<https://www.linkedin.com/pulse/catalyst-tungsten-apache-sparks-speeding-engine-deepak-rajak/?trackingId=GWe8Z5AfjJ3krt3IyAUsCg%3D%3D>

## 0 notifications total

Skip to searchSkip to main contentClose jump menu

0 suggestions found.

## Primary Navigation

* [Home](https://www.linkedin.com/feed/)

**[1](https://www.linkedin.com/mynetwork/)**[1 new 1 new network update. notification](https://www.linkedin.com/mynetwork/)

[My Network](https://www.linkedin.com/mynetwork/)

* [Jobs](https://www.linkedin.com/jobs/)
* [Messaging](https://www.linkedin.com/messaging/)

**[20](https://www.linkedin.com/notifications/)**[20 new 20 new notifications. notifications](https://www.linkedin.com/notifications/)

[Notifications](https://www.linkedin.com/notifications/)

* Me
* Work
* [Retry Premium Free](https://www.linkedin.com/premium/products?upsellOrderOrigin=premium_nav_upsell_text&destRedirectURL=https%3A%2F%2Fwww.linkedin.com%2Ffeed%2F%3Ftrk%3Dhomepage-basic_signin-form_submit%26showPremiumWelcomeBanner%3Dtrue)

A picture containing text, sign, clipart

Description automatically generated

# Catalyst and Tungsten: Apache Spark's Speeding Engine

* Published on June 5, 2020

[[](https://www.linkedin.com/in/deepak-rajak-935b411b/)](https://www.linkedin.com/in/deepak-rajak-935b411b/)

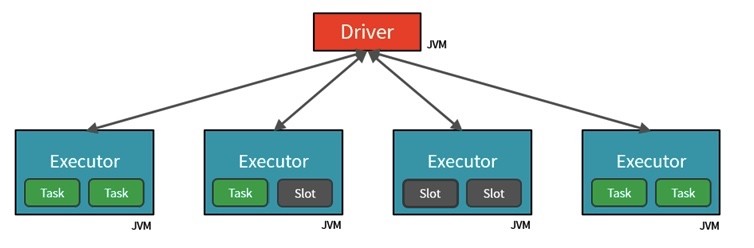
## [Deepak Rajak](https://www.linkedin.com/in/deepak-rajak-935b411b/)

Data Engineering /Advanced Analytics Technical Delivery Lead at Exusia, Inc.

[25 articles](https://www.linkedin.com/in/deepak-rajak-935b411b/detail/recent-activity/posts/)Follow

Spark is a Distributed computing environment. The unit of distribution is a Spark Cluster. Every Cluster has a Driver and one or more executors. Work submitted to the Cluster is split into as many independent Jobs as needed. This is how work is distributed across the Cluster's nodes. Jobs are further subdivided into tasks. The input to a job is partitioned into one or more partitions. These partitions are the unit of work for each slot. In between tasks, partitions may need to be re-organized and shared over the network.

## The cluster: Drivers, executors, slots & tasks



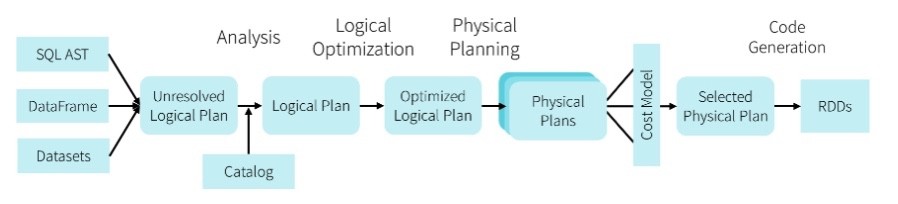
Among the most powerful components of Spark are **Spark SQL**. At its core lies the **Catalyst optimizer**. This extensible query optimizer supports both **rule-based and cost-based optimization.**

**Spark** uses two engines to optimize and run the queries - **Catalyst** and **Tungsten**, in that order. Catalyst basically generates an optimized physical query plan from the logical query plan by applying a series of transformations like **predicate pushdown, column pruning, and constant folding** on the logical plan. This optimized query plan is then used by Tungsten to generate optimized code, that resembles hand written code, by making use of **Whole-stage Codegen** functionality introduced in Spark 2.0. This functionality has improved Spark's efficiency by a huge margin from Spark 1.6, which used the traditional **Volcano Iterator Model**.

