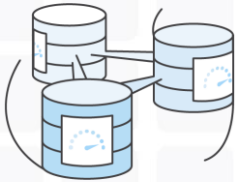
Best Practices for Migrating your Data Warehouse to Amazon Redshift

Darin Briskman, AWS Technical Evangelist
briskman@amazon.com

Why Migrate to Amazon Redshift?

Transactional database

- Redshift can be 100x faster
- Scales from GBs to PBs
- Analyze data without storage constraints



MPP database

- Redshift is 10x cheaper
- Easy to provision and operate
- Higher productivity

Hadoop

- Redshift can be 10x faster
- No programming
- Standard interfaces and integration to leverage BI tools, machine learning, streaming

Migration from Oracle @ Boingo Wireless

2000+ Commercial Wi-Fi locations

1 million+ Hotspots

90M+ ad engagements

100+ countries

Legacy DW: Oracle 11g based DW

Before migration

- Rapid data growth slowed analytics
- Mediocre IOPS, limited memory, vertical scaling
- Admin overhead
- Expensive (license, h/w, support)

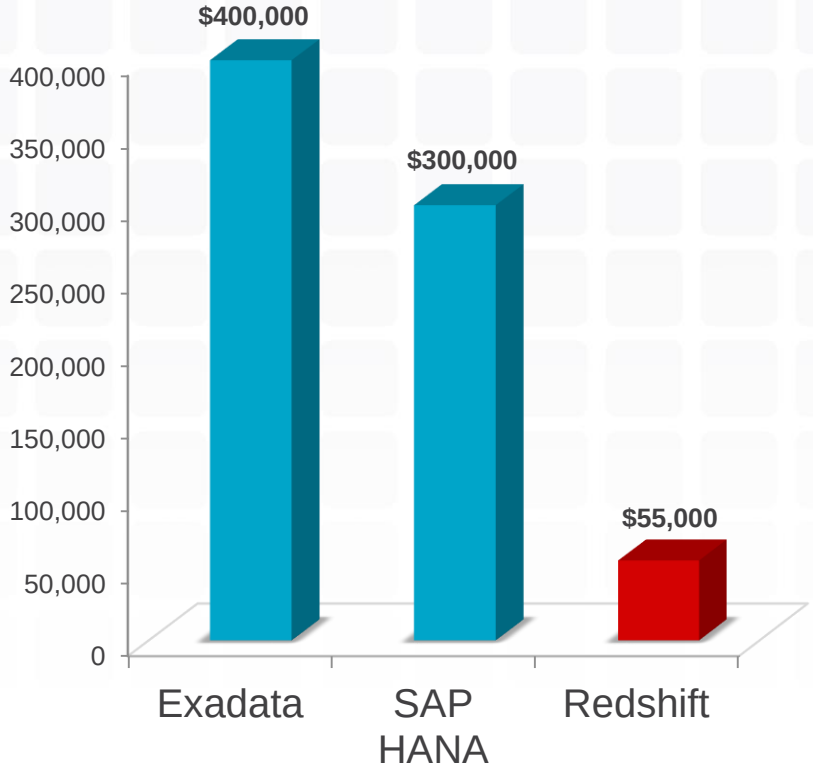
After migration

- 180x performance improvement
- 7x cost savings

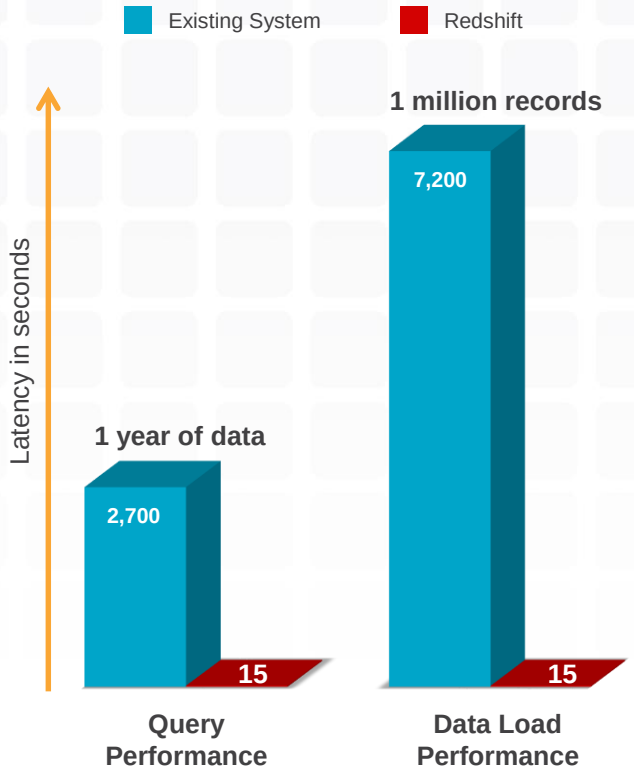
Migration from Oracle @ Boingo Wireless

boingo

7X cheaper than Oracle Exadata



180X faster than Oracle database



aws

Migration from Greenplum @ NTT Docomo

68 million customers

10s of TBs per day of data across mobile network

6PB of total data (uncompressed)

Data science for marketing operations, logistics etc.

