AWS re:Invent

ANT356-R

Building Your First Serverless Data Lake

Damon Cortesi
Big Data Architect
AWS

AWS re:Invent

Data lakes on AWS

Agenda

Data lakes on AWS

Ingestion

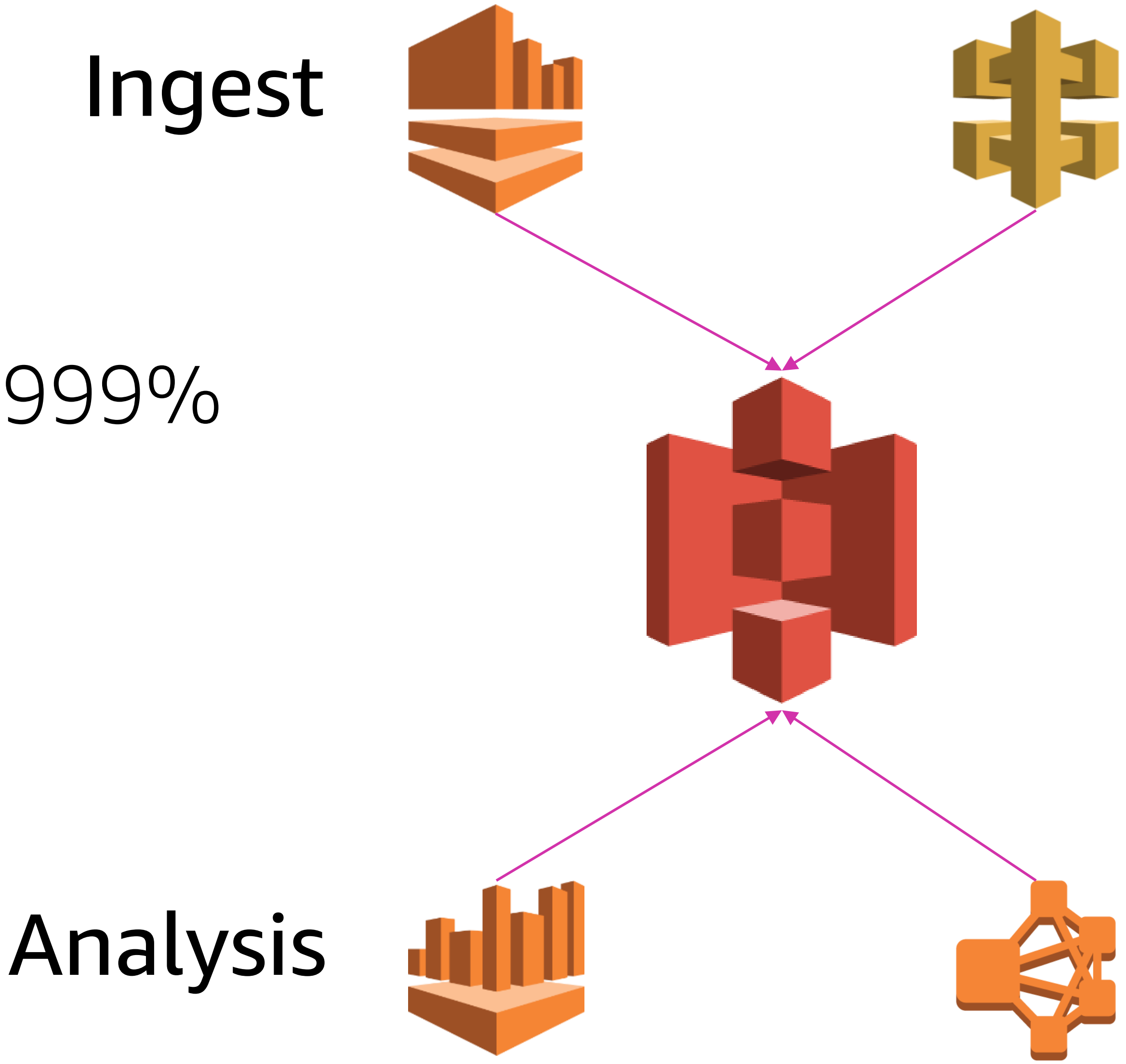
Data prep & optimization

File formats and partitions

A data lake is a centralized repository that allows you to store all your structured and unstructured data at any scale

