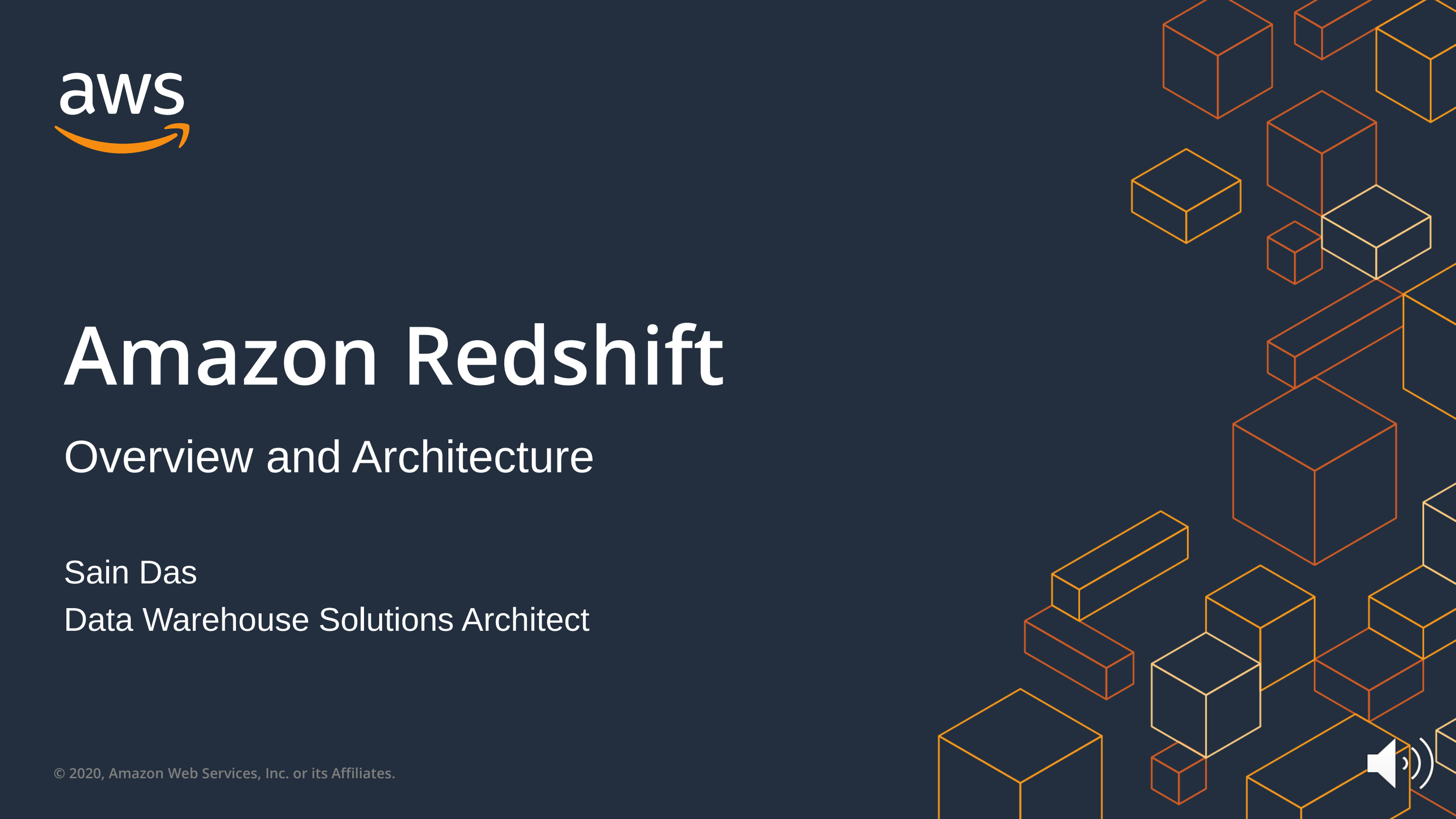


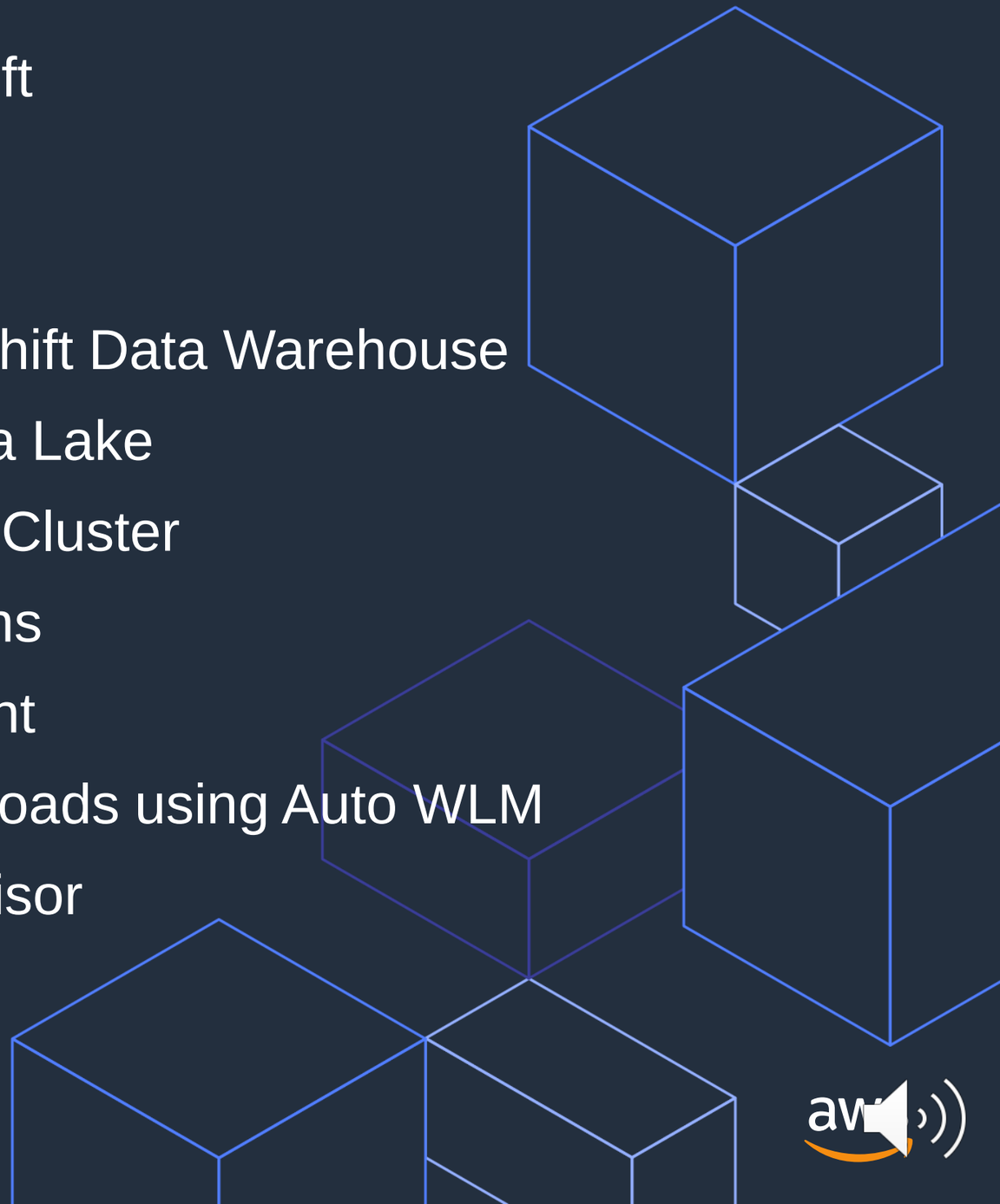
Amazon Redshift

Overview and Architecture

Sain Das
Data Warehouse Solutions Architect

Agenda

- Introduction to Redshift
- Redshift Architecture
- Key Concepts
- Demo: Create a Redshift Data Warehouse
- Redshift and The Data Lake
- Scaling your Redshift Cluster
- Data Ingestion Patterns
- Workload Management
- Demo: Manage Workloads using Auto WLM
- Amazon Redshift Advisor
- Additional Resources
- Q&A



Introduction to Redshift

Amazon Redshift

is the most popular cloud data warehouse

Amazon Redshift benefits

Tens of thousands of customers use Redshift & process over 2 EB of data per day

Data lake & AWS integrated

Lake Formation catalogue & security, Exabyte querying, AWS integrated (e.g., DMS, CloudWatch)

Best performance

Up to 3x faster than other cloud data warehouses

Best value

At least 50% less expensive than other solutions.

Most scalable

Virtually unlimited elastic linear scaling

Most secure & compliant

AWS-grade security, (e.g., VPC, encryption with KMS, Cloud Trail), Certifications such as SOC, PCI, DSS, ISO, FedRAMP, HIPAA

Easy to manage

Easy to provision & manage, automated backups, AWS support, 99.9% SLAs

Amazon Redshift has been innovating quickly

Robust result set caching

Large # of tables support ~20,000

Copy command support for ORC, Parquet

IAM role chaining

Elastic resize

Groups

Amazon Redshift Spectrum: date formats, scalar JSON and ION file formats support, region expansion, predicate filtering

Auto analyze

Health and performance monitoring w/Amazon CloudWatch

Automatic table distribution style

CloudWatch support for WLM queues

Performance enhancements: hash join, vacuum, window functions, resize ops, aggregations, console, union all, efficient compile code cache

Cost controls

Auto WLM

~25 query monitoring rules (QMR) support

AQUA (Advanced Query Accelerator)

Concurrency scaling

200+

DC1 migration to DC2

Resiliency of ROLLBACK processing

Manage multi-part query in AWS console

Auto analyze for incremental changes on table

new features in the past 18 months

Spectrum Request Accelerator

Apply new distribution key

Amazon Redshift Spectrum: Row group filtering in Parquet and ORC, Nested data support, enhanced VPC routing, multiple partitions

Faster classic resize with optimized data transfer protocol

Performance: Bloom filters in joins, complex queries that create internal table, communication layer

Amazon Redshift Spectrum: Concurrency scaling

Integration with AWS Lake Formation

Auto-vacuum sort, auto-analyze, and auto-table sort

Auto WLM with query priorities

Snapshot scheduler

Stored procedures

Performance: Join pushdowns to subquery, mixed workloads temporary tables, rank functions, null handling in join, single row insert

Advisor recommendations for distribution keys

AZ64 compression encoding

Console redesign

Spatial processing

Column level access control with AWS lake formation

RA3

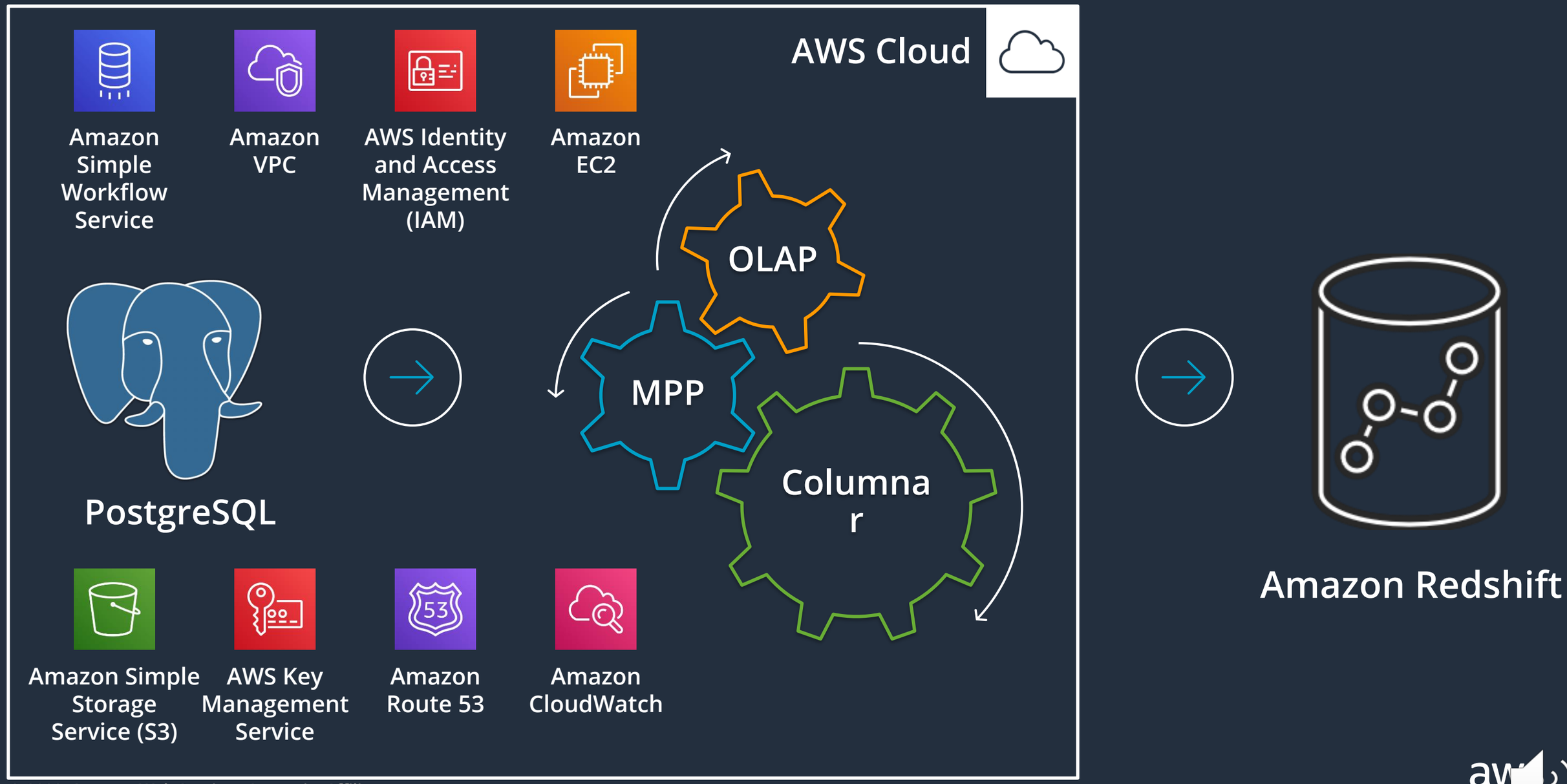
Performance of inter-region snapshot transfers

Federated Query

Materialized views

Pause and resume

Redshift Architecture



Redshift Node types

Amazon Redshift analytics—RA3

- Amazon Redshift Managed Storage (RMS)—Solid-state disks + Amazon S3

Dense compute—DC2

- Solid-state disks

Dense storage—DS2

- Magnetic disks

Instance type	Disk type	Size	Memory	CPUs	Slices
RA3 4xlarge	RMS	Scales to 64 TB	96 GB	12	4
RA3 16xlarge	RMS	Scales to 64 TB	384 GB	48	16
DC2 large	SSD	160 GB	16 GB	2	2
DC2 8xlarge	SSD	2.56 TB	244 GB	32	16
DS2 xlarge	Magnetic	2 TB	32 GB	4	2
DS2 8xlarge	Magnetic	16 TB	244 GB	36	16

