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

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

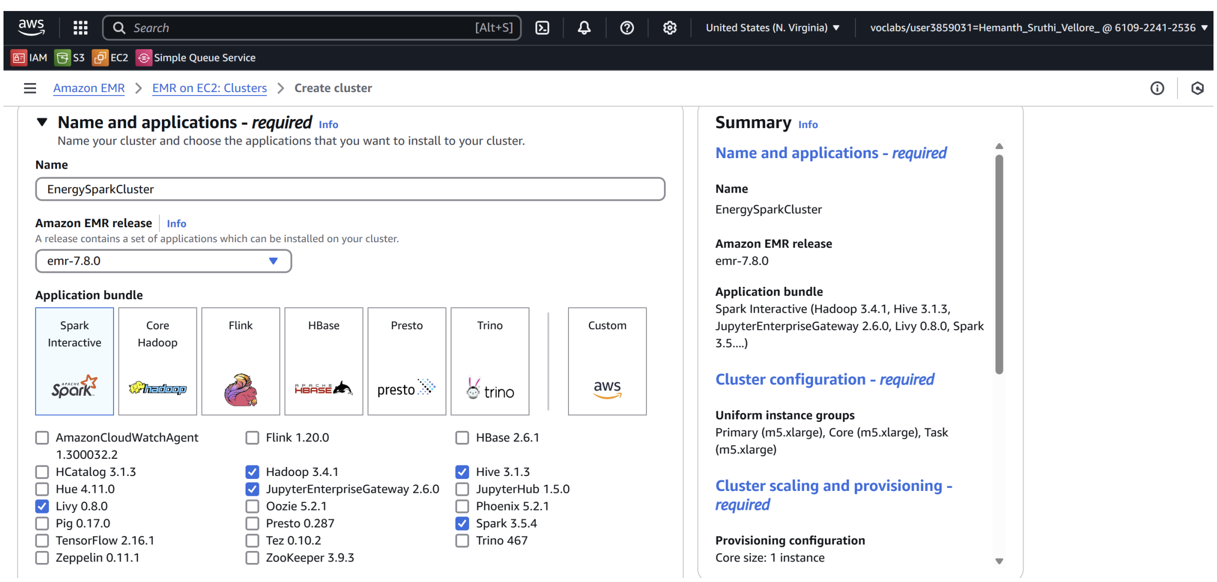
1.1 Distributed Model Training

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

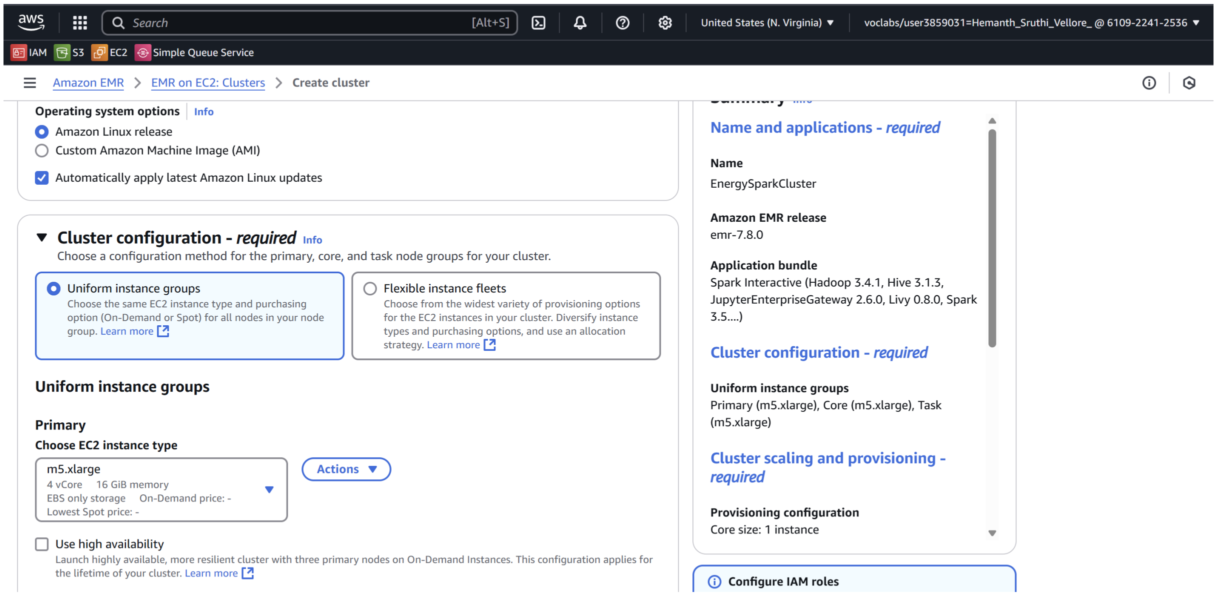


   (d) Configure Hardware Configurations:
      i. Select OS option (Amazon Linux release).
      ii. Select the instance type (m5.xlarge).

