
CS 643 Cloud Computing: Programming Assignment 2

Energy Consumption Forecasting with Apache Spark and Docker

Hemanth Sruthi Vellore [HV234]

May 6, 2025

Abstract: This report presents an energy consumption forecasting system developed for CS 643 Cloud Computing Programming Assignment 2, leveraging Apache Spark on Amazon AWS. The project encompasses parallel model training across multiple EC2 instances, a streamlined single-instance prediction application, and a Docker container for seamless deployment. Powered by a Gradient Boosted Trees regression model from Spark MLlib, the system is trained on `TrainingDataset.csv` and validated with `ValidationDataset.csv`. This document provides a comprehensive guide for configuring the cloud infrastructure, executing the training and prediction workflows, and deploying the Docker container. It also evaluates the role of AI tools (ChatGPT) in code development, offering attribution and insights into the development process.

Contents

| | | |
|----------|--|-----------|
| 1 | Technical Architecture | 2 |
| 1.1 | Distributed Model Training | 2 |
| 1.1.1 | Cluster Provisioning | 2 |
| 1.1.2 | Uploading Files to the EMR Cluster Master Node | 4 |
| 1.1.3 | HDFS Data Integration | 5 |
| 1.1.4 | Model Training Workflow | 5 |
| 1.1.5 | Training Performance Metrics | 6 |
| 1.1.6 | Model Export Process | 6 |
| 1.2 | Single-Instance Prediction System | 7 |
| 1.2.1 | EC2 Instance Creation | 7 |
| 1.2.2 | EC2 Instance Setup | 9 |
| 1.2.3 | Environment Configuration | 9 |
| 1.2.4 | Spark Logging Configuration | 10 |
| 1.2.5 | Load Model and Script | 10 |
| 1.2.6 | Unzip the Model | 11 |
| 1.2.7 | Run the Prediction Script | 11 |
| 1.3 | Docker-Enabled Deployment | 12 |
| 1.3.1 | Dockerfile Configuration | 12 |
| 1.3.2 | Container Build and Deployment | 13 |
| 2 | Deployment Guide | 15 |
| 2.1 | Setting Up Distributed Training | 15 |
| 2.2 | Configuring the Prediction System | 15 |
| 2.3 | Deploying with Docker | 15 |
| 3 | AI-Assisted Development | 16 |
| 4 | Code and Resource Links | 17 |

1 Technical Architecture

The system is structured around three key components: distributed model training, single-instance prediction, and Docker-based deployment.

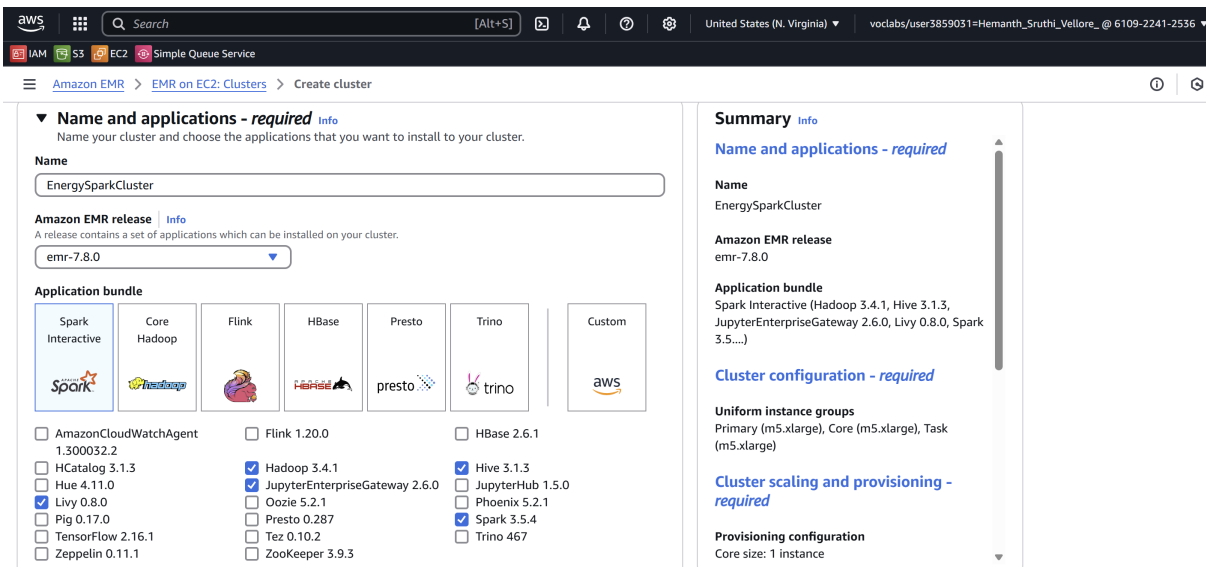
1.1 Distributed Model Training

This component utilizes Apache Spark on a cluster of four EC2 instances to train a GBT regression model on `TrainingDataset.csv`.

1.1.1 Cluster Provisioning

A Spark cluster is provisioned on AWS EC2 instances using the following steps:

1. From the AWS console, go to EMR Service and click Create Cluster.
2. Configure the cluster as follows:
 - (a) Enter cluster name `EnergySparkCluster`
 - (b) Select Release `emr-7.8.0`
 - (c) Choose Spark Interactive: Spark 3.5.4, Hadoop 3.4.1



- (d) Configure Hardware Configurations:
 - i. Select OS option (Amazon Linux release).
 - ii. Select the instance type (m5.xlarge).

