

# Project 2-Hadoop/EMR (PySpark, Jupyter Notebook)

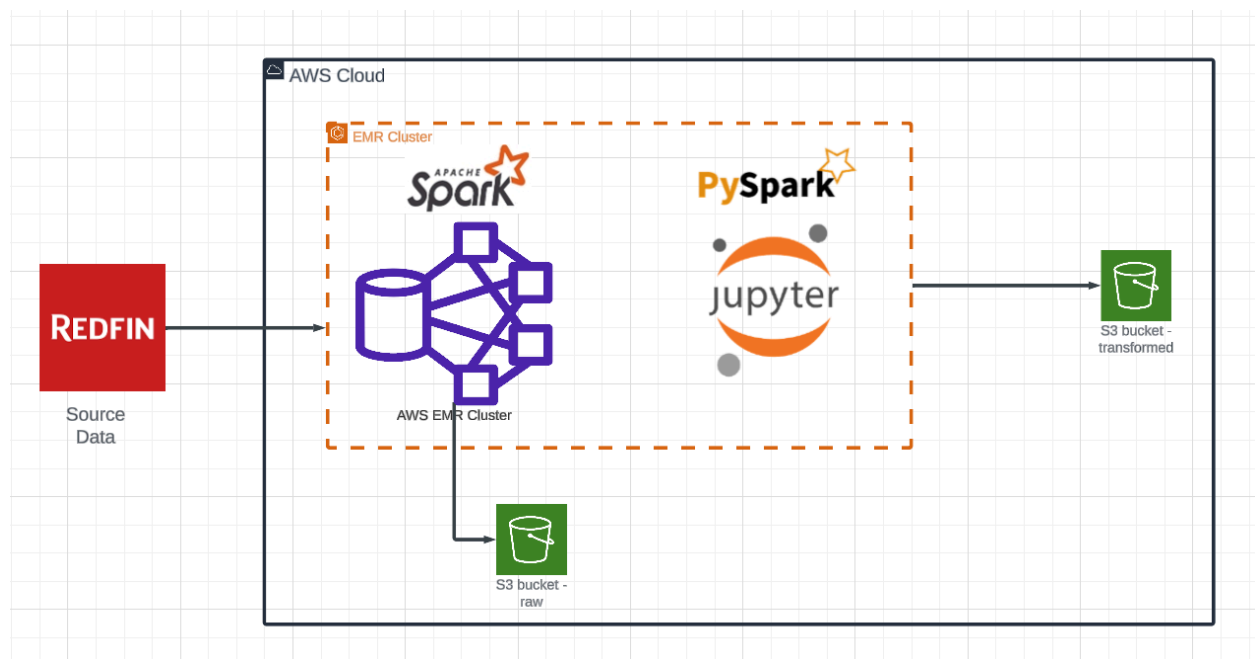
**Note:** unless specified explicitly, keep all other options default while creating any of the services

## Prerequisites:

- Create an IAM user with admin privileges and login to AWS console using that IAM user
- DO NOT DO THIS PROJECT WITH ROOT USER

## Architecture

- Below is a high level architecture we are going to build.



## Step1 - Create VPC

- Select “VPC and more”
- Leave the Auto-generate check mark “on”, and enter “project-hadoop” as the VPC name
- Number of private subnets: Select **Zero**

## Step2 - Create EMR Cluster

- Name: emr-cluster
- Application Bundle, make sure the below options are selected

- Livy
- Spark
- Hadoop
- JupyterEnterpriseGateway
- Hive
- Zeppelin
- Networking
  - VPC - browse and select the VPC you created in Step1 above
- Cluster termination and node replacement
  - Choose “Automatically terminate cluster after idle time (Recommended)”
  - Set Idle time as 1 day
- Amazon EMR Service Role
  - Choose “Create a service role”
  - Security Group -> Choose default security group of the project-hadoop-vpc
- EC2 instance profile for Amazon EMR
  - Choose “Create an instance profile”
  - S3 bucket access -> All S3 buckets in this account with read and write access

*Note: Cluster creation may take up to 15 minutes (you can proceed with next step while waiting)*


## Step3 - Create two S3 bucket

- Bucket names: [initials-mmdd-raw] and [initials-mmdd-transformed]
  - a. Example, my initials SR, so my bucket names will be sr-1224-raw; sr-1224-transformed

## Step4 - Create EMR Studio

- Setup options choose “Custom”
- Service role to let Studio access your AWS resources
  - Choose AmazonEMR-ServiceRole
    - Add admin policy to the above role
- Networking and security
  - VPC: Select the project-hadoop-vpc
  - Subnets -> Select both the subnets
- Create Studio
  - If you get any errors related to ServiceRole lacking permissions attach them
    - S3 full access
    - KMS full access
    - You can also give admin access as this is only a lab
  - If you get any errors saying S3 bucket already exists - go delete the bucket
- Create studio, and wait until creation is completed

## Step5 - Launch Workspace (Jupyter Notebook)

- Navigate to Workspaces, and select Create Workspace and Quick launch
- On the left navigation menu, select cluster 
  - Compute Type: EMR on EC2 cluster
  - Select the cluster name from the dropdown
  - Click Attach
- Under the Notebook options select "PySpark"
  - If you hit any error - you cannot launch workspaces using root account
  - Sign in as IAM user (with admin permissions) and try to access workspace again
  - If you are trying to access Workspace as IAM user (with admin privileges) and hit an error saying the workspace is already in use, try to stop and start the workspace

## Step6- Working with Redfin data, and applying transformations

Open the Jupyter notebook, and try the below PySpark commands to download data from Redfin, apply some transformations, and save the files into sr-12-24-transformed bucket.