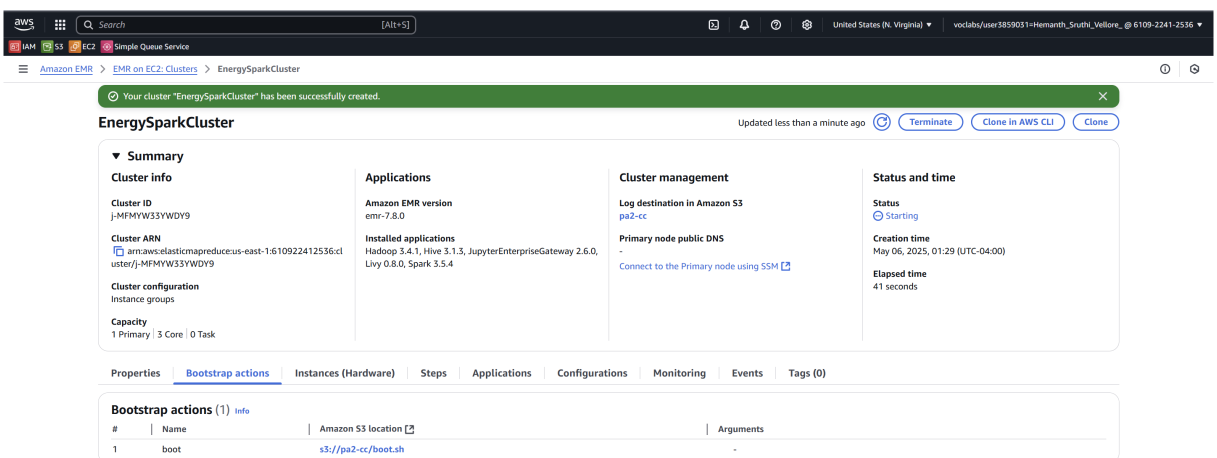
iii. Set number of instances to 4 (1 master, 3 slaves).

(e) Select the "vockey " key pair

(f) Select bootstrap script from the S3 bucket.



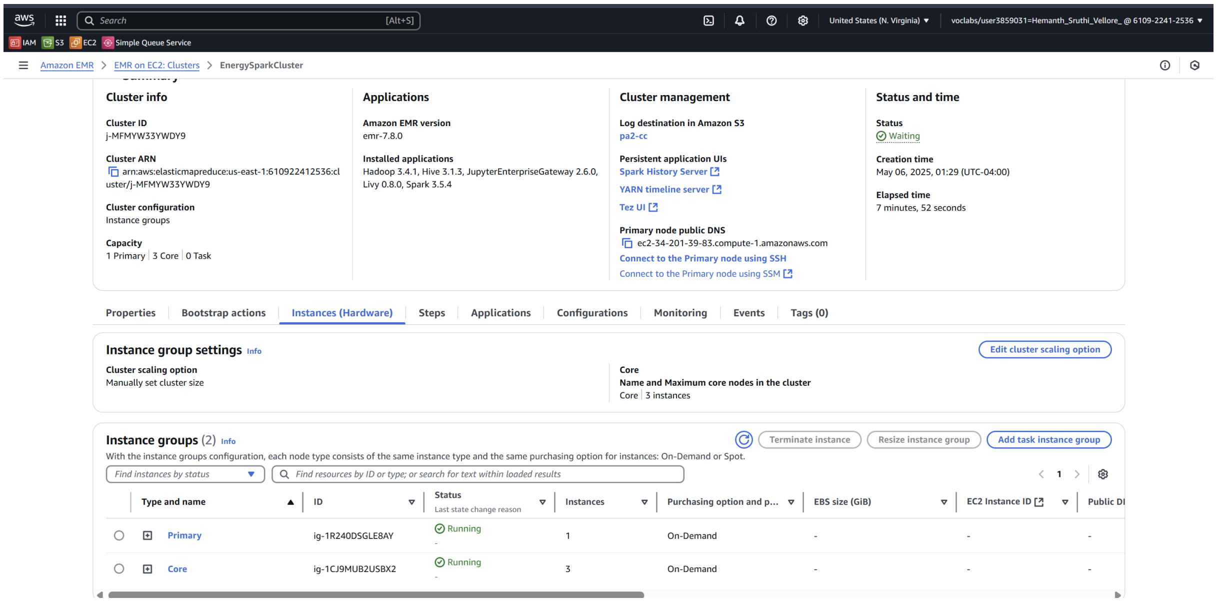
3. Click on Create Cluster.



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

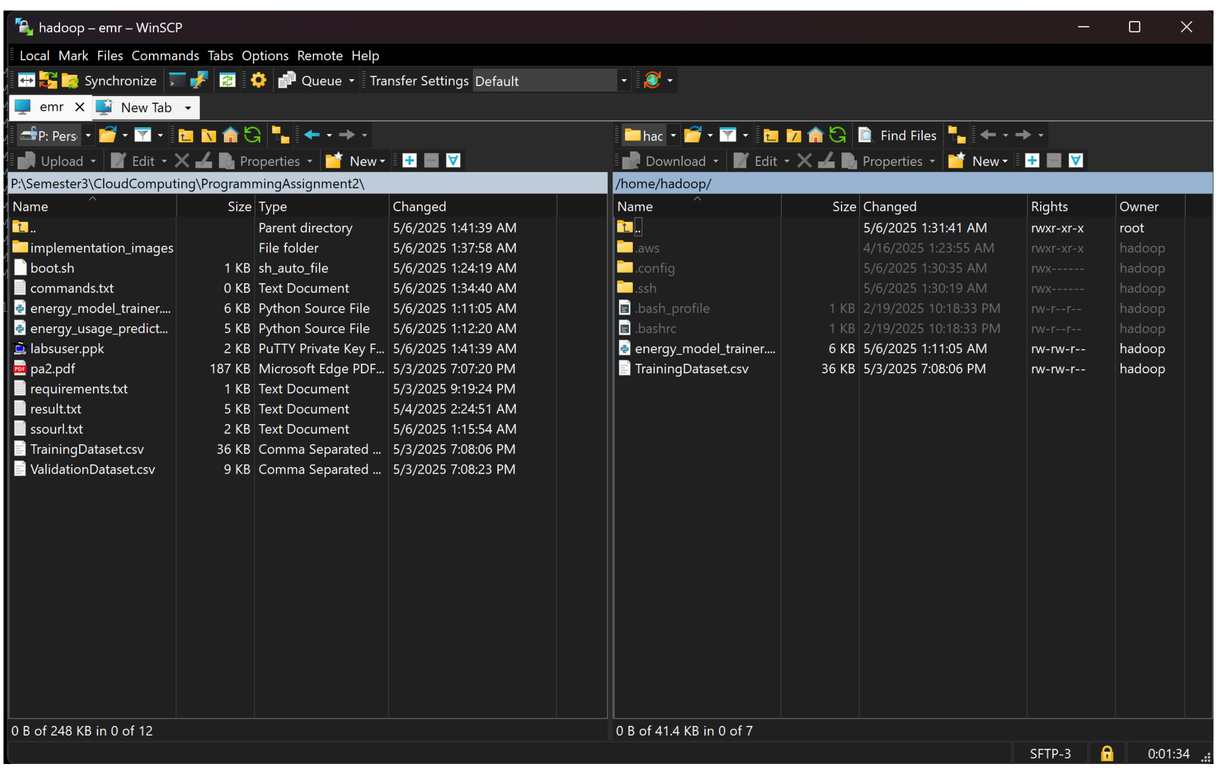
Listing 1: Bootstrap Script (`boot.sh`)

```
sudo pip3 install numpy pandas
```