Amazon Redshift Architecture – DC2 & DS2 Node Types

Massively parallel, shared nothing columnar architecture

Leader node

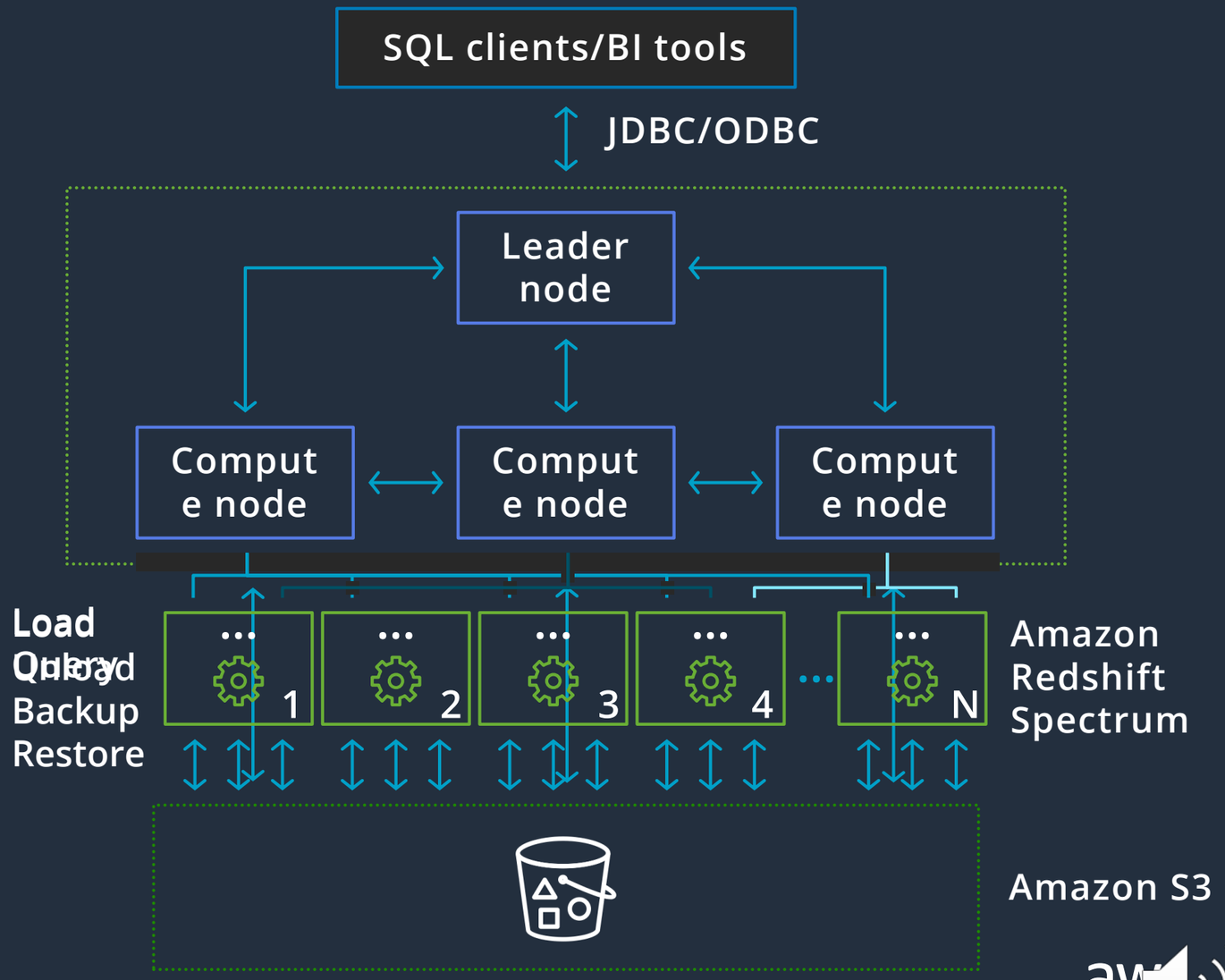
- SQL endpoint
- Stores metadata
- Coordinates parallel SQL processing

Compute nodes

- Local, columnar storage
- Executes queries in parallel
- Load, unload, backup, restore

Amazon Redshift Spectrum nodes

- Execute queries directly against Amazon Simple Storage Service (Amazon S3)



Amazon Redshift Architecture – RA3 Node Types

Massively parallel, columnar architecture

Leader node

Compute nodes

Amazon Redshift Spectrum nodes

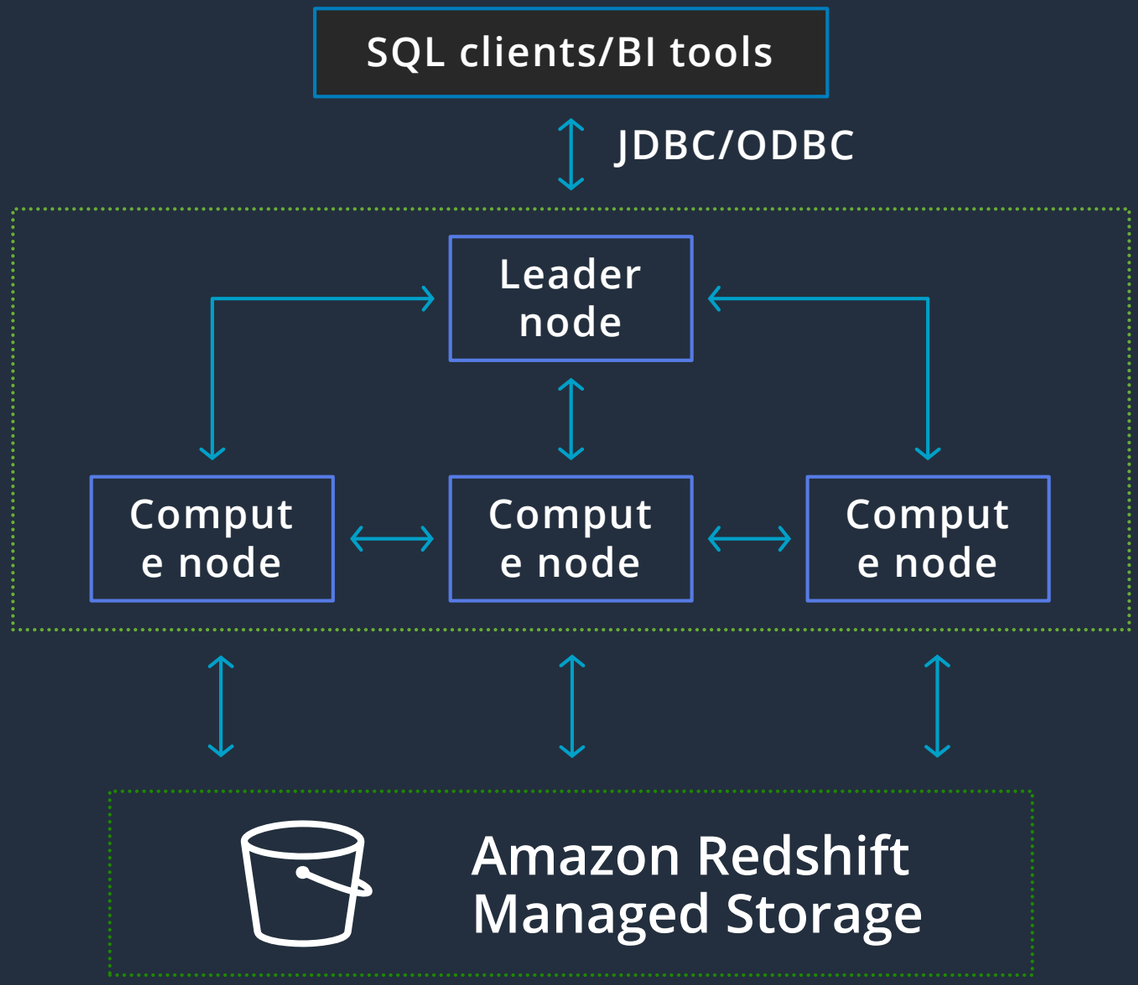
Amazon Redshift Managed Storage

Pay separately for storage and compute

Large high-speed SSD backed cache

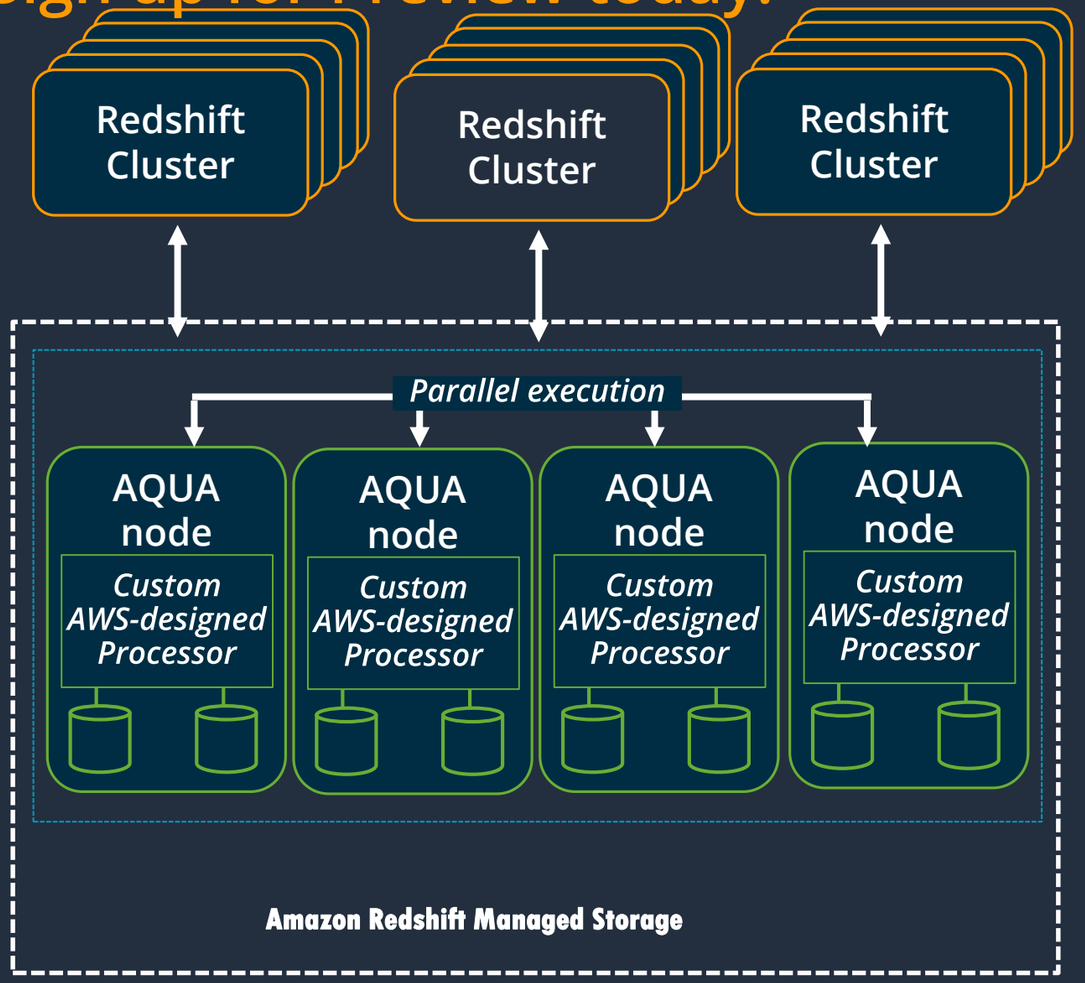
Automatic scaling (up to 64TB/instance)

Supports up to 8.2PB of cluster storage



AQUA (Advanced Query Accelerator) for Amazon Redshift

Sign up for Preview today!



New distributed & hardware-accelerated processing layer

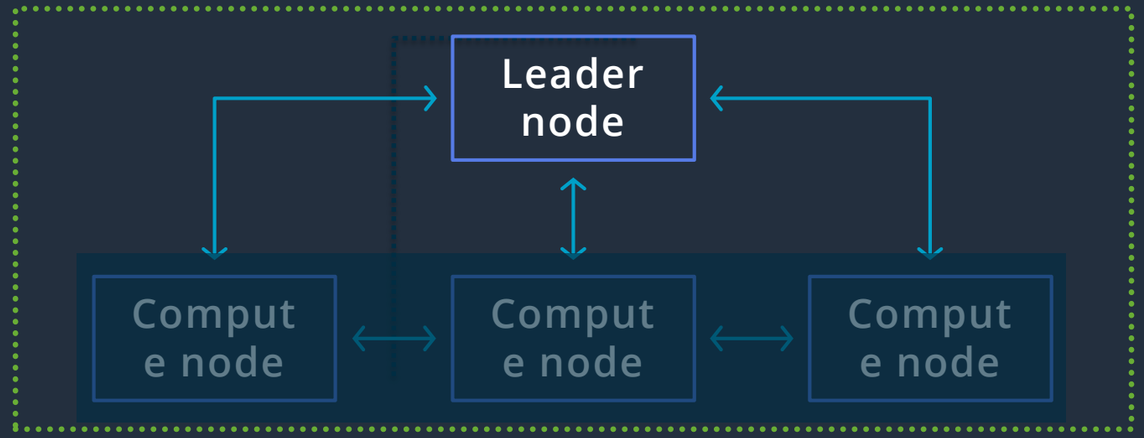
With AQUA, Amazon Redshift is up to 10x faster than any other cloud data warehouse, no extra cost

AQUA Nodes with custom AWS-designed analytics processors to make operations (compression, encryption, filtering, and aggregations) faster than traditional CPUs

Sign up for Preview with RA3. No code changes required

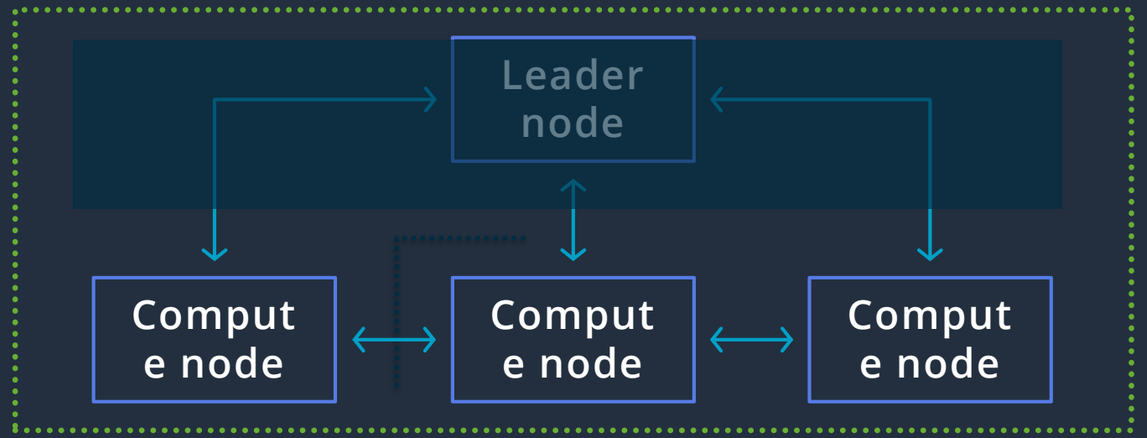
Leader Node – Under the hood

- The leader node is the endpoint to which all SQL connections to the Redshift data warehouse are made.
- It is responsible for query compilation and optimization and managing system metadata
- The leader node also coordinates parallel SQL processing and aggregates results from all the compute nodes before returning the final result set to the user.