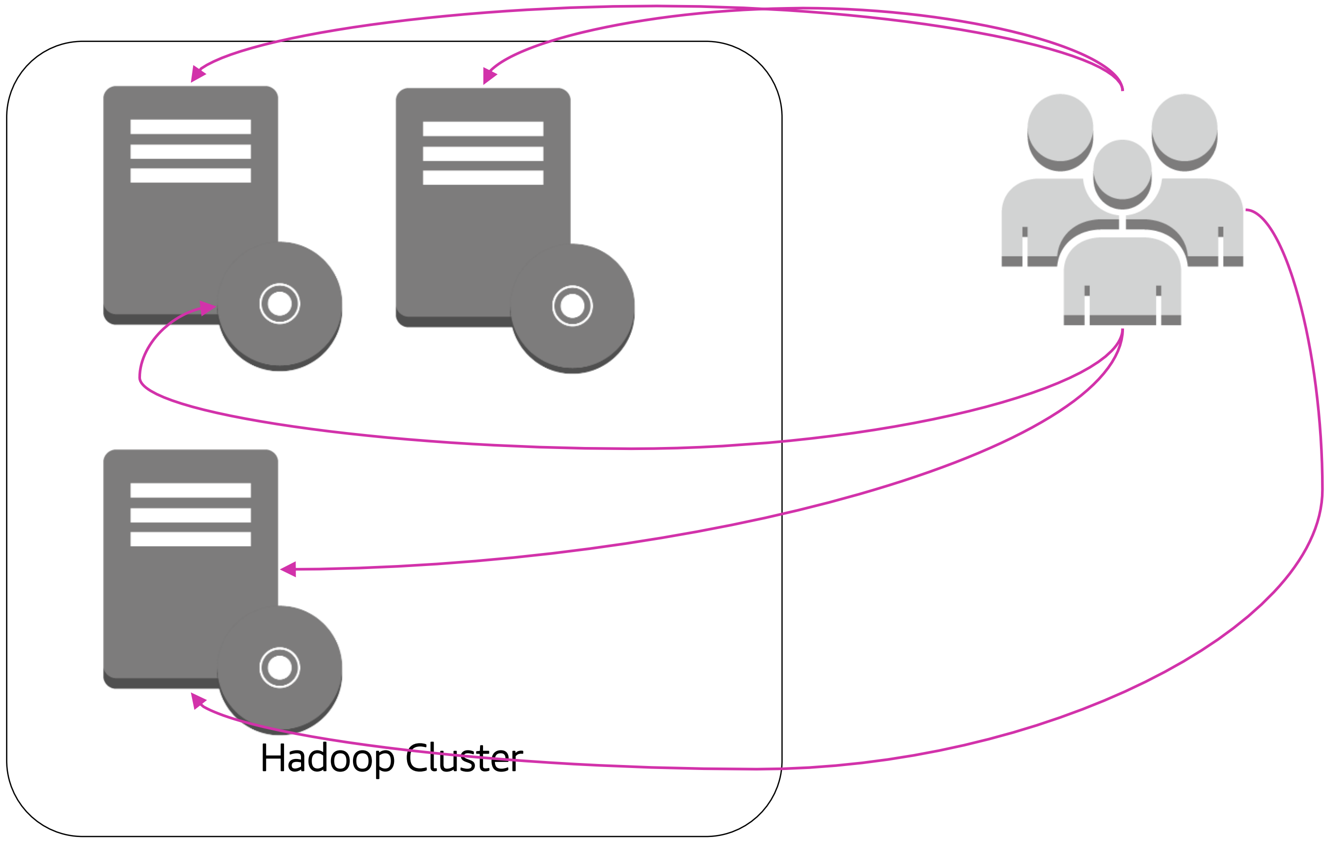
Amazon Simple Storage Service (Amazon S3) is the core

- Durable – designed to be 99.999999999%
- Available – designed to be 99.99%
- Storage – virtually unlimited
- Query in place
- Integrated encryption
- Decoupled storage & compute



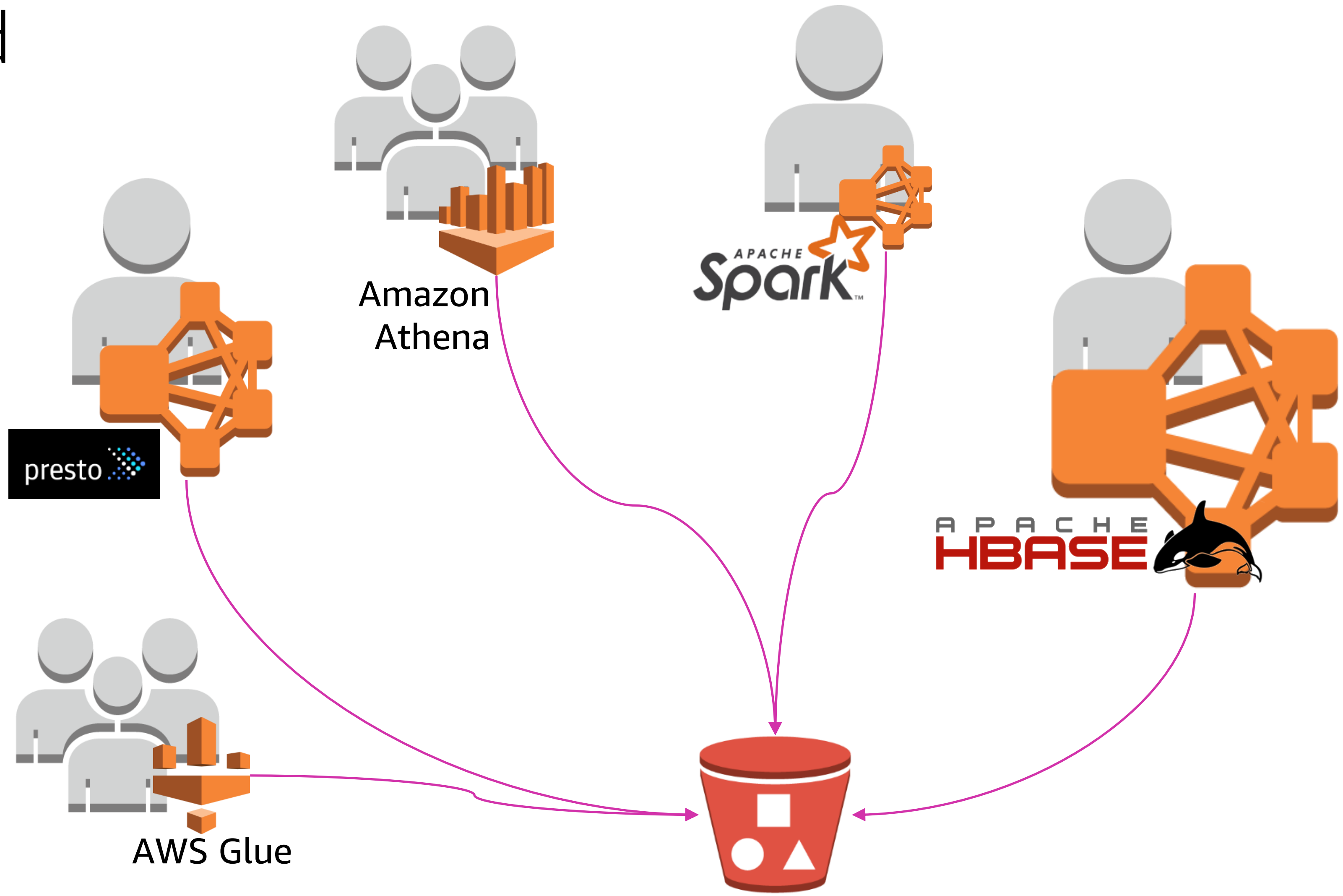
Typical Hadoop environment

- CPU/memory tied directly to disk
- Multiple users competing for resources
- Difficult to upgrade