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.



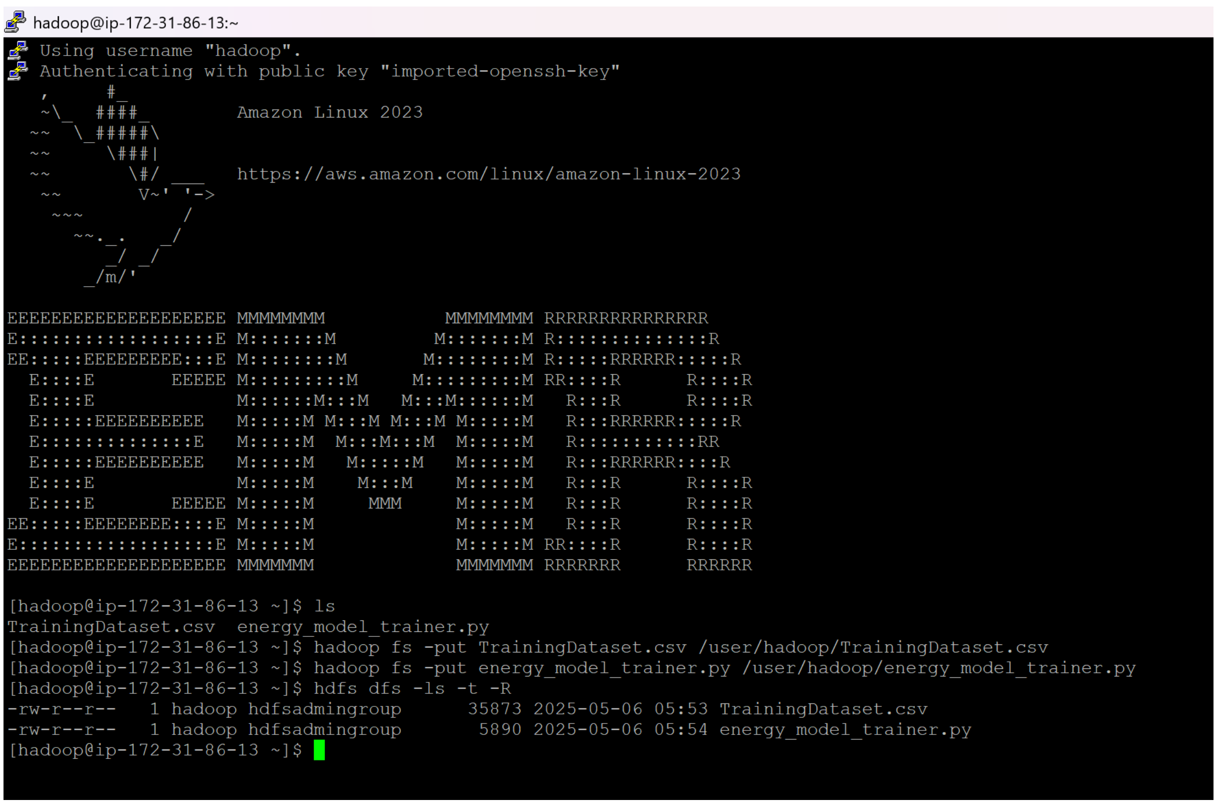
1.1.3 HDFS Data Integration

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

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

Figure 1: Data integration and verification.



1.1.4 Model Training Workflow

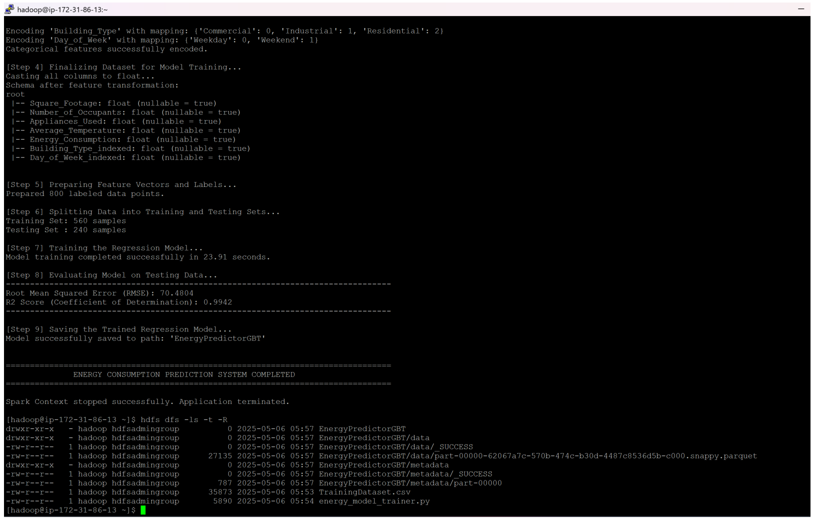
The `energy_model_trainer.py` script, executed via `spark-submit`, trains the GBT model by loading the dataset, encoding categorical features, assembling feature vectors, splitting data into training and testing sets, training the model, evaluating performance (RMSE and R^2), and saving the model to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