**Catalyst** is based on functional programming constructs in Scala and designed with these key two purposes:

* Easily add new optimization techniques and features to Spark SQL
* Enable external developers to extend the optimizer (e.g. adding data source specific rules, support for new data types, etc.)

When you execute code, **Spark SQL** uses Catalyst's general tree transformation framework in four phases, as shown below:



### ****Analysis:****

Spark SQL Optimization starts from relation to be computed. It is computed either from **abstract syntax tree (AST)** returned by **SQL parser** or Dataframe created using API. Both may contain unresolved attribute references or relations. By unresolved attribute, it means we don’t know its type or have not matched it to an input table. Spark SQL make use of Catalyst rules and a **Catalog** object that track data in all data sources to resolve these attributes. It starts by creating an **unresolved logical plan**, and then apply the analysis rules

### ****Logical O****ptimization:

In this phase of Spark SQL optimization, the standard rule-based optimization is applied to the logical plan. It includes **constant folding**, **predicate pushdown**, **projection pruning**and other rules. A logical plan describes computation on datasets without defining how to conduct the computations. i.e. **scan, filter, join, project, aggregate**

1. **Constant folding** is the process of recognizing and evaluating constant expressions at compile time rather than computing them at runtime. This is not in any particular way specific to Catalyst. It is just a standard compilation technique and its benefits should be obvious. It is better to compute expression once than repeat this for each row
2. **Predicate pushdown** corresponds to WHERE clause in the SQL query. If these can be use directly by external system (like Relational Databases ) or for partition pruning (like in Parquet) this means reduced amount of data that has to be transferred / loaded from disk.
3. **Projection pruning** benefits are pretty much the same as for predicate pushdown. If some columns are not used, downstream data source may discard this on read.

### ****Physical Planning:****

There are about 500+ lines of code in the physical planning rules. In this phase, one or more physical plan is formed from the logical plan, using physical operator matches the Spark execution engine. And it selects the plan using the **cost model**. It uses Cost-based optimization only to select join algorithms. For small relation SQL uses broadcast join, the framework supports broader use of cost-based optimization. It can estimate the cost recursively for the whole tree using the rule.

Rule-based physical optimization, such as pipelining projections or filters into one Spark mapOperation is also carried out by the physical planner. Apart from this, it can also push operations from the logical plan into data sources that support predicate or projection pushdown.

A physical plan describes computation on datasets with specific definitions on how to conduct the computation. i.e. **ParquetScan, Sort Merge Join, Filter, Project, Hash Aggregate**

### ****Code Generatio****n:

The final phase of Spark SQL optimization is code generation. It involves generating Java bytecode to run on each machine. Whole-Stage CodeGen is also known as Whole-Stage Java Code Generation, which is a physical query optimization phase in Spakr SQL that clubs multiple physical operations together to form a single Java function.

Whole-Stage Java code generation improves the execution performance by converting a query tree into an optimized function that eliminates unnecessary calls and leverages CPU registers for intermediate data.

Whole-Stage CodeGen is enabled by default in Spark 2.x. This can be controlled by the property **spark.sql.codegen.wholeStage**.

Whole-Stage CodeGen is getting used by some of the modern massively parallel processing (**MPP**) databases to achieve efficiency in execution performance.

## Tungsten

The goal of Project Tungsten is to improve Spark execution by optimising Spark jobs for CPU and memory efficiency (as opposed to network and disk I/O which are considered fast enough).

1. **Off-Heap Memory Management** using binary in-memory data representation aka Tungsten row format and managing memory explicitly,
2. **Cache Locality** which is about cache-aware computations with cache-aware layout for high cache hit rates
3. Whole-Stage Code Generation (aka CodeGen).