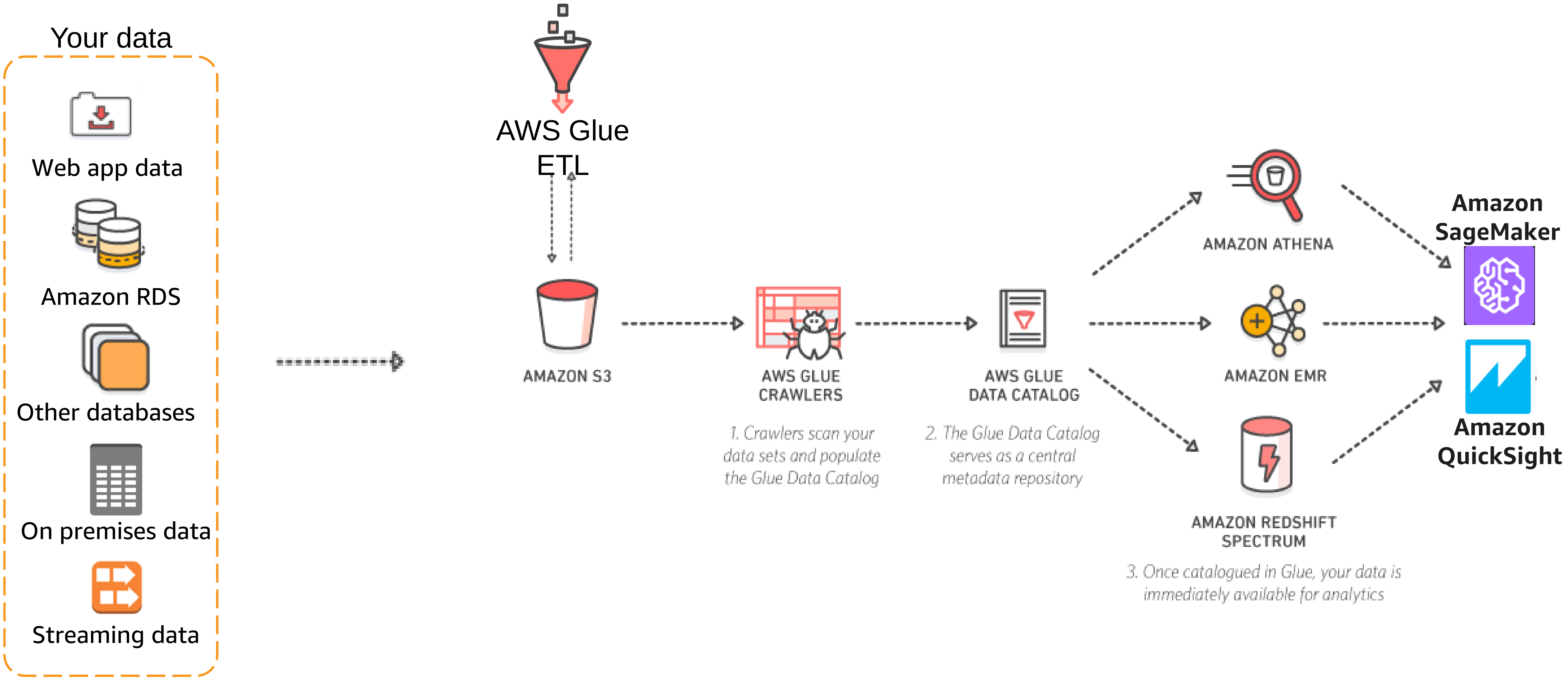


Decoupled storage and compute

- Scale specific to each workload
- Optimize for CPU/memory requirements
- Amazon EMR benefits
  - Spin clusters up/down
  - Easily test new versions with same data
  - Utilize Spot for reduced cost



