



Top 10 AWS Data Engineering Redshift Interview Q & A

Curated by:

Sachin Chandrashekhar

Founder – Data Engineering Hub

🎯 LinkedIn: <https://www.linkedin.com/in/sachincw/>

🎯 WhatsApp Community:
<https://chat.whatsapp.com/FAqHgo4YpUsLFScpiMvtSF>

🎯 Top mate link: <https://lnkd.in/d28ETqaN>

🎯 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

Q1: What are the key components of Amazon Redshift architecture?

A: The key components of Amazon Redshift architecture include:

- **Leader Node:** Manages client connections and receives queries. It then parses, optimizes, and coordinates the execution of these queries.
- **Compute Nodes:** Execute the queries and return results to the leader node. Compute nodes store data and perform the actual query processing.
- **Node Slices:** Each compute node is divided into slices, and each slice is assigned a portion of the node's memory and disk space.


Q2: How does Amazon Redshift achieve high performance for query execution?

A: Amazon Redshift achieves high performance through several mechanisms:

- **Massively Parallel Processing (MPP):** Distributes query processing across multiple nodes.
- **Columnar Storage:** Reduces the amount of data read from disk by reading only the columns involved in the query.
- **Data Compression:** Reduces I/O and storage costs.
- **Result Caching:** To reduce query runtime and improve system performance, Amazon Redshift caches the results of certain types of queries in memory on the leader node. When a user submits a query, Amazon Redshift checks the results cache for a valid, cached copy of the query results. If a match is found in the result cache, Amazon Redshift uses the cached results and doesn't run the query..

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **Query Optimization:** Uses sophisticated algorithms to optimize query execution plans.

Q3: How does Amazon Redshift handle data storage and compression?

Amazon Redshift employs two key techniques for data storage and compression:


1. **Columnar Storage:** Unlike traditional row-based storage, Redshift stores data in a columnar format. This means all the values for a specific column are grouped together, rather than storing each row of data entirely. This approach is particularly beneficial for data warehouses where queries often access specific columns instead of entire rows. Columnar storage significantly reduces the amount of data that needs to be read for a query, improving performance and reducing I/O overhead.
2. **Compression Encodings:** Redshift utilizes compression encodings to further optimize storage space and improve query speeds. Compression applies algorithms to data within each column, reducing its physical size on disk. Redshift offers various compression encodings like LZ0, Zstandard, and its own AZ64 encoding specifically designed for numeric and date/time data types. The choice of encoding depends on the data type and the desired balance between compression ratio and decompression speed during queries.

Here are some additional points to consider:

- **Automated Compression:** Redshift can automatically choose a default compression encoding for new columns based on data types. This helps optimize storage utilization without sacrificing query performance.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **Manual Configuration:** You can also manually specify compression encodings for individual columns during table creation or modification to achieve the best fit for your data.
- **Trade-offs:** While compression saves storage space, it adds some overhead during data insertion and querying due to the decompression process. Finding the right balance between compression and performance is crucial.

By leveraging columnar storage and compression encodings, Amazon Redshift offers a cost-effective and performant solution for storing and analyzing large datasets.


Q4: Explain the purpose and benefits of using the COPY command in Redshift.

A: The COPY command in Redshift serves a vital purpose: efficiently loading large amounts of data into your Redshift tables. Here's why it's beneficial:

- **Bulk Data Loading:** COPY excels at transferring massive datasets from various sources like Amazon S3 buckets, EMR clusters, or even remote hosts accessible through SSH. This bulk loading capability is crucial for data warehouses that deal with significant data volumes.
- **Parallel Processing:** Redshift leverages its MPP architecture to execute the COPY command in parallel. This means data is loaded concurrently across all compute nodes in your cluster, significantly accelerating the loading process compared to traditional methods like individual INSERT statements.
- **Scalability:** The parallel nature of COPY makes it highly scalable. As your cluster size increases with more compute nodes, data loading speeds improve proportionally. This allows you to handle growing data volumes efficiently.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **Simplicity:** The COPY command syntax is relatively straightforward, making it easy to use and manage data loads. You can specify the data source, target table, and data format with minimal complexity.
- **Efficiency:** Compared to individual inserts, COPY optimizes data transfers by minimizing network overhead and maximizing data throughput. This translates to faster loading times and improved overall performance.

Overall, the COPY command is an indispensable tool for data engineers working with Redshift. It provides a fast, scalable, and efficient way to ingest large datasets into your data warehouse, enabling you to perform insightful data analysis and reporting tasks.

