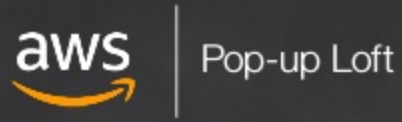
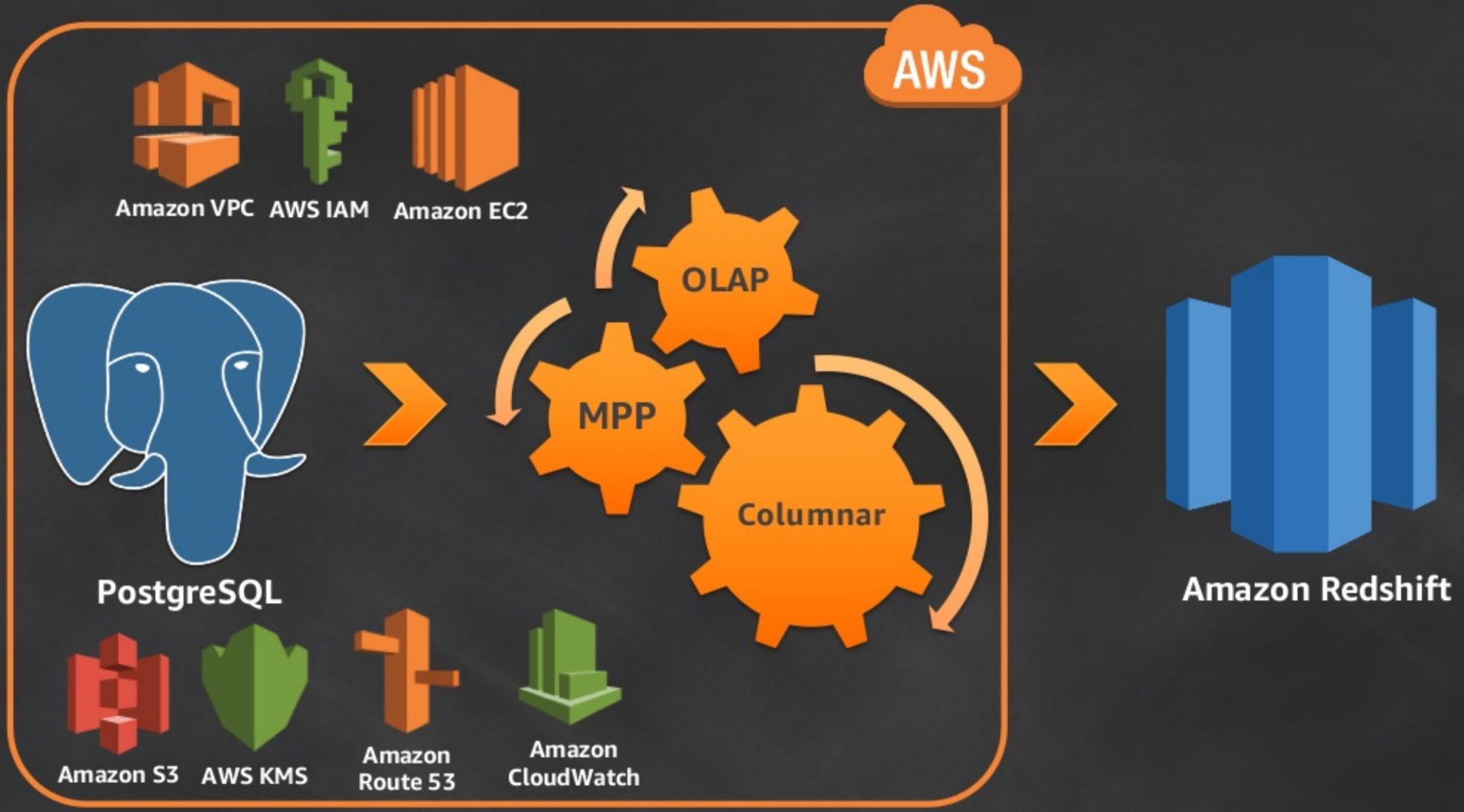