Legacy DW: Greenplum on-premises

After migration:

125 node DS2.8XL cluster

4,500 vCPUs, 30TB RAM

6 PB uncompressed

10x faster analytic queries

50% reduction in time for new BI app. deployment

Significantly less ops. overhead

Migration from SQL on Hadoop @ Yahoo

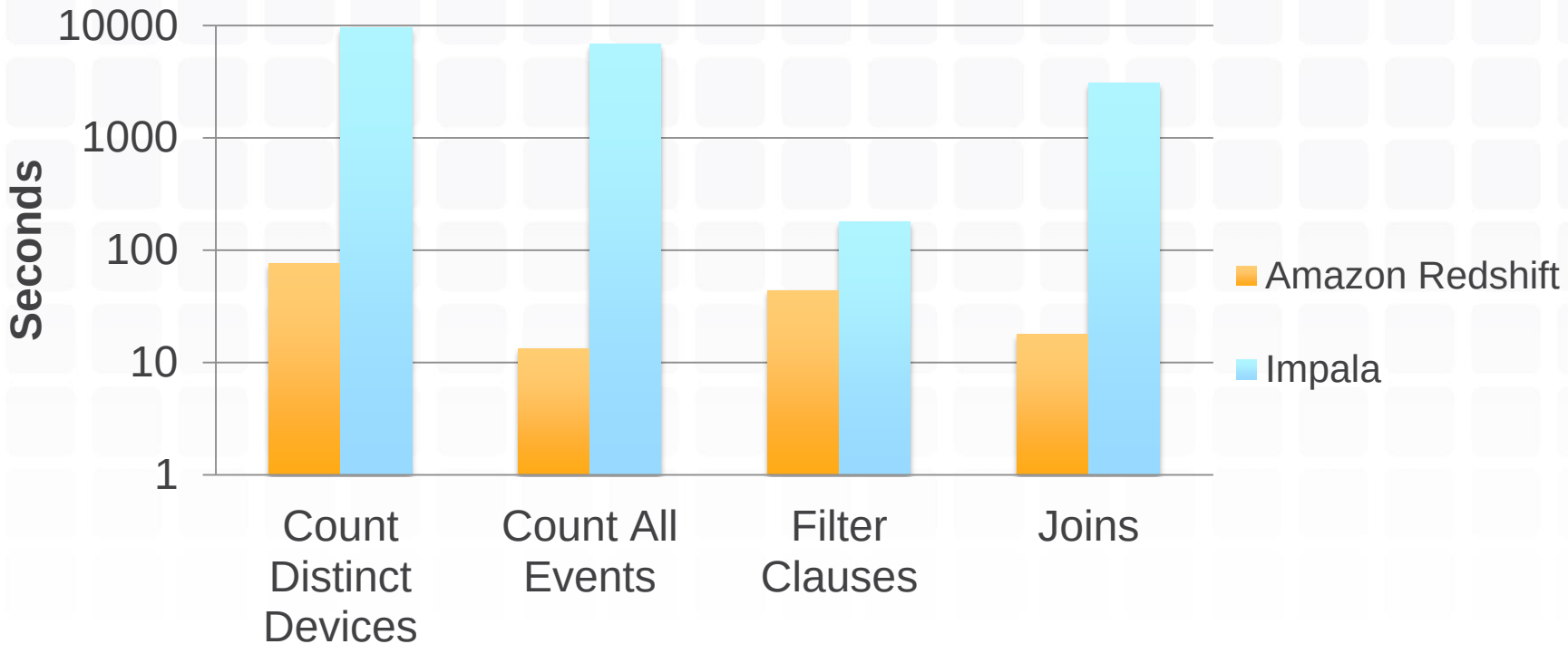
- Analytics for website/mobile events across multiple Yahoo properties
- On an average day
  - 2B events
  - 25M devices

Before migration: Hive – Found it to be slow, hard to use, share and repeat

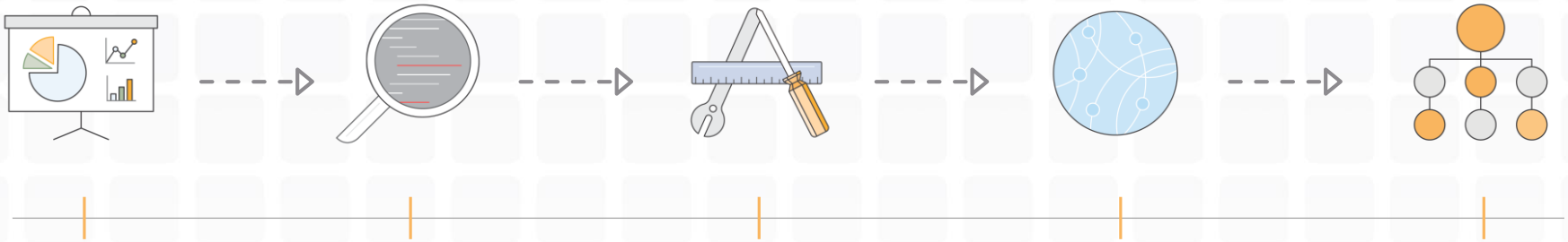
After migration:

- 21 node DC1.8XL (SSD)
- 50TB compressed data
- 100x performance improvement
- Real-time insights
- Easier deployment and maintenance

Migration from SQL on Hadoop @ Yahoo



Amazon Redshift Migration Process



Evaluation

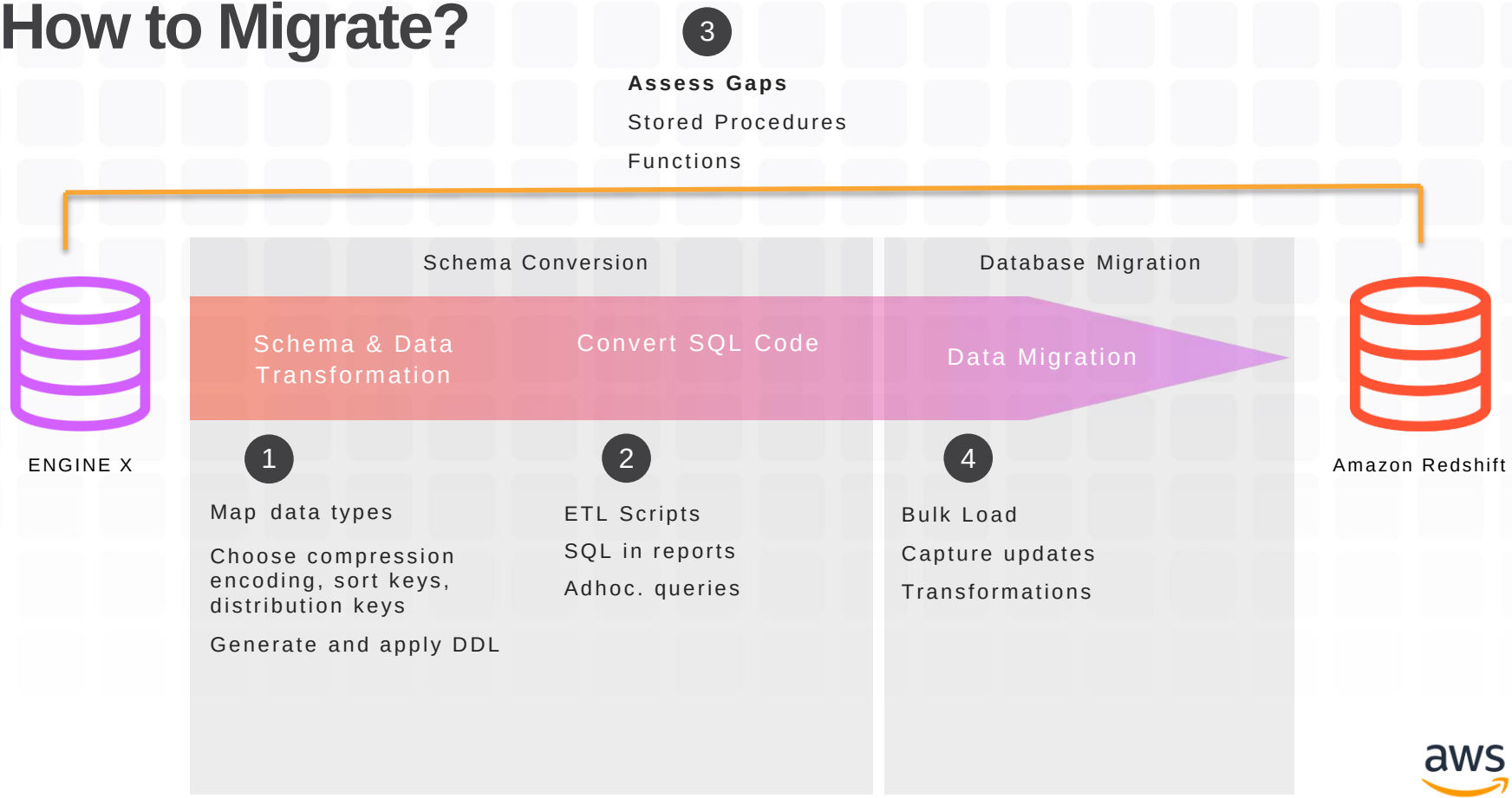
Perform PoC and Validation

Design Migration Plan

Migration, Integration and Validation

Operations and optimization

How to Migrate?