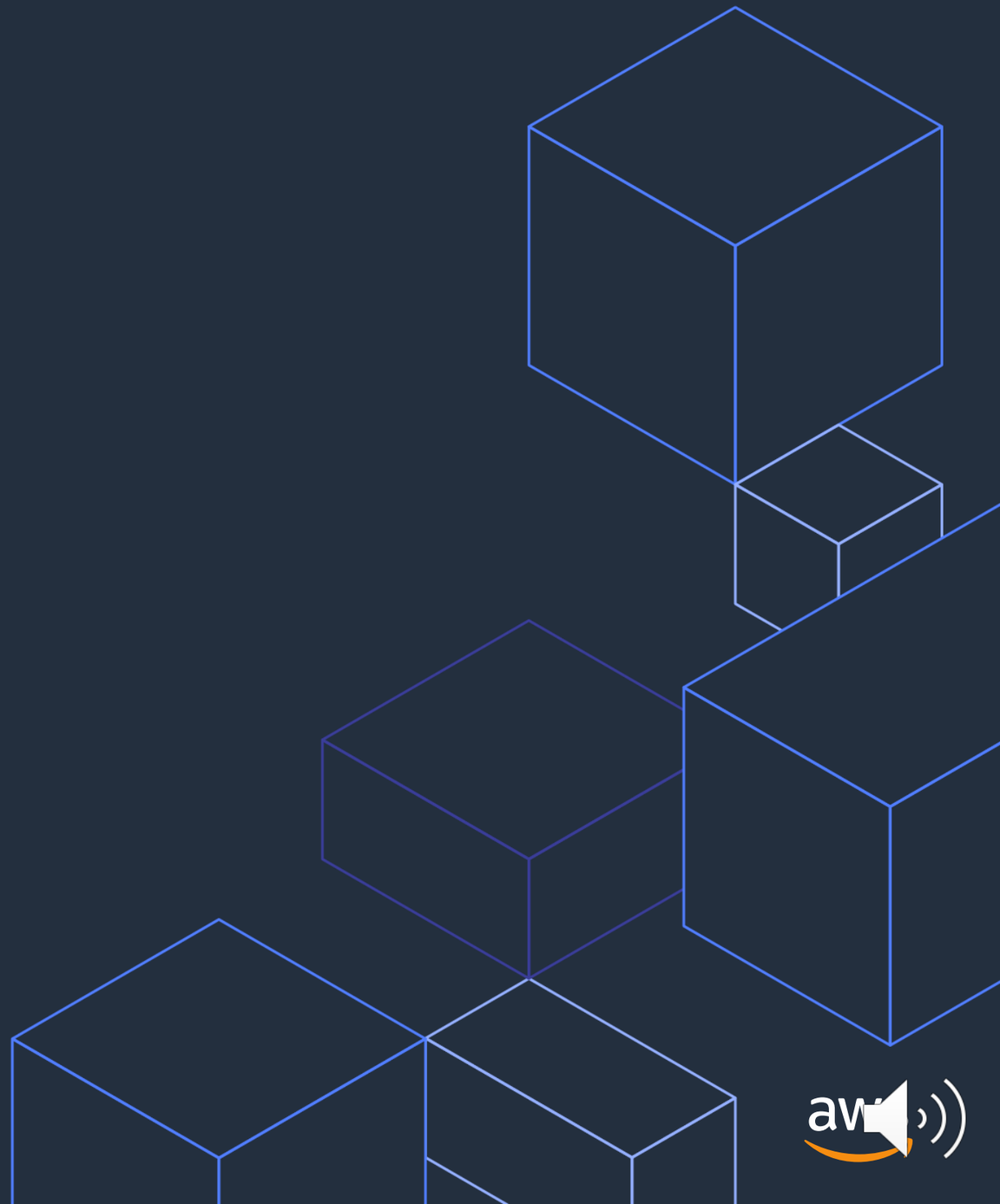


Compute Node – Under the hood (Slices)

- A compute node is partitioned into either 2 ,4 or 16 slices; a slice can be thought of as a “virtual compute node”
- Each slice is allocated a portion of the compute node’s resources, where it processes a portion of the workload assigned to the compute node by the leader node
- The leader node manages distributing data to the slices and apportions the workload for any queries or other database operations to the slices
- Slices are Redshift’s Symmetric Multiprocessing (SMP) mechanism – they work in parallel to complete operations



Key Concepts



Columnar Architecture

Amazon Redshift uses a columnar architecture for storing data on disk

Physically store data on disk by column rather than row

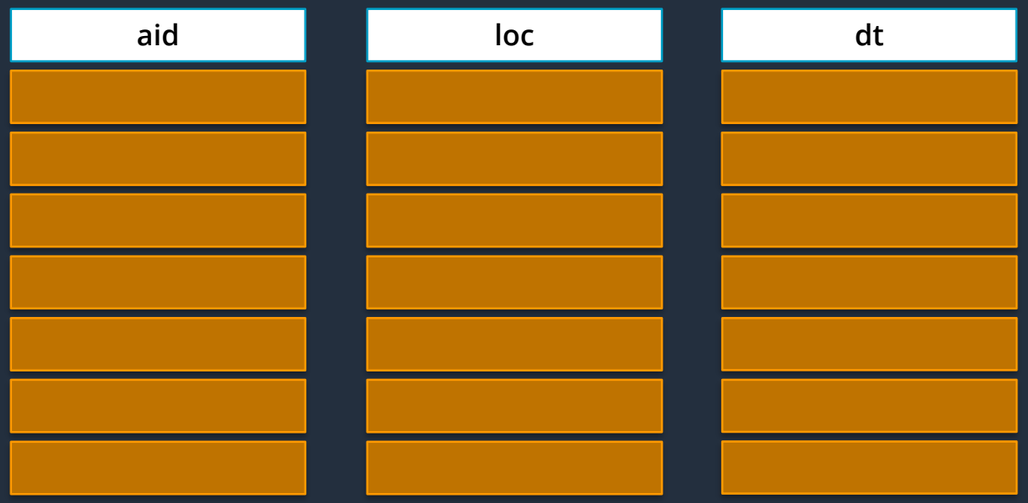
Only read the column data that is required

Goal: Reduce I/O for analytics queries

Columnar Architecture: Example

```
CREATE TABLE deep_dive (
     aid   INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14




```
SELECT min(dt) FROM deep_dive;
```

Row-based storage

- Need to read everything
- Unnecessary I/O

Columnar Architecture: Example

```
CREATE TABLE deep_dive (
     aid   INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14


```
SELECT min(dt) FROM deep_dive;
```

Column-based storage

- **Only scans blocks for relevant column**

Compression

Goals

Allow more data to be stored within an Amazon Redshift cluster

Improve query performance by decreasing I/O

Impact

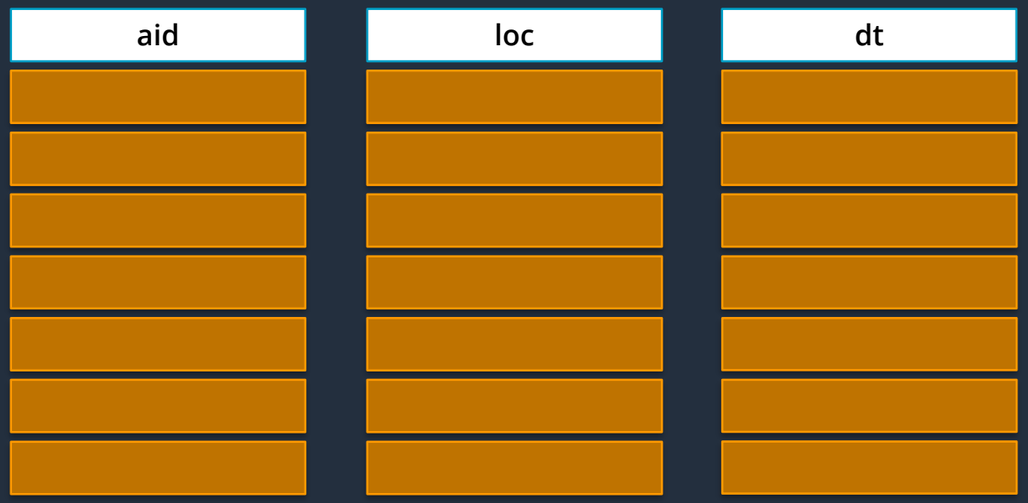
Allows two to four times more data to be stored within the cluster

ANALYZE COMPRESSION is a built-in command that will find the optimal compression for each column on an existing table

Compression: Example

```
CREATE TABLE deep_dive (
     aid   INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14

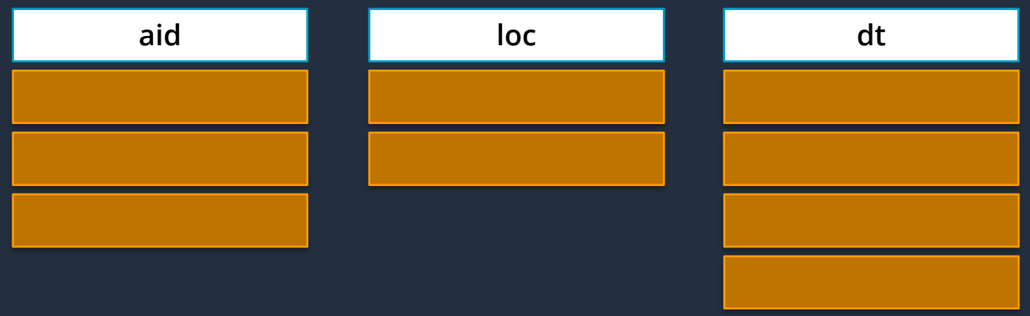


Add 1 of 13 different encodings to each column

Compression: Example

```
CREATE TABLE deep_dive (
     aid   INT         ENCODE AZ64
    ,loc   CHAR(3)     ENCODE BYTEDICT
    ,dt    DATE        ENCODE RUNLENGTH
);
```

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14



More efficient compression is due to storing the same data type in the columnar architecture

Columns grow and shrink independently

Reduces storage requirements

Reduces I/O

Amazon Redshift encoding type: AZ64

AZ64 is Amazon's proprietary compression encoding algorithm designed to achieve a high compression ratio and improved query processing

Goals:

- Increase compression ratio, reducing the required footprint
- Increase query performance by decreasing both encoding/decoding times

Result:

	AZ64 storage savings	AZ64 performance speed ups
RAW	60–70% less storage	25–30% faster
LZO	35% less storage	40% faster
ZSTD	Comparable footprint	70% faster

30TB Cloud DW benchmark is based on TPC-DS (v2.10) with no query modifications done

Best practices: Compression

Apply compression to all tables

- In most cases use AZ64 for INT, SMALLINT, BIGINT, TIMESTAMP, TIMESTAMPTZ, DATE, NUMERIC
- In most cases use LZO/ZSTD for VARCHAR and CHAR

Use ANALYZE COMPRESSION command to find optimal compression

- RAW (no compression) for sparse columns and small tables

Changing column encoding requires a table rebuild

- https://github.com/awslabs/amazon-redshift utils/tree/master/src/ColumnEncodingUtility

Verifying columns are compressed:

```
SELECT "column", type, encoding FROM pg_table_def
WHERE tablename = 'deep_dive';

 column |     type      | encoding
--------+---------------+----------
 aid    | integer       | az64
 loc    | character(3)  | bytedict
 dt     | date          | runlength
```

Data distribution

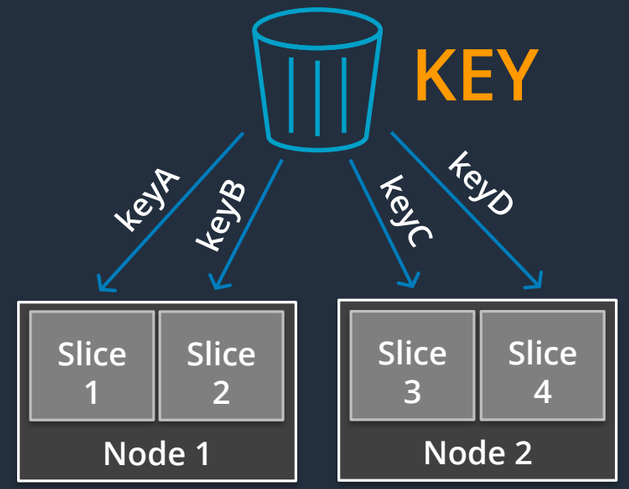
Distribution style is a table property which dictates how that table's data is distributed throughout the cluster

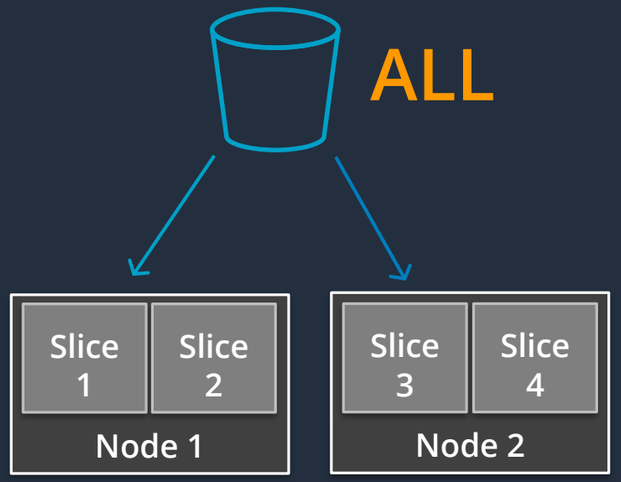
KEY: Value is hashed, same value goes to same location (slice)

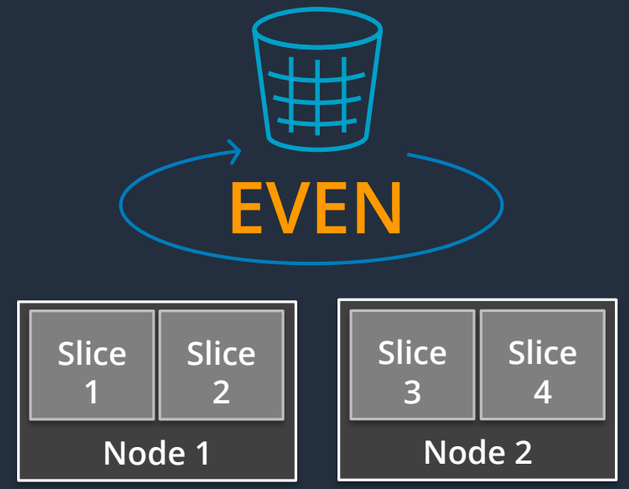
ALL: Full table data goes to the first slice of every node

EVEN: Round robin

AUTO: Combines EVEN and ALL







Goals

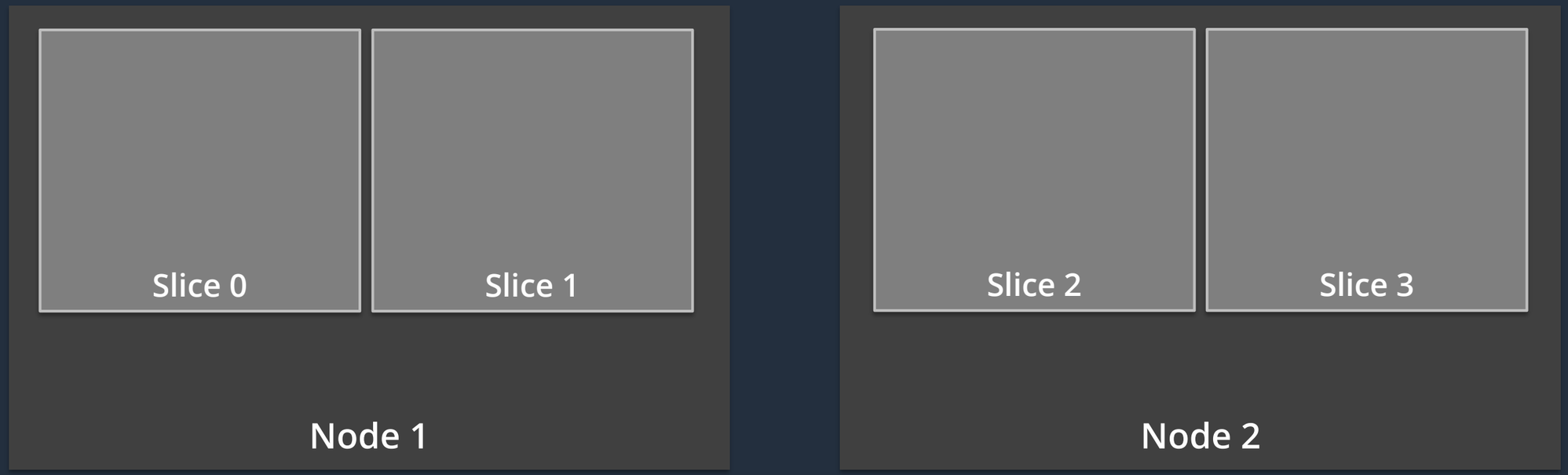
Distribute data evenly for parallel processing

Minimize data movement during query processing

Data distribution: Example