Q5: How can you improve query performance in Amazon Redshift?

A: Optimizing query performance is a crucial aspect of working with Amazon Redshift. Here are several techniques you can employ to achieve faster query execution:


1. Schema Design:

- **Denormalization:** For queries that frequently involve joins, strategically denormalizing tables can reduce the number of joins needed, leading to performance gains. However, this should be done with caution to avoid data redundancy and update anomalies.
- **Distribution Style and Sort Keys:** Redshift utilizes a distributed storage architecture. Choosing the optimal distribution style (e.g., sortkey distribution) and sort keys for your tables aligns data with how queries access it, minimizing data movement across nodes and improving performance.

2. Query Optimization:

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **EXPLAIN Command:** Utilize the EXPLAIN command to analyze the query execution plan and identify potential bottlenecks. This helps pinpoint areas for improvement, such as inefficient joins or missing indexes.
- **Materialized Views:** Pre-compute frequently used aggregates or joins by creating materialized views. These act like pre-calculated tables, reducing query execution time when the underlying data hasn't changed significantly.

3. Cluster Management:


- **Vacuum and Unload:** Regularly vacuuming tables reclaims unused space caused by deletes and updates. Unloading unused tables into S3 frees up cluster resources and optimizes performance.
- **Cluster Sizing:** Ensure your cluster size (number of compute nodes) aligns with your workload demands. An under-provisioned cluster can lead to slow queries, while an over-provisioned one incurs unnecessary costs. Consider using auto-scaling features to dynamically adjust cluster size based on workload fluctuations.

4. Advanced Techniques:

- **Redshift Spectrum:** If your data resides in Amazon S3, leverage Redshift Spectrum to query the data directly from the data lake without physically loading it into Redshift. This approach can be cost-effective for infrequently accessed data.
- **Partitioning for Redshift Spectrum:** Partition Glue catalog tables based on specific columns to restrict scans to relevant data segments, improving query performance for queries that target specific date ranges or other criteria.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

Note that Redshift spectrum supports partitioning whereas Redshift doesn't

Remember, optimizing query performance often involves an iterative process. Continuously monitor query execution times, identify bottlenecks, and implement the appropriate techniques to achieve optimal performance for your Redshift workloads.

Q6: What are Redshift Spectrum and its use cases?


A: Redshift Spectrum is a powerful feature of Amazon Redshift that allows you to query data directly from your Amazon S3 data lake without physically loading it into your Redshift cluster. This offers several advantages and unlocks new use cases for data analysis:

Benefits of Redshift Spectrum:

- **Cost-Effectiveness:** Store large, infrequently accessed data sets in S3's cost-optimized storage tiers. Redshift Spectrum allows you to query this data without incurring the cost of loading it into Redshift, making it a budget-friendly option for historical or archival data.
- **Scalability:** Redshift Spectrum leverages the virtually unlimited storage capacity of S3. You can analyze massive datasets that wouldn't fit within a traditional Redshift cluster, enabling broader data exploration.
- **Flexibility:** Redshift Spectrum supports various data formats commonly used in data lakes, such as Parquet, CSV, and ORC. This eliminates the need for data transformation before querying, simplifying your data pipeline.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **Faster Insights:** For queries that target specific subsets of data in S3, Redshift Spectrum can often retrieve results faster compared to loading the entire dataset into Redshift.

Use Cases for Redshift Spectrum:


- **Ad-hoc Analysis and Data Exploration:** Redshift Spectrum empowers data analysts and scientists to explore vast datasets in S3 for trends or insights without lengthy data loading processes.
- **Log Analysis:** Store and analyze large volumes of application logs or server logs directly in S3 using Redshift Spectrum. This enables efficient log analysis for troubleshooting, security monitoring, or identifying usage patterns.
- **Compliance Reporting:** For data that needs to be retained for legal or regulatory compliance purposes, Redshift Spectrum allows you to query archival data in S3 without incurring the ongoing costs of storing it in a Redshift cluster.
- **Data Warehousing with Historical Data:** Combine frequently accessed hot data stored in Redshift with historical or infrequently accessed cold data residing in S3. Redshift Spectrum seamlessly integrates these datasets for comprehensive data analysis.