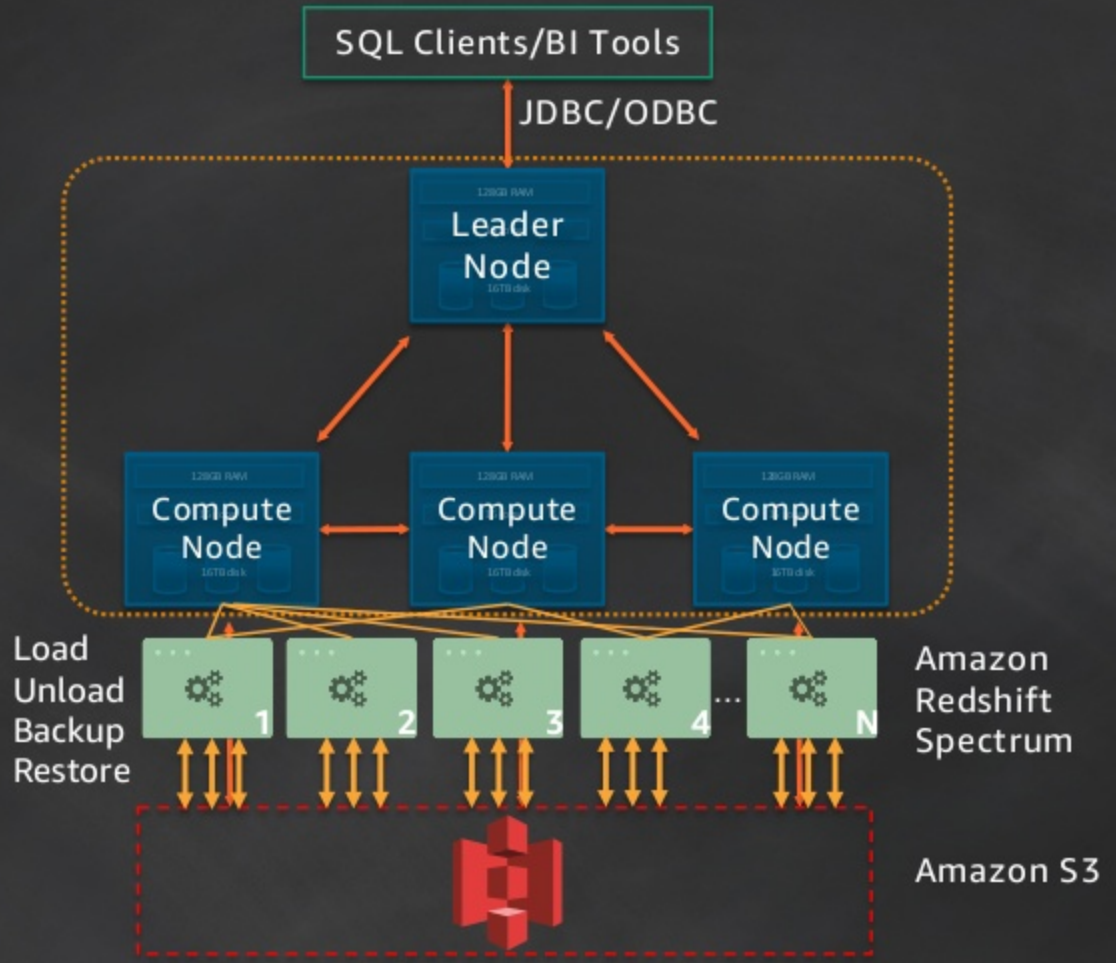
Loading Data into Amazon Redshift





