**Job-Ready Projects: Big Data Platform**

**Amazon EMR (Elastic MapReduce)**

***with***

**Total *4* Important Projects**

**1. Project Name: *Data Transformation Pipeline for Retail Insights***

**2. Project Name: *Machine Learning Model Training on EMR for Customer Churn Prediction***

**3. Project Name: *Real-Time Data Lake Processing for Social Media Analytics***

**4. Project Name: *Scalable Log Analysis System for System Monitoring***

Amazon EMR (Elastic MapReduce) is a cloud-based big data platform provided by AWS. It simplifies processing large amounts of data quickly and cost-effectively by using popular open-source tools like Apache Hadoop, Spark, HBase, and Hive. Here are some key aspects and use cases:

**Key Features**

1. **Scalable Data Processing**: EMR allows the creation of clusters with auto-scaling, automatically adding or removing nodes based on the workload, which optimizes costs.
2. **Managed Service**: AWS manages the infrastructure, so users can focus on data processing without needing to worry about provisioning or maintaining hardware.
3. **Multiple Tools and Frameworks**: EMR supports various big data tools like Hadoop, Spark, HBase, and Presto for **analytics** and **machine** learning tasks.
4. **Customizable**: EMR clusters can be customized with bootstrap actions and can integrate with **S3**, making it flexible for both **batch** and **streaming** data processing.

**Common Use Cases**

* **Data Transformation and ETL**: EMR is often used for transforming raw data (e.g., cleaning, enriching) **before** loading it into a data warehouse.
* **Machine Learning**: Large datasets can be processed in Spark or other machine-learning frameworks for model training.
* **Data Lake Processing**: EMR can analyze **unstructured** and **structured** data stored in S3, making it useful for large-scale data lake architectures.
* **Log Analysis and Processing**: EMR is suitable for processing **logs** and generating reports from logs collected from distributed systems.

**How EMR Works**

1. **Cluster Creation**: Users can define the type and number of nodes and specify the applications to be installed (e.g., Hadoop, Spark).
2. **Data Storage**: EMR integrates directly with **S3**, and data can be stored in other AWS services, like HDFS or DynamoDB.
3. **Data Processing**: Users submit jobs, and the cluster processes the data, leveraging the distributed nature of Hadoop or Spark.
4. **Results**: Processed data can be stored back in S3 or **transferred** to other AWS services like **Redshift** for further analysis.

**Important 4 Projects**

**1. Project Name: *Data Transformation Pipeline for Retail Insights***

* **Use Case**: Data Transformation and ETL
* **Description**: This project focuses on transforming **raw** sales data from various sources (e.g., CSVs in S3) into a clean dataset for business analysis. The pipeline will use EMR to aggregate, clean, and enrich sales and customer data, preparing it for visualization or loading into a data warehouse.
* **Steps**:
  1. **Data Collection**: Collect raw data files (e.g., daily sales, customer details) and store them in S3.
  2. **Cluster Setup**: Configure an EMR cluster with **Spark**, **Hive**, and HDFS.
  3. **Data Transformation**: Write Spark jobs to clean, deduplicate, and transform data into a usable format (e.g., calculate daily sales summaries).
  4. **Data Enrichment**: Enrich sales data with customer segmentation details.
  5. **Save Processed Data**: Store the cleaned and transformed data back in S3 or transfer it to Redshift for further analysis.
  6. **Automation**: Use **Lambda** functions to **trigger** EMR jobs automatically when **new** data is uploaded to **S3**.

**Details with Commands:**

1. **Upload Data to S3**:

aws s3 **cp** /path/to/your/data s3://your-bucket/raw-data/ --recursive

1. **Create an EMR Cluster with Spark**:

aws **emr** create-cluster --name "Data Transformation Cluster" \

--use-default-roles --release-label emr-6.5.0 \

--applications Name=Spark \

--instance-type m5.xlarge --instance-count 3 \

--log-uri s3://your-bucket/emr-logs/

**3. Submit a Spark Job for Data Transformation**:

**In Console:** Go to **Amazon EMR > Jobs > Steps** to submit a step with your Spark script (e.g., s3://your-bucket/scripts/data-transformation.py).