- iii. Set number of instances to 4 (1 master, 3 slaves).
- (e) Select the "vockey " key pair
- (f) Select bootstrap script from the S3 bucket.

3. Click on Create Cluster.

The cluster is initialized with a bootstrap script to install dependencies.

Listing 1: Bootstrap Script (boot.sh)

```
1 sudo pip3 install numpy pandas
```

The screenshot displays the AWS Management Console interface for an Amazon EMR cluster. The cluster is named 'EnergySparkCluster' and is in the 'Waiting' status. The console shows various tabs for cluster management, including Properties, Bootstrap actions, Instances (Hardware), Steps, Applications, Configurations, Monitoring, Events, and Tags (0).

Cluster info:

- Cluster ID: j-MFMYW33YWDY9
- Cluster ARN: arnaws:elasticmapreduce:us-east-1:610922412536:cluster/j-MFMYW33YWDY9
- Cluster configuration: Instance groups
- Capacity: 1 Primary | 3 Core | 0 Task

Applications:

- Amazon EMR version: emr-7.8.0
- Installed applications: Hadoop 3.4.1, Hive 3.1.3, JupyterEnterpriseGateway 2.6.0, Livy 0.8.0, Spark 3.5.4

Cluster management:

- Log destination in Amazon S3: pa2-cc
- Persistent application Uls: Spark History Server, YARN timeline server, Tez UI
- Primary node public DNS: ec2-34-201-39-83.compute-1.amazonaws.com
- Connect to the Primary node using SSH
- Connect to the Primary node using SSH

Status and time:

- Status: Waiting
- Creation time: May 06, 2025, 01:29 (UTC-04:00)
- Elapsed time: 7 minutes, 52 seconds

Instance group settings:

- Cluster scaling option: Manually set cluster size
- Core: Name and Maximum core nodes in the cluster, Core | 3 instances

Instance groups (2):

| Type and name | ID | Status | Instances | Purchasing option and p... | EBS size (GiB) | EC2 Instance ID | Public DI |
|---------------|------------------|---------|-----------|----------------------------|----------------|-----------------|-----------|
| Primary | ig-1R240DSGLE8AY | Running | 1 | On-Demand | - | - | - |
| Core | ig-1CJ9MUBZUSBX2 | Running | 3 | On-Demand | - | - | - |

1.1.2 Uploading Files to the EMR Cluster Master Node

After the EMR cluster has been successfully provisioned, transfer the required files to the master node using a secure file transfer client such as WinSCP. Establish a connection to the master node with the appropriate SSH credentials. Upload the `TrainingDataset.csv` file and the `energy_model_trainer.py` script to the home directory of the hadoop user.

The screenshot shows the WinSCP interface with a local file explorer on the left and a remote file explorer on the right. The local file explorer shows the directory structure of the user's home directory, including files like `boot.sh`, `commands.txt`, `energy_model_trainer.py`, `energy_usage_predict.py`, `labsuser.ppk`, `pa2.pdf`, `requirements.txt`, `result.txt`, `ssourl.txt`, `TrainingDataset.csv`, and `ValidationDataset.csv`. The remote file explorer shows the directory structure of the master node, including files like `aws`, `.config`, `.ssh`, `.bash_profile`, `.bashrc`, `energy_model_trainer.py`, and `TrainingDataset.csv`.

The status bar at the bottom indicates that 0 B of 248 KB is being transferred in 0 of 12, and 0 B of 41.4 KB is being transferred in 0 of 7. The connection is established via SFTP-3.

1.1.3 HDFS Data Integration

Now that all the files are on the master node, we want to migrate them to HDFS so that all the slave nodes can access them without having to physically copy them to all EC2 nodes. SSH into the master node and run the below commands to copy the files to HDFS.

Listing 2: HDFS Integration Commands

```
1 hadoop fs -put TrainingDataset.csv /user/hadoop/TrainingDataset.csv
2 hadoop fs -put energy_model_trainer.py /user/hadoop/energy_model_trainer.py
3 hdfs dfs -ls -t -R
```

Figure 1: Data integration and verification.

```
hadoop@ip-172-31-86-13:~  
Using username "hadoop".  
Authenticating with public key "imported-openssh-key"  
#  
Amazon Linux 2023  
  
https://aws.amazon.com/linux/amazon-linux-2023  
  
EEEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRRRRRRRRR  
E:::EEEEE E:::M M:::MM R:::R  
EE:::EEEEEEEEEE E:::M M:::MM R:::RRRRRR:::R  
E:::E EEEE M:::MM M:::MM RR:::R R:::R  
E:::E M:::MM M:::MM R:::R R:::R  
E:::EEEEEEEEEE M:::MM M:::MM M:::MM R:::RRRRRR:::R  
E:::EEEEE E:::M M:::MM M:::MM R:::RRRRRR:::R  
E:::E M:::MM M:::MM R:::R R:::R  
E:::E EEEE M:::MM MMM M:::MM R:::R R:::R  
EE:::EEEEEEEEEE E:::M M:::MM R:::R R:::R  
E:::EEEEE E:::M M:::MM RR:::R R:::R  
EEEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRR RRRRRR
```

```
[hadoop@ip-172-31-86-13 ~]$ ls  
TrainingDataset.csv energy_model_trainer.py  
[hadoop@ip-172-31-86-13 ~]$ hadoop fs -put TrainingDataset.csv /user/hadoop/TrainingDataset.csv  
[hadoop@ip-172-31-86-13 ~]$ hadoop fs -put energy_model_trainer.py /user/hadoop/energy_model_trainer.py  
[hadoop@ip-172-31-86-13 ~]$ hdfs dfs -ls -t -R  
-rw-r--r-- 1 hadoop hdfsadmingroup 35873 2025-05-06 05:53 TrainingDataset.csv  
-rw-r--r-- 1 hadoop hdfsadmingroup 5890 2025-05-06 05:54 energy_model_trainer.py  
[hadoop@ip-172-31-86-13 ~]$
```

1.1.4 Model Training Workflow

The `energy_model_trainer.py` script, executed via `spark-submit`, trains the GBT model by loading the dataset, encoding categorical features, assembling feature vectors, splitting data into training and testing sets, training the model, evaluating performance (RMSE and R^2), and saving the model to `EnergyPredictorGBT`.