Important Considerations:

- **Query Performance:** Redshift Spectrum queries might not always be as performant as querying data loaded directly into Redshift. Factors like data format, access patterns, and query complexity can affect performance.
- **Cost Optimization:** While S3 storage can be cost-effective, consider data transfer costs when querying data from S3 with Redshift Spectrum. Utilize efficient data partitioning and filtering techniques to minimize data scanned.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

Overall, Redshift Spectrum is a valuable tool for extending the capabilities of your Redshift data warehouse by enabling cost-effective and scalable querying of data directly from your S3 data lake. It opens doors for broader data exploration, log analysis, and historical data retention while optimizing storage costs.

Q7: Elaborate about Distribution keys and sort keys?

Redshift Distribution Keys and Sort Keys: Optimizing for Performance


In Amazon Redshift, distribution keys and sort keys play a crucial role in optimizing query performance. These keys determine how data is physically stored and accessed within your cluster, impacting how efficiently Redshift retrieves data for your queries.

1. Distribution Key (DISTKEY):

- **Function:** The distribution key specifies a column (or a set of columns) used to distribute data rows across the compute nodes in your Redshift cluster. It acts like a partitioning mechanism, ensuring rows with similar DISTKEY values reside on the same node or a small number of nodes.
- **Benefits:**
 - **Reduced Data Movement:** During query execution, Redshift primarily scans data on relevant nodes based on the DISTKEY. This minimizes data movement across the network, significantly improving query performance, especially for joins and aggregations that involve filtering based on the DISTKEY column(s).

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>


- **Parallel Processing:** Redshift can leverage its parallel processing architecture to execute queries on multiple nodes simultaneously when data is distributed efficiently using a proper DISTKEY.
- **Choosing a DISTKEY:**
 - Select a column that is frequently used in WHERE clauses or joins.
 - Ideally, the chosen column should have a high cardinality (number of distinct values) to ensure even distribution of data across nodes.
 - Avoid using low cardinality columns like unique identifiers, as they can lead to data skewing (uneven distribution) on a single node, impacting performance.

2. Sort Key (SORTKEY):

- **Function:** The sort key defines the order in which data rows within each node are physically stored. It acts like an ordering mechanism within each node's data partitions.
- **Benefits:**
 - **Faster Scans:** When a query involves filtering or sorting data based on the SORTKEY columns, Redshift can efficiently scan through the data in its pre-sorted order, minimizing the number of rows to be processed. This significantly improves query performance compared to scanning unsorted data.
 - **Range Queries:** Sort keys are particularly beneficial for queries that involve range filters or aggregations on the SORTKEY columns. Redshift can leverage the sorted order to quickly identify relevant data segments.
- **Choosing a SORTKEY:**

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- Consider the columns that are frequently used in ORDER BY clauses, WHERE clauses with range predicates (e.g., dates between a specific range), or group by clauses.
- You can define multiple columns in the SORTKEY, with the leading column having the most significant impact on sorting order.

Key Considerations:

- **Choosing the Right Keys:** Selecting optimal DISTKEY and SORTKEY combinations is crucial for performance. Analyze your workload patterns and identify the most frequently used columns in WHERE clauses, joins, and aggregations to guide your choices.
- **Trade-offs:** There can be trade-offs when choosing keys. For instance, a good DISTKEY might not be the best SORTKEY, and vice versa. It's essential to consider your specific query patterns to find the best balance.

By effectively utilizing distribution keys and sort keys, you can significantly improve the performance of your Redshift queries, enabling faster data retrieval and analysis.

Q8: Explain about the different Distribution Style that you have in Redshift?


Amazon Redshift offers four distribution styles for tables, providing flexibility to optimize data storage and query performance based on your workload:

1. Even Distribution:

- **Description:** This is the default style for new tables. Redshift distributes data in round-robin fashion across all compute nodes in the cluster, aiming for an even spread of data size.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **Benefits:**

- Simplicity: No need to define a distribution key.
- Might be suitable for small tables or tables with infrequent joins and low write volumes.

- **Drawbacks:**

- Potential performance overhead: Redshift may need to shuffle data across nodes during joins or aggregations if the relevant columns are not evenly distributed, impacting query speed.
- Not ideal for large tables or frequent joins: As data volume grows, even distribution can become inefficient due to potential data skewness on some nodes.

2. Key Distribution:

- **Description:** This style utilizes a user-defined distribution key (DISTKEY) to distribute data. Rows with the same DISTKEY value reside on the same node or a small set of nodes.


- **Benefits:**

- Improved join and aggregation performance: When the join or aggregation columns are chosen as the DISTKEY, Redshift minimizes data movement across nodes during these operations, leading to faster queries.
- Reduced data skewness: By using a high-cardinality column (many distinct values) as the DISTKEY, you can ensure even data distribution across nodes, preventing performance bottlenecks.