**Cli:**

aws emr add-steps --cluster-id <ClusterId> \

--steps Type=Spark,Name="Data Transformation",ActionOnFailure=CONTINUE,Args=[s3://your-bucket/scripts/data-transformation.py]

**4**. **Output Transformed Data to S3**:

Your Spark script should save transformed data back to S3:

**output\_path = "s3://your-bucket/processed-data/"**

**df.write.format("parquet").save(output\_path)**

**5**. **Automate with Lambda (Optional)**:

**Create Lambda function** to trigger EMR jobs upon data upload:

***import boto3***

***def lambda\_handler(event, context):***

# Trigger EMR job here

**2. Project Name: *Machine Learning Model Training on EMR for Customer Churn Prediction***

* **Use Case**: Machine Learning
* **Description**: This project will train a machine learning model to predict customer churn using Spark MLlib on Amazon EMR. The dataset includes customer behavioral data, and the model will help identify customers who may churn based on usage patterns.
* **Steps**:
  1. **Data Collection**: Load customer behavioral data (e.g., usage metrics, purchase history) into S3.
  2. **Cluster Setup**: Configure an EMR cluster with Spark and Jupyter for interactive model training.
  3. **Data Preprocessing**: Clean and preprocess the data using Spark jobs (e.g., handle missing values, normalize features).
  4. **Model Training**: Use Spark MLlib to train a **classification** model (e.g., logistic regression) to predict churn likelihood.
  5. **Model Evaluation**: Split the data, evaluate the model with metrics like accuracy or AUC, and tune hyperparameters as needed.
  6. **Deployment**: **Store** the model in S3 and use a Lambda function or **SageMaker** to deploy it as a service for predictions.

**Details with Commands:**

1. **Upload Data to S3**:

aws s3 **cp** /path/to/your/customer-data.csv s3://your-bucket/data/

1. **Launch an EMR Cluster with Spark and Jupyter Notebook**:

aws emr create-cluster --name "ML Training Cluster" \

--use-default-roles --release-label emr-6.5.0 \

--applications Name=Spark Name=JupyterHub \

--instance-type m5.xlarge --instance-count 4 \

--log-uri s3://your-bucket/emr-logs/

**3. Train Model using Jupyter Notebook (Console)**:

Access Jupyter on EMR via the Console and use Spark MLlib in Python to train your model.

**Example code** for Jupyter:

**from pyspark.ml.classification import LogisticRegression**

**from pyspark.ml.feature import VectorAssembler**

# Load and preprocess data, train model

**4. Save the Trained Model to S3**:

model.save("s3://your-bucket/models/customer-churn-model")

**5. Deploy and Automate**:

Optionally, save the model to be used with SageMaker or a Lambda function.

**3. Project Name: *Real-Time Data Lake Processing for Social Media Analytics***

* **Use Case**: Data Lake Processing
* **Description**: This project involves setting up a data lake with raw social media data stored in S3 and using EMR to run analytics and sentiment analysis on that data. This setup is ideal for understanding user sentiment over time.
* **Steps**:
  1. **Data Ingestion**: Stream social media data (e.g., tweets) into S3 buckets, organized by date.
  2. **Cluster Setup**: Launch an EMR cluster with tools like Spark and Presto for querying.
  3. **Data Processing**: Use Spark jobs to process the data (e.g., filter by hashtags, extract sentiment).
  4. **Analysis**: Run queries with Presto on EMR to generate insights, such as popular hashtags and general sentiment analysis over time.
  5. **Result Storage**: Store aggregated data and insights back in S3 for visualization.
  6. **Reporting**: Use Athena or QuickSight to create dashboards for visualizing sentiment trends, engagement metrics, and top keywords.