Listing 3: Training Workflow Command

```
1 spark-submit energy_model_trainer.py
```

1.1.5 Training Performance Metrics

The model achieved a Root Mean Squared Error (RMSE) of 70.4804 and a coefficient of determination (R^2) of 0.9942 on the training dataset, indicating a strong fit and high predictive accuracy.

```

hadoop@ip-172-31-86-13:~$
Encoding 'Building_Type' with mappings: {'Commercial': 0, 'Industrial': 1, 'Residential': 2}
Encoding 'Day_of_Week' with mappings: {'Weekday': 0, 'Weekend': 1}
Categorical features successfully encoded.

[Step 4] Finalizing Dataset for Model Training...
Casting all columns to float...
Schema after feature transformation:
root
  |-- Square_Footage: float (nullable = true)
  |-- Number_of_Occupants: float (nullable = true)
  |-- Appliances_Used: float (nullable = true)
  |-- Average_Temperature: float (nullable = true)
  |-- Energy_Consumption: float (nullable = true)
  |-- Building_Type_Indexed: float (nullable = true)
  |-- Day_of_Week_Indexed: float (nullable = true)

[Step 5] Preparing Feature Vectors and Labels...
Prepared 860 labeled data points.

[Step 6] Splitting Data into Training and Testing Sets...
Training Set: 560 samples
Testing Set: 300 samples

[Step 7] Training the Regression Model...
Model training completed successfully in 23.91 seconds.

[Step 8] Evaluating Model on Testing Data...
Root Mean Squared Error (RMSE): 70.4804
R2 Score (Coefficient of Determination): 0.9942

[Step 9] Saving the Trained Regression Model...
Model successfully saved to path: 'EnergyPredictorGBT'

=====
ENERGY CONSUMPTION PREDICTION SYSTEM COMPLETED
=====

Spark Context stopped successfully. Application terminated.

[hadoop@ip-172-31-86-13 ~]$ hdfs dfs -ls -t -R
-hexa----- 1 hadoop hdfs@hadoopgroup 0 2025-05-06 05:57 EnergyPredictorGBT/
-hexa----- 1 hadoop hdfs@hadoopgroup 0 2025-05-06 05:57 EnergyPredictorGBT/data/
-hexa----- 1 hadoop hdfs@hadoopgroup 2713 2025-05-06 05:57 EnergyPredictorGBT/data/part-00000-62067a7c-570b-474c-b3bd-4407b0536d5b-c000.snappy.parquet
-hexa----- 1 hadoop hdfs@hadoopgroup 0 2025-05-06 05:57 EnergyPredictorGBT/metadata/
-hexa----- 1 hadoop hdfs@hadoopgroup 0 2025-05-06 05:57 EnergyPredictorGBT/metadata/_SUCCESS
-hexa----- 1 hadoop hdfs@hadoopgroup 107 2025-05-06 05:57 EnergyPredictorGBT/metadata/part-00000
-hexa----- 1 hadoop hdfs@hadoopgroup 35073 2025-05-06 05:53 TrainingDataset.csv
-hexa----- 1 hadoop hdfs@hadoopgroup 5080 2025-05-06 05:54 energy_model_trainer.py
[hadoop@ip-172-31-86-13 ~]$

```

1.1.6 Model Export Process

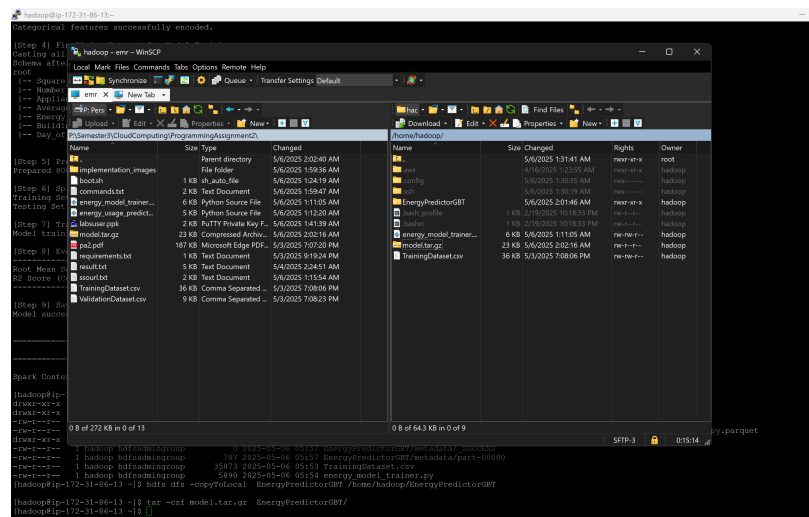
The trained model was exported to the master node's local file system, compressed into an archive, and downloaded to the local machine using WinSCP. After verifying the download, the EMR cluster was shut down to avoid unnecessary resource usage.