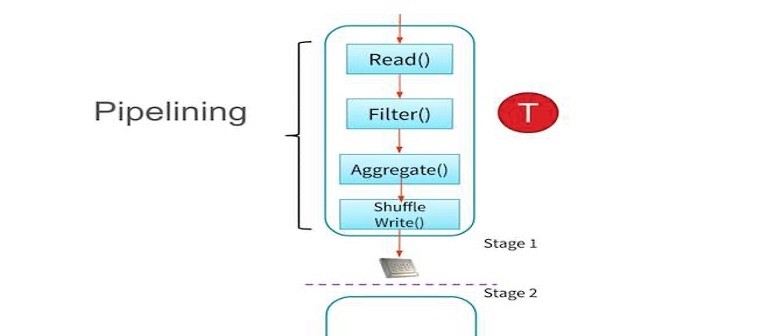
**property: spark.sql.tungsten.enabled to true**

### Lets see how Spark utilises it with an example:

As opposed to narrow transformations, wide transformations cause data to shuffle between executors. This is because a wide transformation requires sharing data across workers. Pipelining helps us optimize our operations based on the differences between the two types of transformations.

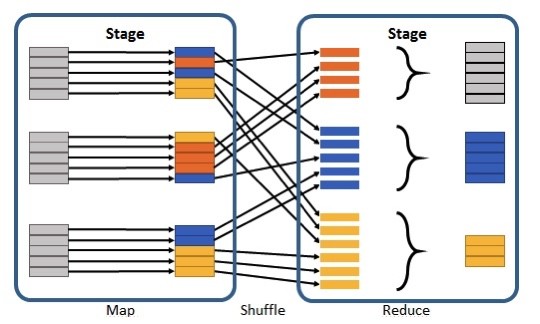
## Pipelining

* Pipelining is the idea of executing as many operations as possible on a single partition of data.
* Once a single partition of data is read into RAM, Spark will combine as many narrow operations as it can into a single Task
* Wide operations force a shuffle, conclude a stage, and end a pipeline.



## Shuffles

A shuffle operation is triggered when data needs to move between executors.



To carry out the shuffle operation Spark needs to:

* Convert the data to the **UnsafeRow, commonly referred to as Tungsten Binary Format.**
* Write that data to disk on the local node - at this point the slot is free for the next task.
* Send that data across the wire to another executor
* Technically the Driver decides which executor gets which piece of data.
* Then the executor pulls the data it needs from the **other executor's shuffle files.**
* Copy the data back into RAM on the new executor
* The concept, if not the action, is just like the initial read "every" DataFrame starts with.
* The main difference being it's the 2nd+ stage.

As we will see in a moment, this amounts to a **free cache from what is effectively temp files**. Some actions induce in a shuffle. Good examples would include the operations count() and reduce(..).

## UnsafeRow (also known as Tungsten Binary Format)

Sharing data from one worker to another can be a costly operation.

Spark has optimized this operation by using a format called Tungsten.

Tungsten prevents the need for expensive serialization and de-serialization of objects in order to get data from one JVM to another.

The data that is "**shuffled**" is in a format known as UnsafeRow, or more commonly, the Tungsten Binary Format.

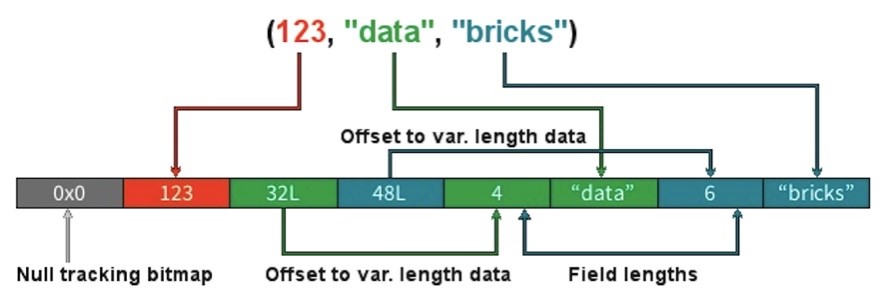
UnsafeRow is the in-memory storage format for Spark SQL, DataFrames & Datasets.

Advantages include:

* Compactness:
* Column values are encoded using custom encoders, not as JVM objects (as with RDDs).
* The benefit of using Spark 2.x's custom encoders is that you get almost the same compactness as Java serialization, but significantly **faster encoding/decoding speeds**.
* Also, for custom data types, it is possible to write custom encoders from scratch.
* Efficiency: Spark can operate directly out of Tungsten, without first deserializing Tungsten data into JVM objects.

