Workshop: Training your Immo Eliza Model in AWS Environment

Hi there,

My name is Andreia Negreira, I am an ex-becodian and I am happy to bring this workshop for you! The idea of our workshop is to use AWS cloud computing resources to train our most loved immo eliza model.

To ensure our workshop runs smoothly, I would like to ask you to complete some small preparations in advance. These are basic tasks but can take some time, so please read these instructions carefully and ensure everything is set up and tested beforehand.

What You Need to Have Ready:

- An AWS account;
- A file with your AWS credentials;
- Installation of AWS CLI;
- Import of your AWS profile to the CLI;
- Installation of Terraform;
- Import of your AWS profile to Terraform.
- Prepare your script to run in an AWS Ray Cluster.

Step-by-Step Instructions:

1. Create an AWS Account

If you do not already have an AWS account, please create one:

1. Go to the AWS Sign-Up page: https://aws.amazon.com/
2. Click on "Create an AWS Account"
3. Follow the on-screen instructions to complete the registration process

2. Obtain AWS Credentials

Once you have an AWS account, you need to create an access key (Access Key ID and Secret Access Key):

1. Sign in to the AWS Management Console
2. Navigate to the IAM (Identity and Access Management) service
3. In the navigation pane, choose "Users" and then select your username

4. Go to the "Security credentials" tab
5. Under "Access keys," click "Create access key"
6. Download the .csv file containing your credentials and keep it secure

3. Install AWS CLI

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services:

1. Follow the instructions on the AWS CLI installation guide to install the AWS CLI on your operating system: https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

4. Configure AWS CLI with Your AWS Profile

After installing the AWS CLI, configure it with your credentials:

1. Open a terminal or command prompt
2. Run the command: **aws configure**
3. Enter your Access Key ID, Secret Access Key, default region name (e.g., eu-west-1), and default output format (e.g., json) when prompted

5. Install Terraform

Terraform is an open-source infrastructure as code software tool:

1. Download the appropriate package for your operating system from the Terraform downloads page: https://developer.hashicorp.com/terraform/install
2. Follow the installation instructions for your operating system (e.g., unzip the package and move the binary to your system's PATH)

6. Configure Terraform with Your AWS Profile

To use Terraform with your AWS credentials, you need to set up the AWS provider:

1. Ensure your AWS CLI configuration is working correctly by running: **aws sts get-caller-identity**
2. Create a directory for your Terraform configuration files called on VScode, for ex.: terraform-workshop
3. Inside this directory, create a file named “provider.tf” with the following content:

```
provider "aws" {
region = " eu-west-1"
}
```

4. Initialize your Terraform configuration: **terraform init**

Testing Your Setup

AWS CLI Test:

1. Run: **aws s3 ls**
   This should list your S3 buckets if everything is configured correctly

Terraform Test:

1. In your terraform-workshop directory, create a simple configuration file called "main.tf", with the following content:

   ```
   resource "aws_s3_bucket" "test_bucket" {
   bucket = "my-unique-bucket-name"
   }
   ```

2. Run: **terraform plan**
   This should output the execution plan if everything is set up correctly

7. Update your script to run in a Ray Cluster

You probably got a script allongside with these instructions. I will not ask you to understand what Ray is for now, as it is one of the topics that will be covered during our workshop, but if you want to use your own script, it is important for you to adapt it to be supported in a Ray cluster. It is optional, you can also use my own script and data, it will be provided as well.

1. **The import block:** I use Random Forest Regressor and pandas to perform the cleaning and have a dataframe in my operations. I also save my model at the end in pickle format. Please import the necessary packages, but keep in mind that pandas, boto3, ray and BytesIO are necessary. Also check if the import of other models are supported in ray: https://docs.aws.amazon.com/glue/latest/dg/edit-script-ray-env-dependencies.html

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as MSE
from sklearn.ensemble import RandomForestRegressor
```

```
import pickle
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.pipeline import Pipeline
import ray
import boto3
from io import BytesIO
```

2. In the following block, we will be initialising our cluster. The functions for reading from and writing to S3 are already defined with the @ray.remote decorator. This decorator indicates that these functions can be executed in parallel across the Ray cluster. There's no need for you to change this block, only if you want another format for your model at the end.

```
ray.init()

@ray.remote
def read_csv_from_s3(bucket_name, file_key):
    s3 = boto3.client('s3')
    response = s3.get_object(Bucket=bucket_name, Key=file_key)
    data = response['Body'].read()
    df = pd.read_csv(BytesIO(data))
    return df

@ray.remote
def write_model_to_s3(bucket_name, file_key, model):
    s3 = boto3.client('s3')
    buffer = BytesIO()
    pickle.dump(model, buffer)
    s3.put_object(Bucket=bucket_name, Body=buffer.getvalue(), Key=file_key)
```

3. The functions **cleaning** and **model** are the most important ones for training a model with efficiency. Change the content of those functions with your own logic and be mindful about what the function is returning.

```
def cleaning(df):
    df.rename(columns={'swimming-pool': 'swimming_pool', 'state-building': 'state_building', 'land-surface': 'land_surface'}, inplace=True)
    df.drop(df[df['state_building'] == "0"].index, inplace=True)
    df['state_building'] = df['state_building'].astype(str)
    df.duplicated()
    df.fillna(0, inplace=True)
    df.drop(df[df['locality'] == 0].index, inplace=True)
    df.drop(columns=['type-transaction', 'url', 'area_terrace', 'area-garden', 'n-facades'], inplace=True)
    return df

def model(df):
    df.dropna(inplace=True)
    X = df.drop(['price'], axis=1)
    y = df['price']
    trans_1 = ColumnTransformer([
        ('ohe_trans', OneHotEncoder(sparse=False, handle_unknown='ignore'), [0, 1, 2, 12])
    ], remainder='passthrough')
    trans_2 = ColumnTransformer([
        ('scale', MinMaxScaler(), slice(0, len(X)+1))
    ], remainder='passthrough')
    trans_3 = RandomForestRegressor(random_state=3)
    pipeline = Pipeline(steps=[('trans_1', trans_1), ('trans_2', trans_2), ('trans_3', trans_3)])
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    pipeline.fit(X_train, y_train)
    pred = pipeline.predict(X_test)
    rmse = np.sqrt(MSE(y_test, pred))
    return pipeline
```

4. The value of the variables in the **main block** will be updated during the workshop, so don't worry about it for the moment. Just try to already have a directory with the script updated and your csv file, in case you want to use your own content.

And next... Joking haha, for now, that's it!

If you got it all, that's great, now you have everything ready for the workshop! If you encounter any issues, please reach out for help before the workshop begins. Thank you, and I look forward to seeing you!

Best regards,

Andreia Negreira

My e-mail: andreia.negreira@bigindustries.be

Useful stackoverflow: https://stackoverflow.com/questions/64124063/how-to-make-terraform-to-read-aws-credentials-file

Offical hashicorp documentation for AWS provider: https://registry.terraform.io/providers/hashicorp/aws/latest/docs