**Details with Commands:**

1. **Ingest Data to S3 (Simulated)**:

aws s3 **cp** /path/to/social-media-data s3://your-bucket/social-data/ --recursive

**2. Create EMR Cluster with Spark and Presto**:

aws emr create-cluster --name "Data Lake Processing Cluster" \

--use-default-roles --release-label emr-6.5.0 \

--applications Name=Spark Name=Presto \

--instance-type m5.xlarge --instance-count 4 \

--log-uri s3://your-bucket/emr-logs/

**3**. **Submit Sentiment Analysis Job with Spark**:

Console: Go to **Amazon EMR > Steps** and add your sentiment analysis Spark script as a step.

aws emr add-steps --cluster-id <ClusterId> \

--steps Type=Spark,Name="Sentiment Analysis",Args=[s3://your-bucket/scripts/sentiment-analysis.py]

aws emr add-steps --cluster-id <ClusterId> \

--steps Type=Spark,Name="Sentiment Analysis",Args=[s3://your-bucket/scripts/sentiment-analysis.py]

**Cli:**

aws emr add-steps --cluster-id <ClusterId> \

--steps Type=Spark,Name="Sentiment Analysis",Args=[s3://your-bucket/scripts/sentiment-analysis.py]

**4. Run Aggregations with Presto:**

Console: Use EMR Notebooks to run SQL queries with Presto.

CLI example for interactive queries:

**aws emr add-steps --cluster-id <ClusterId> \**

**--steps Type=Presto,Name="Social Data Aggregation",Args=["SELECT hashtag, COUNT(\*) FROM social\_data GROUP BY hashtag"]**

**5. Store Aggregated Results in S3:**

# Inside your Presto or Spark job

output.write.format("parquet").save("s3://your-bucket/aggregated-results/")

**4. Project Name: *Scalable Log Analysis System for System Monitoring***

* **Use Case**: Log Analysis and Processing
* **Description**: This project sets up a scalable solution for processing and analyzing system logs, such as application or web server logs. It allows teams to monitor system health and generate alerts on specific events.
* **Steps**:
  1. **Data Collection**: Collect application logs from multiple sources and store them in S3.
  2. **Cluster Setup**: Configure an EMR cluster with Hadoop, Spark, and HDFS.
  3. **Log Parsing**: Use **Spark** jobs to **parse** **logs**, extract fields of interest (e.g., IP address, error codes), and filter based on criteria like error severity.
  4. **Data Aggregation**: Aggregate log data to find patterns (e.g., frequency of errors, peak usage times).
  5. **Alert Generation**: Use Lambda to trigger alerts based on specific patterns detected in logs (e.g., high frequency of 500 errors).
  6. **Storage and Visualization**: Store processed log data in S3 and use QuickSight or Elasticsearch for visualizing patterns and trends.

**Details with Commands:**

1. **Upload Logs to S3**:

aws s3 **cp** /path/to/application-logs s3://your-bucket/logs/ --recursive

**2. Launch EMR Cluster with Hadoop and Spark:**

aws emr create-cluster --name "Log Analysis Cluster" \

--use-default-roles --release-label emr-6.5.0 \

--applications Name=Hadoop Name=Spark \

--instance-type m5.xlarge --instance-count 3 \

--log-uri s3://your-bucket/emr-logs/

**3. Run Log Parsing and Error Filtering with Spark:**

**Console**: Go to **Amazon EMR > Steps** to add a step with your log-parsing script (e.g., s3://your-bucket/scripts/log-parser.py).

**CLI:**

aws emr add-steps --cluster-id <ClusterId> \

--steps Type=Spark,Name="Log Parser",Args=[s3://your-bucket/scripts/log-parser.py]

**4. Aggregate and Store Results**:

After parsing, store the filtered and aggregated data in S3:

**output\_path = "s3://your-bucket/processed-logs/"**

**log\_data.write.format("parquet").save(output\_path)**

**5. Set Up Alerts with CloudWatch Logs Insights (Optional):**

Create an alert to monitor log patterns with CloudWatch Logs Insights for real-time monitoring and alerts.