Listing 4: Model Export Commands

```

1 hdfs dfs -copyToLocal EnergyPredictorGBT /home/hadoop/EnergyPredictorGBT
2 tar -czf model.tar.gz EnergyPredictorGBT/

```



```

hadoop@ip-172-31-86-13:~$
[Step 4] Finalizing Dataset for Model Training...
Casting all columns to float...
Schema after feature transformation:
root
  |-- Square_Footage: float (nullable = true)
  |-- Number_of_Occupants: float (nullable = true)
  |-- Appliances_Used: float (nullable = true)
  |-- Average_Temperature: float (nullable = true)
  |-- Energy_Consumption: float (nullable = true)
  |-- Building_Type_Indexed: float (nullable = true)
  |-- Day_of_Week_Indexed: float (nullable = true)

[Step 5] Preparing Feature Vectors and Labels...
Prepared 860 labeled data points.

[Step 6] Splitting Data into Training and Testing Sets...
Training Set: 560 samples
Testing Set: 300 samples

[Step 7] Training the Regression Model...
Model training completed successfully in 23.91 seconds.

[Step 8] Evaluating Model on Testing Data...
Root Mean Squared Error (RMSE): 70.4804
R2 Score (Coefficient of Determination): 0.9942

[Step 9] Saving the Trained Regression Model...
Model successfully saved to path: 'EnergyPredictorGBT'

=====
ENERGY CONSUMPTION PREDICTION SYSTEM COMPLETED
=====

Spark Context stopped successfully. Application terminated.

[hadoop@ip-172-31-86-13 ~]$ hdfs dfs -copyToLocal EnergyPredictorGBT /home/hadoop/EnergyPredictorGBT
[hadoop@ip-172-31-86-13 ~]$ tar -czf model.tar.gz EnergyPredictorGBT/
[hadoop@ip-172-31-86-13 ~]$

```

1.2 Single-Instance Prediction System

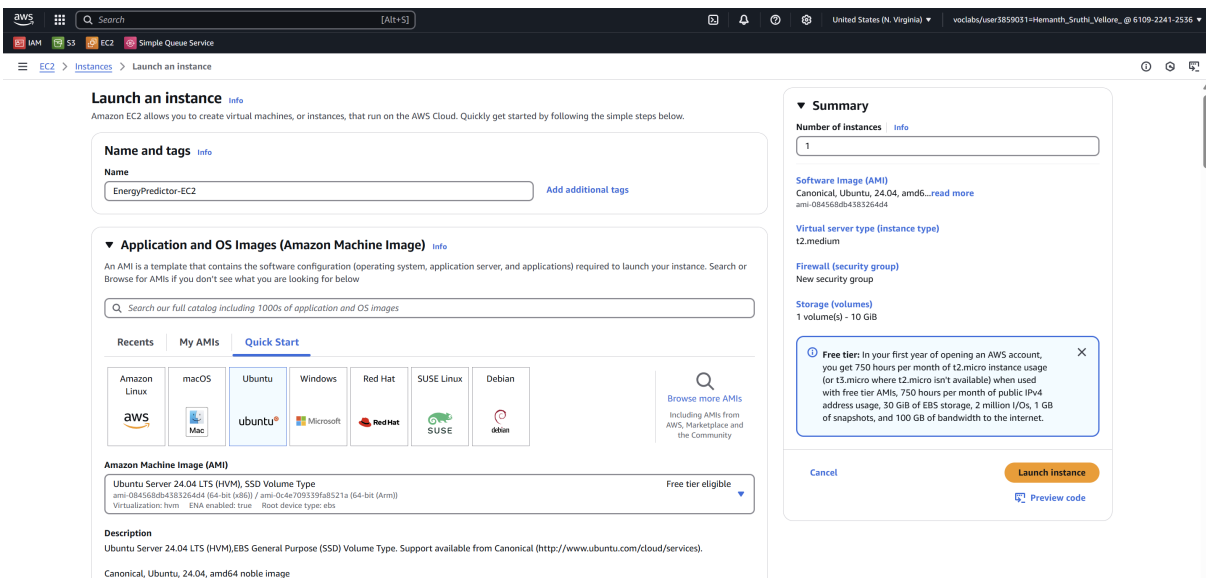
The prediction system runs on a single EC2 instance using `energy_usage_predictor.py` and the saved model.

1.2.1 EC2 Instance Creation

Provisioned an EC2 instance via the AWS Management Console by:

- Naming the instance `EnergyPredictor-EC2`
- Selecting `Ubuntu Server 24.04 LTS (HVM) AMI`
- Choosing `t2.medium` instance type
- Setting root volume to 10 GB (gp3)
- Assigning the `EMR_EC2_DefaultRole` IAM role
- Selecting the vockey key pair
- Launching the instance

Instance initialized and ready for setup.



AWS Search [Alt+S] IAM S3 EC2 Simple Queue Service

United States (N. Virginia) voclabs/user3859031-Hemanth_Sruthi_Velloru_@ 6109-2241-2536

EC2 > Instances > Launch an instance

Instance type Info | Get advice

Instance type

t2.medium

Family: t2 2 vCPU 4 GiB Memory Current generation: true

On-Demand Ubuntu Pro base pricing: 0.0499 USD per Hour On-Demand Linux base pricing: 0.0464 USD per Hour

On-Demand HWE base pricing: 0.0752 USD per Hour On-Demand Windows base pricing: 0.0644 USD per Hour

On-Demand SUSE base pricing: 0.1464 USD per Hour

Additional costs apply for AMIs with pre-installed software

Key pair (login) Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

vockey

Create new key pair

Network settings Info

Network

vpc-00bb3ab99faa17a85

Subnet

No preference (Default subnet in any availability zone)

Auto-assign public IP

Enable

Additional charges apply when outside of free tier allowance

Firewall (security groups)

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group Select existing security group

Summary

Number of instances Info

1

Software Image (AMI)

Canonical, Ubuntu, 24.04, amd64...read more

ami-084568db4383264d4

Virtual server type (instance type)

t2.medium

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 10 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GiB of snapshots, and 100 GiB of bandwidth to the internet.