Listing 3: Training Workflow Command

```
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.

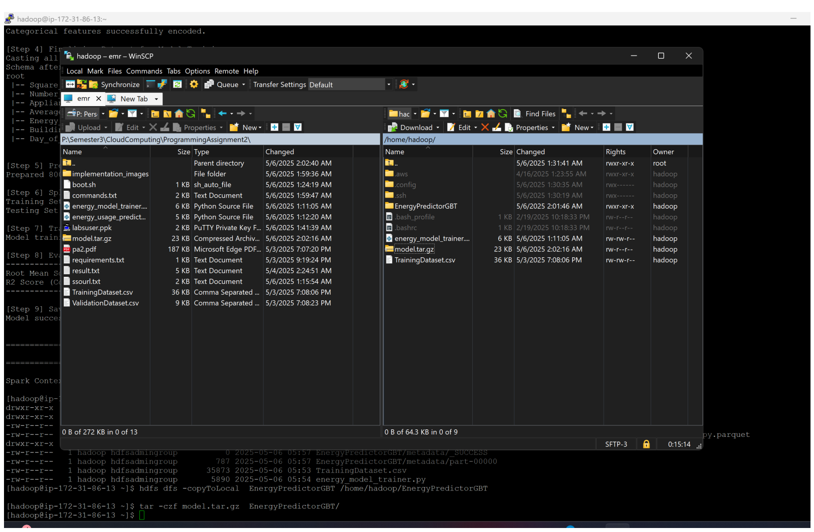


1.1.6 Model Export Process

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

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



1.2 Single-Instance Prediction System

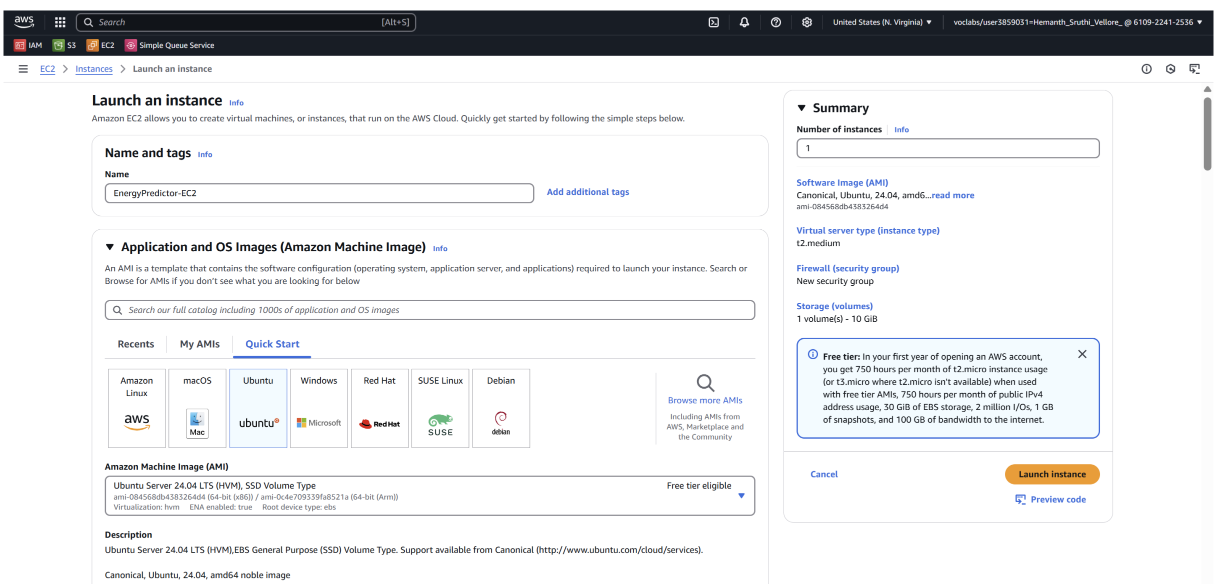
The prediction system runs on a single EC2 instance using `energy_usage_predictor.py` and the saved model.

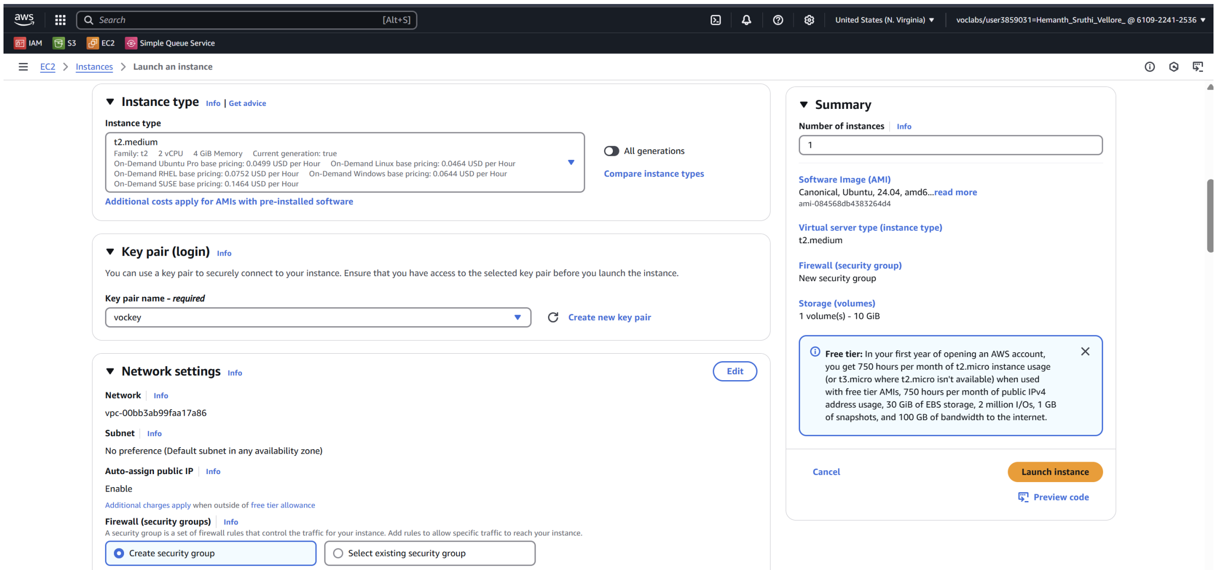
1.2.1 EC2 Instance Creation

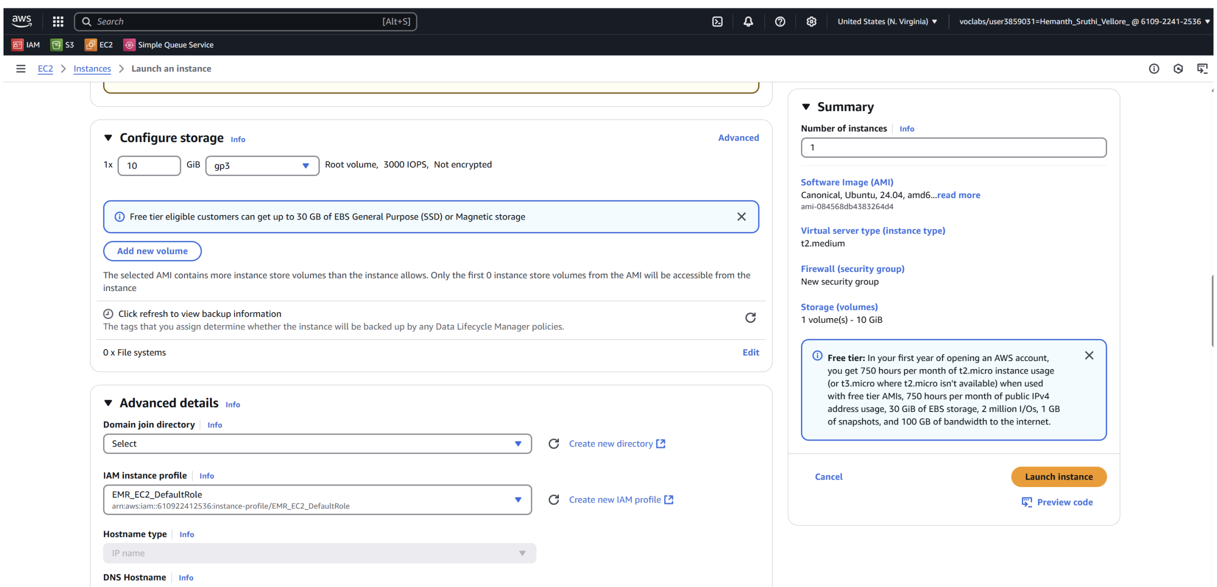
Provisioned an EC2 instance via the AWS Management Console by:

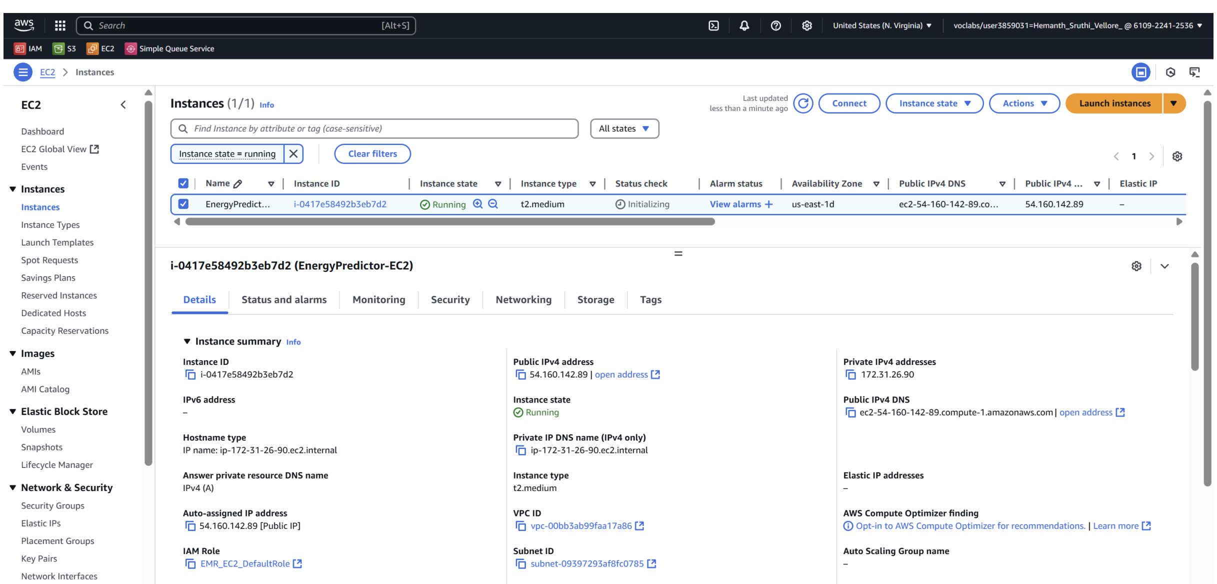
- Naming the instance `EnergyPredictor-EC2`
- Selecting `Ubuntu Server 24.04 LTS (HVM)` AMI
- Choosing `t2.medium` instance type
- Setting root volume to 10 GB (gp3)
- Assigning the `EMR_EC2_DefaultRole` IAM role
- Selecting the `vockey` key pair
- Launching the instance

Instance initialized and ready for setup.




EC2 > Instances > Launch an instance
Instance type
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 RHEL 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
All generations
Compare instance types
Additional costs apply for AMIs with pre-installed software
Key pair (login)
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
Edit
Network
vpc-00bb3ab99faa17a86
Subnet
No preference (Default subnet in any availability zone)
Auto-assign public IP
Enable
Firewall (security groups)
Create security group
Select existing security group
Summary
Number of instances
1
Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...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 GB of snapshots, and 100 GB of bandwidth to the internet.
Cancel