### How UnsafeRow works

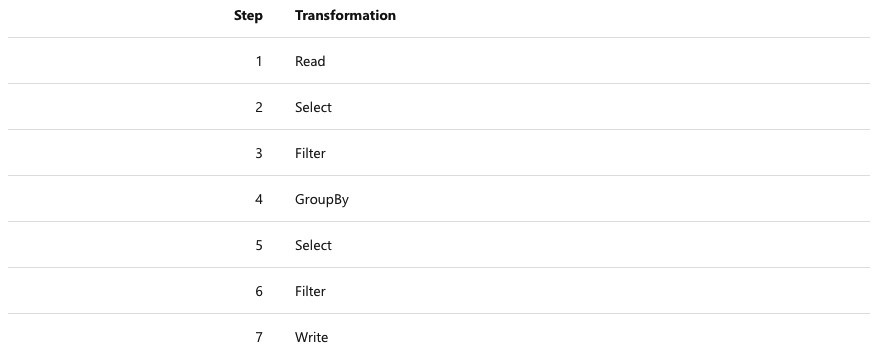
* The first field, "123", is stored in place as its primitive.
* The next 2 fields, "data" and "bricks", are strings and are of variable length.
* An offset for these two strings is stored in place (32L and 48L respectively shown in the picture below).
* The data stored in these two offset's are of format "length + data".
* At offset 32L, we store 4 + "data" and likewise at offset 48L we store 6 + "bricks".



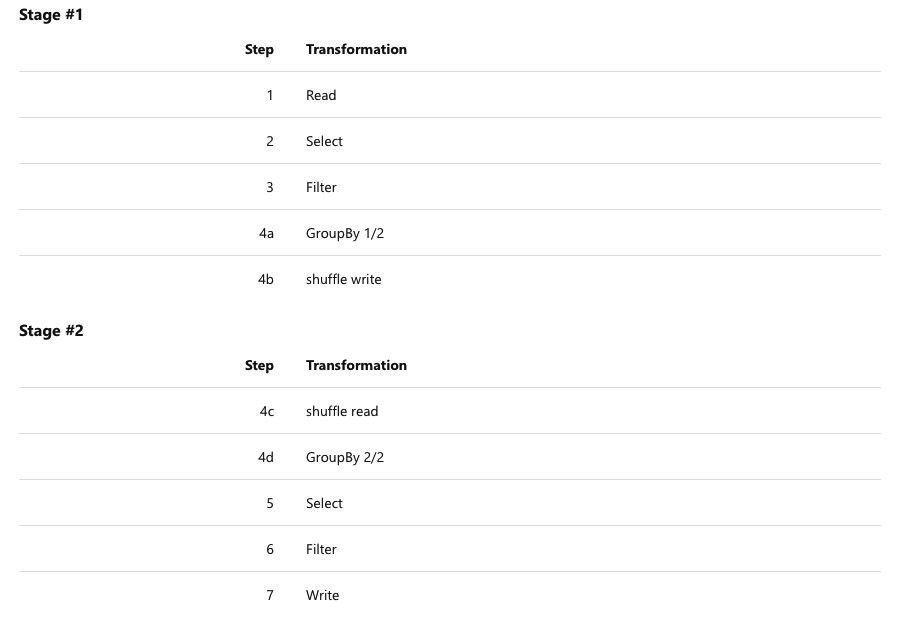
## Stages

* When we shuffle data, it creates what is known as a **stage boundary.**
* Stage boundaries represent a process bottleneck.

Take for example the following transformations:



Spark will break this one job into two stages (steps 1-4b and steps 4c-7):



In **Stage #1**, Spark will create a pipeline of transformations in which the data is read into RAM (Step #1), and then perform steps #2, #3, #4a & #4b

All partitions must complete **Stage #1** before continuing to Stage #2

* It's not possible to group all records across all partitions until every task is completed.
* This is the point at which all the tasks must synchronize.
* This creates our bottleneck.
* Besides the bottleneck, this is also a significant performance hit: disk IO, network IO and more disk IO.