Amazon Redshift Architecture

- 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)



Terminology and Concepts: Disks

- Amazon Redshift utilizes locally attached storage devices
  - Compute nodes have 2½ to 3 times the advertised storage capacity

- Each disk is split into two partitions
  - Local data storage, accessed by local CN
  - Mirrored data, accessed by remote CN

- Partitions are raw devices
  - Local storage devices are ephemeral in nature
  - Tolerant to multiple disk failures on a single node

Terminology and Concepts: Redundancy

- Global **commit** ensures all permanent tables have written blocks to another node in the cluster to ensure data redundancy

- Asynchronously backup blocks to Amazon S3—always consistent snapshot
  - Every 5 GB of changed data or eight hours
  - User on-demand manual snapshots

- Temporary tables
  - Blocks are not mirrored to the remote partition—two-times faster write performance
  - Do not trigger a full commit or backups

Disable backups at the table level:

```
CREATE TABLE example(id int) BACKUP NO;
```

Terminology and Concepts: Transactions

- Amazon Redshift is a fully transactional, ACID compliant data warehouse
  - Isolation level is *serializable*
  - Two phase commits (local and global commit phases)

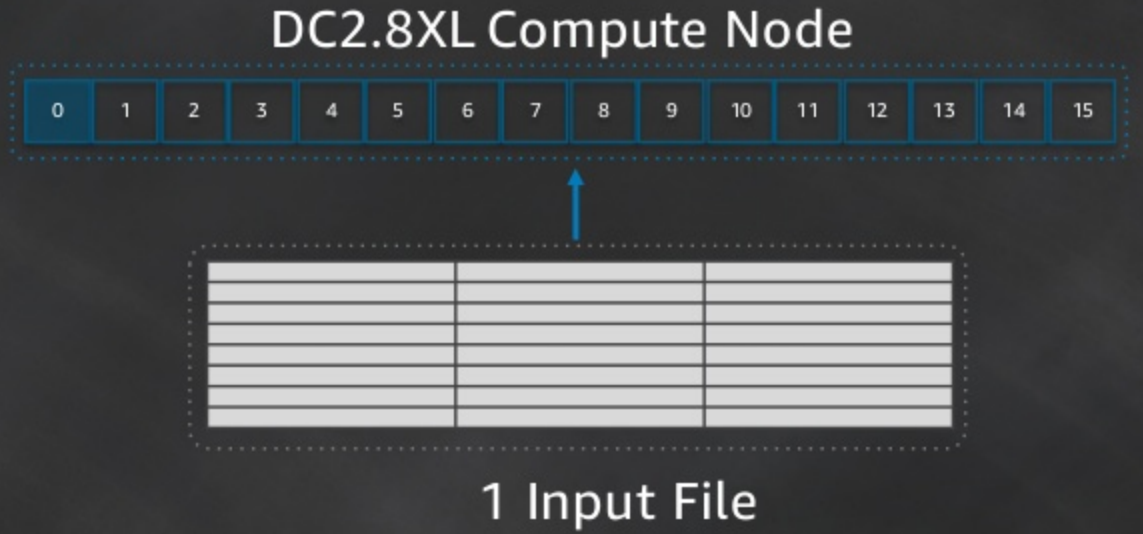
- Cluster commit statistics:
  https://github.com/awslabs/amazon-redshift-utils/blob/master/src/AdminScripts/commit_stats.sql

Design consideration:

- Because of the expense of commit overhead, limit commits by explicitly creating transactions

