Deploying and Running PySpark ETL on Amazon EMR

For Scalable Data Processing

Brent Hyman
Data and Automation Developer
MLOps Meetup • September 2025

Why Apache Spark for Large-Scale Data Processing?

Distributed Computing

Breaks large datasets into chunks
Processes in parallel across clusters

In-Memory Processing

Remarkably fast compared to
traditional disk-based systems

Unified Analytics

Batch jobs, streaming, SQL queries,
and machine learning in one platform

Scalability

Handles petabytes of data
across thousands of nodes

What is Amazon Elastic MapReduce aka EMR?

AWS's Fully Managed Big Data Platform

Amazon EMR supports a wide range of open-source big data frameworks such as Spark, Hadoop, Presto, and more.

Used for Cluster Management

EMR provisions and manages a cluster of Amazon EC2 instances (or serverless options). AWS handles setup, configuration, and scaling. You don't need to manually install Hadoop or Spark on servers.

Cost Efficient

Only pay for the compute and storage you use. You can run EMR on cheaper spot instances, on-demand, or mix them. EMR also integrates tightly with S3 (storage) so you don't need costly HDFS clusters.

Core Components

- **Compute layer**: Runs on Amazon EC2, EKS, or a serverless option (EMR Serverless).
- **Storage Layer**: Usually S3 for storage, but can also use HDFS, DynamoDB, or Glue Data Catalog for metadata.
- **Frameworks**: Spark, Hadoop MapReduce, etc.
- **Management**: Built-in security and monitoring

Spark/EMR vs Traditional Methods: Performance Comparison

Aspect	Traditional Processing	Spark/EMR
Data Size	Limited by single machine	Scales to petabytes
Processing Speed	Disk I/O bottlenecks	10-100x faster (in-memory)
Fault Tolerance	Manual recovery needed	Automatic failure recovery
Resource Scaling	Manual scaling	Auto-scaling clusters
Cost	Always-on infrastructure	Pay-per-use, on-demand
Development	Complex distributed code	Simplified APIs (Python/SQL)
Time to Insight	Hours to days	Minutes to hours

Proof of Concept

The idea is to run just a REALLY SIMPLE Spark SQL statement to fill a new table with transformed data:

```
-- Simple PySpark SQL ETL Example

 INSERT OVERWRITE TABLE high_volume_stocks
    SELECT ticker, the_date, open, high, low, close, vol
    FROM 2016_stock_data
    WHERE vol > 250000
```

Now… How does Spark/EMR come into play?

- Need to set up a batch processing architecture for just this single purpose transformation.
- Will set up an EC2-backed transient (short lived) EMR cluster
  - Just one task node which means 3 nodes total
    - Master node
    - Core node
    - Task node

All the infrastructure code for the job, and the ETL logic, is stored in a single repo.

High-Level Architecture Flow of the POC

1. GitHub Repository	Version control for PySpark ETL job CI/CD pipelines for the deployment and running of the ETL job.
2. Amazon S3 Storage	Upload scripts, JARs, and dependencies Intermediate and final data outputs
3. Amazon EMR Cluster	Launch EC2-backed cluster Submit and execute Spark jobs
4. Data Outputs	Results saved back to S3 Queryable via Amazon Athena

AWS Environment Setup

S3 Bucket Configuration

- Create a bucket for storing scripts and dependencies
- Have either separate folders for the input and output data. Or perhaps even a separate bucket for that.
- Option for lifecycle policies for cost optimization

IAM Roles & Permissions

- EMR_DefaultRole: Grants EMR service access
- EMR_EC2_DefaultRole: EC2 instance permissions
- Required policies: AmazonS3FullAccess
- Additional: CloudWatchLogsFullAccess

Networking Setup

- Configure VPC, subnets, security groups, and gateways (NAT for private subnets, Internet for public).

DDL for Amazon Athena Tables

These are all prerequisites to have setup in AWS before you can even deploy and run the job.

How I Generated the Code for the ETL Job/EMR/Spark Infrastructure

My practical approach towards building out this repo.

- Before jumping into the code, I started with the AWS console…
- I manually created an EMR cluster (researched instructions on how to do this)
  - Provisioned Spark as part of the cluster setup
  - Configured a step to execute the ETL routine
  - Stored the ETL script in S3 for EMR to access during execution
- After I got that working, I used a PowerShell script (windows guy…) to export the definition and configurations of the cluster to a text file.
- Fed that cluster Definition and Configurations file to AI (ChatGPT + GitHub Copilot) for assistance on generating all the needed infrastructure files (YAML + terraforms)
- Did my best to keep it simple and exclude unnecessary files that overcomplicated the repo.

Repository Structure & Workflow Details
GitHub Integration & CI/CD

📂 .github/workflows/

GitHub Actions CI/CD pipelines for automated deployment and ETL execution.
- deploy.yml → Automatic deployment triggered on pushes to main. Syncs ETL job code from ./jobs/ to S3 using secure OIDC authentication. Keeps S3 bucket updated.
- run-emr-etl.yml → Manual workflow for complete ETL pipeline execution. Creates transient EMR cluster, uploads and runs Spark job, waits for completion, exports logs and artifacts.
- terraform-emr.yml → Manual infrastructure deployment using Terraform. Creates persistent EMR cluster with job pre-loaded. Useful for development environments.

📂 infra/

Terraform definitions for infrastructure.
- main.tf → Defines EMR cluster, instance groups, roles, steps, log URIs, etc.
- variables.tf → Input variables for region, subnet, instance type, bucket names, roles. Makes infrastructure reusable across environments.

📂 jobs/

Spark jobs (the actual ETL code you want to run).
- poc_etl_job.py → Python Spark script that performs Extract → Transform → Load. This is the business logic of your ETL pipeline.

📂 scripts/

Developer helper scripts (not production jobs).
- Get-EMRClusterInfo.ps1 → PowerShell script to fetch/export cluster details, configs, and steps into a single text file. Useful for debugging, documentation, and learning.

Demo

Live Demo: Deploying and Running PySpark ETL on EMR from GitHub

🚀 Repository walkthrough
⚡ Cluster launch and job execution

Other things to Consider
📊 Monitoring and logging
💰 Cost optimization strategies

Thank you so much!!!

Please feel free to ask any questions and connect.

Personal Brand

BytePeak Engineering LLC