```
CREATE TABLE deep_dive (
    aid    INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) (EVEN|KEY|ALL|AUTO);
```

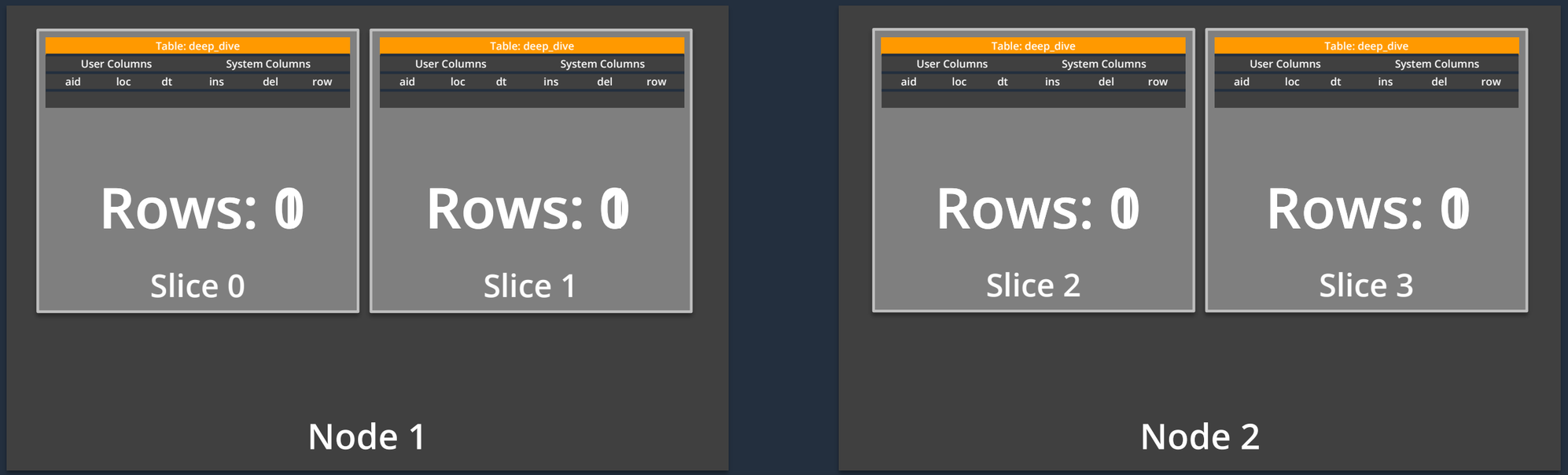
Table: deep_dive					
User columns			System columns		
aid	loc	dt	ins	del	row



Data distribution: EVEN Example

```
CREATE TABLE deep_dive (
    aid    INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) DISTSTYLE EVEN;
```

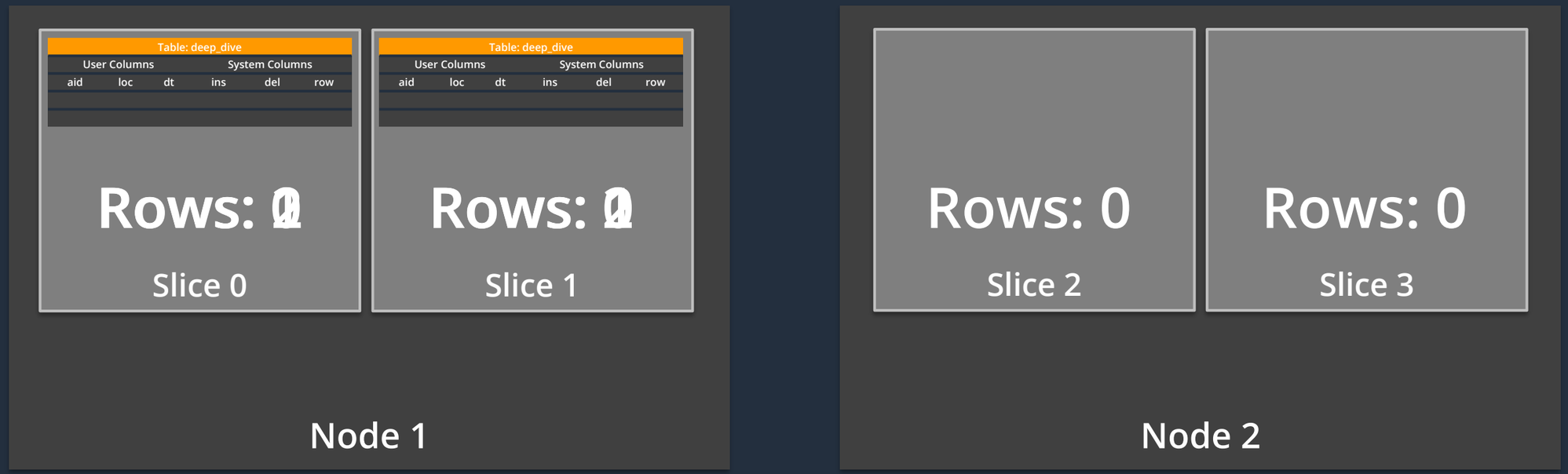
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution: KEY Example #1

```
CREATE TABLE deep_dive (
    aid    INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) DISTSTYLE KEY DISTKEY (loc);
```

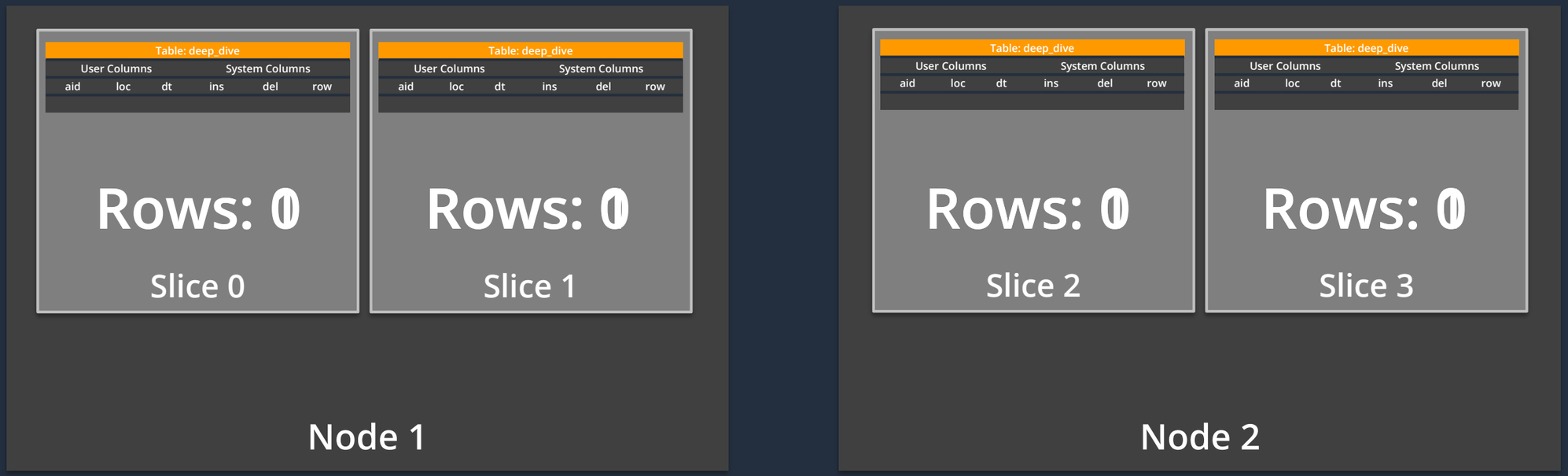
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```



Data distribution: KEY Example #2

```
CREATE TABLE deep_dive (
    aid    INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) DISTSTYLE KEY DISTKEY (aid);
```

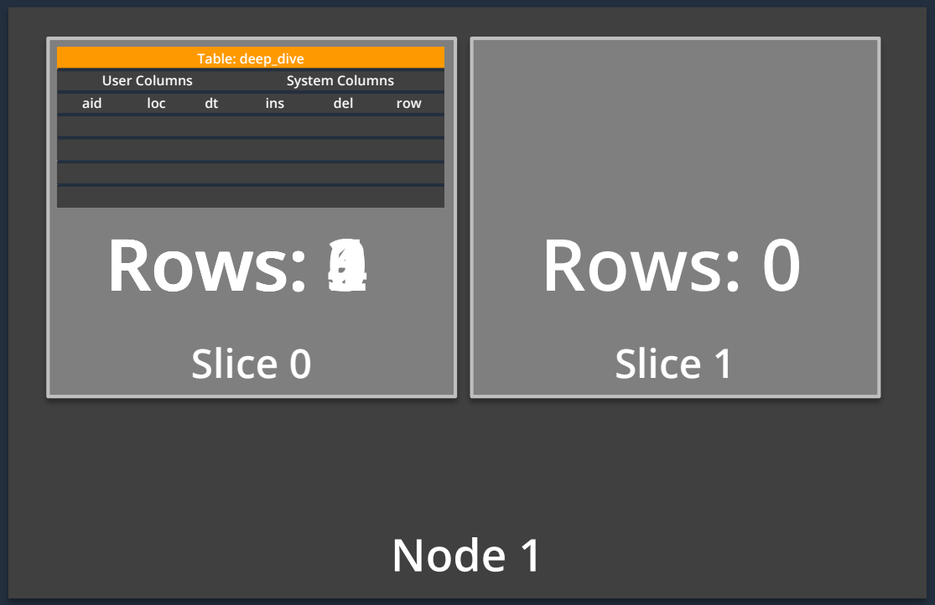
```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```

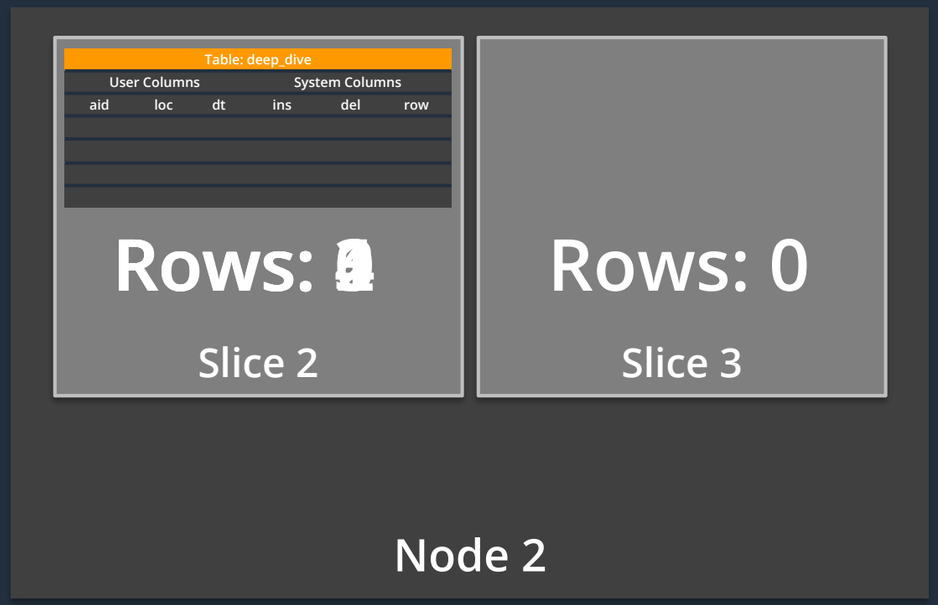


Data distribution: ALL Example

```
CREATE TABLE deep_dive (
    aid    INT         --audience_id
    ,loc   CHAR(3)     --location
    ,dt    DATE        --date
) DISTSTYLE ALL;
```

```
INSERT INTO deep_dive VALUES
(1, 'SFO', '2016-09-01'),
(2, 'JFK', '2016-09-14'),
(3, 'SFO', '2017-04-01'),
(4, 'JFK', '2017-05-14');
```





Summary: Data distribution

DISTSTYLE KEY

- **Goals**
  - Optimize **JOIN** performance between large tables by distributing on columns used in the **ON** clause
  - Optimize **INSERT INTO SELECT** performance
  - Optimize **GROUP BY** performance
- The column that is being distributed on should have a high cardinality and not cause row skew.