Cancel Launch instance Preview code

AWS Search [Alt+S] IAM S3 EC2 Simple Queue Service

United States (N. Virginia) voclabs/user3859031-Hemanth_Sruthi_Velloru_@ 6109-2241-2536

EC2 > Instances > Launch an instance

Configure storage Info Advanced

1x 10 GiB gp3 Root volume, 3000 IOPS, Not encrypted

Free tier eligible customers can get up to 30 GiB of EBS General Purpose (SSD) or Magnetic storage

Add new volume

The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

Click refresh to view backup information

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

0 x File systems

Advanced details Info

Domain join directory

Select Create new directory

IAM Instance profile

EMR_EC2_DefaultRole

Create new IAM profile

Hostname type

IP name

DNS Hostname

Enable IP name IPv4 (A record) DNS requests

Summary

Number of instances Info

1

Software Image (AMI)

Canonical, Ubuntu, 24.04, amd64...read more

ami-084568db4383264d4

Virtual server type (instance type)

t2.medium

Firewall (security group)

New security group

Storage (volumes)

1 volume(s) - 10 GiB

Free tier: In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GiB of snapshots, and 100 GiB of bandwidth to the internet.

Cancel Launch instance Preview code

AWS Search [Alt+S] IAM S3 EC2 Simple Queue Service

United States (N. Virginia) voclabs/user3859031-Hemanth_Sruthi_Velloru_@ 6109-2241-2536

EC2 > Instances

EC2 Instances

Dashboard EC2 Global View Events

Instances

Instances Instance Types Launch Templates Spot Requests Savings Plans Reserved Instances Dedicated Hosts Capacity Reservations

Images

AMIs AMI Catalog

Elastic Block Store

Volumes Snapshots Lifecycle Manager

Network & Security

Security Groups Elastic IPs Placement Groups Key Pairs Network Interfaces

Load Balancing

Instances (1/1) Info

Find Instance by attribute or tag (case-sensitive) All states

Instance state = running Clear filters

| Name | Instance ID | Instance state | Instance type | Status check | Alarm status | Availability Zone | Public IPv4 DNS | Public IPv4 ... | Elastic IP |
|------------------|---------------------|----------------|---------------|--------------|--------------|-------------------|-------------------------|-----------------|------------|
| EnergyPredict... | i-0417e58492b3eb7d2 | Running | t2.medium | Initializing | View alarms | us-east-1d | ec2-54-160-142-89.co... | 54.160.142.89 | - |

i-0417e58492b3eb7d2 (EnergyPredictor-EC2)

Details Status and alarms Monitoring Security Networking Storage Tags

Instance summary Info

Instance ID

i-0417e58492b3eb7d2

IPv6 address

-

Hostname type

IP name: ip-172-31-26-90.ec2.internal

Answer private resource DNS name

IPv4 (A)

Auto-assigned IP address

54.160.142.89 [Public IP]

IAM Role

EMR_EC2_DefaultRole

IMDSv2

Public IPv4 address

54.160.142.89 [open address]

Instance state

Running

Private IP DNS name (IPv4 only)

ip-172-31-26-90.ec2.internal

Instance type

t2.medium

VPC ID

vpc-00bb3ab99faa17a85

Subnet ID

subnet-09397293af8c0785

Instance ARN

Private IPv4 addresses

172.31.26.90

Public IPv4 DNS

ec2-54-160-142-89.compute-1.amazonaws.com [open address]

Elastic IP addresses

-

AWS Compute Optimizer finding

Opt-in to AWS Compute Optimizer for recommendations. [Learn more]

Auto Scaling Group name

-

Managed

1.2.2 EC2 Instance Setup

After creating the instance, SSH access is established using the key pair. The system is updated, and essential dependencies, including Java, Python, and Apache Spark, are installed and configured to prepare the environment.

Listing 5: Instance Setup Commands

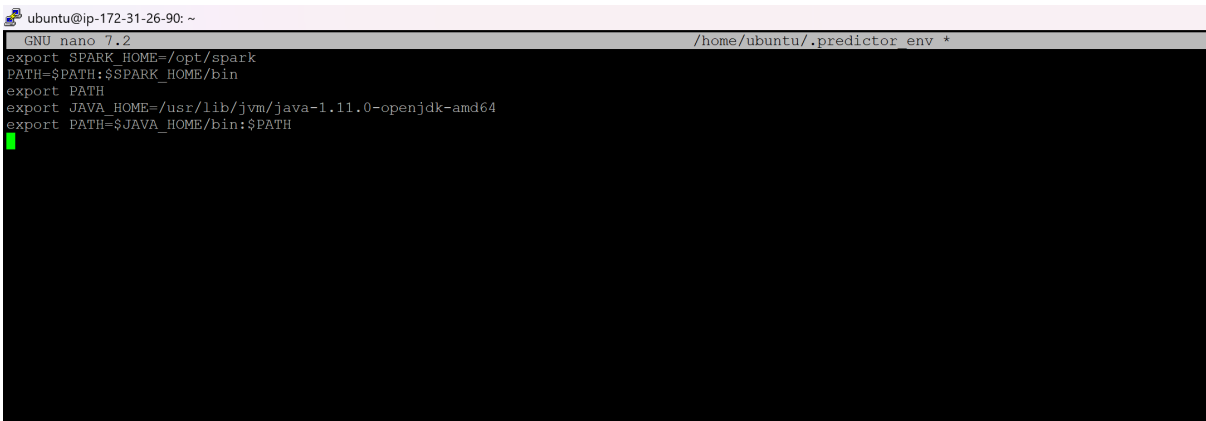
```
1 sudo apt-get update
2 sudo apt-get install -y python3-pip python3-numpy python3-pandas openjdk-11-
  jdk
3 wget https://archive.apache.org/dist/spark/spark-3.5.5/spark-3.5.5-bin-
  hadoop3.tgz
4 sudo tar xvf spark-3.5.5-bin-hadoop3.tgz -C /opt
5 sudo chown -R ubuntu:ubuntu /opt/spark-3.5.5-bin-hadoop3
6 sudo ln -fs spark-3.5.5-bin-hadoop3 /opt/spark
```