Launch instance
Preview code


EC2 > Instances > Launch an instance
Configure storage
Advanced
1x 10 GiB gp3 Root volume, 3000 IOPS, Not encrypted
Free tier eligible customers can get up to 30 GB 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
Edit
Advanced details
Domain join directory
Select
Create new directory
IAM instance profile
EMR_EC2_DefaultRole
Create new IAM profile
Hostname type
DNS Hostname
Summary
Number of instances
1
Software Image (AMI)
Canonical, Ubuntu, 24.04, amd6...read more
Virtual server type (instance type)
t2.medium
Firewall (security group)
New security group
Storage (volumes)
1 volume(s) - 10 GiB
Cancel
Launch instance
Preview code


EC2 > Instances
Instances (1/1)
Connect
Instance state
Actions
Launch instances
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
Instance ID
i-0417e58492b3eb7d2
Public IPv4 address
54.160.142.89 | open address
Private IPv4 addresses
172.31.26.90
Instance state
Running
Public IPv4 DNS
ec2-54-160-142-89.compute-1.amazonaws.com | open address
Hostname type
IP name: ip-172-31-26-90.ec2.internal
Private IP DNS name (IPv4 only)
ip-172-31-26-90.ec2.internal
Instance type
t2.medium
VPC ID
vpc-00bb3ab99faa17a86
Subnet ID
subnet-09397293af8fc0785
IAM Role
EMR_EC2_DefaultRole
Auto-assigned IP address
54.160.142.89 [Public IP]

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

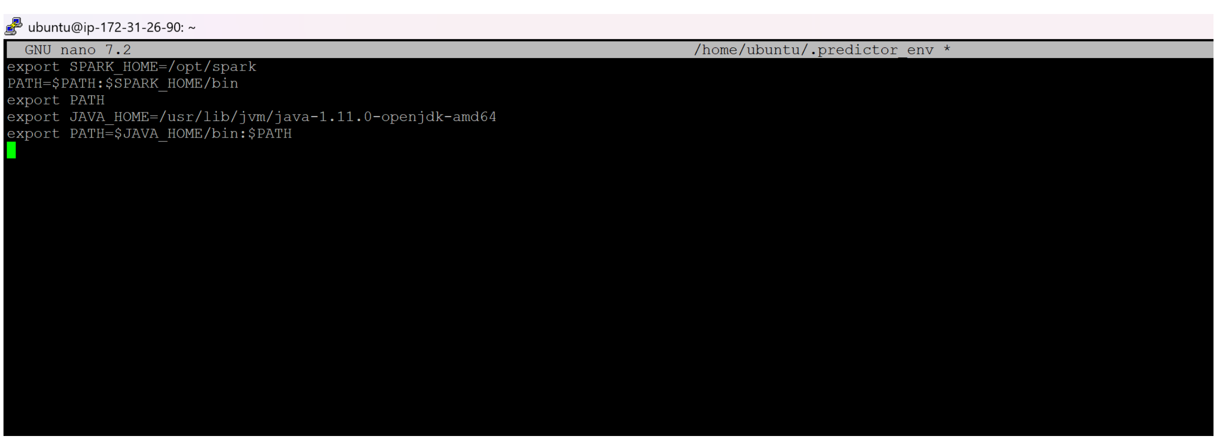
```
sudo apt-get update
sudo apt-get install -y python3-pip python3-numpy python3-pandas openjdk-11-
    jdk
wget https://archive.apache.org/dist/spark/spark-3.5.5/spark-3.5.5-bin-
    hadoop3.tgz
sudo tar xvf spark-3.5.5-bin-hadoop3.tgz -C /opt
sudo chown -R ubuntu:ubuntu /opt/spark-3.5.5-bin-hadoop3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