DISTSTYLE ALL

- **Goals**
  - Optimize **JOIN** performance with dimension tables
  - Reduces disk usage on small tables
- Small and medium size dimension tables (< 3M rows)

DISTSTYLE EVEN

- If neither **KEY** or **ALL** apply

DISTSTYLE **AUTO**

- Default distribution—combines DISTSTYLE **ALL** and **EVEN**

Sort Keys

Goal

Make queries run faster by increasing the effectiveness of zone maps and reducing I/O

Impact

Enables range-restricted scans to prune blocks by leveraging zone maps

---

Achieved with the table property SORTKEY defined on one or more columns

Optimal sort key is dependent on:

- Query patterns
- Business requirements
- Data profile

Sort keys: Example

```
CREATE TABLE deep_dive (
     aid   INT        --audience_id
    ,loc   CHAR(3)    --location
    ,dt    DATE       --date
) SORTKEY (dt, loc);
```

Add a sort key to one or more columns to physically sort the data on disk

deep_dive

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
3	SFO	2017-04-01
4	JFK	2017-05-14

deep_dive (sorted)

aid	loc	dt
3	SFO	2017-04-01
4	JFK	2017-05-14
2	JFK	2017-10-20
1	SFO	2017-10-20

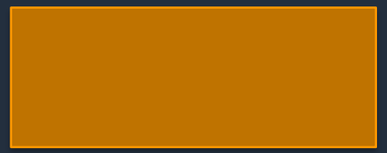
Zone maps and sorting: Example

```
SELECT count(*) FROM deep_dive WHERE dt = '06-09-2017';
```

Unsorted table

MIN: 01-JUNE-2017
MAX: 20-JUNE-2017

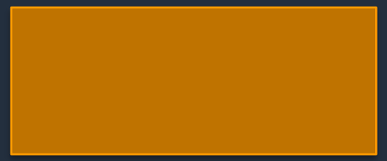
MIN: 08-JUNE-2017
MAX: 30-JUNE-2017



MIN: 12-JUNE-2017
MAX: 20-JUNE-2017

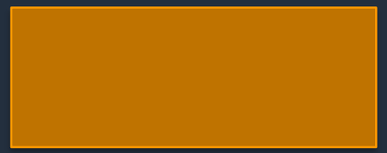
MIN: 02-JUNE-2017
MAX: 25-JUNE-2017

Sorted by date

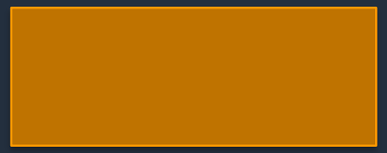


MIN: 01-JUNE-2017
MAX: 06-JUNE-2017

MIN: 07-JUNE-2017
MAX: 12-JUNE-2017



MIN: 13-JUNE-2017
MAX: 21-JUNE-2017



MIN: 21-JUNE-2017
MAX: 30-JUNE-2017

Best practices: Sort keys

Place the sort key on columns that are frequently filtered on placing the lowest cardinality columns first

- On most fact tables, the first sort key column should be a temporal column
- Columns added to a sort key after a high-cardinality column are not effective

With an established workload, use the following scripts to help find sort key suggestions:

- https://github.com/awslabs/amazon-redshift-utils/blob/master/src/AdminScripts/filter_used.sql
- https://github.com/awslabs/amazon-redshift-utils/blob/master/src/AdminScripts/predicate_columns.sql

Design considerations:

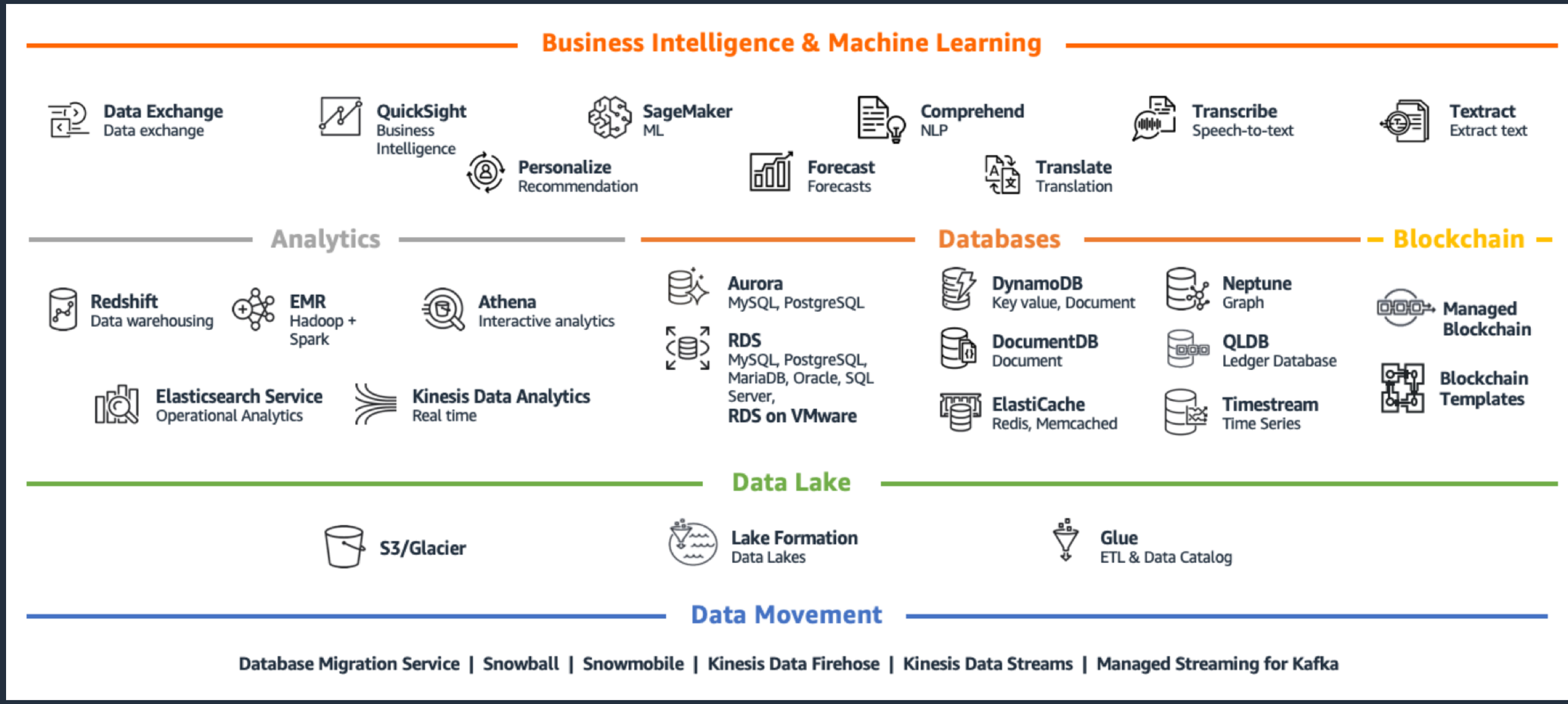
- Sort keys are less beneficial on small tables
- Define four or less sort key columns—more will result in marginal gains and increased ingestion overhead

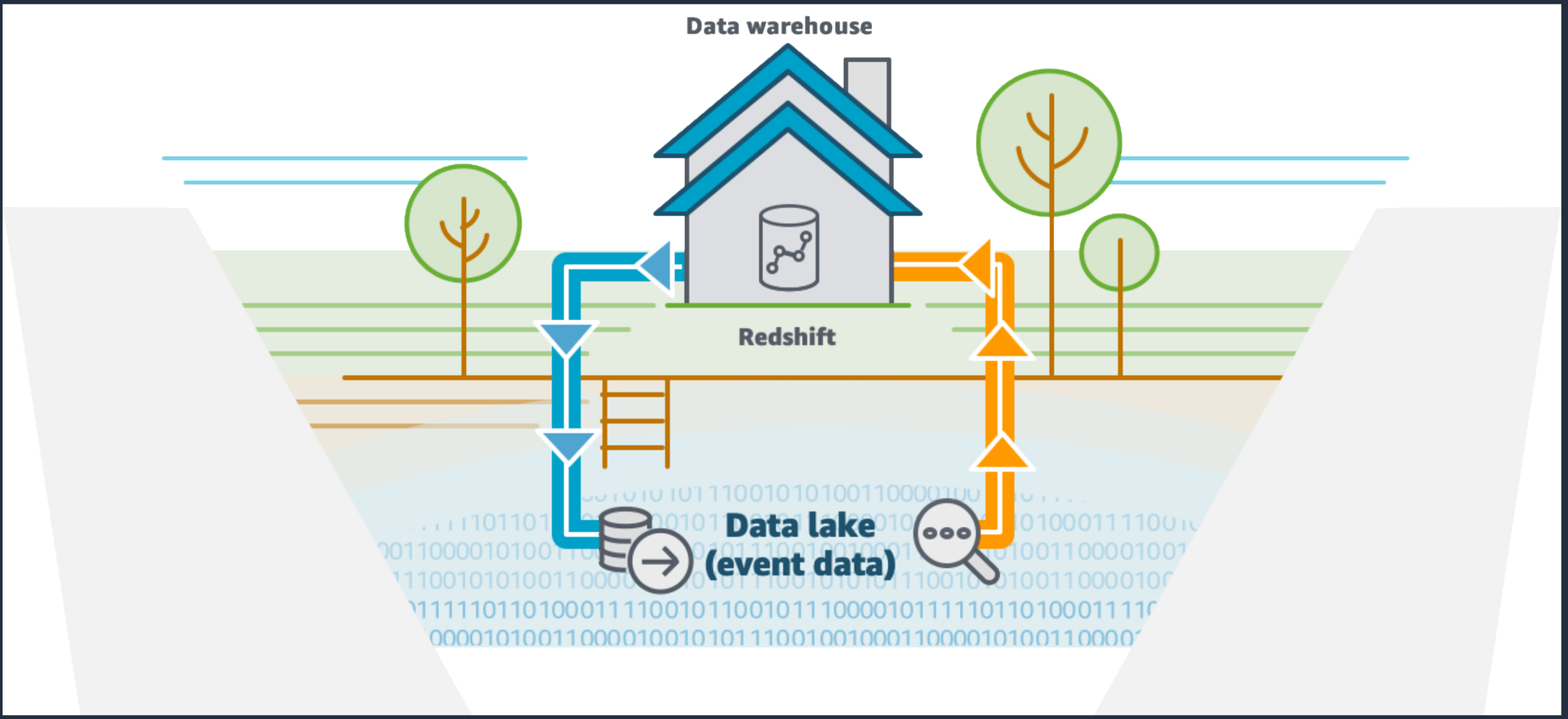
Demo: Create a Redshift Data Warehouse

Redshift and the Data Lake

Amazon Redshift is part of a complete portfolio

Broad and deep portfolio, purpose-built for builders





Customers are moving to data lake architectures

Redshift enables you to have a lake house approach

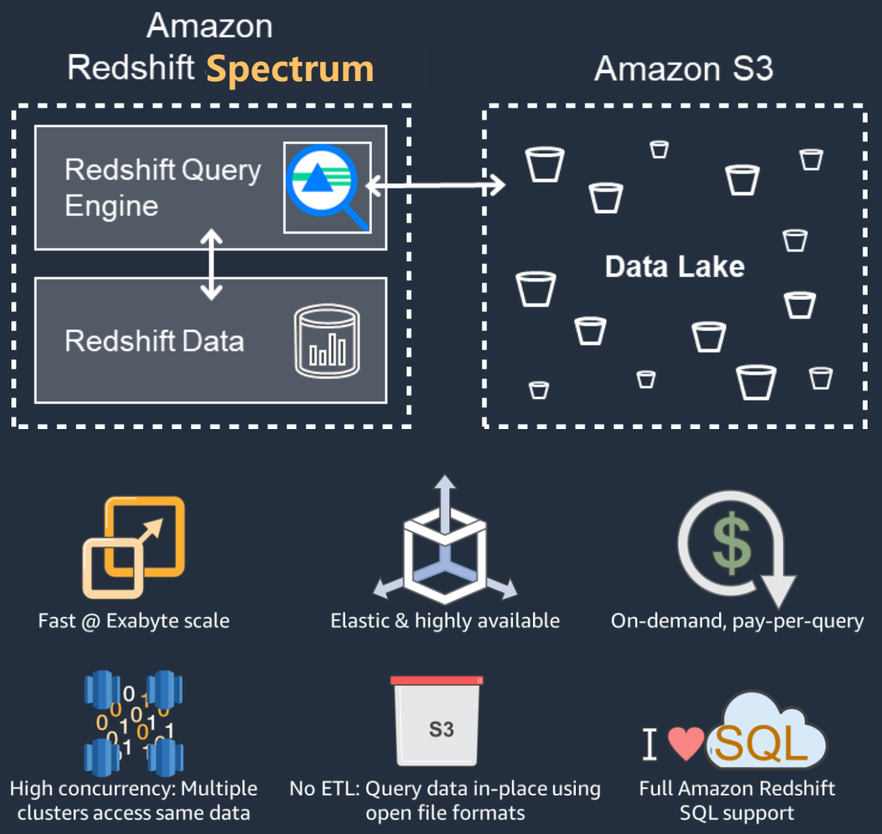
Redshift Spectrum Overview

Redshift Spectrum is a feature of Redshift that allows Redshift SQL queries to reference external data on Amazon S3 as they would any other table in Amazon Redshift

Benefits

- Enables the Lake House pattern out-of-the-box
- Allows for querying of potentially exabytes of data in an S3 data lake from within Amazon Redshift
- Data is queried in-place, so no loading of data into your Redshift cluster is required
- Keeps your data warehouse lean by ingesting warm data locally while keeping other data in the data lake within reach
- Powered by a separate fleet of powerful Amazon Redshift Spectrum nodes

Run SQL queries directly against data in S3 using thousands of nodes



Life of a Redshift Spectrum Query

1. Query
```
select count(*)
from S3.EXT_TABLE
where col_1= 'ABC';
```

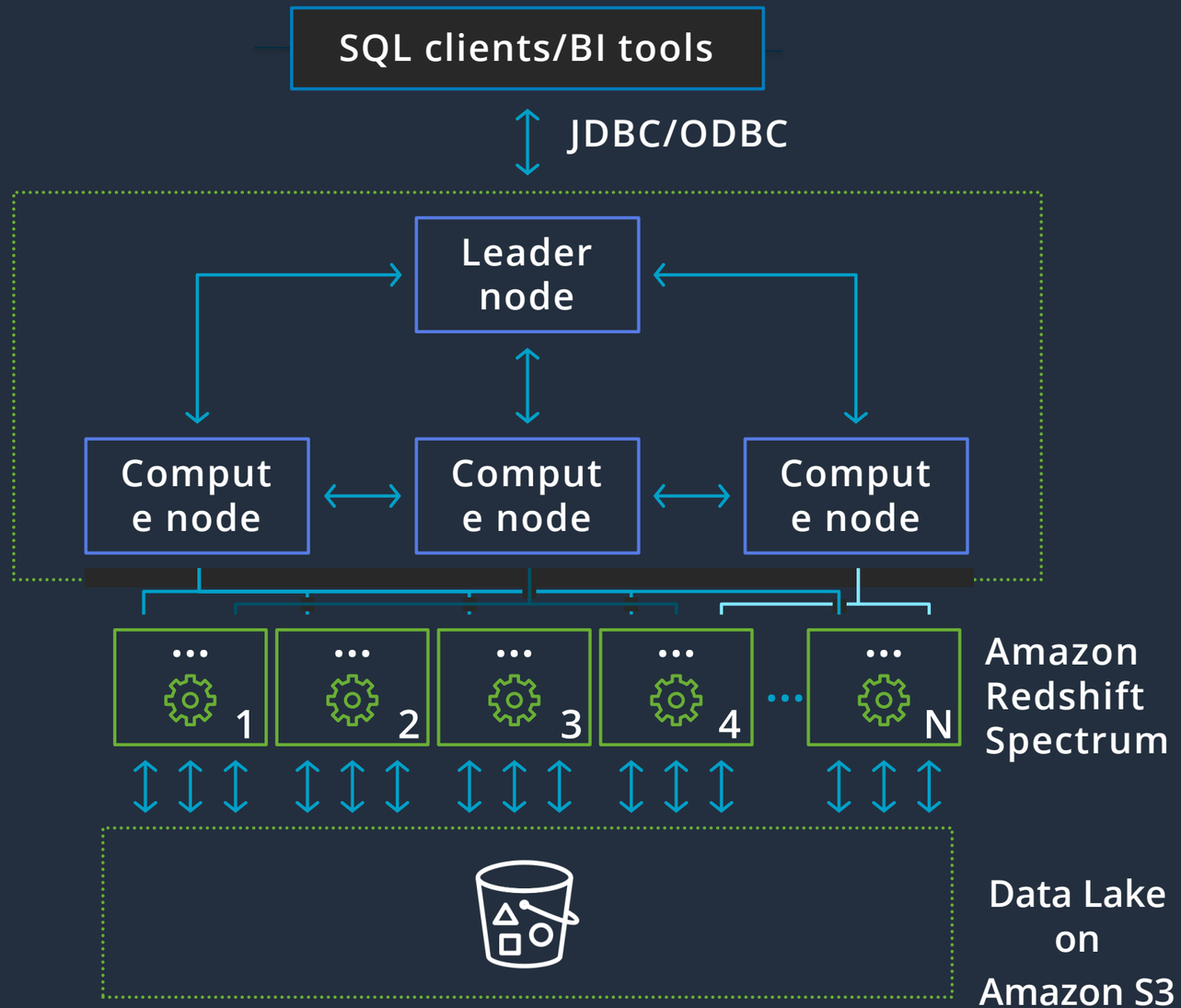
2. Query is optimized and compiled by the leader node and determines what gets run locally and what goes to Amazon Redshift Spectrum

