClasses = 2,
        resultsToEvaluate = "computeClassLabels|computeClassProbabilities"
).compute(X_test, daal_model)
```

Enable the Scikit-Learn Extension

The **Intel® Extension for Scikit-Learn*** provides CPU accelerations for many scikit-learn libraries. Below is an example of how to import the scikit-learn extension and relevant libraries for the **data.py** python module.

```
# Call patch_sklearn() before importing scikit-learn libraries
from sklearnex import patch_sklearn
patch_sklearn()

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, PowerTransformer
```

Next Steps

Download the module from GitHub ›

Register for office hours to get help on your implementation ›

Check out the full suite of Intel Cloud Optimization Modules ›

Come chat with us on our DevHub Discord server to keep interacting with other developers ›