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

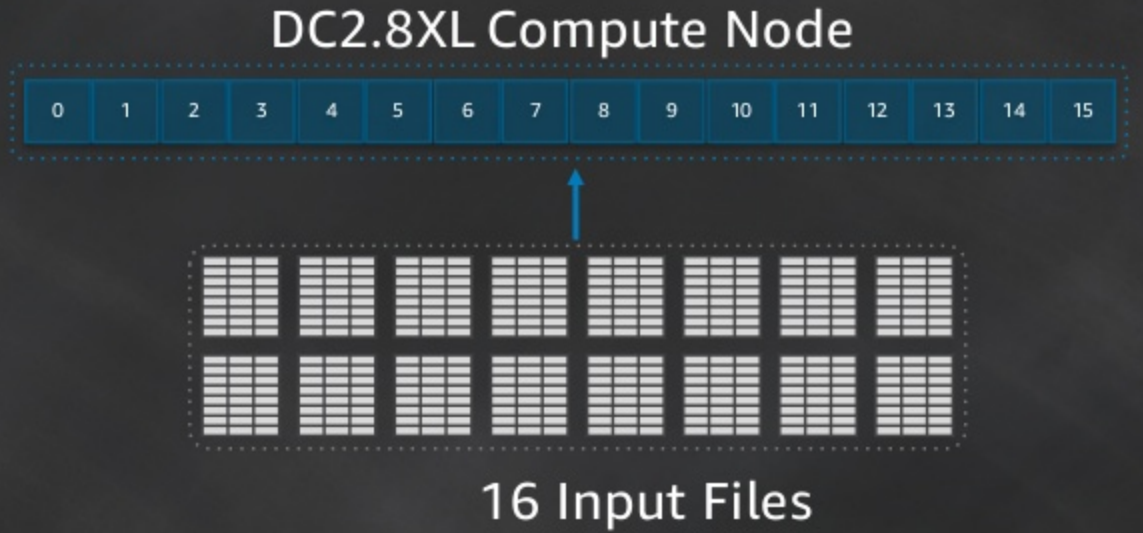


Data Ingestion: COPY Statement

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 recommend
  - 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 so there is a multiple of the number of slices
- Files sizes should be 1 MB–1 GB after gzip compression

- Useful COPY options
  - MAXERRORS
  - ACCEPTINVCHARS
  - NULL AS

Data Ingestion: Amazon Redshift Spectrum

- Use INSERT INTO SELECT against external Amazon S3 tables
  - Ingest additional file formats: Parquet, ORC, Grok
  - Aggregate incoming data
  - Select subset of columns and/or rows
  - Manipulate incoming column data with SQL

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

- Design considerations:
  - Repeated reads against Amazon S3 are not transactional
  - $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
  - Must VACUUM or DEEP COPY to remove ghost rows from table

Data Ingestion: Deduplication/UPSERT

Table: 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
5	SJC	**2017-10-10**
6	SEA	**2017-11-29**

s3://bucket/dd.csv

aid	loc	dt
1	SFO	2017-10-20
2	JFK	2017-10-20
5	SJC	2017-10-10
6	SEA	2017-11-29

Data Ingestion: Deduplication/UPSERT

Steps:

1. Load CSV data into a staging table
2. Delete duplicate data from the production table
3. Insert (or append) data from the staging into the production table

Data Ingestion: Deduplication/UPSERT

```
BEGIN;
CREATE TEMP TABLE staging(LIKE deep_dive);
COPY staging FROM 's3://bucket/dd.csv'
:'creds' COMPUPDATE OFF;
DELETE deep_dive d
USING staging s WHERE d.aid = s.aid;
INSERT INTO deep_dive SELECT * FROM staging;
DROP TABLE staging;
COMMIT;
```

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
  - Turn off automatic compression—COMPUPDATE OFF
  - Copy compression settings from production table or use ANALYZE COMPRESSION statement
    - Use CREATE TABLE LIKE or write encodings into the DDL
  - For copying a large number of rows (> hundreds of millions) consider using ALTER TABLE APPEND instead of INSERT INTO SELECT

Vacuum and Analyze

- VACUUM will globally sort the table and remove rows that are marked as deleted
  - For tables with a sort key, ingestion operations will locally sort new data and write it into the unsorted region

- ANALYZE collects table statistics for optimal query planning

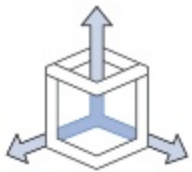
Best Practices:

- VACUUM should be run only as necessary
  - Typically nightly or weekly
  - Consider **Deep Copy** (recreating and copying data) for larger or wide tables
- ANALYZE can be run periodically after ingestion on just the columns that WHERE predicates are filtered on
- Utility to VACUUM and ANALYZE all the tables in the cluster:
  https://github.com/awslabs/amazon-redshift-utils/tree/master/src/AnalyzeVacuumUtility