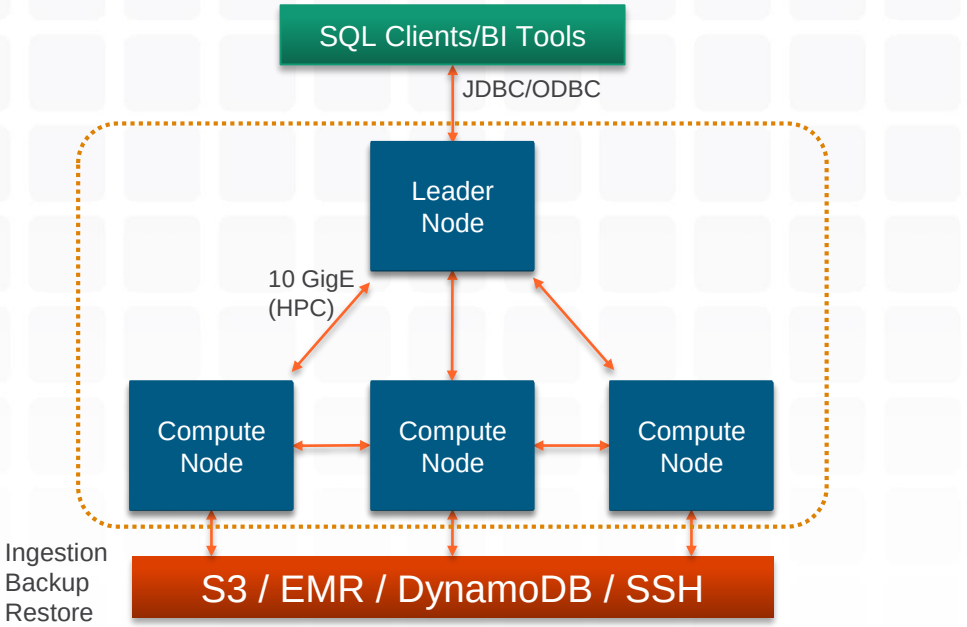


If you forget everything else…

- Lift-and-Shift is **NOT** an ideal approach
  - Depending where you are coming from, it is sure to fail
- AWS has a rich ecosystem of solutions
  - Your final solution will use other AWS services
  - AWS Solution Architects, ProServ, and Partners can help

Amazon Redshift Cluster Architecture

- **Massively parallel, shared nothing**
- **Leader node**
  - SQL endpoint
  - Stores metadata
  - Coordinates parallel SQL processing
- **Compute nodes**
  - Local, columnar storage
  - Executes queries in parallel
  - Load, backup, restore



Designed for I/O Reduction

- Columnar storage
- Data compression
- Zone maps


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

aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

- Accessing dt with row storage:
  - Need to read everything
  - Unnecessary I/O

aws

Designed for I/O Reduction

- Columnar storage
- Data compression
- Zone maps


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

aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

- Accessing dt with columnar storage:
  - Only scan blocks for relevant column

aws

Designed for I/O Reduction

- Columnar storage
- Data compression
- Zone maps


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

aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

- Columns grow and shrink independently
- Effective compression ratios due to like data
- Reduces storage requirements
- Reduces I/O

aws

Designed for I/O Reduction

- Columnar storage
- Data compression
- Zone maps

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

aid	loc	dt
1	SFO	2016-09-01
2	JFK	2016-09-14
3	SFO	2017-04-01
4	JFK	2017-05-14

- In-memory block metadata
- Contains per-block MIN and MAX value
- Effectively prunes blocks which cannot contain data for a given query
- Eliminates unnecessary I/O

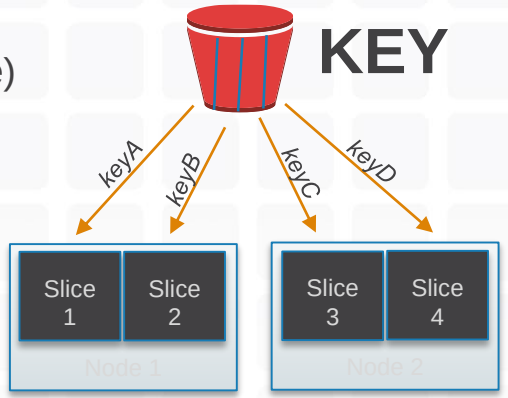
aws

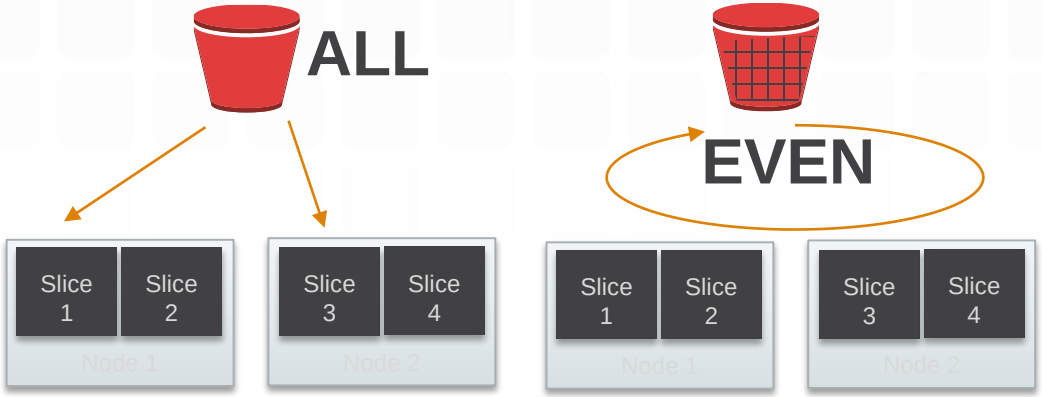
Terminology and Concepts: Slices

- A **slice** can be thought of like a “virtual compute node”
  - Unit of data partitioning
  - Parallel query processing
- Facts about slices:
  - Each compute node has either 2, 16, or 32 slices
  - Table rows are distributed to slices
  - A slice processes only its own data

