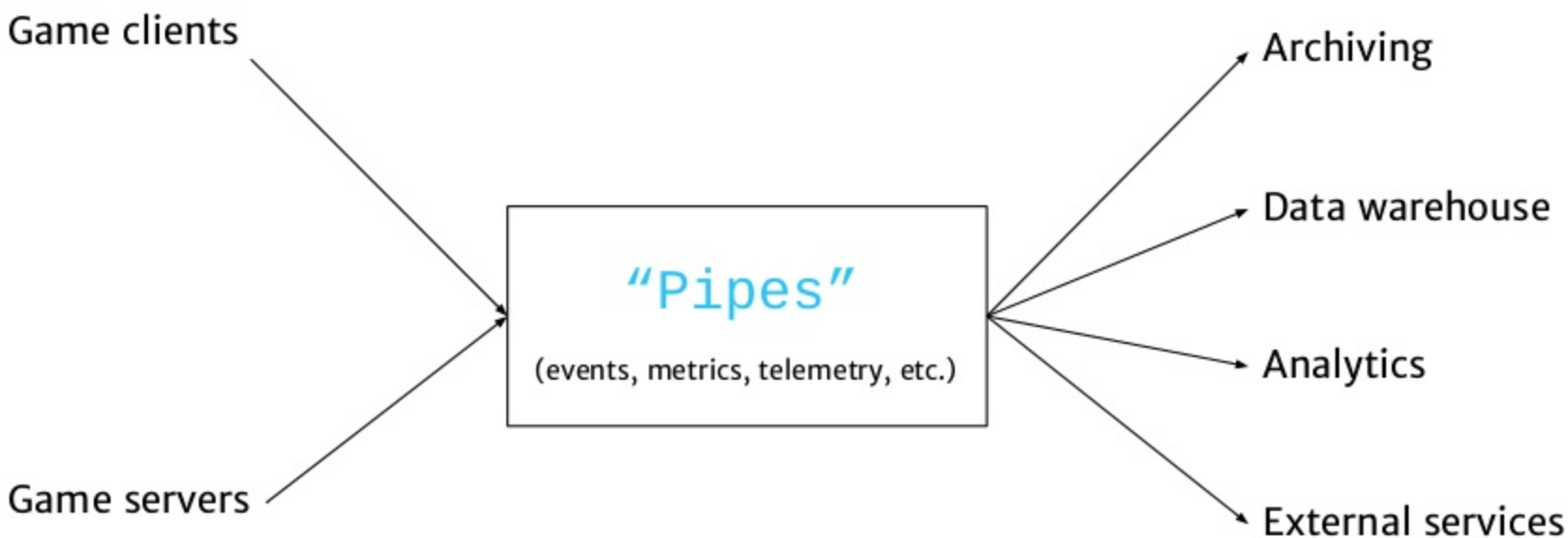
Querying Data Pipeline with AWS Athena

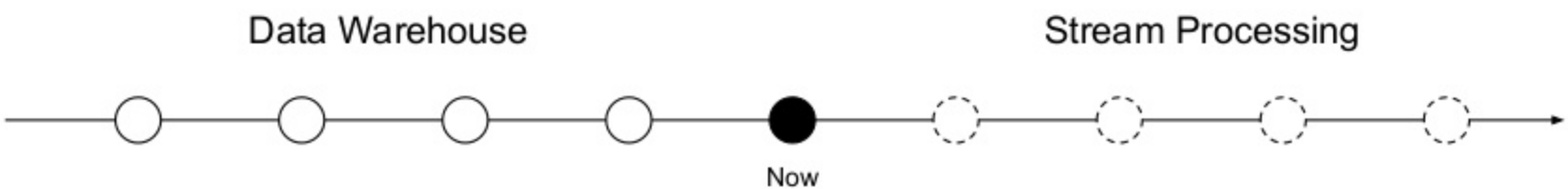
demonware

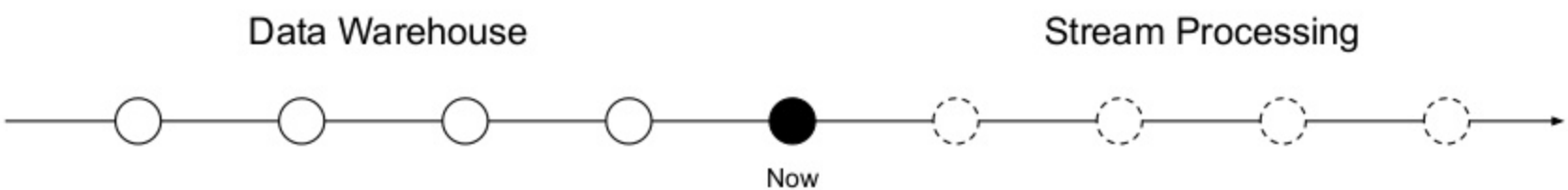
Yaroslav Tkachenko, Senior Software Engineer

demonware

Our main titles


Game clients
Game servers
"Pipes"
(events, metrics, telemetry, etc.)
Archiving
Data warehouse
Analytics
External services


Data Warehouse
Stream Processing
Now