Once the data is shuffled, We can resume execution...

For **Stage #2**, Spark will again create a pipeline of transformations in which the shuffle data is read into RAM (Step #4c) and then perform transformations #4d, #5, #6 and finally the write action, step #7.

## Lineage

From the developer's perspective, we start with a read and conclude (in this case) with a write:

However, **Spark starts with the action (write(..) in this case**).

Next, it asks the question, what do I need to do first?

It then proceeds to determine which transformation precedes this step until it identifies the first transformation.

## Why Work Backwards?

**Question**: So what is the benefit of working backward through your action's lineage? **Answer**: It allows Spark to determine if it is necessary to execute **every**transformation.

Take another look at our example:

* Say we've executed this once already
* On the first execution, step #4 resulted in a shuffle
* Those shuffle files are on the various executors (src & dst)
* Because the transformations are **immutable**, no aspect of our lineage can change.
* That means the results of our last shuffle (if still available) can be reused.

In this case, what we end up executing is only the operations from **Stage #2**.

This saves us the initial network read and all the transformations in **Stage #1**

## Caching

The reuse of shuffle files (also known as our temp files) is just one example of Spark optimising queries anywhere it can. We cannot assume this will be available to us.

Shuffle files are by definition **temporary files and will eventually be removed.**

However, we cache data to explicitly accomplish the same thing that happens inadvertently with shuffle files.

In this case, the lineage plays the same role. Take for example:

In this case we **cached** the result of the select(..).

We never even get to the part of the lineage that involves the shuffle, let alone Stage #1.

Instead, we pick up with the **cache** and **resume** execution from there:

## Conclusion:

So we have seen how spark optimises the entire execution of our code into multiple ways. It utilises the capabilities of Catalyst and Tungsten, it works backward after an action being called upon, it defines the stage boundaries and create pipelines for a particular stage and finally if we cache the RDDs explicitly then it guarantees faster and efficient query execution.

Thanks for reading it. I hope you have found yourself informed. Please let me know in case you have any questions on this. Please click the "like" button if you have liked it.

**credits**: [**https://www.youtube.com/watch?v=AoVmgzontXo**](https://www.youtube.com/watch?v=AoVmgzontXo) and Azure Databricks Learn Documents.

Report this

### Published by

**[Error! Filename not specified.](https://www.linkedin.com/in/deepak-rajak-935b411b/)**

[**Deepak Rajak**](https://www.linkedin.com/in/deepak-rajak-935b411b/)

Data Engineering /Advanced Analytics Technical Delivery Lead at Exusia, Inc.

Published • 11mo

[25 articles](https://www.linkedin.com/in/deepak-rajak-935b411b/detail/recent-activity/posts/)Follow

Hi All, In this article, We will understand why Apache Spark is lightening fast? How it optimizes the code at various level via Catalyst and Tungsten ? What happens when we encounter a Data Shuffle ? What is Constant Folding, Predicate Pushdown and Projection Pruning? What is the Unsafe row format? What exactly the Whole-Stage CodeGen ? How spark defines the stage boundaries ? [hashtag#costantfolding](https://www.linkedin.com/feed/hashtag/?keywords=costantfolding) [hashtag#predicatepushdown](https://www.linkedin.com/feed/hashtag/?keywords=predicatepushdown) [hashtag#projectionpruning](https://www.linkedin.com/feed/hashtag/?keywords=projectionpruning) [hashtag#logicalplanning](https://www.linkedin.com/feed/hashtag/?keywords=logicalplanning) [hashtag#physicalplanning](https://www.linkedin.com/feed/hashtag/?keywords=physicalplanning) [hashtag#wholestagecodegen](https://www.linkedin.com/feed/hashtag/?keywords=wholestagecodegen) [hashtag#apachespark](https://www.linkedin.com/feed/hashtag/?keywords=apachespark) [hashtag#dataengineering](https://www.linkedin.com/feed/hashtag/?keywords=dataengineering) [hashtag#exusia](https://www.linkedin.com/feed/hashtag/?keywords=exusia) [hashtag#catalyst](https://www.linkedin.com/feed/hashtag/?keywords=catalyst) [hashtag#tungsten](https://www.linkedin.com/feed/hashtag/?keywords=tungsten) [hashtag#unsaferowformat](https://www.linkedin.com/feed/hashtag/?keywords=unsaferowformat) [Exusia, Inc.](https://www.linkedin.com/company/exusia-inc-/)