1.2.3 Environment Configuration

Environment variables are set for Spark and Java.

Listing 6: Environment Configuration (~/.predictor_env)

```
1 export SPARK_HOME=/opt/spark
2 PATH=$PATH:$SPARK_HOME/bin
3 export PATH
4 export JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
5 export PATH=$JAVA_HOME/bin:$PATH
6 source ~/.predictor_env
```



The screenshot shows a terminal window with the title bar 'ubuntu@ip-172-31-26-90: ~'. The terminal is running the GNU nano 7.2 text editor, editing the file '/home/ubuntu/.predictor_env *'. The file contains the following lines: 'export SPARK_HOME=/opt/spark', 'PATH=\$PATH:\$SPARK_HOME/bin', 'export PATH', 'export JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64', and 'export PATH=\$JAVA_HOME/bin:\$PATH'. A green cursor is visible at the end of the last line.

1.2.4 Spark Logging Configuration

To reduce console output during execution, adjust the default Spark logging level:

- Copy the template logging configuration file.
- Edit the `log4j2.properties` file, changing `rootLogger.level` from `INFO` to `ERROR`.

Listing 7: Logging Configuration

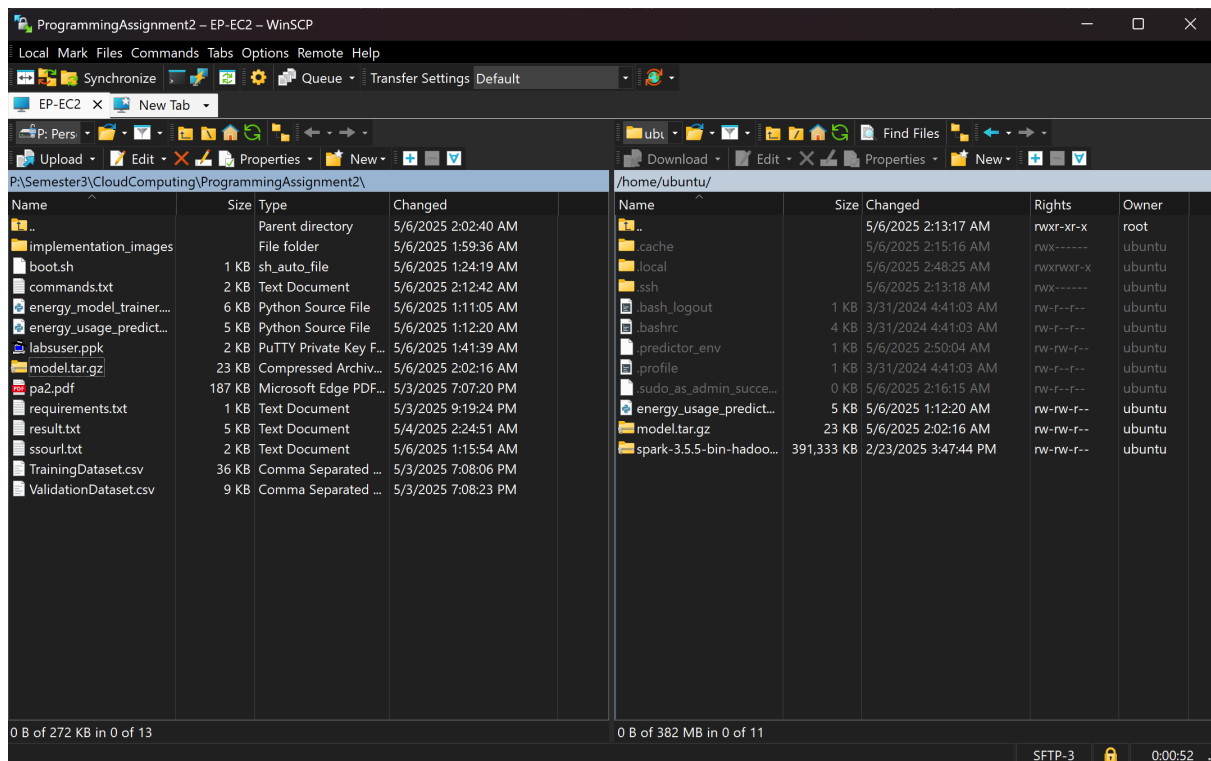
```

1 cp $SPARK_HOME/conf/log4j2.properties.template $SPARK_HOME/conf/log4j2.
   properties
2 nano $SPARK_HOME/conf/log4j2.properties
3 # Change rootLogger.level = info to ERROR

```

1.2.5 Load Model and Script

Using WinSCP, transfer the previously exported model and the `energy_usage_predictor.py` script to the EC2 instance.