Amazon Kinesis: Streaming Data Done the AWS Way

Makes it easy to capture, deliver, and process real-time data streams

Easy to provision, deploy, and manage



Elastically scalable

Real-time latencies

Pay as you go, no up-front costs

Right services for your specific use cases

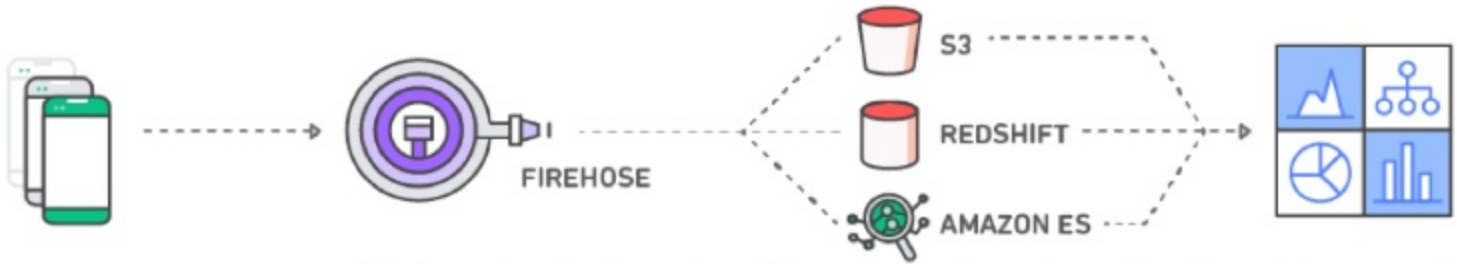
amazon web services

Streaming Data Scenarios Across Verticals

Scenarios/ Verticals	Accelerated Ingest-Transform-Load	Continuous Metrics Generation	Responsive Data Analysis
Digital Ad Tech/Marketing	Publisher, bidder data aggregation	Advertising metrics like coverage, yield, and conversion	User engagement with ads, optimized bid/buy engines
IoT	Sensor, device telemetry data ingestion	Operational metrics and dashboards	Device operational intelligence and alerts
Gaming	Online data aggregation, e.g., top 10 players	Massively multiplayer online game (MMOG) live dashboard	Leader board generation, player-skill match
Consumer Online	Clickstream analytics	Metrics like impressions and page views	Recommendation engines, proactive care

Amazon Kinesis Firehose

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



- **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.

Capture IT & App Logs, Device & Sensor Data, and more

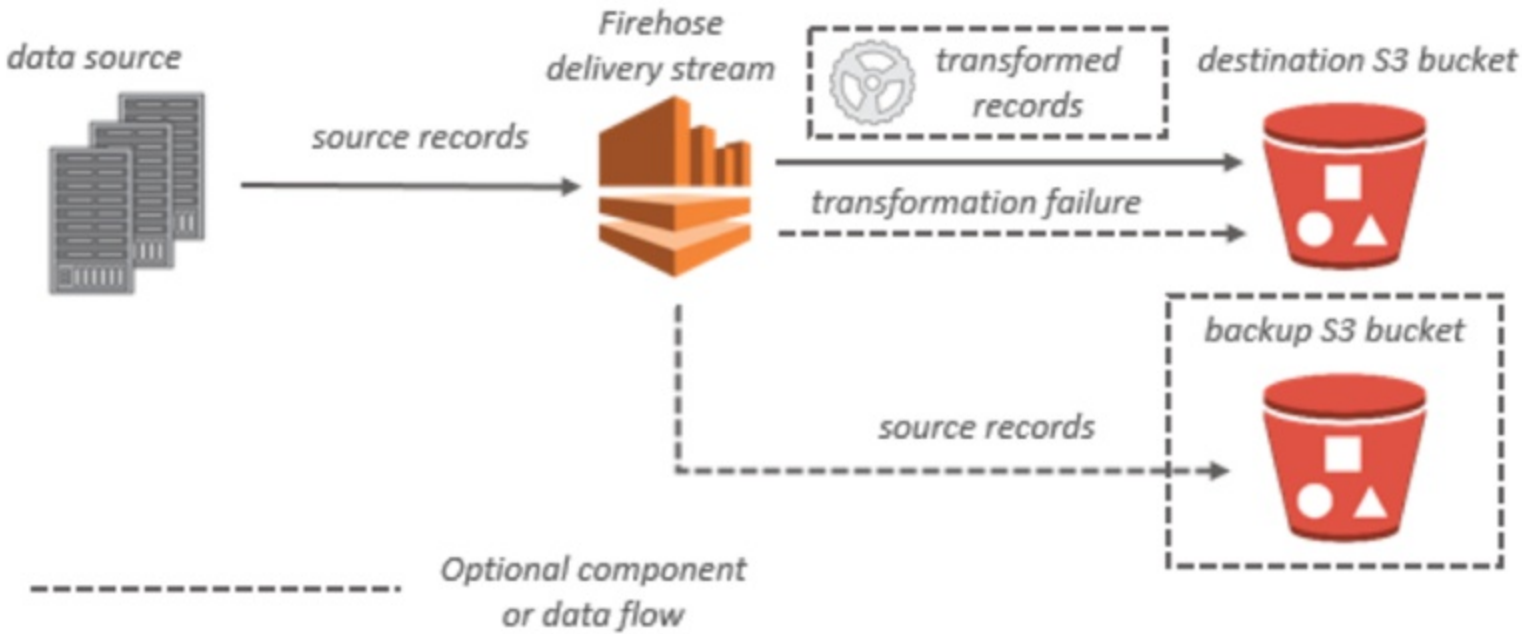
Enable near-real time analytics using existing tools