**Note:** make sure to replace bucket names with your bucket names

```
# Import necessary PySpark modules
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
```

```
# Download the Redfin market tracker dataset and upload it to an S3 bucket
%%bash
wget -O -
https://redfin-public-data.s3.us-west-2.amazonaws.com/redfin_market_tracker/city_market_tracker.tsv000.gz | aws s3 cp - s3://sr-1224-raw/city_market_tracker.tsv000.gz
```

```
# Create a Spark session
spark = SparkSession.builder.appName("RedfinDataAnalysis").getOrCreate()
```

```
# Read the Redfin dataset from the S3 bucket in CSV format with specified options
redfin_data = spark.read.csv("s3://sr-1224-raw/city_market_tracker.tsv000.gz",
header=True, inferSchema=True, sep="\t")
```

```
# Display the first 3 rows of the dataset
redfin_data.show(3)
```

```
# Print the schema of the dataset
redfin_data.printSchema()
```

```
# List the columns in the dataset
redfin_data.columns
```

```
# Select specific columns from the dataset
df_redfin = redfin_data.select(['period_end', 'period_duration', 'city', 'state',
    'property_type',
    'median_sale_price', 'median_ppsf', 'homes_sold', 'inventory', 'months_of_supply',
    'median_dom', 'sold_above_list', 'last_updated'])
# Display the first 3 rows of the selected columns
df_redfin.show(3)
```

```
# Print the total number of rows in the selected dataset
print(f"Total number of rows: {df_redfin.count()}")
```

```
from pyspark.sql.functions import isnull
# Count null values in each column of the selected dataset
null_counts = [df_redfin.where(isnull(col_name)).count() for col_name in
df_redfin.columns]
null_counts
```

```
# Display the count of null values for each column
for i, col_name in enumerate(df_redfin.columns):
    print(f"{col_name}: {null_counts[i]} null values")
```

```
# Count the number of rows without any missing values
remaining_count = df_redfin.na.drop().count()
# Print the number of rows with missing values
print(f"Number of missing rows: {df_redfin.count() - remaining_count}")
# Print the total number of remaining rows after dropping rows with missing values
print(f"Total number of remaining rows: {remaining_count}")
```

```
# Remove rows with any missing values and print the total number of remaining rows
df_redfin = df_redfin.na.drop()
print(f"Total number of rows: {df_redfin.count()}")
```

```
# Re-count null values in each column to confirm removal of rows with missing values
null_counts = [df_redfin.where(isnull(col_name)).count() for col_name in
df_redfin.columns]
null_counts
```

```
from pyspark.sql.functions import year, month
# Extract year from 'period_end' and save in a new column "period_end_yr"
df_redfin = df_redfin.withColumn("period_end_yr", year(col("period_end")))
# Extract month from 'period_end' and save in a new column "period_end_month"
df_redfin = df_redfin.withColumn("period_end_month", month(col("period_end")))
```

```
# Drop 'period_end' and 'last_updated' columns from the dataset
df_redfin = df_redfin.drop("period_end", "last_updated")
# Display the first 3 rows of the modified dataset
df_redfin.show(3)
```

```
from pyspark.sql.functions import when
# Map the month number to their respective month name
df_redfin = df_redfin.withColumn("period_end_month",
    when(col("period_end_month") == 1, "January")
    .when(col("period_end_month") == 2, "February")
    .when(col("period_end_month") == 3, "March")
    .when(col("period_end_month") == 4, "April")
    .when(col("period_end_month") == 5, "May")
    .when(col("period_end_month") == 6, "June")
    .when(col("period_end_month") == 7, "July")
    .when(col("period_end_month") == 8, "August")
    .when(col("period_end_month") == 9, "September")
    .when(col("period_end_month") == 10, "October")
    .when(col("period_end_month") == 11, "November")
    .when(col("period_end_month") == 12, "December")
    .otherwise("Unknown"))
# Display the first 3 rows with the mapped month names
df_redfin.show(3)
```

```
# Write the final DataFrame to an S3 bucket as a Parquet file
s3_bucket = "s3://sr-1224-transformed/redfin_data.parquet"
df_redfin.write.mode("overwrite").parquet(s3_bucket)
```

Once you're done, make sure to delete all the AWS resources you created for this lab.

-----END OF LAB-----

#####