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



1.2.4 Spark Logging Configuration

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

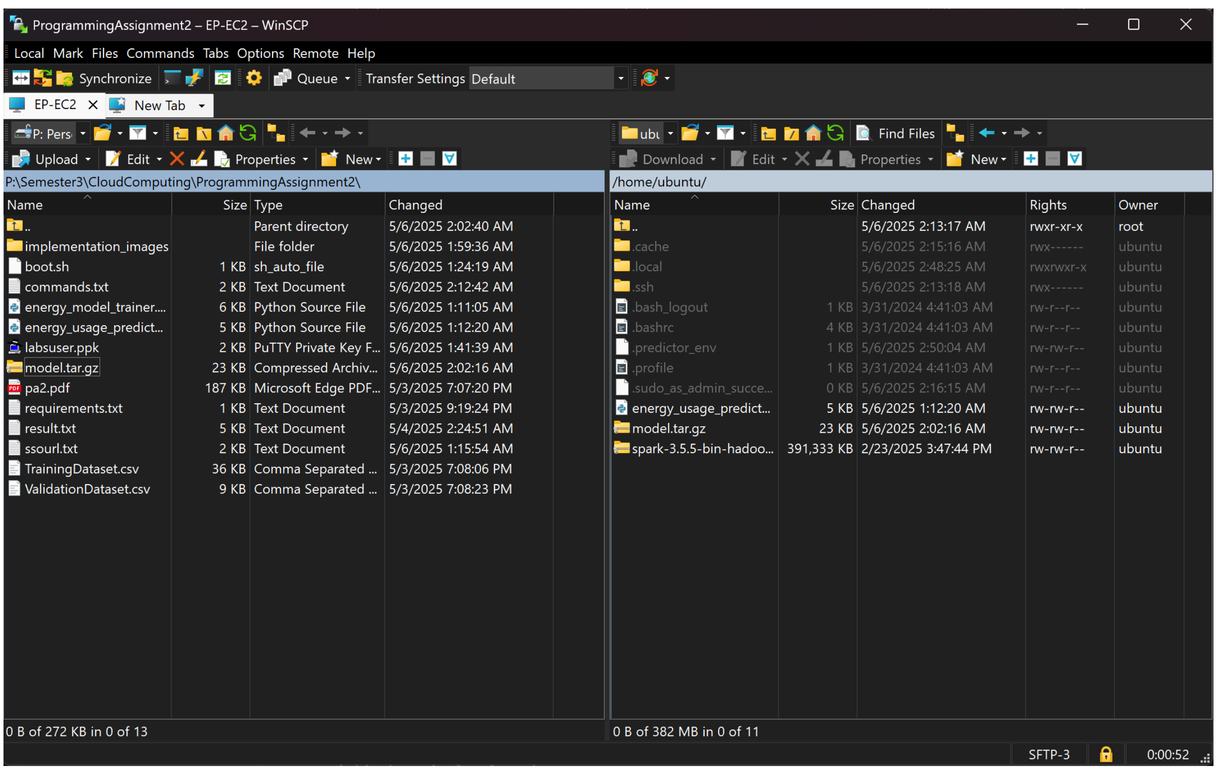
- Copy the template logging configuration file.
- Edit the `log4j2.properties` file, changing `rootLogger.level` from `INFO` to `ERROR`.