Terminology and Concepts: 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 first slice of every node
  - EVEN: Round robin
- Goals:
  - Distribute data evenly for parallel processing
  - Minimize data movement during query processing





aws

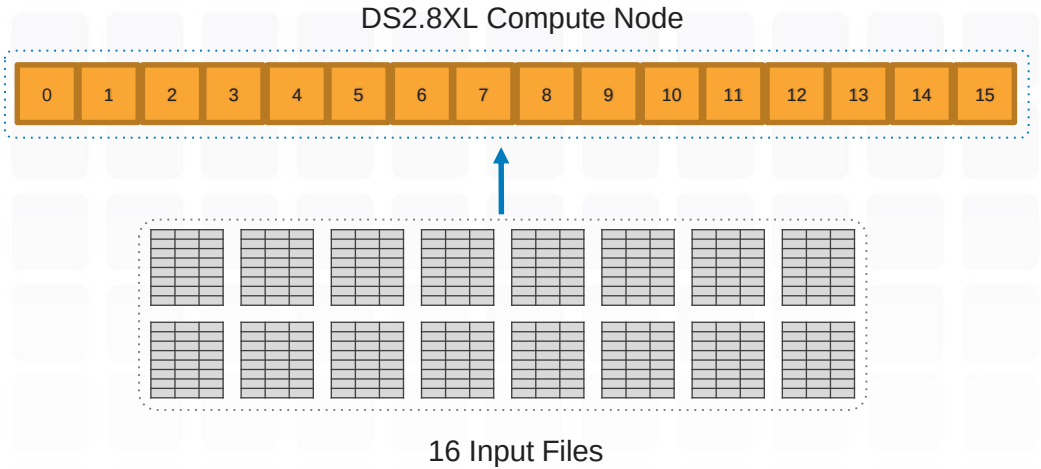
Data Loading Best Practices

- 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 Loading Best Practices Continued

- Use at least as many input files as there are slices in the cluster
- With 16 input files, all slices are working so you maximize throughput
- COPY continues to scale linearly as you add nodes



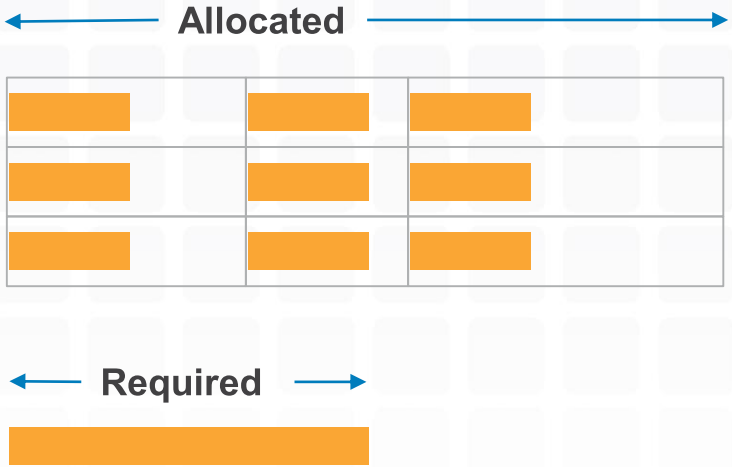
Data Preparation

- Export Data from Source System
  - CSV Recommend (**Delimiter '|'**)
    - Be aware of UTF-8 varchar columns (UTF-8 take 4 bytes per char)
    - Be aware of your NULL character (**\N**)
  - GZIP Compress Files
  - Split Files (1MB – 1GB after gzip compression)

- Useful COPY Options for PoC Data
  - MAXERRORS
  - ACCEPTINVCHARS
  - NULL AS

Keep Columns as Narrow as Possible

- Buffers allocated based on declared column width
- Wider than needed columns mean memory is wasted
- Fewer rows fit into memory; increased likelihood of queries spilling to disk
- Check SVV_TABLE_INFO(max_varchar)
- SELECT max(len(col)) FROM table



Amazon Redshift is a Data Warehouse

- Optimized for batch inserts
  - The time to insert a single row in Redshift is roughly the same as inserting 100,000 rows
- Updates are delete + insert of the row
  - Deletes mark rows for deletion
- Blocks are immutable
  - Minimum space used is one block per column, per slice