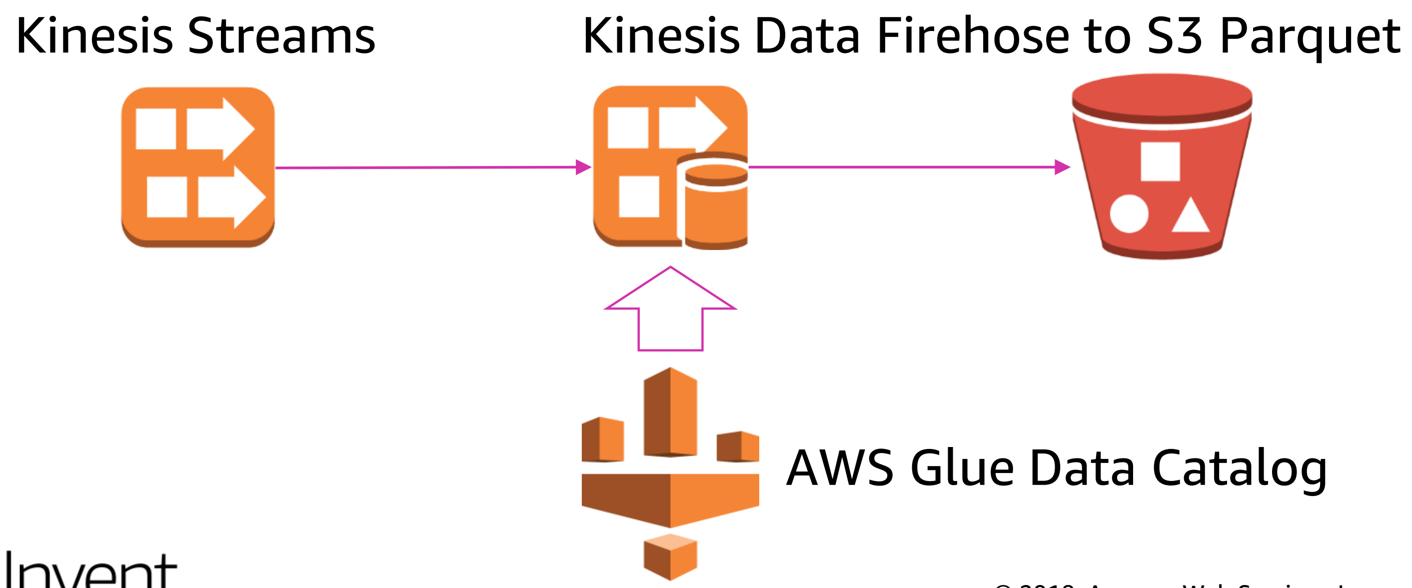
Data lake on AWS



Ingestion

Real-time ingestion

- May 2018: Amazon Kinesis Data Firehose adds support for Parquet and ORC
- One schema per Kinesis Data Firehose, integrates with AWS Glue Data Catalog
- Automatically uses an Amazon S3 prefix in “YYYY/MM/DD/HH” UTC time format
    - Caveat: Partitions not automatically managed in AWS Glue



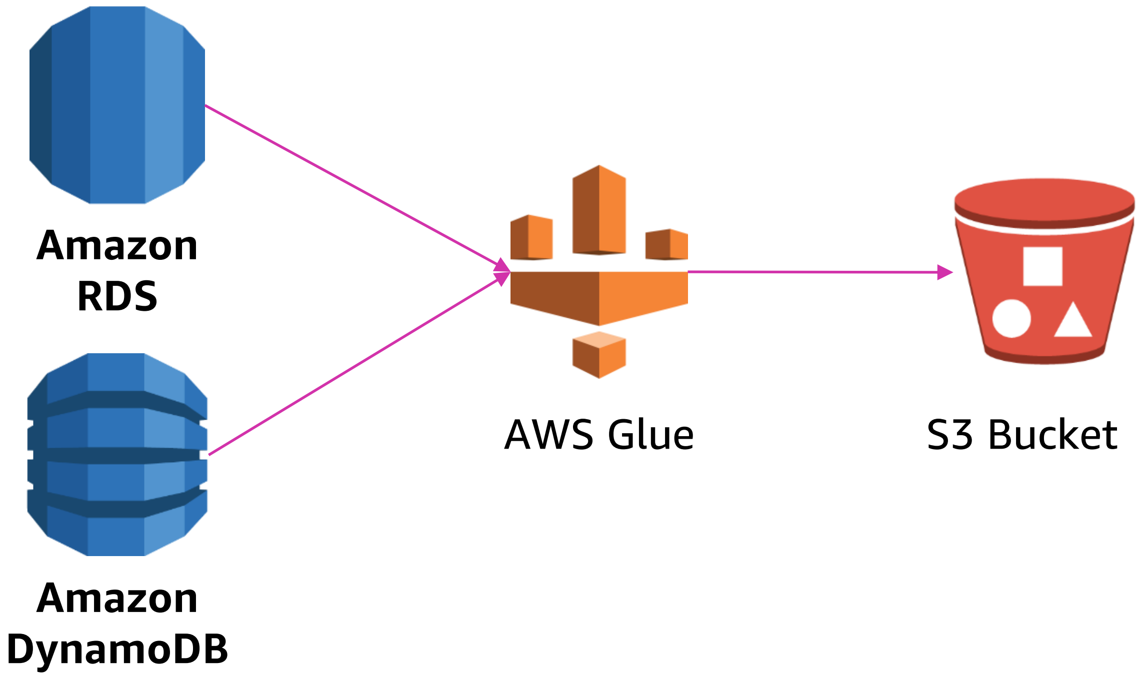
Spark on Amazon EMR for splitting generic data

```
def createContext():
  sc = SparkContext.getOrCreate()
  ssc = StreamingContext(sc, 1)                       # Set a batch interval of 1 second
  ssc.checkpoint("s3://<bucket>/<checkpoint>")  # Enable _Spark_ checkpointing in S3
  lines = KinesisUtils.createStream(
    ssc, "Strlogger", "dcortesi-logger",          # _Kinesis_ checkpointing in "StrLogger" DymamoDB table
    "https://kinesis.us-east-1.amazonaws.com", "us-east-1",
    InitialPositionInStream.LATEST, 2
  )
  parsedDF = lines.map(lambda v: map_json(v)).  # map_json creates standard schema with nested JSON
  parsedDF.foreachRDD(save_rdd)
  return ssc


# For each RDD, write it out to S3 partitioned by the log type ("tag") and date ("dt")
def save_rdd(rdd):
  if rdd.count() > 0:
    sqlContext = SQLContext(rdd.context)
    df = sqlContext.createDataFrame(rdd)
    df.write.partitionBy("tag", "dt").mode("append").parquet("s3://<bucket>/<prefix>/")


ssc = StreamingContext.getOrCreate("s3://<bucket>/<cp>/", lambda: createContext())
ssc.start()
```

https://spark.apache.org/docs/latest/streaming-kinesis-integration.html

Database sources

- AWS Database Migration Service (AWS DMS)
  - Can extract in semi-real-time to S3 in change data capture format
- Export from DB
  - AWS Glue JDBC connections or native DB utilities
  - Crawl using AWS Glue