- Send data from IT infra, mobile devices, sensors
- Integrated with AWS SDK, Agents, and AWS IoT

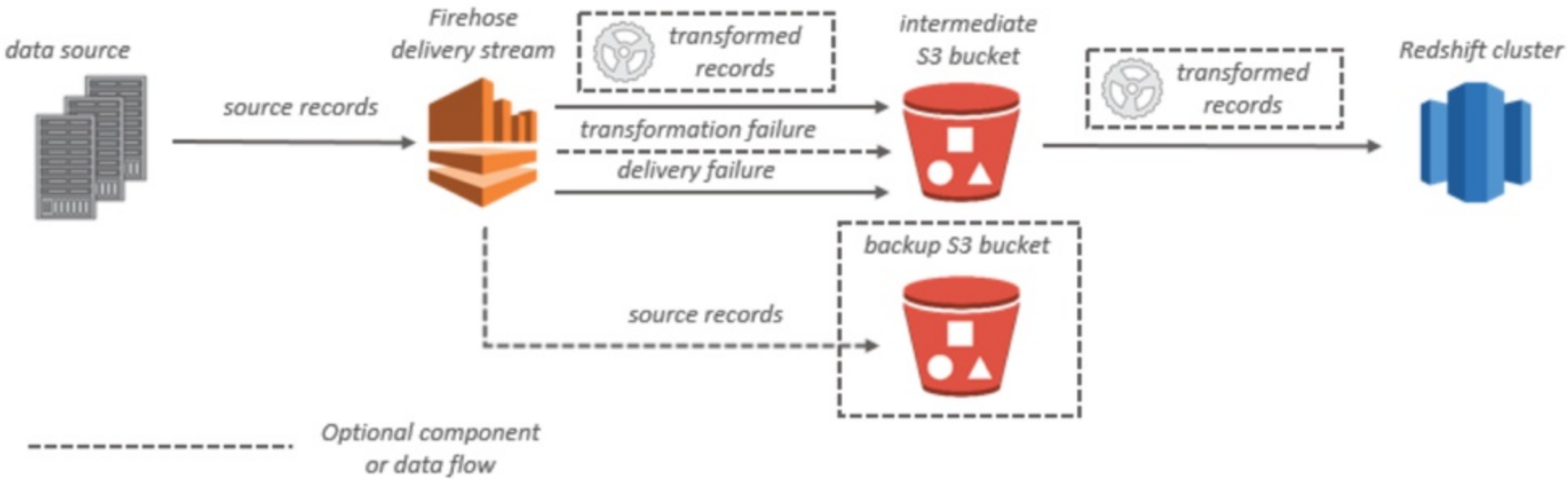
- Fully-managed service to capture streaming data
- Elastic w/o resource provisioning
- Pay-as-you-go: 3.5 cents/ GB transferred

- Batch, compress, and encrypt data before loads
- Loads data into Redshift tables base on COPY command