Column Compression

- Auto Compression
  - Samples data automatically when COPY into an empty table
    - Samples up to 100,000 rows and picks optimal encoding
  - Turn off Auto Compression for Staging Tables
    - Bake encodings into your DDL or use CREATE TABLE (LIKE …)
- Analyze Compression
  - Data profile has changed
  - Run after changing sort key

Column (Compression) Encoding Types

- Raw encoding (RAW)
- Byte-dictionary (BYTEDICT)
- Delta encoding (DELTA / DELTA32K)
- Mostly encoding (MOSTLY8 / MOSTLY16 / MOSTLY32)
- Runlength encoding (RUNLENGTH)
- Text encoding (TEXT255 / TEXT32K)
- LZO encoding (LZO)
- Zstandard (ZSTD)

Average: 2-4x

Primary/Unique/Foreign Key Constraints

- Primary/Unique/Foreign Key constraints are NOT enforced
  - If you load data multiple times, Amazon Redshift won't complain
  - If you declare primary keys in your DDL, the optimizer will expect the data to be unique
- Redshift optimizer uses declared constraints to pick optimal plan
  - In certain cases it can result in performance improvements

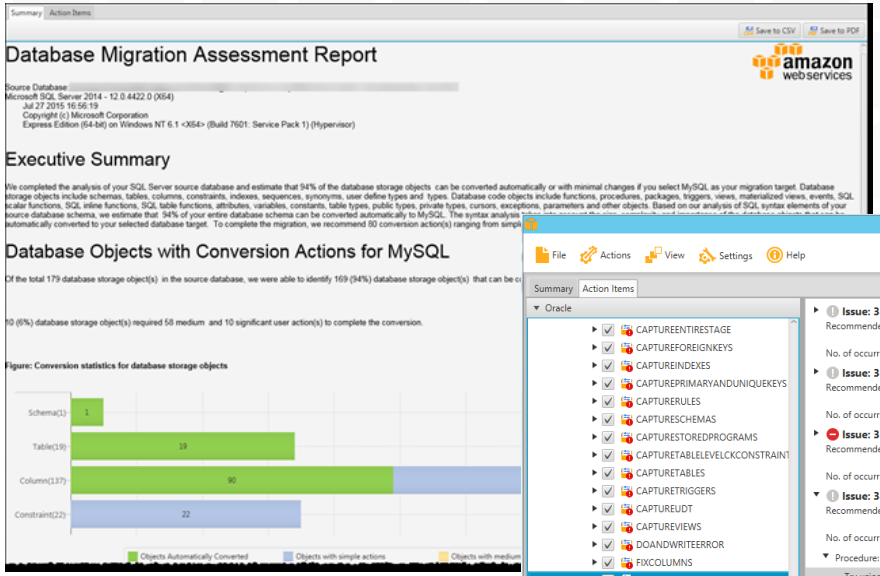
Benchmarking Tips

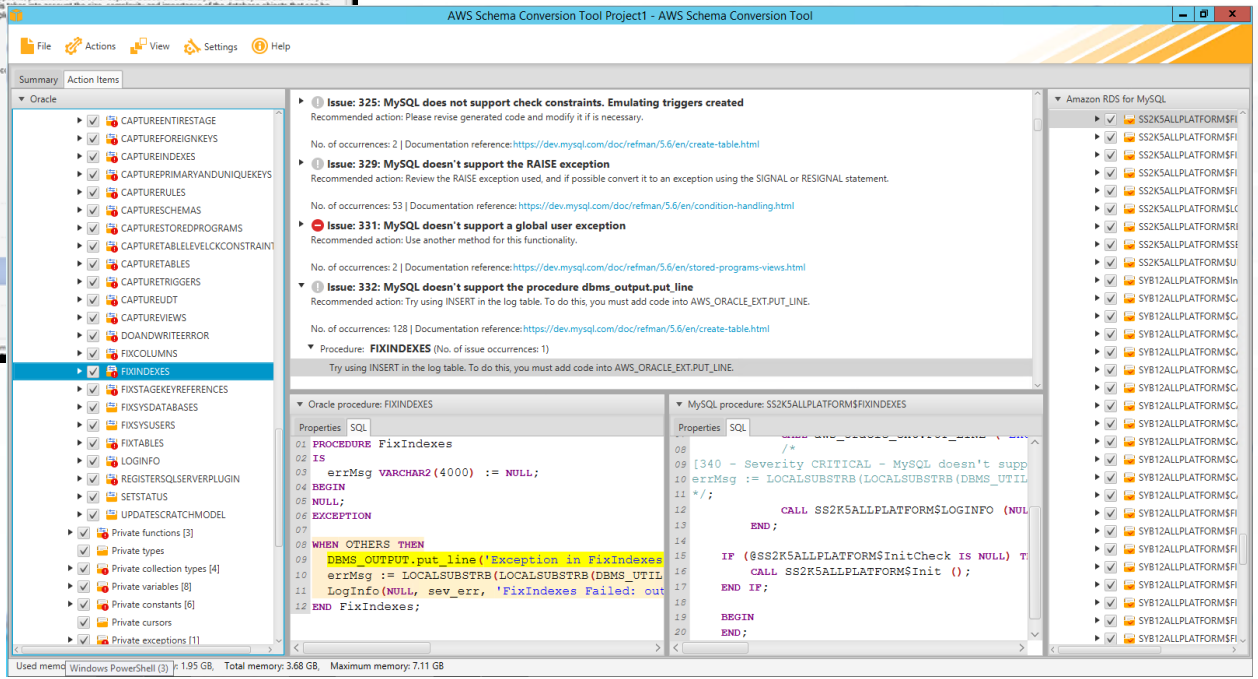
- Verify tables are vacuumed and analyzed
  - Check SVV_TABLE_INFO (STATS_OFF, UNSORTED)
  - Vacuum & Analyze
    - https://github.com/awslabs/amazon-redshift-utils/tree/master/src/AnalyzeVacuumUtility
- Verify column encodings
  - Check PG_TABLE_DEF
  - Re-encode tables
    - https://github.com/awslabs/amazon-redshift-utils/tree/master/src/ColumnEncodingUtility
- Verify good distribution keys
  - SVV_TABLE_INFO (SKEW_ROWS)