- **Drawbacks:**

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- Requires defining a DISTKEY: You need to carefully choose the appropriate column(s) based on your access patterns and query workloads.
- Might not be suitable for all workloads: If your queries rarely involve joins or aggregations, key distribution might not offer significant performance gains compared to even distribution.

3. All Distribution:


- **Description:** This style replicates the entire table data on every compute node in the cluster.
- **Benefits:**
 - Fast access for specific rows: Since the entire table resides on each node, Redshift can quickly locate rows without data shuffling.
 - Might be suitable for very small tables or infrequently accessed reference data.
- **Drawbacks:**
 - Storage inefficiency: This approach consumes significantly more storage space due to data redundancy.
 - Not ideal for large tables: As your data volume grows, all distribution becomes expensive due to storage overhead.
 - Increased management overhead: Managing and updating replicated data across nodes can be more complex.

4. Auto Distribution (AUTO):

- **Description:** This is a dynamic approach where Redshift automatically chooses the distribution style for your table. Internally, Redshift uses ALL distribution for small tables and switches to EVEN distribution as the table size grows.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **Benefits:**

- Simplicity: No need to explicitly define a distribution style during table creation.
- Adapts to changing data volume: Redshift automatically adjusts the distribution style as your table size increases, potentially improving efficiency.

- **Drawbacks:**

- Less granular control: You relinquish control over the initial distribution style, which might not be optimal for all scenarios.
- Performance considerations: While AUTO can be convenient, it might not always choose the best distribution style for your specific workload patterns.

Choosing the Right Distribution Style:

The optimal choice depends on factors like:


- **Table size:** For small tables, AUTO or EVEN distribution might suffice. For larger tables, consider KEY distribution.
- **Workload patterns:** If your queries involve frequent joins or aggregations, KEY distribution is generally preferred. For simple select queries, AUTO or EVEN might be adequate.
- **Access patterns:** Analyze how you access data. If specific columns are often used for filtering or joins, choose them as the DISTKEY.

While AUTO offers convenience, consider analyzing your workload and data access patterns to potentially gain performance benefits by explicitly defining the DISTKEY for key distribution. This allows for more granular control over data distribution and query optimization.

Q9: What are the use cases of Unloading the data from redshift?

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

Unloading data from Amazon Redshift serves various purposes in your data management workflows. Here are some key use cases:

1. Data Warehousing Pipelines:

- **Moving Data to Data Lakes:** Redshift is optimized for data warehousing and analytics, but not necessarily for raw data storage. You can unload data from Redshift to a data lake in Amazon S3 for long-term archiving, cost-effective storage of historical data, or further processing with big data frameworks like Spark or Hadoop.
- **Transferring Data to Other Warehouses:** If you need to migrate data to another data warehouse or analytical platform, unloading from Redshift to S3 provides a convenient intermediate storage location before loading it into the new destination.

2. Data Sharing and Collaboration:

- **Sharing Data with Analysts:** You can unload specific datasets from Redshift and share them with external analysts or collaborators who might not have direct access to your Redshift cluster. By storing the unloaded data in S3, you can grant controlled access to relevant personnel for further analysis or visualization using various tools.
- **Providing Data for Downstream Applications:** Some applications might require specific data subsets from Redshift for processing or reporting purposes. Unloading data to S3 allows these applications to access the required information efficiently without directly querying the Redshift cluster.

3. Data Backup and Disaster Recovery:

- **Creating Backups:** Regularly unloading critical data from Redshift to S3 creates backups that can be used for disaster recovery in case of unforeseen events like cluster failures. S3 offers high durability and

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

data redundancy, ensuring the availability of your backups when needed.

- **Archival for Compliance:** For regulatory compliance purposes, you might need to retain historical data for extended periods. Unloading data to S3's Glacier storage class provides a cost-effective archiving solution for compliance requirements.

4. Data Transformation and Enrichment:

- **Pre-processing for Advanced Analytics:** You can unload specific data sets from Redshift and perform additional transformations or enrichment processes on them using tools like AWS Glue or Spark in a separate environment before loading them back into Redshift or another data store. This approach can offload compute-intensive tasks from your Redshift cluster.
- **Data Lake Analytics:** By unloading data to S3, you can leverage the processing power of big data frameworks like Spark to perform complex analytics or machine learning tasks on the data using tools outside of Redshift, potentially achieving faster processing times for specific use cases.