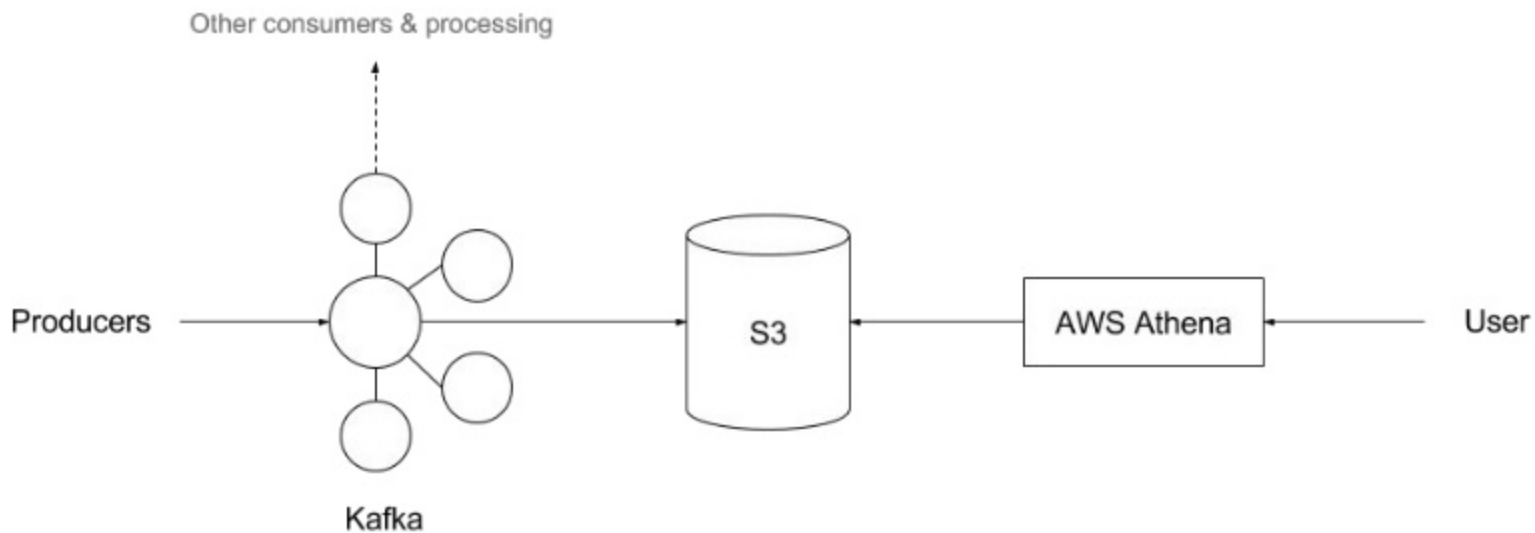


- Need raw data
- Don’t want to support complex infrastructure
- Retention is usually short

- Not an option - want to query historical data

AWS Athena

Amazon Athena is an interactive query service that makes it easy to analyze data in Amazon S3 using standard SQL. Athena is serverless, so there is no infrastructure to manage, and you pay only for the queries that you run.