AWS Schema Conversion Tool (AWS SCT)

- Convert schema in a few clicks
- Sources include Oracle, Teradata, Greenplum, Netezza, and MS SQL DW
- Automatic schema optimization
- Converts application SQL code
- Detailed assessment report

AWS Schema Conversion Tool

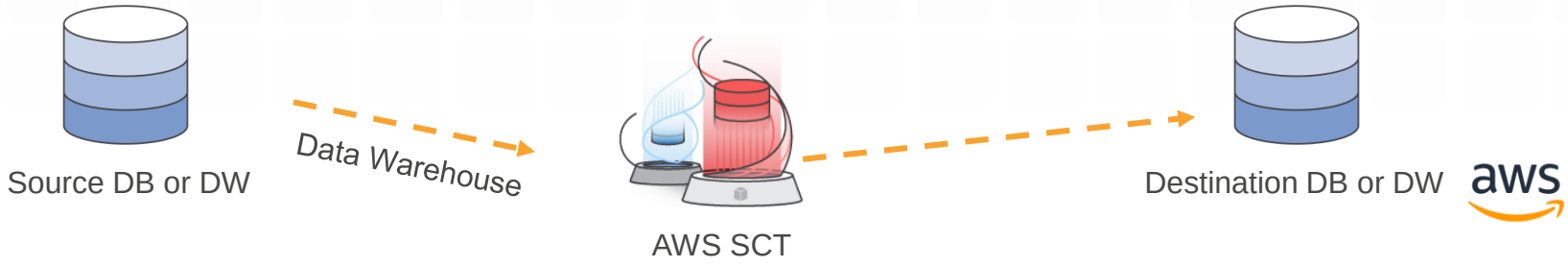




Database migration process

Step 1: Convert or Copy your Schema

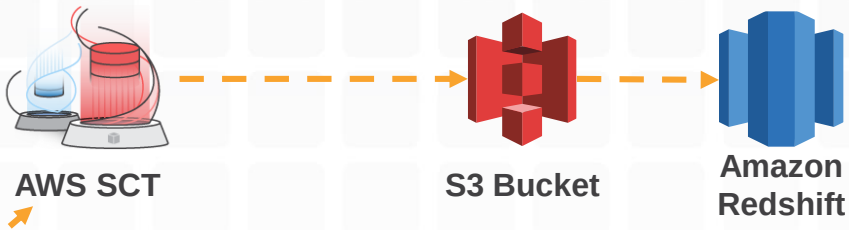
Step 2: Move your data



SCT data extractors

Extract Data from your data warehouse and migrate to Amazon Redshift

- Extracts through local migration agents
- Data is optimized for Redshift and Saved in local files
- Files are loaded to an Amazon S3 bucket (through network or Amazon Snowball) and then to Amazon Redshift



TERADATA. NETEZZA
Greenplum VERTICA
ORACLE

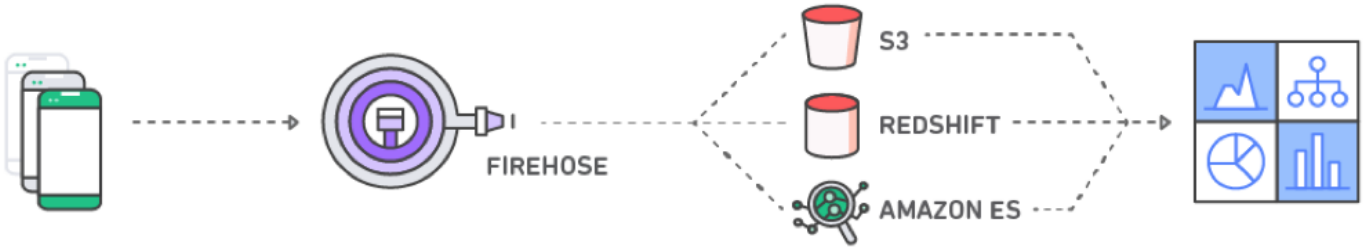
aws

For Large Data Transfers

- AWS Snowball
- AWS Snowball Edge
- AWS Snowmobile

Amazon Kinesis Firehose

Load massive volumes of streaming data into Amazon S3, Redshift and Elasticsearch