3. Query plan is sent to all compute nodes

4. Compute nodes obtain partition info from Data Catalog; dynamically prune partitions

5. Each compute node issues multiple requests to the Amazon Redshift Spectrum layer

6. Amazon Redshift Spectrum nodes scan your S3 data



7. Amazon Redshift Spectrum projects, filters, joins and aggregates

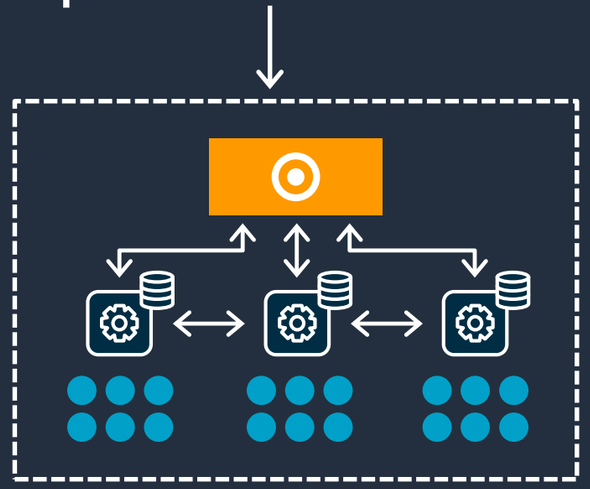
8. Final aggregations and joins with local Amazon Redshift tables done in-cluster

9. Result is sent back to client

Scaling your Redshift Cluster

Concurrency scaling

Amazon Redshift automatically adds transient clusters,
in seconds, to serve sudden spike in concurrent requests with consistently fast performance



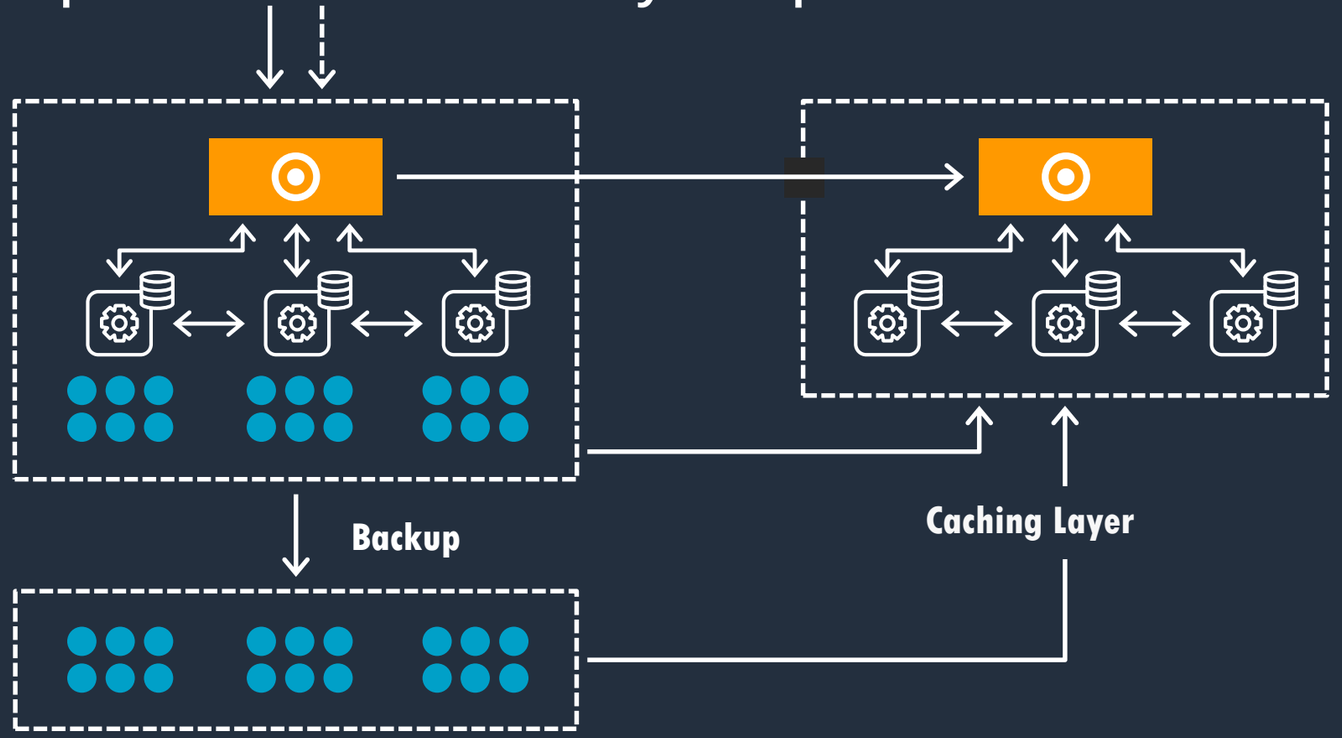
For every 24 hours that your main cluster is in use, you accrue a one-hour credit for Concurrency Scaling. This means that Concurrency Scaling is free for > 97% of customers.

How it works:

1. All queries go to the leader node, user only sees less wait for queries
2. When queries in designated WLM queue begin queuing, Amazon Redshift automatically routes them to the new clusters, enabling Concurrency Scaling automatically
3. Amazon Redshift automatically spins up a new cluster, processes waiting queries and automatically shuts down the Concurrency Scaling cluster

Concurrency scaling

Amazon Redshift automatically adds transient clusters,
in seconds, to serve sudden spike in concurrent requests with consistently fast performance



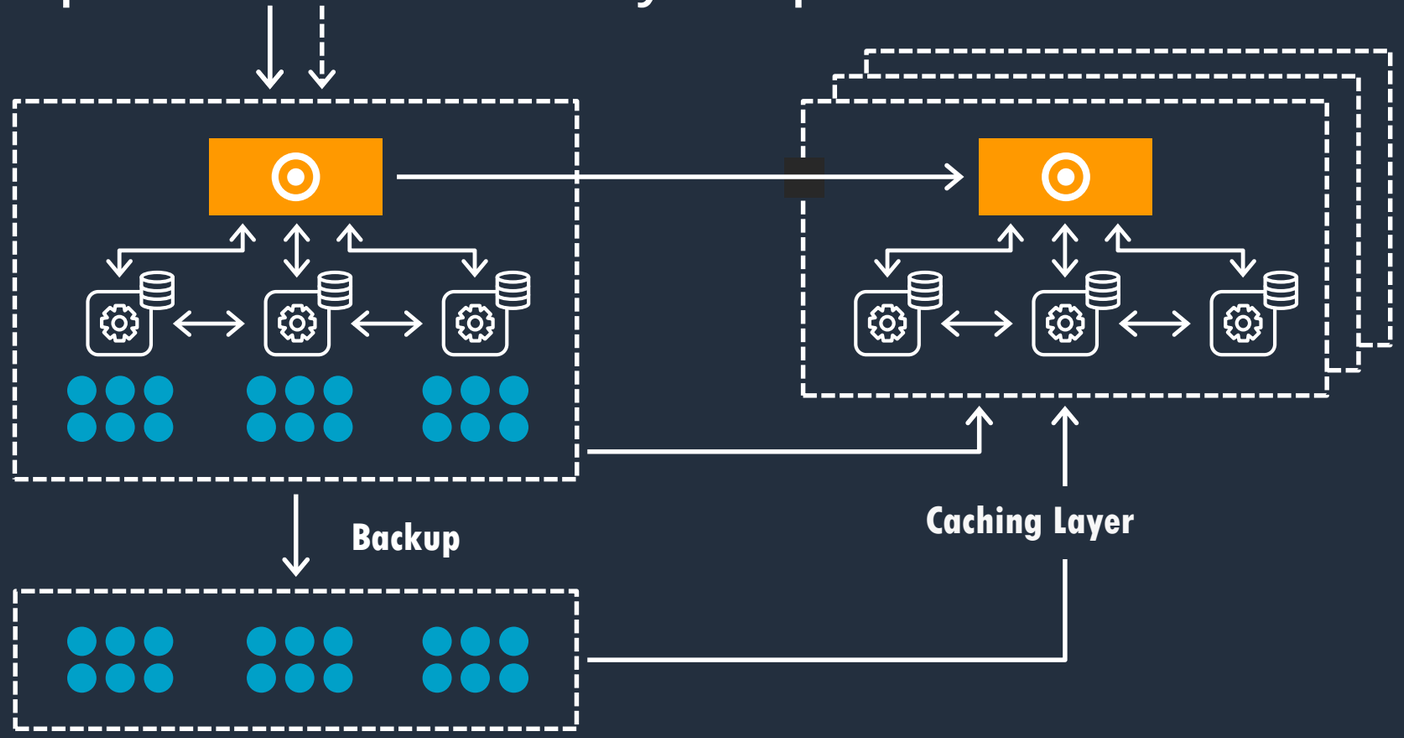
For every 24 hours that your main cluster is in use, you accrue a one-hour credit for Concurrency Scaling. This means that Concurrency Scaling is free for > 97% of customers.

How it works:

1. All queries go to the leader node, user only sees less wait for queries
2. When queries in designated WLM queue begin queuing, Amazon Redshift automatically routes them to the new clusters, enabling Concurrency Scaling automatically
3. Amazon Redshift automatically spins up a new cluster, processes waiting queries and automatically shuts down the Concurrency Scaling cluster

Concurrency scaling

Amazon Redshift automatically adds transient clusters,
in seconds, to serve sudden spike in concurrent requests with consistently fast performance



For every 24 hours that your main cluster is in use, you accrue a one-hour credit for Concurrency Scaling. This means that Concurrency Scaling is free for > 97% of customers.

How it works:

1. All queries go to the leader node, user only sees less wait for queries
2. When queries in designated WLM queue begin queuing, Amazon Redshift automatically routes them to the new clusters, enabling Concurrency Scaling automatically
3. Amazon Redshift automatically spins up a new cluster, processes waiting queries and automatically shuts down the Concurrency Scaling cluster

Resizing Amazon Redshift

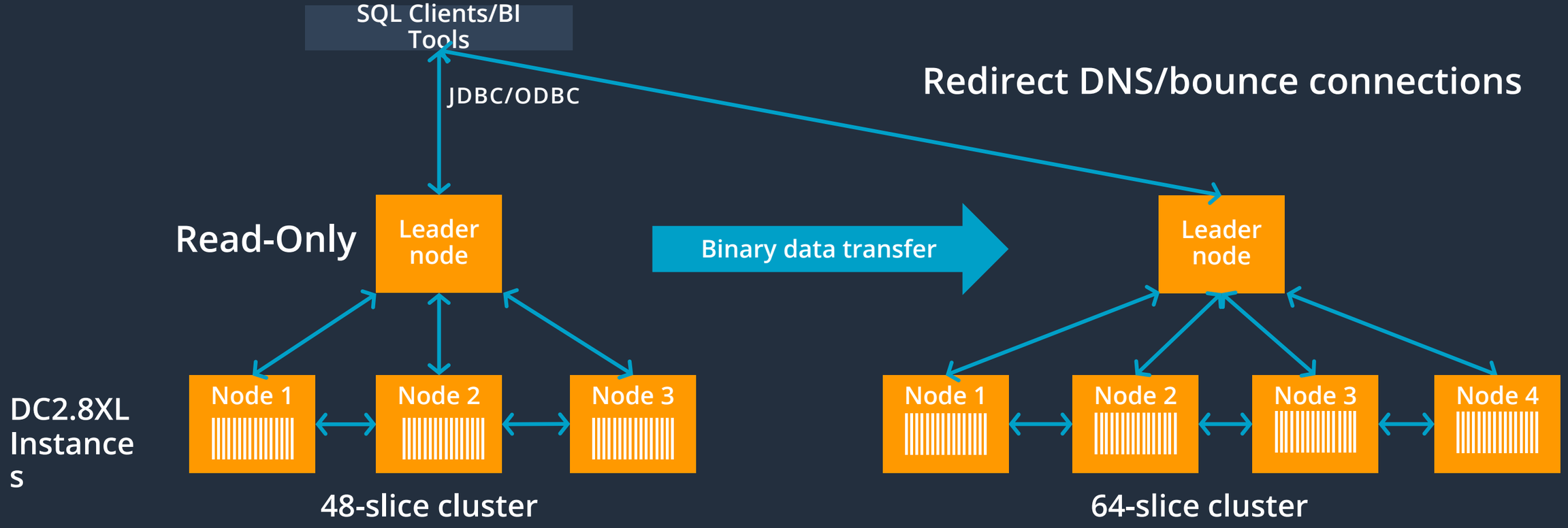
Classic resize

- Data is transferred from old cluster to new cluster (within hours)

Elastic resize

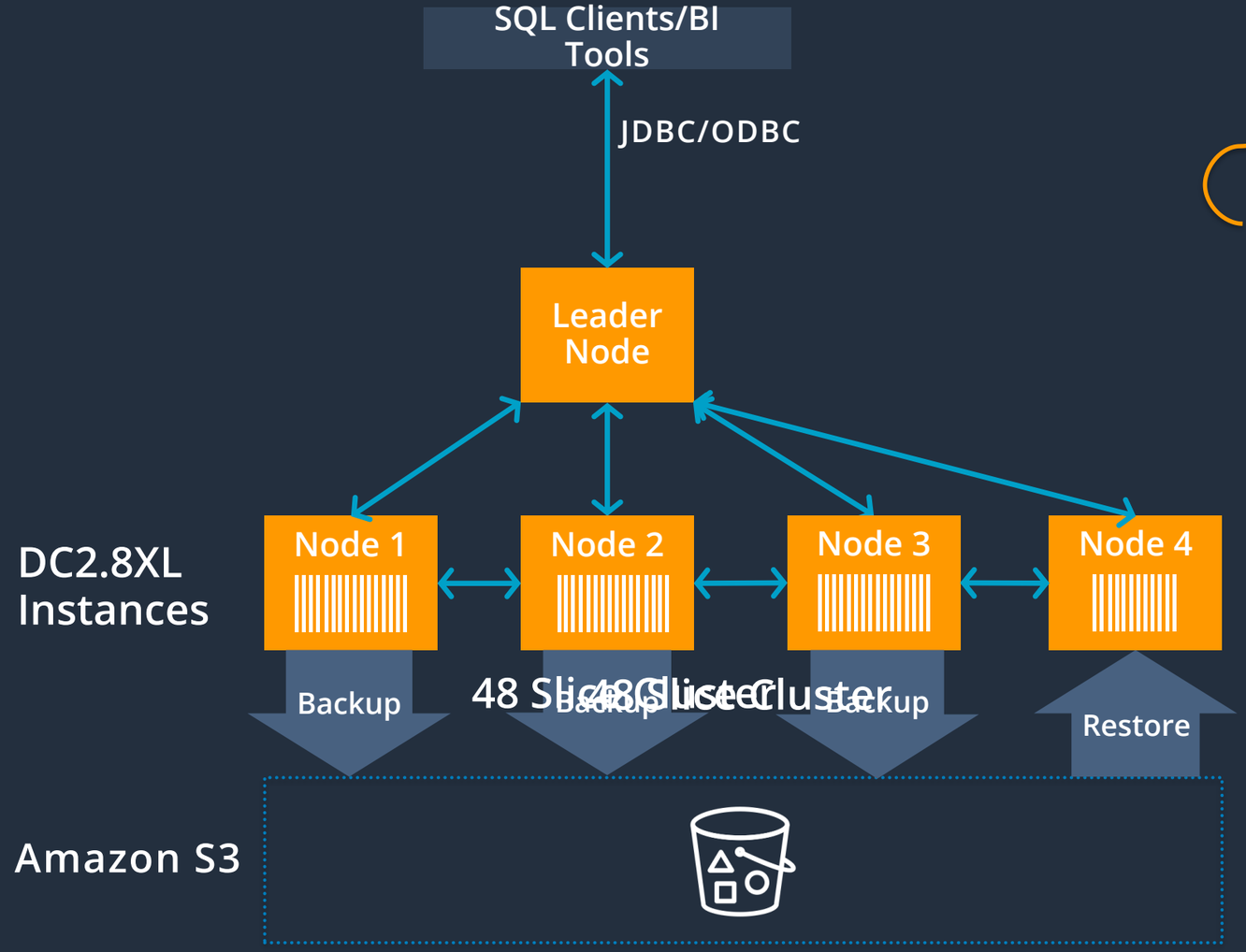
- Nodes are added/removed to/from existing cluster (within minutes)