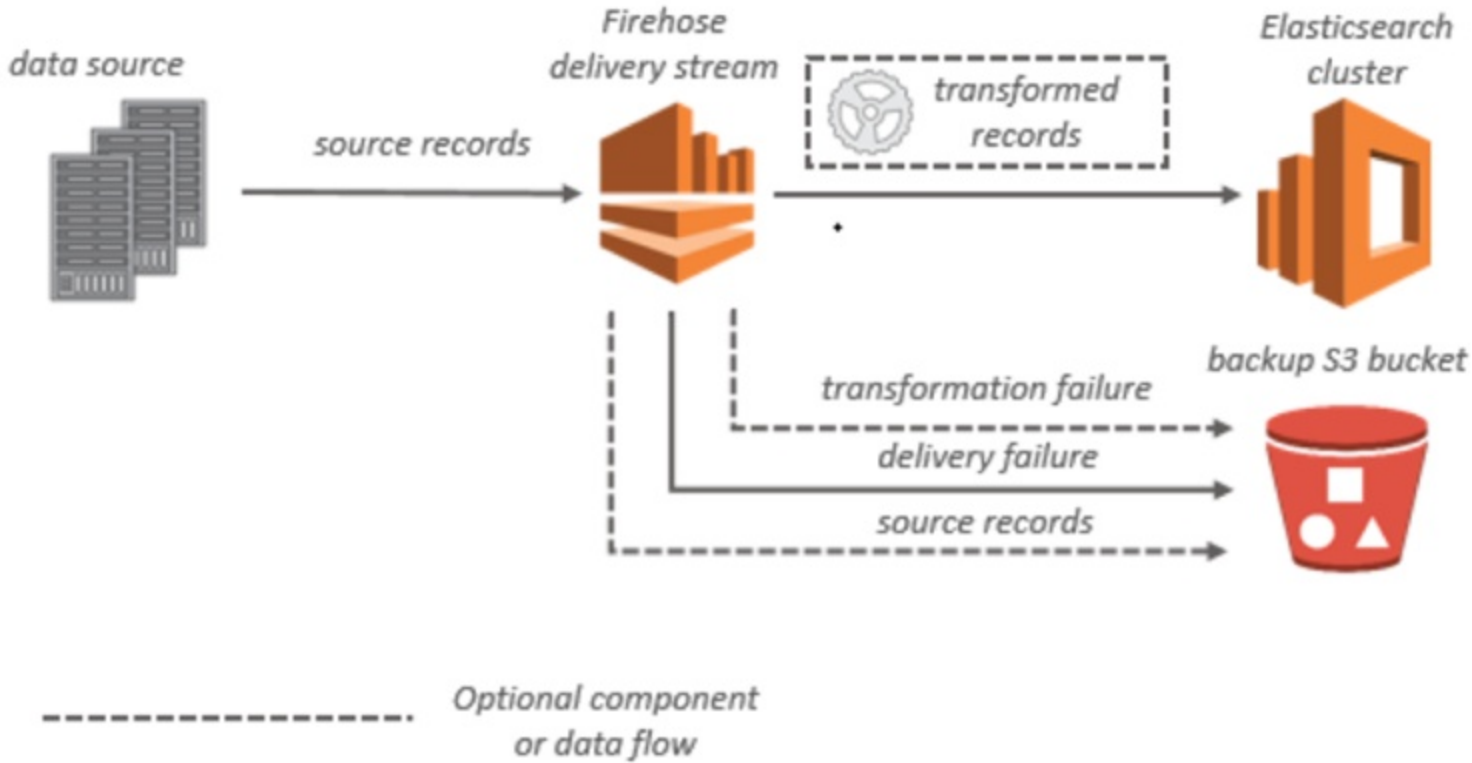
Amazon Kinesis Firehose to Amazon S3



Amazon Kinesis Firehose to Amazon Redshift



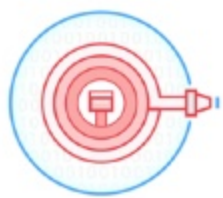
Amazon Kinesis Firehose to Amazon Elasticsearch



Amazon Kinesis Data Firehose vs. Amazon Kinesis Data Streams

Amazon Kinesis Streams

Amazon Kinesis Data Streams is for use cases that require **custom processing**, per incoming record, with sub-1 second processing latency, and a choice of stream processing frameworks.

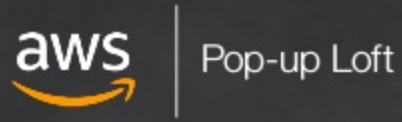


Amazon Kinesis Firehose

Amazon Kinesis Data Firehose is for use cases that require zero administration, ability to **use existing analytics tools based on Amazon S3, Amazon Redshift and Amazon Elasticsearch**, and a data latency of 60 seconds or higher.

Hands-on Lab: Redshift Basics

http://github.com/wrbaldwin/da-week/



Pop-up Loft

Everything and Anything Startups Need to Get Started on AWS

aws.amazon.com/activate