1.2.6 Unzip the Model

Once the files are transferred, unzip the model archive using the following command:

Listing 8: Unzipping the Model

```
1 tar -xzf model.tar.gz
```

```
ubuntu@ip-172-31-26-90: ~
ubuntu@ip-172-31-26-90:~$ tar -xzf model.tar.gz
EnergyPredictorGBT/
EnergyPredictorGBT/data/
EnergyPredictorGBT/data/ SUCCESS
EnergyPredictorGBT/data/part-00000-62067a7c-570b-474c-b30d-4487c8536d5b-c000.snappy.parquet
EnergyPredictorGBT/metadata/
EnergyPredictorGBT/metadata/ SUCCESS
EnergyPredictorGBT/metadata/part-00000
```

1.2.7 Run the Prediction Script

Execute the `energy_usage_predictor.py` script using `spark-submit` to generate predictions from the validation dataset. The script loads the dataset, preprocesses it, loads the GBT model, generates predictions, and reports RMSE and R^2 metrics. The results for the validation set are:

- RMSE: 73.8957
- R^2 : 0.9933

Listing 9: Prediction Execution Commands

```
1 spark-submit energy_usage_predictor.py ValidationDataset.csv
```

```
ubuntu@ip-172-31-26-90: ~
ubuntu@ip-172-31-26-90:~$ ls
EnergyPredictorGBT ValidationDataset.csv energy_usage_predictor.py model.tar.gz spark-3.5.5-bin-hadoop3.tgz
ubuntu@ip-172-31-26-90:~$ spark-submit energy_usage_predictor.py ValidationDataset.csv

=====
ENERGY CONSUMPTION PREDICTOR
=====
A Spark-powered application for estimating energy usage on unseen datasets,
leveraging a pre-trained Gradient Boosted Trees regression model.

[STEP] Setting up Spark environment..... Completed
[STEP] Importing test data..... Completed
[STEP] Standardizing column names..... Completed
[STEP] Transforming categorical features..... Completed
[STEP] Removing original categorical fields..... Completed
[STEP] Casting columns to float..... Completed
[STEP] Preparing features and target variable..... Completed
[STEP] Converting DataFrame to RDD format..... Completed
[STEP] Retrieving pre-trained model..... Completed
[STEP] Generating predictions..... Completed
[STEP] Aligning predictions with true values..... Completed
[STEP] Evaluating prediction performance..... Completed

=====
Prediction Performance Report
=====
Root Mean Squared Error (RMSE)          73.8957
R2 (Coefficient of Determination)         0.9933
=====

=====
ENERGY CONSUMPTION PREDICTOR COMPLETED
=====
[STEP] Terminating Spark environment..... Completed
ubuntu@ip-172-31-26-90:~$
```

1.3 Docker-Enabled Deployment

A Docker container is built to facilitate portable deployment of the prediction system.

1.3.1 Dockerfile Configuration

The Dockerfile sets up the environment and installs dependencies to run the Spark application.

- Base Image: Uses `ubuntu:20.04`.
- Dependencies: Installs Python 3, Java 11, Spark, and necessary tools.
- Environment Variables: Configures `JAVA_HOME` and `SPARK_HOME`.
- Copy Files: Transfers `energy_usage_predictor.py` and model files to the container.
- Entry Point: Uses `spark-submit` to run the script.

Listing 10: Dockerfile

```

1 FROM ubuntu:20.04
2 ENV DEBIAN_FRONTEND=noninteractive
3 WORKDIR /app
4 RUN apt-get update && \
5     apt-get install -y python3 python3-pip python3-numpy python3-pandas
6     openjdk-11-jdk wget nano && \
7     rm -rf /var/lib/apt/lists/*
8 ENV JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
9 ENV PATH=$JAVA_HOME/bin:$PATH
10 RUN wget https://archive.apache.org/dist/spark/spark-3.5.5/spark-3.5.5-bin-
11     hadoop3.tgz && \
12     tar xvf spark-3.5.5-bin-hadoop3.tgz -C /opt && \
13     rm spark-3.5.5-bin-hadoop3.tgz && \
14     ln -fs /opt/spark-3.5.5-bin-hadoop3 /opt/spark
15 ENV SPARK_HOME=/opt/spark
16 ENV PATH=$PATH:$SPARK_HOME/bin
17 RUN cp $SPARK_HOME/conf/log4j2.properties.template $SPARK_HOME/conf/log4j2.
18     properties && \
19     sed -i 's/rootLogger.level = info/rootLogger.level = ERROR/g'
20     $SPARK_HOME/conf/log4j2.properties
21 COPY energy_usage_predictor.py /app/
22 COPY EnergyPredictorGBT /app/EnergyPredictorGBT
23 ENTRYPOINT ["spark-submit", "energy_usage_predictor.py"]
24 CMD [""]

```

1.3.2 Container Build and Deployment

The Docker image is built, executed, and deployed as follows:

- Install Docker: Install Docker on the EC2 instance.

Listing 11: Install Docker

```
1 sudo apt-get install docker.io
```

- Build Image: Build the Docker image with the tag energy-predict.

Listing 12: Build Docker Image

```
1 sudo docker build -t energy-predict .
```

- Run Container: Run the container with the validation dataset mounted.