Like

Comment

Share

* 164
* 14 comments

### Reactions

**[Error! Filename not specified.](https://www.linkedin.com/in/shubhamkanungo?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAABXEpZYBUIYMiWcpHFCjXgVr-0F2I0AQU-8" \t "_blank)**

[Shubham Waghmare](https://www.linkedin.com/in/shubham-waghmare-2778b6142?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAACKpJPsB4FZmNLkzwkimBv_JXRu630jbTkw" \t "_blank)

**[Error! Filename not specified.](https://www.linkedin.com/in/ravikiranvandhanapu?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAAAR6JU0Bax9Wa7PrpdtFCbBPaDyscWCVQY0" \t "_blank)**

**[Error! Filename not specified.](https://www.linkedin.com/in/debabrat-sarma-7431b6a0?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAABVi86MBlVUsH9NnCfuFN5OQrY1I5HXLc8c" \t "_blank)**

[Poorna Bogadi](https://www.linkedin.com/in/poorna-bogadi-65050785?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAABIM_twB2kIIC8wzm4IU6so4cUSdFbv0PIc" \t "_blank)

**[Error! Filename not specified.](https://www.linkedin.com/in/kuldeeptandon?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAABbgJIEBSfQzbv_FYu_DXc5EpWYEShaj7kI" \t "_blank)**

[Keerthivass Bs](https://www.linkedin.com/in/keerthivass-bs-863a75171?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAACjqJYsBGEnX2B_40eHr6saHcfVTb41l7u4" \t "_blank)

**[Error! Filename not specified.](https://www.linkedin.com/in/prabukasthurisamy?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAAA0FvW4BqrV5gV7O20CzGEoQV5txPPi1Foo" \t "_blank)**

**[Error! Filename not specified.](https://www.linkedin.com/in/narender-sharma-5b07205b?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAAAy-KVQBSaJSYTUFrzOHo3tFBHiOTZ9_-r8" \t "_blank)**

**[Error! Filename not specified.](https://www.linkedin.com/in/pariksheet-de-994775100?miniProfileUrn=urn%3Ali%3Afs_miniProfile%3AACoAABm98kABolcnbfe0lVycud117icKeczJlu8" \t "_blank)**

* …

### 14 CommentsComments on Deepak Rajak’s article

**Most relevant**



Top of Form

0 suggestions found.

Open Emoji Keyboard

Bottom of Form

**[Error! Filename not specified.](https://www.linkedin.com/in/ramsjha/" \t "_self)**

### [Ram Jhaout of network 3rd+VP (IT) @ Morgan Stanley | Data Engineering | Ex - J P Morgan, Capgemini | PMP® TOGAF®](https://www.linkedin.com/in/ramsjha/" \t "_self)

10mo

Awesome post. Just a note, if you are running out of time then [**Sarfaraz Hussain**](https://www.linkedin.com/in/ACoAACBzXo4BgGJqXGtmNphxt2qiqIbPEBfMvEo/) had a abstract post on same lines and a video which can be referred.

Like

 3

Reply2 Replies2 Replies on Ram Jha’s comment

**[Error! Filename not specified.](https://www.linkedin.com/in/deepak-rajak-935b411b/" \t "_self)**

### [Deepak Rajak2nd degree connection 2ndData Engineering /Advanced Analytics Technical Delivery Lead at Exusia, Inc.](https://www.linkedin.com/in/deepak-rajak-935b411b/" \t "_self)

10mo

Thanks [**Ram Jha**](https://www.linkedin.com/in/ACoAAAGI1GsB422LjPUqWAXl1Gbd8iM3CCkWVqU/) ✅

Like

Reply

**[Error! Filename not specified.](https://www.linkedin.com/in/sarfaraz-hussain-8123b4132/" \t "_self)**

### [Sarfaraz Hussain2nd degree connection 2ndData Engineer @ Walmart Labs || Big Data Enthusiast || Spark | Scala | FP | Snowflake || Fitness Freak](https://www.linkedin.com/in/sarfaraz-hussain-8123b4132/" \t "_self)