Capture and submit streaming data

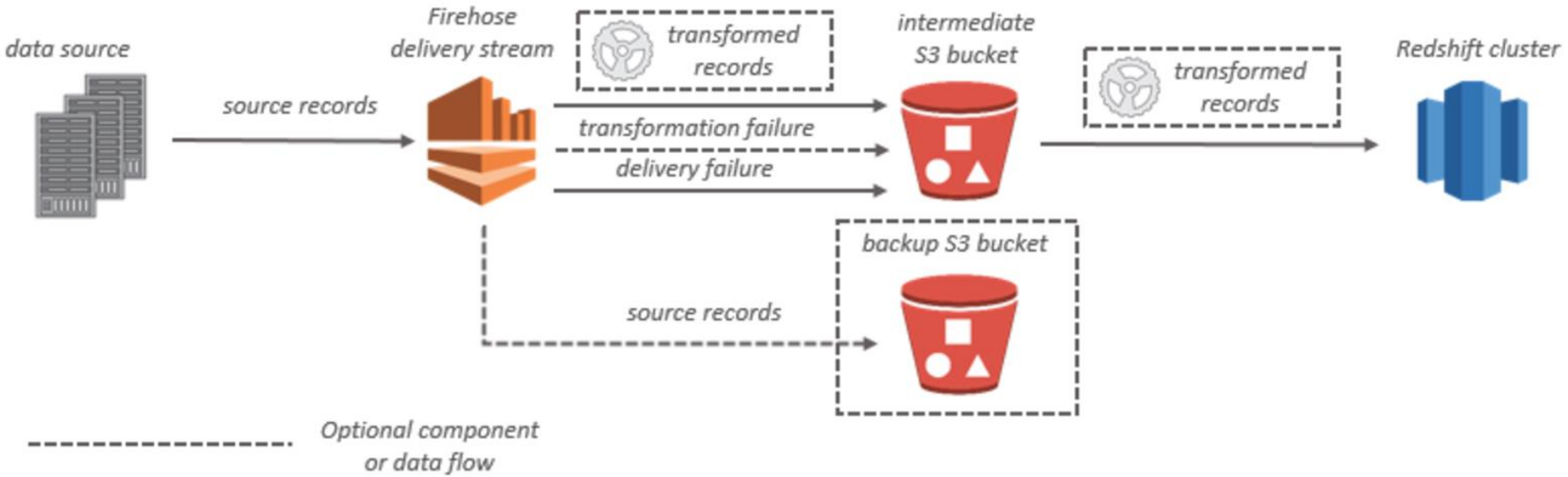
Firehose loads streaming data continuously into Amazon S3, Redshift and Elasticsearch

Analyze streaming data using your favorite BI tools

- **Zero administration:** Capture and deliver streaming data into Amazon S3, Amazon Redshift, and other destinations **without writing an application or managing infrastructure.**
- **Direct-to-data store integration: Batch, compress**, and **encrypt** streaming data for delivery into data destinations **in as little as 60 secs** using simple configurations.
- **Seamless elasticity:** Seamlessly scales to match data throughput w/o intervention
- **Serverless ETL using AWS Lambda** - Firehose can invoke your Lambda function to transform incoming source data.

aws

Amazon Kinesis Firehose to Amazon Redshift



Data Integration Partners

Amazon Redshift

Data Integration

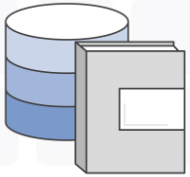
- ATTUNITY CloudBeam
- bryte Systems
- FlyData
- INFORMATICA
- Matillion
- pentaho
- Segment
- snapLogic
- talend integration at any scale
- Qubole
- xplenty

Systems Integrators

- 2ND WATCH
- accenture
- BEEVA
- BRIGHTLIGHT Ask more of your data
- Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING
- cloudpack
- Cognizant
- FULL 360
- iOLAP
- isobar
- MAGNUS DATA
- NorthBay solutions
- SLALOM CONSULTING

aws

AWS Glue for Automated, Serverless ETL



Data Catalog

Hive metastore compatible metadata repository of data sources

Crawls data source to infer table, data type, partition format

Job Authoring

Generates Python code to move data from source to destination

Edit with your favorite IDE; share code snippets using Git

Job Execution

Runs jobs in Spark containers – automatic scaling based on SLA

Glue is serverless – only pay for the resources you consume

aws

Resources

- 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, etc.
- **ColumnEncodingUtility**
  Gives you the ability to apply optimal column encoding to an established schema with data already loaded
- **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/

Thank you!

briskman@amazon.com