Third-party sources

- Still need some tooling to extract from third-party sources like Segment/Mixpanel
- Some sources provide a JDBC connector (Salesforce)
- Similar to DB sources
    - Export → S3 → Catalog with AWS Glue
    - If needed, iteratively process using AWS Glue bookmarks

Example structures

Kinesis Data Firehose to Parquet	YYYY/MM/DD/HH/name-TS-UUID.parquet
Log data	<pre>/<logtype>/dt=YYYY-MM-DD/json.gz

DB exports	<pre>/<schema>/<db>/<table>/Y-m-dTH/
	<pre>/<table>/LOAD00[1-9].CSV
	<pre>/<table>/<timestamp>.csv

Third-party exports	<pre>/full_export/latest-YMD/csv.gz
	<pre>/incremental/YYYY/MM/DD/csv.gz

Data prep

Business drivers

- “Hey, can you just add this one data source real quick?”

- Question everything/understand the business drivers
  - Do you really need real-time access?
  - What business decisions are you making?
  - What fields are important to you?
  - What questions are you trying to answer?

- The quickest way to a data swamp is by trying to tackle everything at one time. You won’t understand why any data is needed or important, and worse, neither will your customers.

Optimize your data

- Most raw data will *not* be optimized for querying or might also contain sensitive data
  - There will be CSVs ☺
  - There will be multiple date/time formats
  - Carefully grant access to raw
  - Architect for idempotency

- Storage tiers: raw → landing → production
  1. Optimized file format
  2. Partitioned data
  3. Masked data

1. Optimized file format

- ✓ Compress
- ✓ Compact
- ✓ Convert

Compaction	Number of files	Run time
SELECT count(*) FROM lineitem	5000 files	8.4 seconds
SELECT count(*) FROM lineitem	1 file	2.31 seconds
Speedup		3.6x faster

Conversion	File format	Number of Files	Size	Run time
SELECT count(*) FROM events	json.gz	46,182	176 GB	463.33 seconds
SELECT count(*) FROM events	snappy parquet	11,640	213 GB	6.25 seconds
Speedup				74x faster