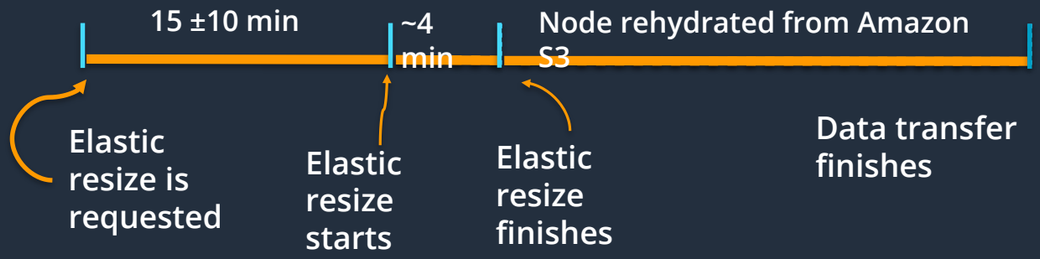
Classic resize



- Source cluster is placed into read-only mode during resize
- All data is copied and redistributed on the target cluster
- Allows for changing node types

Elastic resize





- Cluster is read only while we take a snapshot to Amazon S3 and provision the new node(s)
- Incoming queries/connections are put on hold
- Data blocks are moved first
- Cluster is fully available for reads and writes
- Some queries within transactions maybe rolled back

Elastic resize node increments

Instance type	Allowed increments	Max change from original size	Example: valid sizes for 4-node cluster
DC2 large DS2 xlarge	2x or ½ original cluster size only	Double, ½ size	2, 4, 8
RA3 4xlarge RA316xlarge DC2 8xlarge DS2 8xlarge	Can allow ± single node increments so long as slices remain balanced	Double, ½ size	2, 3, 4, 5, 6, 7, 8

When to use elastic vs. classic resize

Scenario	Elastic resize	Classic resize
Scale up and down for workload spikes	✓	
Incrementally add/remove storage	✓	
If elastic resize is not an option because of sizing limits		✓
Limited availability during resize	< 5 minutes (parked connections)	1–24 hours (read-only)

Data Ingestion Patterns

Data ingestion: COPY statement

Use COPY to load large volumes of records into Redshift tables.

COPY command supports a wide variety of formats including CSV, ORC and Parquet

There are other parameters you can specify in your command to allow invalid characters, perform compression analysis etc.

Data ingestion: COPY statement

Ingestion throughput

Each slice's query processors can load one file at a time:

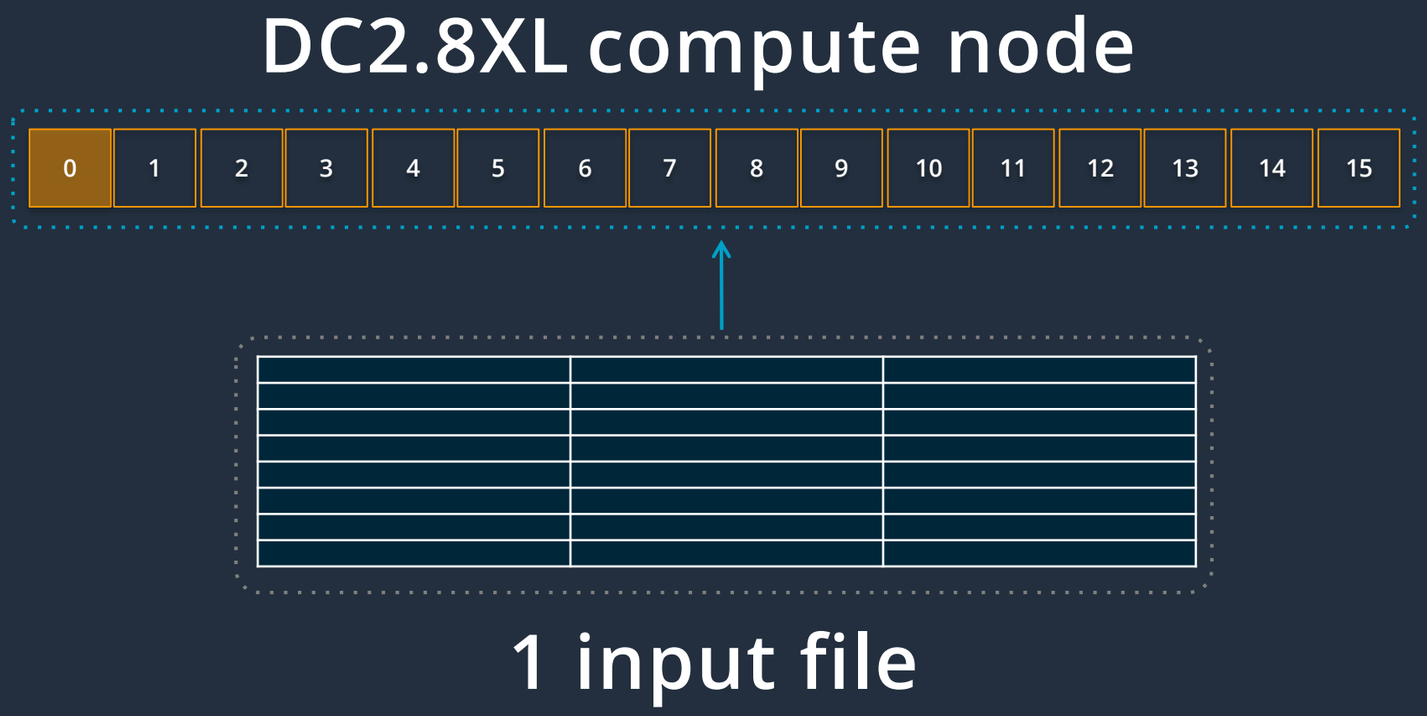
Streaming decompression

Parse

Distribute

Write

Realizing only partial node usage as 6.25% of slices are active

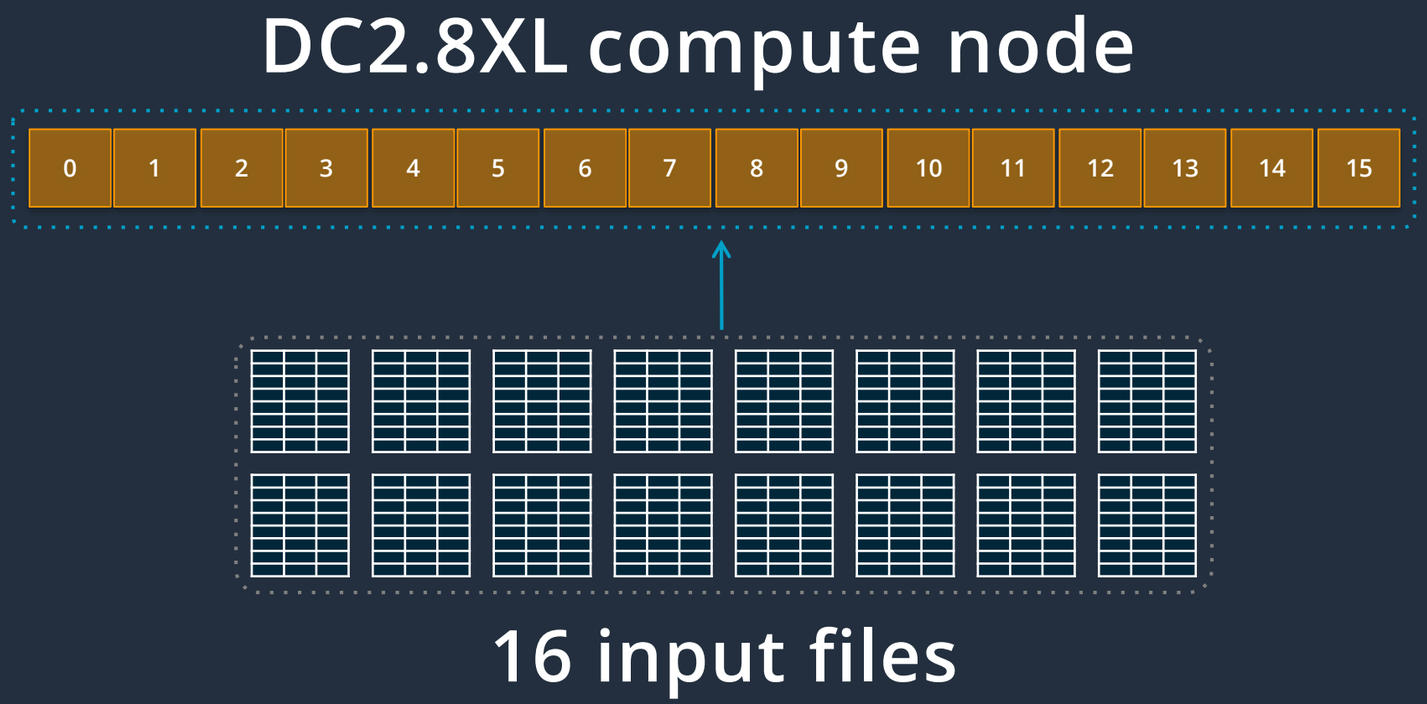


Data ingestion: COPY statement

Number of input files should be a multiple of the number of slices

Splitting the single file into 16 input files, all slices are working to maximize ingestion performance

COPY continues to scale linearly as you add nodes



Recommendation is to use delimited files—1 MB to 1 GB after gzip compression

Best practices: COPY ingestion

Delimited files are recommended

- Pick a simple delimiter '|' or ',' or tabs
- Pick a simple NULL character (\N)
- Use double quotes and an escape character (' \ ') for varchars
- UTF-8 varchar columns take four bytes per char

Split files into a number that is a multiple of the total number of slices in the Amazon Redshift cluster

- `SELECT count(slice) from stv_slices;`

Files sizes should be 1 MB to 1 GB after gzip compression

Data ingestion: Amazon Redshift Spectrum

Use INSERT INTO SELECT from external Amazon S3 tables

- Aggregate incoming data
- Select subset of columns and/or rows
- Manipulate incoming column data with SQL
- **Load data in alternative file formats: Amazon ION, Grok, RCFile, and Sequence**

Best practices

- Save cluster resources for querying and reporting rather than on ELT
- Filtering/aggregating incoming data can improve performance over COPY

Design considerations

- $5/TB of (compressed) data scanned

Design considerations: Data ingestion

Designed for large writes

- Batch processing system, optimized for processing massive amounts of data
- 1 MB size plus immutable blocks means that we clone blocks on write so as not to introduce fragmentation
- Small write (~1-10 rows) has similar cost to a larger write (~100K rows)

UPDATE and DELETE

- Immutable blocks means that we only logically delete rows on UPDATE or DELETE

Best practices: ELT

Wrap workflow/statements in an explicit transaction

Consider using DROP TABLE or TRUNCATE instead of DELETE

Staging tables:

- Use temporary table or permanent table with the "BACKUP NO" option
- If possible use DISTSTYLE KEY on both the staging table and production table to speed up the INSERT INTO SELECT statement
- With COPY turn off automatic compression—COMPUPDATE OFF
- Copy compression settings from the production table (using LIKE keyword) or manually apply compression to CREATE TABLE DDL (from ANALYZE COMPRESSION output)
- For copying a large number of rows (> hundreds of millions) consider using ALTER TABLE APPEND instead of INSERT INTO SELECT

(AUTO) VACUUM

The VACUUM process runs either manually or automatically in the background

Goals

VACUUM will remove rows that are marked as deleted

VACUUM will globally sort tables

- For tables with a sort key, ingestion operations will locally sort new data and write it into the unsorted region

Best practices

- VACUUM should be run only as necessary
- For the majority of workloads AUTO VACUUM DELETE will reclaim space and AUTO TABLE SORT will sort the needed portions of the table
- In cases where you know your workload—VACUUM can be run manually
- Use VACUUM BOOST at off peak times (blocks deletes)

(AUTO) ANALYZE

- The ANALYZE process collects table statistics for optimal query planning

- In the vast majority of cases AUTO ANALYZE automatically handles statistics gathering

Best practices

  - ANALYZE can be run periodically after ingestion on just the columns that WHERE predicates are filtered on

Utility to manually run VACUUM and ANALYZE on all the tables in the cluster: https://github.com/awslabs/amazon-redshift-utils/tree/master/src/AnalyzeVacuumUtility

  -

Workload Management

Workload Management (WLM) modes

Manual WLM

- Define different queues based on user groups or query groups
- Define how many concurrent queries can run within each queue
- Allocate a certain percentage of the cluster memory to each queue

Auto WLM

- Define different queues based on user groups or query groups
- Redshift will determine the number of concurrent queries to run at any given point in time
- Redshift will also decide how much memory to allocate for each query based on the need.

Workload Management – SQA and QMR

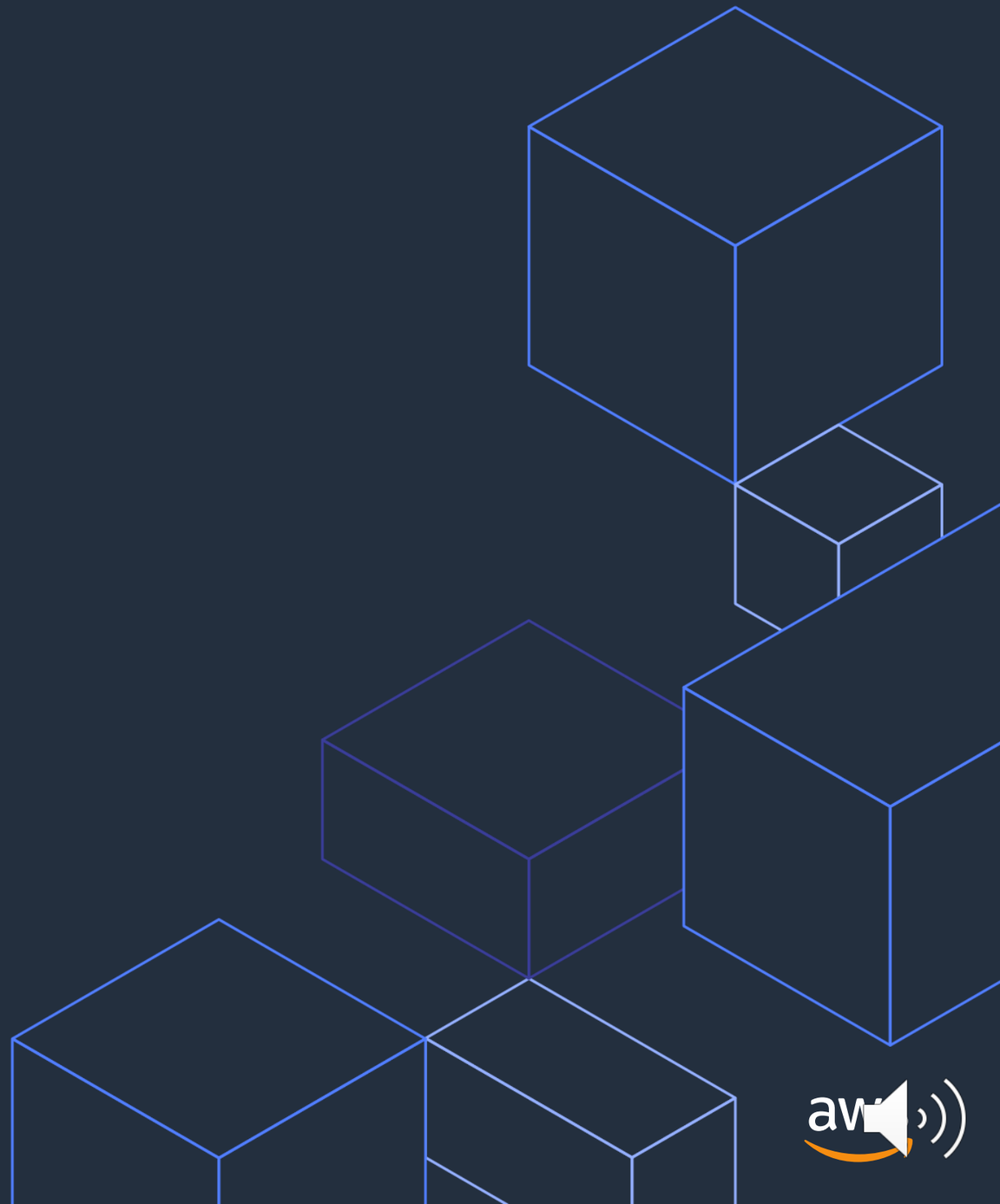
Short Query Acceleration (SQA)

- SQA is enabled via a check box and this would send short queries to a separate queue
- Amazon Redshift uses machine learning to predict the query's execution time.

