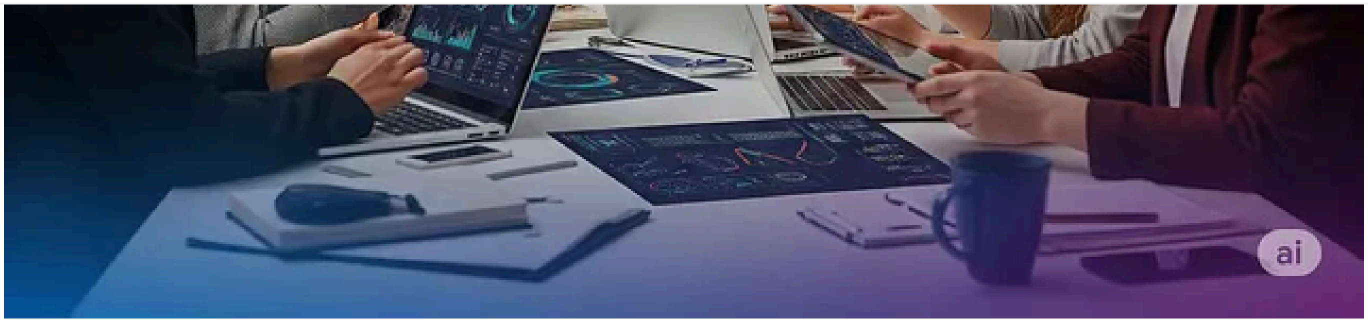
Empowering Interns with Meaningful
Data Analytics Projects


ai

Empowering Interns with Meaningful Data Analytics Projects: A Spark-Based Approach

Anurag Sharma 6 min read · Jul 28, 2025

3

At our organization, we frequently onboard fresh college graduates as interns, eager to dive into the world of data analytics. Our goal is to provide them with hands-on, impactful work that accelerates their learning while introducing them to the tools and workflows central to our team, most notably, Apache Spark. Spark is a cornerstone of our data analytics ecosystem, used for querying data, building machine learning models, constructing data pipelines, and performing in-depth analysis. However, assigning critical or time-sensitive tasks to interns, who are still on their learning journey, isn't practical. Instead, we needed to craft projects that were educational, meaningful, and aligned with our team's objectives.
As a data analytics lead, I was tasked with creating a stream of technical debt items — small, self-contained projects that improve our systems while offering interns a chance to grow their skills. These tasks had to strike a balance: they needed to be impactful enough to contribute to our project's health but forgiving enough to allow interns to experiment, learn, and make mistakes. After some brainstorming, I landed on an approach that combined code optimization, pipeline development, and cloud infrastructure exposure. Here's how we turned this idea into a practical, Spark-based use case that empowered our interns and kept our project running smoothly.

The Challenge: Creating Meaningful Work for Interns
Our team relies heavily on Databricks, a unified analytics platform built on Apache Spark, to manage our data workflows. We run hundreds of Databricks jobs daily, each performing tasks like data processing, ETL pipelines, and machine learning model training. Over time, we noticed that some jobs were running slower than expected, hinting at opportunities to optimize code before resorting to scaling up cluster resources — an expensive and often unnecessary fix.
This observation inspired a project that was perfect for interns: building a system to monitor and analyze Databricks job runtime metrics. The goal was to create a dataset of job performance metrics, store it efficiently, and enable analytics to identify jobs that needed optimization. This project would allow interns to work with Spark, learn about cloud storage (AWS S3), explore data partitioning, and gain hands-on experience with Databricks' Hive metastore — all while contributing to reducing technical debt.

The Solution: A Spark-Based Job Monitoring Pipeline
To kick things off, I designed a Python script that leveraged Spark to query runtime details for all our Databricks jobs. The script collected key metrics — such as job name, execution date, and runtime duration — and transformed

them into a structured dataset. This dataset was then saved as a Parquet file in an AWS S3 bucket, a format chosen for its efficiency and compatibility with Spark. To make the data accessible for analytics, we created a daily Databricks Hive table on top of the Parquet files, partitioned by job name. With over 400 jobs running in our environment, this resulted in a table with 400+ partitions, each containing details like:

• **Date**: When the job ran

• **Job Name**: The unique identifier for the job

• **Runtime**: The duration of the job's execution in seconds

Here's a simplified version of the Python script that powered this pipeline:

```
from pyspark.sql import SparkSession
from datetime import datetime
# Initialize Spark session
spark = SparkSession.builder.appName("JobRuntimeMonitoring").getOrCreate()
# Query Databricks job runtime details (simplified example)
job_data = spark.sql("""
   SELECT job_name, execution_date, runtime_seconds
   FROM databricks_job_logs
   WHERE execution_date = current_date()
""")
# Save dataset as Parquet to S3, partitioned by job_name
s3_path = "s3://my-bucket/job_runtime_data/"
job_data.write \
   .format("parquet") \
   .partitionBy("job_name") \
   .mode("append") \
   .save(s3_path)
# Register as a Hive table
spark.sql(f"""
   CREATE TABLE IF NOT EXISTS job_runtime_metrics
   USING PARQUET
   PARTITIONED BY (job_name)
   LOCATION '{s3_path}'
""")
```

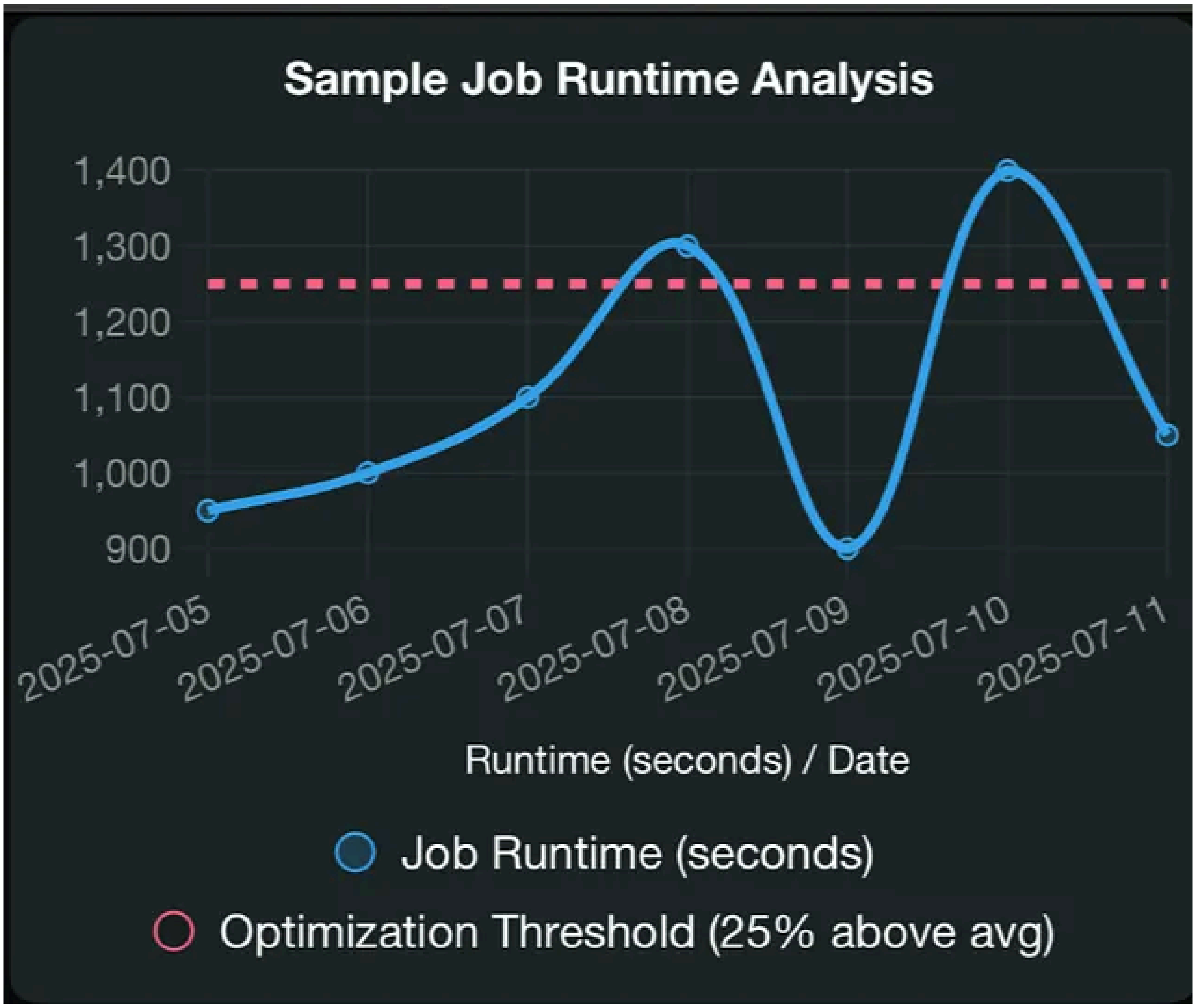
This pipeline ran daily, ensuring fresh data was available for analysis. By partitioning the table by job_name, we optimized query performance for analytics tasks, as Spark could efficiently read only the relevant partitions.