Listing 13: Run Docker Container

```
1 sudo docker run -v /home/ubuntu/ValidationDataset.csv:/app/
  ValidationDataset.csv energy-predict /app/ValidationDataset.csv
```

```
ubuntu@ip-172-31-26-90:~$ sudo docker run -v /home/ubuntu/ValidationDataset.csv:/app/ValidationDataset.csv energy-predict /app/ValidationDataset.csv
=====
ENERGY CONSUMPTION PREDICTOR
=====
A Spark-powered application for estimating energy usage on unseen datasets,
leveraging a pre-trained Gradient Boosted Trees regression model.

[STEP] Setting up Spark environment..... Completed
[STEP] Importing test data..... Completed
[STEP] Standardizing column names..... Completed
[STEP] Transforming categorical features..... Completed
[STEP] Removing original categorical fields..... Completed
[STEP] Casting columns to float..... Completed
[STEP] Preparing features and target variable..... Completed
[STEP] Converting DataFrame to RDD format..... Completed
[STEP] Retrieving pre-trained model..... Completed
[STEP] Generating predictions..... Completed
[STEP] Aligning predictions with true values..... Completed
[STEP] Evaluating prediction performance..... Completed

=====
Prediction Performance Report
=====
Root Mean Squared Error (RMSE)          73.8957
R2 (Coefficient of Determination)        0.9933
=====

=====
ENERGY CONSUMPTION PREDICTOR COMPLETED
=====
[STEP] Terminating Spark environment..... Completed
ubuntu@ip-172-31-26-90:~$
```

- Login, Tag, and Push: Login to Docker Hub, tag the image, and push it to Docker Hub.

Listing 14: Login, Tag, and Push Docker Image

```
1 sudo docker login
2 sudo docker tag energy-predict sruthivellore/energy-predict
3 sudo docker push sruthivellore/energy-predict
```

```

ubuntu@ip-172-31-26-90: ~$ sudo docker logout
sudo docker login
Removing login credentials for https://index.docker.io/v1/
Log in with your Docker ID or email address to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com/ to create one.
You can log in with your password or a Personal Access Token (PAT). Using a limited-scope PAT grants better security and is required for organizations using SSO. Learn more at https://docs.docker.com/go/access-tokens/

Username: sruthivellore
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu@ip-172-31-26-90:~$ sudo docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
energy-predict       latest      74c23eef6156  33 minutes ago 1.57GB
sruthivellore/energy-predict latest      74c23eef6156  33 minutes ago 1.57GB
ubuntu              20.04      b7bab04fd9aa  3 weeks ago   72.8MB
ubuntu@ip-172-31-26-90:~$ sudo docker push sruthivellore/energy-predict
Using default tag: latest
The push refers to repository [docker.io/sruthivellore/energy-predict]
9db460db4baa: Pushed
c4a56f47841c: Pushed
6447ec9391b9: Pushed
92e75ac5672e: Pushed
25370d984a94: Pushed
c068a4ea61c1: Pushed
470b66ea5123: Mounted from library/ubuntu
latest: digest: sha256:25b4c3faabdb782d9c03bf427ed5898f8e661ef36ed46b6a4c27f38707ade028 size: 1786
ubuntu@ip-172-31-26-90:~$

```

- Pull Image: Pull the image from Docker Hub for use elsewhere.

Listing 15: Pull Docker Image

```
1 sudo docker pull sruthivellore/energy-predict
```

```

ubuntu@ip-172-31-26-90: ~$ sudo docker pull sruthivellore/energy-predict
Using default tag: latest
latest: Pulling from sruthivellore/energy-predict
Digest: sha256:25b4c3faabdb782d9c03bf427ed5898f8e661ef36ed46b6a4c27f38707ade028
Status: Image is up to date for sruthivellore/energy-predict:latest
docker.io/sruthivellore/energy-predict:latest
ubuntu@ip-172-31-26-90:~$

```

2 Deployment Guide

This section provides a comprehensive guide to replicating the project, ensuring clarity and reproducibility.

2.1 Setting Up Distributed Training

1. Provision four EC2 instances on AWS using tools (EMR).
2. Apply the bootstrap script (`boot.sh`) during instance setup.
3. Upload `TrainingDataset.csv` and `energy_model_trainer.py` to HDFS.
4. Execute `spark-submit energy_model_trainer.py`.
5. Export the trained model using `hdfs dfs -copyToLocal` and `tar`.

2.2 Configuring the Prediction System

1. Launch a single EC2 instance running Ubuntu 20.04.
2. Install dependencies and Spark
3. Configure environment variables (`~/predictor_env`) and logging settings.
4. Transfer `model.tar.gz`, `energy_usage_predictor.py`, and `ValidationDataset.csv`.
5. Extract the model and run `spark-submit energy_usage_predictor.py ValidationDataset.csv`.

2.3 Deploying with Docker

1. Install Docker on an EC2 instance.
2. Build the Docker image using the provided `Dockerfile`.
3. Run the container, mapping the validation dataset.
4. Optionally, pull the image from Docker Hub: `sruthivellore/energy-predict`.

3 AI-Assisted Development

For this project, I used ChatGPT mainly to get some basic templates for scripts and setup steps. It helped with getting started faster, especially for things like the Dockerfile and a few bash commands. However, most of the actual code, including model training, prediction scripts, environment setup, and AWS configuration, was done manually based on what the project needed.

While ChatGPT saved time on some repetitive parts, a lot of debugging, fixing compatibility issues with Spark, and making everything work properly had to be done manually. Overall, AI tools were helpful for speeding up small parts.

4 Code and Resource Links

- GitHub Repository: <https://github.com/sruthivellore/EnergyPredictSparkAWS.git>
- Docker Hub: <https://hub.docker.com/r/sruthivellore/energy-predict>