Additional Considerations:

- **Cost Optimization:** Choose the optimal S3 storage class (Standard, Intelligent-Tiering, Glacier) based on your data access frequency and compliance needs to balance cost and accessibility.
- **Performance:** The unloading process itself consumes Redshift resources. Consider scheduling unload tasks during off-peak hours to minimize impact on query performance.

By understanding these use cases, you can effectively leverage the UNLOAD command in Redshift to manage your data effectively, integrate with broader data pipelines, and support various data sharing, archival, and transformation needs within your organization.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

Q10: What are the different ETL architectures involving Redshift?

ETL Architectures with Amazon Redshift:


Extracting, Transforming, and Loading (ETL) is a fundamental process for building data warehouses. Here are some common ETL architectures involving Amazon Redshift:

1. Vanilla ETL:

- **Description:** This is a traditional approach where data is extracted from source systems, transformed in a separate layer, and then loaded into Redshift.
- **Tools:** Extraction tools like AWS Glue or custom scripts can be used for data extraction. Transformation can be done in various tools like AWS Lambda, Spark, or custom code. Redshift's COPY command is used for data loading.
- **Benefits:**
 - Flexibility: Allows customization of each ETL stage using various tools.
 - Control: Provides granular control over the data transformation logic.
- **Drawbacks:**
 - Complexity: Can be complex to manage, especially for large-scale data pipelines.
 - Performance: Transformation in a separate layer might add processing overhead.

Sachin Chandrashekhar - Data Engineering Hub

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

2. ELT with Staging Area:

- **Description:** Similar to vanilla ETL, but introduces a temporary staging area (often in S3) to store the extracted data before transformation. Data is then loaded transformed and loaded into curated layer in S3 and also into Redshift tables using COPY command.
- **Benefits:**
 - Decoupling: Separates data extraction from transformation, allowing independent scaling and scheduling of each stage.
 - Flexibility: Staging area can hold data in its raw format, enabling flexible transformation logic to load to curated layer and to Redshift tables using COPY Command later.
- **Drawbacks:**
 - Increased complexity: Adds an additional layer to manage (staging area).
 - Storage costs: Storing raw data in the staging area can incur additional storage costs.

3. ELT with Redshift Spectrum:

- **Description:** Leverages Redshift Spectrum to directly query and transform data residing in your S3 data lake without physically loading it into Redshift.
- **Benefits:**
 - Cost-effective: Stores data in S3's cost-optimized storage classes and avoids unnecessary data movement.
 - Scalability: Enables querying and processing massive datasets directly from S3.

Sachin Chandrashekhar - Data Engineering Hub

🔗 LinkedIn: <https://www.linkedin.com/in/sachincw/>

🔗 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

- **Drawbacks:**

- Performance: Querying data from S3 might be slower compared to loading it directly into Redshift.
- Transformation limitations: Not all transformations can be efficiently performed using Redshift Spectrum.

Choosing the Right Architecture:

The optimal architecture depends on your specific needs:

- **Data Volume and Complexity:** For smaller ETL processes, vanilla ETL might suffice. For large-scale data with complex transformations, consider ELT with a staging area or managed services.
- **Performance Requirements:** If query performance within Redshift is critical, traditional ETL with in-cluster transformations might be preferable. Redshift Spectrum can be a good choice for cost-sensitive scenarios with acceptable query performance trade-offs.
- **Development and Maintenance Resources:** Managed ETL services can simplify development but might add costs. If you have the resources for custom development, vanilla ETL or ELT with a staging area offer greater flexibility.

By understanding these architectures and their trade-offs, you can make informed decisions on how to leverage Redshift as part of your overall data warehousing and analytics strategy

Sachin Chandrashekhar - Data Engineering Hub

🔗 LinkedIn: <https://www.linkedin.com/in/sachincw/>

🔗 AWS DE Program Waitlist: <https://waitlist.sachin.cloud>

Thank you so much for reading this document. I genuinely wish you all the best in your AWS Data Engineering Interview interviews.

- Sachin Chandrashekhar

Follow me on LinkedIn and click the bell 

 LinkedIn: <https://www.linkedin.com/in/sachincw/>

I conduct Real-world AWS Data Engineering (RADE) Programs.

Get on the waitlist

 AWS RADE Waitlist: <https://waitlist.sachin.cloud>

I also post updates regularly on

 WhatsApp Community:
<https://chat.whatsapp.com/FAqHgo4YpUsLFScpiMvtSF>

Look at other resources at:

 Top mate link: <https://lnkd.in/d28ETqaN>