** Tests run using Athena*

Row and column formats

ID	Age	State
123	20	CA
345	25	WA
678	40	FL
999	21	WA

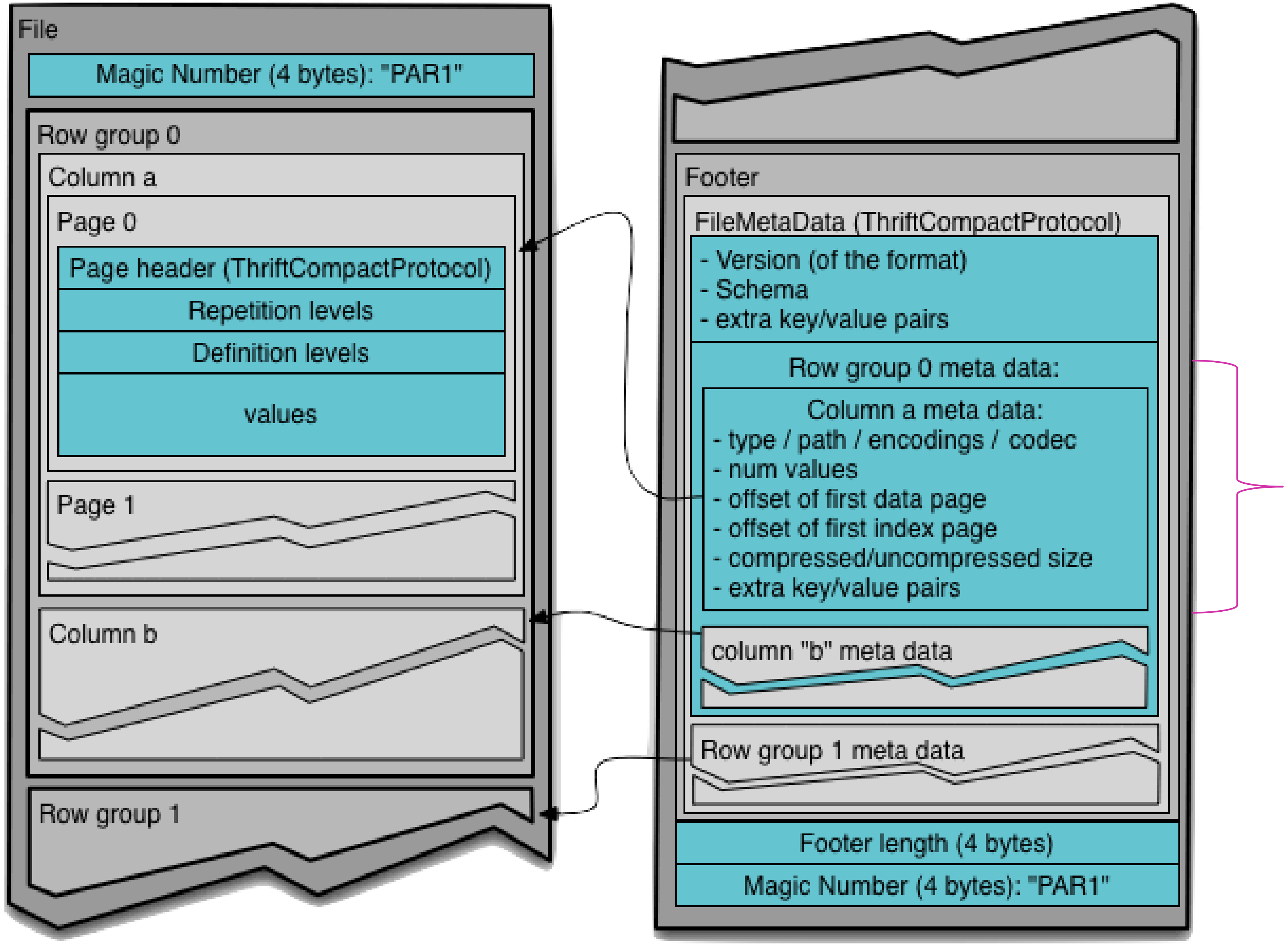
Row format

123	20	CA	345	25	WA	678	40	FL	999	21	WA

Column format

123	345	678	999	20	25	40	21	CA	WA	FL	WA

Parquet file format



Row group metadata allows Parquet reader to skip portions of, or all, files

Parquet challenges

- Schema evolution, particularly with nested data
  - https://github.com/prestodb/presto/pull/6675 (2016)
  - https://github.com/prestodb/presto/pull/10158 (March 2018)

- Can't use predicate pushdowns with nested data
- Strings are just binary – can lead to performance issues in extreme cases

- Use `parquet-tools` to debug, investigate your Parquet files
  - github.com/apache/parquet-mr/tree/master/parquet-tools
  - Most recent version doesn't show string min/max stats (patch)

AWS re:Invent

aws

Flatten data using AWS Glue DynamicFrame

```
datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = "dcortesi",
    table_name = "events_relationalize",
    transformation_ctx = "datasource0"
)

datasource1 = Relationalize.apply(
    frame = datasource0,
    staging_path = glue_temp_path,
    name = dfc_root_table_name,
    transformation_ctx = "datasource1"
)
flattened_dynf = datasource1.select(dfc_root_table_name)

(
    flattened_dynf.toDF()
    .repartition("date")
    .write
    .partitionBy("date")
    .option("maxRecordsPerFile", OUTPUT_LINES_PER_FILE)
    .mode("append")
    .parquet(EXPORT_PATH)
)
```

AWS re:Invent

https://aws.amazon.com/blogs/big-data/simplify-querying-nested-json-with-the-aws-glue-relationalize-transform/

Flatten data using AWS Glue DynamicFrame

```
{
    "player": {
        "username": "user1",
        "characteristics": {
            "race": "Human",
            "class": "Warlock",
            "subclass": "Dawnblade",
            "power": 300,
            "playercountry": "USA"
        },
        "arsenal": {
            "kinetic": {
                "name": "Sweet Business",
                "type": "Auto Rifle",
                "power": 300,
                "element": "Kinetic"
            },
            "energy": {
                "name": "MIDA Mini-Tool",
                "type": "Submachine Gun",
                "power": 300,
                "element": "Solar"
            }}}}
```

```
{
    "player.username": "user1",
    "player.characteristics.race": "Human",
    "player.characteristics.class": "Warlock",
    "player.characteristics.subclass": "Dawnblade",
    "player.characteristics.power": 300,
    "player.characteristics.playercountry": "USA",
    "player.arsenal.kinetic.name": "Sweet Business",
    "player.arsenal.kinetic.type": "Auto Rifle",
    "player.arsenal.kinetic.power": 300,
    "player.arsenal.kinetic.element": "Kinetic",
    "player.arsenal.energy.name": "MIDA Mini-Tool",
    "player.arsenal.energy.type": "Submachine Gun",
    "player.arsenal.energy.power": 300,
    "player.arsenal.energy.element": "Solar"
}
```

parquet-tools output

Row count Total size Data type

Compression

Column stats

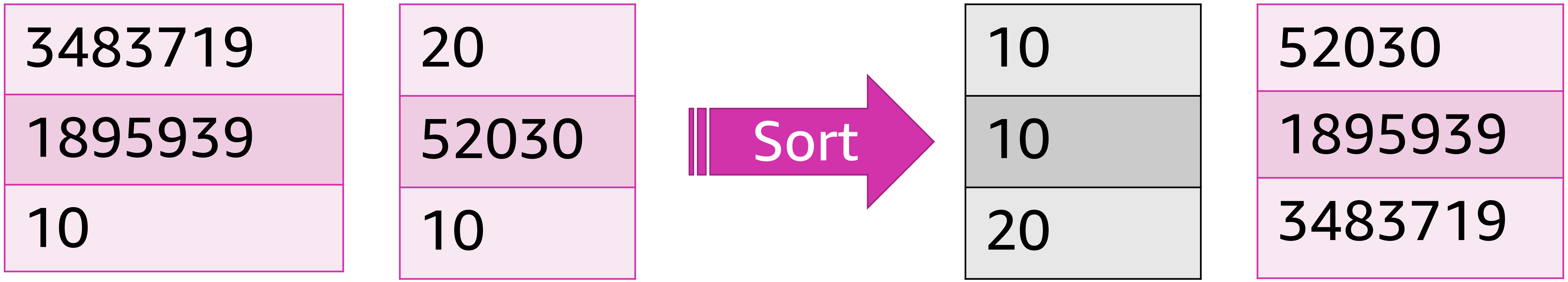
```
row group 1:     RC:177 TS:16692 OFFSET:4
--------------------------------------------------------------------------------
operation:        BINARY SNAPPY DO:0 FPO:4 SZ:250/315/1.26 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[min: REST.COPY.OBJECT, max:
REST.PUT.OBJECT, num_nulls: 0]
bytes_sent:       BINARY SNAPPY DO:0 FPO:254 SZ:436/512/1.17 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[min: 1161, max: 8, num_nulls:
34]
object_size:      BINARY SNAPPY DO:0 FPO:3882 SZ:134/130/0.97 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[min: 0, max: 79, num_nulls:
134]
remote_ip:        BINARY SNAPPY DO:0 FPO:4016 SZ:323/354/1.10 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[min: 10.1.2.3, max: 54.2.4.5,
num_nulls: 0]
bucket:           BINARY SNAPPY DO:0 FPO:7270 SZ:126/122/0.97 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[min: dcortesi-test-us-west-2,
max: dcortesi-test-us-west-2, num_nulls: 0]
time:             INT96 SNAPPY DO:0 FPO:8230 SZ:518/739/1.43 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[no stats for this column]
user_agent:       BINARY SNAPPY DO:0 FPO:8748 SZ:625/771/1.23 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[min: , aws-sdk-java/1.11.132
Linux/4.9.76-3.78.amzn1.x86_64 OpenJDK_64-Bit_Server_VM/25.141-b16/1.8.0_141, presto, max: aws-cli/1.14.17 Python/2.7.10 Darwin/16.7.0
botocore/1.8.21, num_nulls: 1]
http_status:      BINARY SNAPPY DO:0 FPO:9373 SZ:160/160/1.00 VC:177 ENC:PLAIN_DICTIONARY,RLE,BIT_PACKED ST:[min: 200, max: 404, num_nulls:
0]
```