Listing 7: Logging Configuration

```
cp $SPARK_HOME/conf/log4j2.properties.template $SPARK_HOME/conf/log4j2.properties
nano $SPARK_HOME/conf/log4j2.properties
# 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.

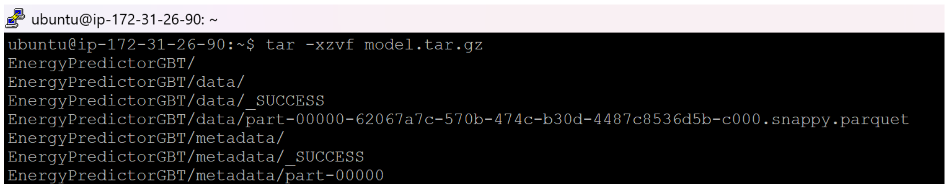


1.2.6 Unzip the Model

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

Listing 8: Unzipping the Model

```
tar -xzvf model.tar.gz
```



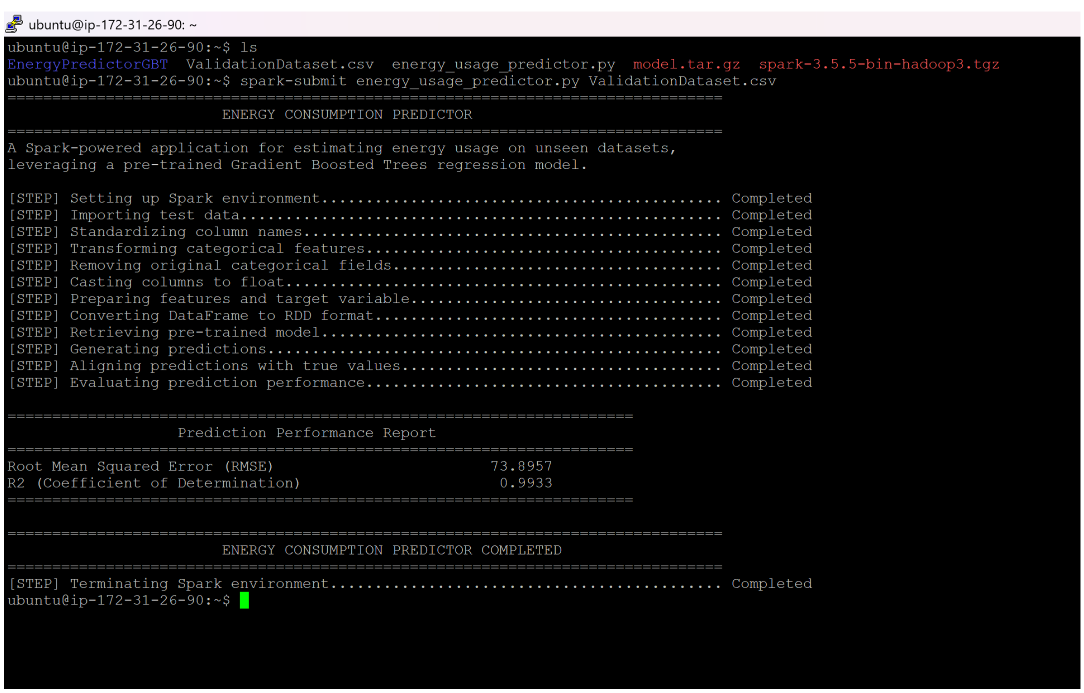
1.2.7 Run the Prediction Script

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

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



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

```
FROM ubuntu:20.04
ENV DEBIAN_FRONTEND=noninteractive
WORKDIR /app
RUN apt-get update && \
    apt-get install -y python3 python3-pip python3-numpy python3-pandas
       openjdk-11-jdk wget nano && \
    rm -rf /var/lib/apt/lists/*
ENV JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-amd64
ENV PATH=$JAVA_HOME/bin:$PATH
RUN wget https://archive.apache.org/dist/spark/spark-3.5.5/spark-3.5.5-bin-
   hadoop3.tgz && \
    tar xvf spark-3.5.5-bin-hadoop3.tgz -C /opt && \
    rm spark-3.5.5-bin-hadoop3.tgz && \
    ln -fs /opt/spark-3.5.5-bin-hadoop3 /opt/spark
ENV SPARK_HOME=/opt/spark
ENV PATH=$PATH:$SPARK_HOME/bin
RUN cp $SPARK_HOME/conf/log4j2.properties.template $SPARK_HOME/conf/log4j2.
   properties && \
    sed -i 's/rootLogger.level = info/rootLogger.level = ERROR/g'
       $SPARK_HOME/conf/log4j2.properties
COPY energy_usage_predictor.py /app/
COPY EnergyPredictorGBT /app/EnergyPredictorGBT
ENTRYPOINT ["spark-submit", "energy_usage_predictor.py"]
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

