PySpark coding best Practices guidelines

Surendra Panpaliya

Use SparkSession

Always use the SparkSession object

to interact with Spark.

It's the entry point to any Spark functionality and

ensures proper management of resources.

Avoid Global Imports

Import only the necessary modules and functions.

Avoid using global imports like

from pyspark.sql import *,

as it can lead to namespace conflicts.

Use DataFrame API

Prefer using the DataFrame API over the RDD API.

The DataFrame API provides optimizations

Improvements in terms of performance and ease of use.

Lazy Evaluation

PySpark uses lazy evaluation

Means transformations are not executed immediately.

Perform actions like .show() or .count()

to trigger execution when needed.

Caching and Persistence

Cache intermediate DataFrames

when you plan to reuse them

in multiple operations.

Use .cache() or .persist()

to improve performance.

Avoid UDFs

Use built-in DataFrame functions

instead of User-Defined Functions (UDFs)

As they are more optimized

Partitioning

When reading or writing data,

take advantage of partitioning

to optimize query performance.

Useful for large datasets.

Avoid Data Shuffling

Minimize data shuffling,

which involves moving

data across nodes.

Shuffling can be expensive

in terms of performance.

Configure Resources

Adjust the Spark configuration

to allocate resources

Memory, cores, etc.

based on your cluster's capabilities and

workload requirements.

Broadcast Variables

Use broadcast variables

to efficiently share

small read-only data

across all nodes.

Avoid Collecting Large Data

Avoid collecting large DataFrames

to the driver node.

Instead, write the data

to a storage system or

perform aggregations

to reduce data size.

Avoid Collecting Large Data

- Data Format and Compression
- Choose appropriate file formats
- (Parquet, Delta, etc.)
- compression methods
- to optimize storage and
- query performance.

Error Handling

Implement proper error handling and

logging to ensure that

issues are identified and

addressed promptly.

Testing

Write unit tests for your PySpark code.

Use tools like pyspark.sql.SparkSession’s

createDataFrame

to create test data.

Code Formatting

- Follow a consistent
- Coding style,
- Such as PEP 8,
- To make your code
- More readable.

Documentation

Document your code,

especially complex transformations and logic,

To make it easier for others

Your future self to understand.

Code Review

Encourage code reviews

To catch potential issues

Ensure adherence

To best practices.

Dynamic Partitioning

If using partitioning

Avoid too many dynamic partitions,

as they can negatively

Impact performance.

Use Broadcast Joins

When joining a small DataFrame

with a large one,

use a broadcast join

to avoid shuffling.

Cluster Considerations

- Keep in mind the characteristics
- Resources of your cluster
- When designing
- Optimizing your code.

Summary

By following these best practices,

You'll be able to write PySpark code

That is more maintainable,

Efficient

aligned with industry standards.

Thank You

Surendra Panpaliya