AWS re:Invent

Optimizing columnar even more

- 300GB dataset in S3, but only retrieving one row per Athena query
- Used AWS Glue to sort entire dataset before writing to Parquet
- Parquet reader can determine whether to skip an entire file
- Cost optimization

```
WHERE id = 52030
```

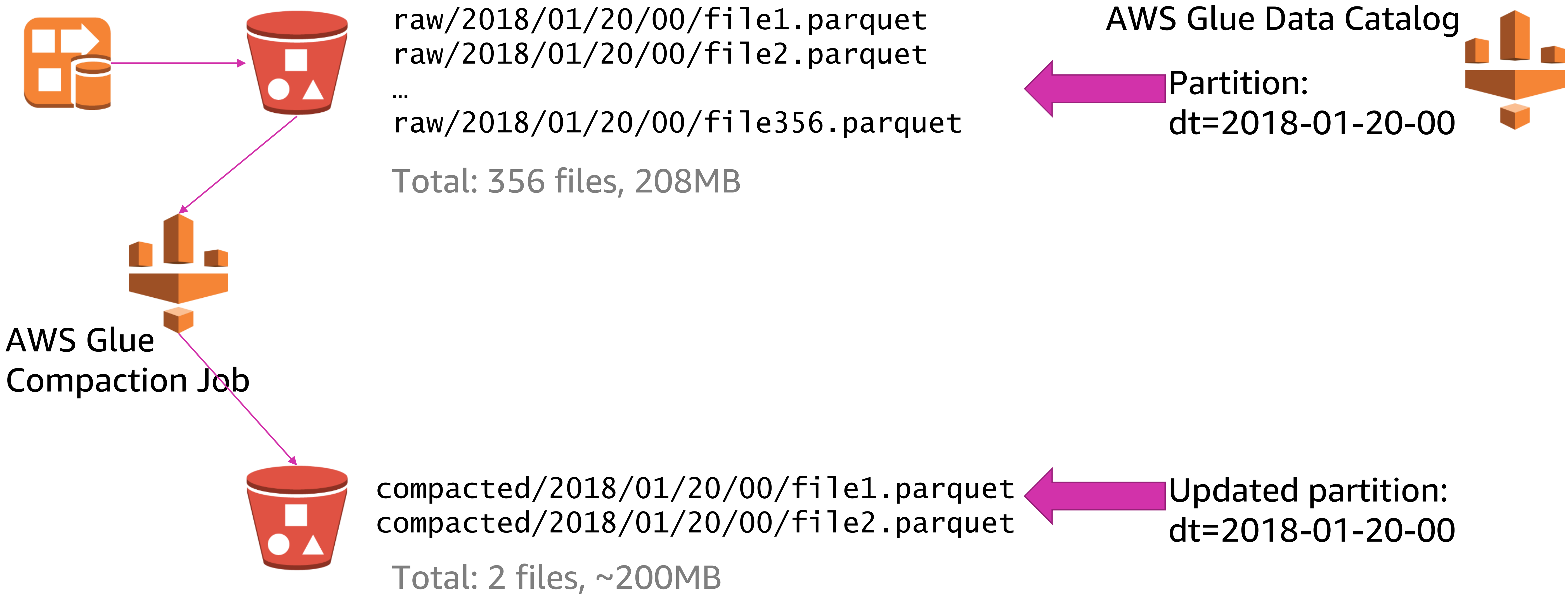


AWS re:Invent

Compact small files

- Regular “janitor” job that uses Spark to copy and compact data from one prefix into another
    - Swap AWS Glue Data Catalog partition location
- If you know your dataset …
    - Spark ”repartition” into optimal sizes
- Brute force it!
    - S3 list, group objects based on size
    - repartition(<some_small_number>) + “maxRecordsPerFile”

Compact small files



2. Partitioned data

- “Be careful when you repartition the data so you don’t write out too few partitions and exceed your partition rate limits” 😫

- “Be careful when you repartition the data (in Spark) do you don’t write out too few (Hive-style) partitions and exceed your (S3) partition rate limits”

Partitions, partitions, partitions—Hive-style

- S3 prefix convention
    - s3://<bucket>/<prefix>/year=2018/month=01/day=20/
    - S3://<bucket>/<prefix>/dt=2018-01-20/
- Cost and performance optimization
- Enables predicate pushdown
- Design based on query patterns
- Don't go overboard

```
SELECT remote_ip,
COUNT(*)
FROM access_logs
WHERE dt='2018-01-20'
```

https://docs.aws.amazon.com/athena/latest/ug/partitions.html

Partitions, partitions, partitions—S3

S3 automatically partitions based on key prefix

- Bucket: service-logs
- Partition: service-logs/2
- Impact at (per prefix)
- ~3,500 PUT/LIST/DELETE requests/second
- ~5,500 GET requests/second