Query Monitoring Rules (QMR)

- Query Monitoring Rules allow you to make adjustments to your workload in real time
- You can define rules to reprioritize/ cancel badly written queries.
- You can also log queries that meet certain criteria for future analysis

Auto WLM Demo



Amazon Redshift Advisor

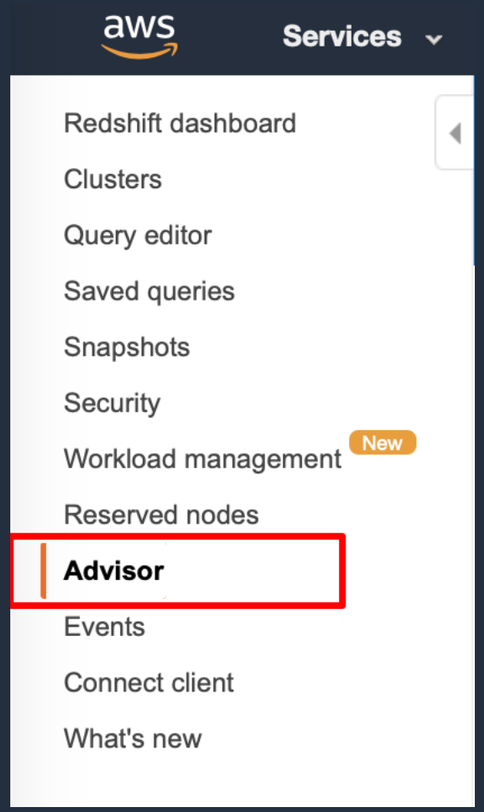
Amazon Redshift Advisor

Amazon Redshift Advisor available in Amazon Redshift Console

Runs daily scanning operational metadata

Observes with the lens of best practices

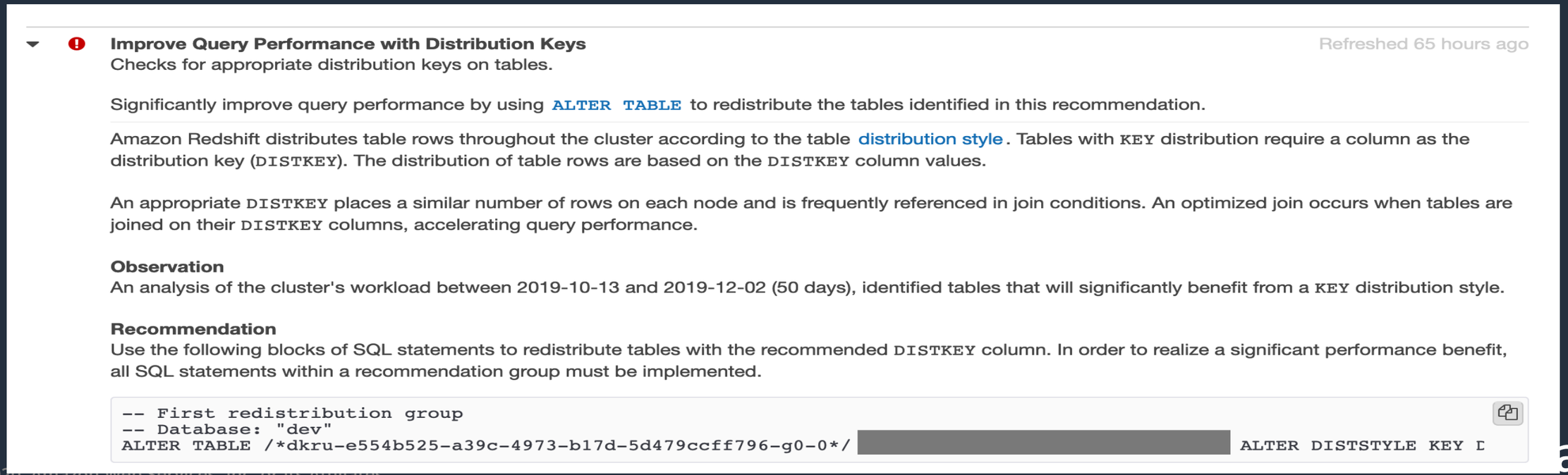
Provides tailored high-impact recommendations to optimize your Amazon Redshift cluster for performance and cost savings



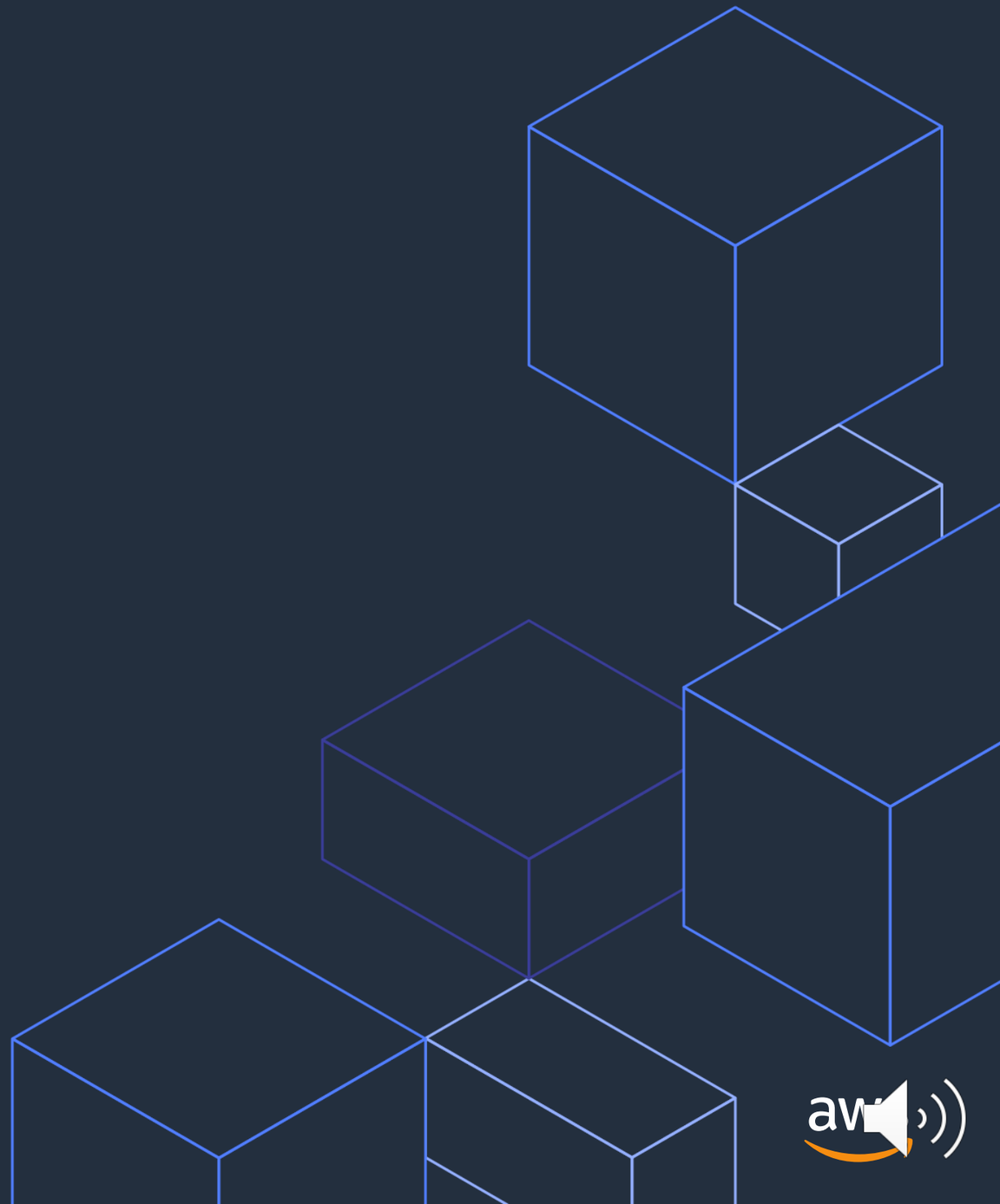
Amazon Redshift Advisor: recommendations

Recommendations include

- Skip compression analysis during COPY
- Split Amazon S3 objects loaded by COPY
- Compress Amazon S3 file objects loaded by COPY
- Compress table data
- Reallocate Workload Management (WLM) memory
- Cost savings
- Enable short query acceleration
- Alter distribution keys on tables



AWS Partners



Amazon Redshift ISV partners

We have a rich ecosystem of partners that offer complementary technology for data transformation and loading, and Business Intelligence.

View all Amazon Redshift partners: https://aws.amazon.com/redshift/partners

Amazon Redshift service partners

Work with APN partners to follow best practices to build and grow your solutions.

View all Amazon Redshift partners: https://aws.amazon.com/redshift/partners

Additional Resources

AWS Labs on GitHub: Amazon Redshift

https://github.com/awslabs/amazon-redshift-utils

https://github.com/awslabs/amazon-redshift-monitoring

https://github.com/awslabs/amazon-redshift-udfs

Admin scripts

Collection of utilities for running diagnostics on your cluster

Admin views

Collection of utilities for managing your cluster, generating schema DDL, and so on

Column Encoding utility

Utility that will apply optimal column encoding to an established schema with data already loaded

AWS big data blog: Amazon Redshift

Amazon Redshift Engineering's Advanced Table Design Playbook

- https://aws.amazon.com/blogs/big-data/amazon-redshift-engineerings-advanced-table-design-playbook-preamble-prerequisites-and-prioritization/

—Zach Christopherson

Top 10 Performance Tuning Techniques for Amazon Redshift

- https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-techniques-for-amazon-redshift/

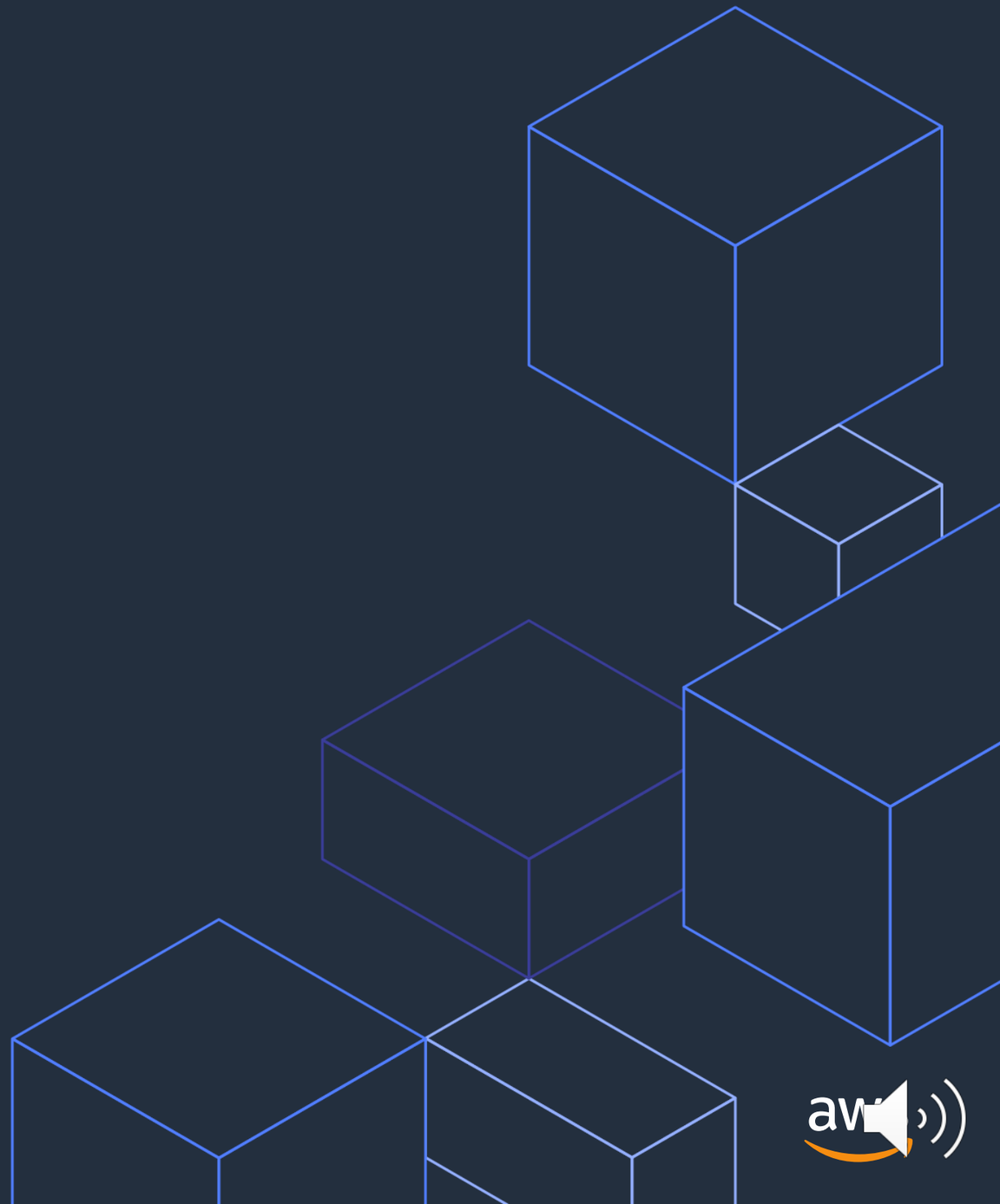
—Ian Meyers and Zach Christopherson

Twelve Best Practices for Amazon Redshift Spectrum

- https://aws.amazon.com/blogs/big-data/10-best-practices-for-amazon-redshift-spectrum/

—Po Hong and Peter Dalton

Q&A



Thank You!