Turning Data into Action: Analytics for Code Optimization
With the dataset in place, the next step was to enable interns to perform analytics on the job runtime data. The goal was to identify jobs that were running inefficiently — specifically, those with runtimes exceeding 25% above their historical average. This threshold was chosen to flag jobs that might benefit from code-level optimizations, such as rewriting inefficient Spark transformations, optimizing joins, or caching intermediate results. Interns were tasked with writing Spark SQL queries to compute average runtimes per job and flag outliers.

Here's an example query they developed:

```
WITH job_stats AS (
   SELECT
       job_name,
       AVG(runtime_seconds) AS avg_runtime,
       STDDEV(runtime_seconds) AS stddev_runtime
   FROM job_runtime_metrics
   GROUP BY job_name
)
SELECT
   j.job_name,
   j.execution_date,
   j.runtime_seconds,
        js.avg_runtime,
        CASE
            WHEN j.runtime_seconds > (js.avg_runtime * 1.25) THEN 'Needs Optimization
            ELSE 'Normal'
        END AS status
    FROM job_runtime_metrics j
    JOIN job_stats js ON j.job_name = js.job_name
    WHERE j.execution_date = current_date()
    AND j.runtime_seconds > (js.avg_runtime * 1.25);
```

> *This query helped interns identify jobs that were running longer than expected.*
> *For each flagged job, they would:*
> *1.* ***Debug the Code:*** *Dive into the job's Spark code to identify inefficiencies, such as unnecessary shuffles, poorly optimized joins, or lack of caching.*
> *2.* ***Propose Fixes:*** *Suggest improvements, like using broadcast joins for small datasets or repartitioning data to reduce skew.*
> *3.* ***Test Optimizations:*** *Run the optimized code in a sandbox environment to measure performance improvements.*
> *4.* ***Document Findings:*** *Create reports summarizing their analysis and recommendations, fostering a culture of knowledge sharing.*
> *This process not only helped interns learn Spark's internals but also gave them exposure to debugging and performance tuning — critical skills in data analytics.*



Why This Worked for Interns
This project was a win-win for both the interns and our team.

Here's why:
• **Hands-On Learning:** Interns gained practical experience with Spark, Python, and SQL, learning how to query, transform, and analyze data at scale.
• **Cloud Exposure:** By saving data to S3 and creating Hive tables, they learned about cloud storage and metadata management.
• **Real Impact:** Their work directly contributed to identifying and resolving technical debt, improving the efficiency of our data pipelines.

• **Safe Environment**: Since the tasks weren't time-critical, interns could experiment, make mistakes, and learn without risking production systems.
• **Scalable Framework**: The partitioned dataset and analytics queries provided a reusable framework for ongoing monitoring, ensuring the project had lasting value.

Key Takeaways for Engaging Interns

From this experience, we distilled a few principles for creating meaningful intern projects in data analytics:
1. **Focus on Technical Debt**: Low-risk, high-impact tasks like code optimization or monitoring pipelines are perfect for interns.
2. **Leverage Core Tools**: Design projects around the tools your team uses (e.g., Spark, Databricks) to align with real-world workflows.
3. **Provide Structure, Allow Freedom**: Give interns clear guidelines (e.g., a script template) but let them explore and innovate within those boundaries.
4. **Emphasize Learning Outcomes**: Ensure projects teach transferable skills like debugging, cloud infrastructure, and data analysis.
5. **Celebrate Contributions**: Highlight how interns' work improves the project, boosting their confidence and sense of ownership.

Conclusion

By creating a Spark-based pipeline to monitor Databricks job runtimes, we gave our interns a chance to dive into the data analytics world, learn Apache Spark, and contribute to our project's success. The project not only helped us address technical debt but also empowered interns to develop practical skills in coding, cloud infrastructure, and performance optimization. As data teams continue to grow, finding ways to engage interns with meaningful, educational projects will remain a key strategy for building the next generation of data professionals.