Object keys:

2018/01/20/data001.csv.gz
2018/01/20/data002.csv.gz
2018/01/20/data003.csv.gz
2018/01/21/data001.csv.gz
2018/01/21/data002.csv.gz
2018/01/22/data001.csv.gz

https://docs.aws.amazon.com/AmazonS3/latest/dev/request-rate-perf-considerations.html

Partitions, partitions, partitions—Spark

- “Logical chunk of a large distributed data set”
- Enables optimal distributed performance
- You can repartition – or redistribute – data in Spark
  - Explicit number of partitions
  - Partition by one or more columns (defaults to 200 partitions)
    - `spark.sql.shuffle.partitions`
- Spark can read/write Hive-style partitions

```
write.partitionBy("date").parquet("s3://<bucket>/<prefix>/")
```

Data and workflow management

- Emerging area – lots of active development
- Apache Atlas (1.0 – June 2018) - https://atlas.apache.org/
- Netflix Metacat (June 2018) - https://github.com/Netflix/metacat
- LinkedIn WhereHows (2016) - https://github.com/linkedin/WhereHows
- AWS Glue (2016) - https://aws.amazon.com/glue/
- FINRA Herd (2015) - http://finraos.github.io/herd/

- Apache Airflow - https://airflow.apache.org/
- Spotify Luigi - https://luigi.readthedocs.io/
- Apache Oozie - http://oozie.apache.org/

Data security

- Limit access to data on S3 using IAM policies
- Encrypt data at rest using S3 or AWS Key Management Service (AWS KMS)-managed keys
- Amazon EMR integrates with LDAP for authentication
- Amazon Macie for discovering sensitive data
- Enable S3 access logs for auditing
    - Bonus – Query your S3 access logs on S3 using Athena!

Monitoring

- Track latency on your different data sources/pipelines
    - Alarm on data freshness to avoid data drift
    - Can use AWS Glue metadata properties or external system
- CloudWatch integrates with AWS Glue/Amazon EMR/Athena
- Build trust
    - Notice data latency before your customers
    - Data validation – "Is that metric right?"
- When something goes wrong, how will you recover?

Overview

- So far we've …
- Crawled and discovered our raw datasets
- Converted data into Parquet
- Organized data into structures that make sense for our business
- Compacted small files into larger ones
- Optimized the heck out of Parquet
- Partitioned our data when it needs to be

- So what does this look like?

Storage tiers—Raw

```
raw-bucket/
|___ apache-logs/
    |___ dt=2018-01-20-00/
        |___ log-2018-01-20T00:01:00Z.log
        |___ log-2018-01-20T00:01:03Z.log
        |___ log-2018-01-20T00:02:10Z.log
        |___ log-2018-01-20T00:20:00Z.log
|
|___ syslog-logs/
    |___ dt=2018-01-20-02/
        |___ syslog-2018-01-20T02:00:00Z.gz
        |___ syslog-2018-01-20T02:15:00Z.gz
|
|___ kinesis-events/
    |___ 2018
         |___ 01
              |___ 20
                   |___ 00
                        |___ events-01:15.parquet
                        |___ events-01:16.parquet
```

```
raw-bucket/
|___ db-backups/
     |___ prod-webapp/
          |___ 2018-01-19/
               |___ users_20180120.tsv.gz
               |___ products_20180120.tsv.gz
               |___ plans_20180120.tsv.gz
               |___ orgs_20180120.tsv.gz
          |___ 2018-01-18/
               |___ users_20180119.tsv.gz
               |___ products_20180119.tsv.gz
               |___ plans_20180119.tsv.gz
               |___ orgs_20180119.tsv.gz
```

Storage tiers—Landing

```
raw-processed-bucket/
|___ log-data/
     |___ apache/
          |___ dt=2018-01-20/
               |___ part00000.snappy.parquet
               |___ part00001.snappy.parquet
     |___ syslog/
          |___ dt=2018-01-20/
               |___ part00000.snappy.parquet
|
|___ product/
     |___ orgs/
          |___ part00000.snappy.parquet
     |___ plans/
          |___ part00000.snappy.parquet
     |___ events/
          |___ dt=2018-01-20/
               |___ part00000.snappy.parquet
               |___ part00001.snappy.parquet
```

Storage tiers—Production model

```
production-marketing-bucket/
|___ product/
     |___ users/
          |___ part00000.snappy.parquet
     |___ events/
          |___ dt=2018-01-20/
               |___ part00000.snappy.parquet
               |___ part00001.snappy.parquet
|___ email-analytics/
     |___ email-clicks/
          |___ dt=2018-01-20/
               |___ part00000.snappy.parquet
```

```
production-marketing-bucket/
|___ social-analytics/
     |___ reach-dashboard/
          |___ aggregates-by-date/
               |___ dt=2018-01-20/
                    |___
part00000.snappy.parquet
          |___ aggregates-by-region/
               |___ region=na/
                    |___ dt=2018-01-20/
                         |___ part00000.parquet
               |___ region=apac/
                    |___ dt=2018-01-20/
                         |___ part00000.parquet
```

Optimizing for analytics

AWS re:Invent

aws

Optimizing for analytics

- Data prep optimizations apply
  - Compress, compact, convert
  - Tailor to your customer's use case
  - It's OK to duplicate data

- Create production data models
  - Use case dependent – usually by organization
  - Same data is typically represented different ways depending on need
  - Always running COUNT/MIN/MAX queries? Create aggregates!

Application-specific tuning and use cases

- “What database should I use for our data lake?”
- “Yes.”

- Athena
- Amazon EMR
- AWS Glue
- Amazon QuickSight
- Amazon SageMaker
- Amazon Redshift Spectrum

Thank you!

Damon Cortesi

Please complete the session survey in the mobile app.