10mo

Thanks [**Ram Jha**](https://www.linkedin.com/in/ACoAAAGI1GsB422LjPUqWAXl1Gbd8iM3CCkWVqU/) 🙂

Like

 1

Reply

**[Error! Filename not specified.](https://www.linkedin.com/in/harsh-patne-66b711145/" \t "_self)**

### [Harsh Patne2nd degree connection 2ndSr.Data Architect at Globant](https://www.linkedin.com/in/harsh-patne-66b711145/" \t "_self)

10mo

[**Komal Tolani**](https://www.linkedin.com/in/ACoAAA2PmogB8FzbKtJnrnHLX6cRWtDwn54Yymo/) this is very useful stuff

Like

 2

Reply1 Reply1 Comment on Harsh Patne’s comment

**[Error! Filename not specified.](https://www.linkedin.com/in/deepak-rajak-935b411b/" \t "_self)**

### [Deepak Rajak2nd degree connection 2ndData Engineering /Advanced Analytics Technical Delivery Lead at Exusia, Inc.](https://www.linkedin.com/in/deepak-rajak-935b411b/" \t "_self)

10mo

Thanks [**Harsh Patne**](https://www.linkedin.com/in/ACoAACMqv3MBDYKB1Vz-JJ2XgNRT74CpEz624Ig/)

Like

Reply

Load more comments

[[](https://www.linkedin.com/in/deepak-rajak-935b411b/)](https://www.linkedin.com/in/deepak-rajak-935b411b/)

## [Deepak Rajak](https://www.linkedin.com/in/deepak-rajak-935b411b/)

### Data Engineering /Advanced Analytics Technical Delivery Lead at Exusia, Inc.

Follow

### More from Deepak Rajak

**[Error! Filename not specified.](https://www.linkedin.com/pulse/deploying-databricks-google-cloud-platform-deepak-rajak/)**

**[Deploying Databricks on Google Cloud Platform](https://www.linkedin.com/pulse/deploying-databricks-google-cloud-platform-deepak-rajak/)**

[Deepak Rajak on LinkedIn](https://www.linkedin.com/pulse/deploying-databricks-google-cloud-platform-deepak-rajak/)

**[Error! Filename not specified.](https://www.linkedin.com/pulse/ci-cd-azure-databricks-using-devops-deepak-rajak/)**

**[CI / CD in Azure Databricks using Azure DevOps](https://www.linkedin.com/pulse/ci-cd-azure-databricks-using-devops-deepak-rajak/)**

[Deepak Rajak on LinkedIn](https://www.linkedin.com/pulse/ci-cd-azure-databricks-using-devops-deepak-rajak/)

**[Error! Filename not specified.](https://www.linkedin.com/pulse/read-write-from-aws-s3-azure-datalake-storage-google-cloud-rajak/)**

**[Read / Write from AWS S3 , Azure DataLake Storage & Google Cloud Storage without mounting via Databricks](https://www.linkedin.com/pulse/read-write-from-aws-s3-azure-datalake-storage-google-cloud-rajak/)**

[Deepak Rajak on LinkedIn](https://www.linkedin.com/pulse/read-write-from-aws-s3-azure-datalake-storage-google-cloud-rajak/)

**[Error! Filename not specified.](https://www.linkedin.com/pulse/reading-from-azure-datalake-writing-google-bigquery-via-deepak-rajak/)**

**[Reading from Azure DataLake & Writing to Google BigQuery via Databricks](https://www.linkedin.com/pulse/reading-from-azure-datalake-writing-google-bigquery-via-deepak-rajak/)**

[Deepak Rajak on LinkedIn](https://www.linkedin.com/pulse/reading-from-azure-datalake-writing-google-bigquery-via-deepak-rajak/)

[See all 25 articles](https://www.linkedin.com/in/deepak-rajak-935b411b/detail/recent-activity/posts/)



Status is online

#### MessagingYou are on the messaging overlay. Press enter to open the list of conversations.

Compose message

You are on the messaging overlay. Press enter to open the list of conversations.