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

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

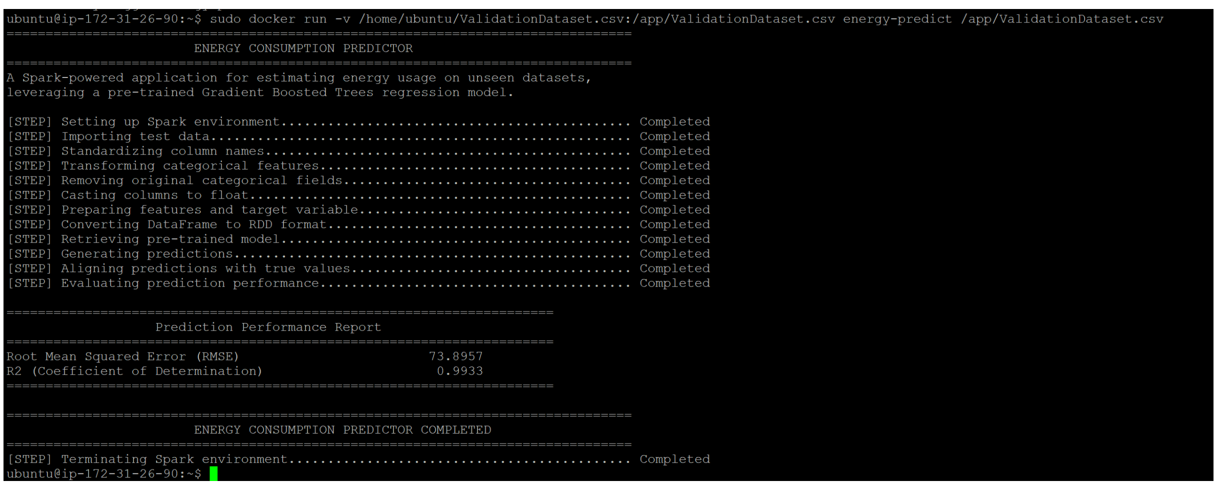
Listing 12: Build Docker Image

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

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

Listing 13: Run Docker Container

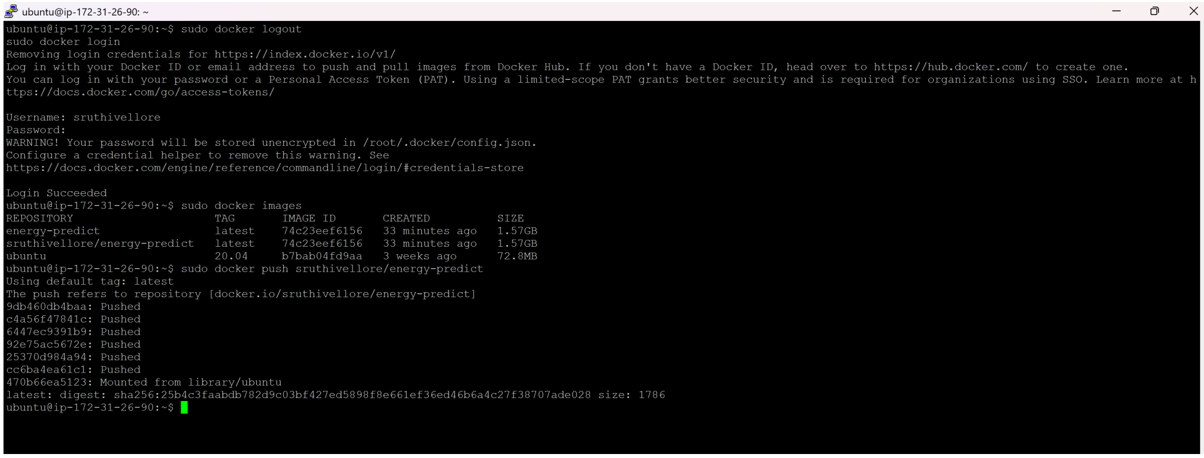
```
sudo docker run -v /home/ubuntu/ValidationDataset.csv:/app/
    ValidationDataset.csv energy-predict /app/ValidationDataset.csv
```



- 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

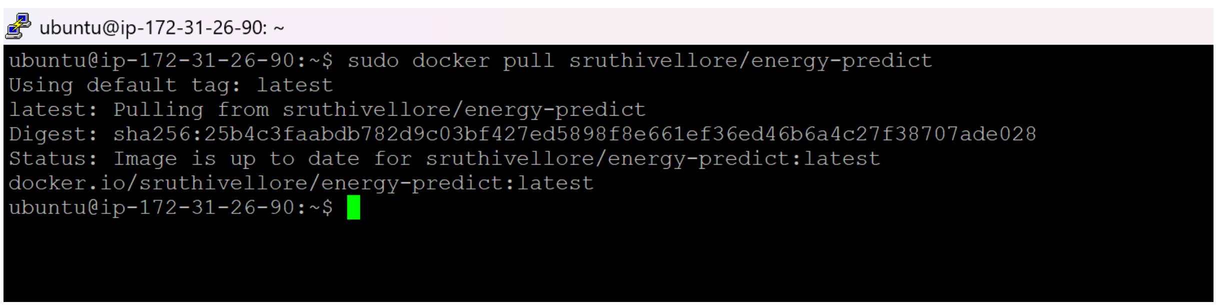
```
sudo docker login
sudo docker tag energy-predict sruthivellore/energy-predict
sudo docker push sruthivellore/energy-predict
```



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

Listing 15: Pull Docker Image

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



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`