Other consumers & processing
Producers
Kafka
S3
AWS Athena
User

Building blocks

- AWS S3 – persisting streaming data
- Schema definition – Apache Hive[1] DDL for describing schemas
- Query language – Presto[2] (ANSI-compatible) SQL for querying

[1] Apache Hive – distributed data warehouse software

[2] Presto – distributed SQL query engine

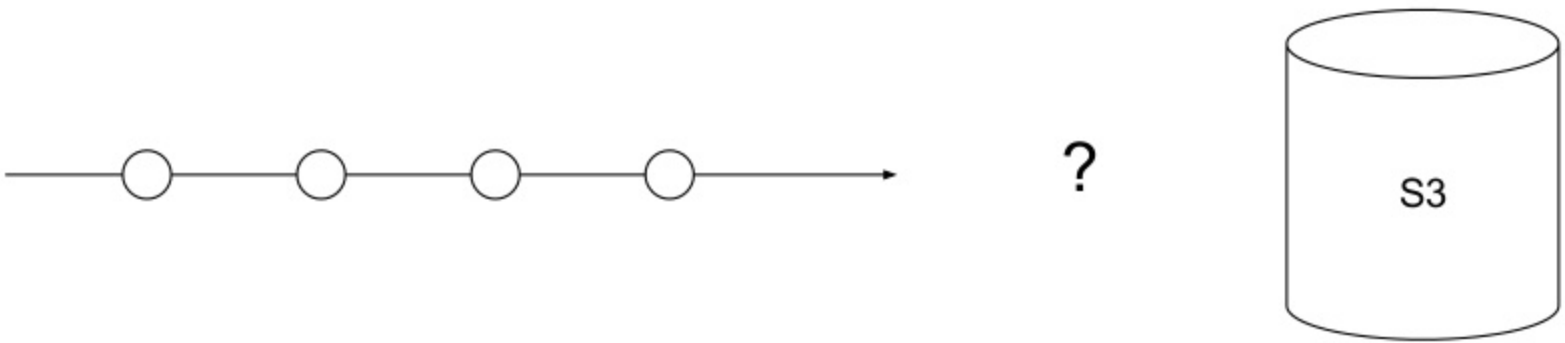
AWS S3

Supported formats

- JSON
- CSV
- Avro
- Parquet
- ORC
- Apache Web Server logs
- Logstash Grok
- CloudTrail

Supported compression

- Snappy
- Zlib
- GZIP
- LZO


?
S3

Streaming data to S3

Non-AWS pipelines:

- Kafka -> Kafka Connect, Secor
- ActiveMQ, RabbitMQ, etc. -> Camel
- ??? -> Custom Consumer

AWS pipelines:

- Lambda <- SQS
- Kinesis Stream -> ~~Lambda~~ -> Kinesis Firehose

Kinesis Pipeline

1. **Kinesis Stream** as an input
2. ~~**Lambda** to forward to Firehose and transform (optional)~~
3. **Kinesis Firehose** as a buffer (size or time), compression and another transformation (optional, using Lambda)

Schema definition

Apache Hive Data Definition Language (DDL) is used for describing tables and databases:

ALTER DATABASE SET DBPROPERTIES
ALTER TABLE ADD PARTITION
ALTER TABLE DROP PARTITION
ALTER TABLE RENAME PARTITION
ALTER TABLE SET LOCATION
ALTER TABLE SET TBLPROPERTIES
CREATE DATABASE
CREATE TABLE
DESCRIBE TABLE

DROP DATABASE
DROP TABLE
MSCK REPAIR TABLE
SHOW COLUMNS
SHOW CREATE TABLE
SHOW DATABASES
SHOW PARTITIONS
SHOW TABLES
SHOW TBLPROPERTIES
VALUES

```
CREATE EXTERNAL TABLE table_name (
  id STRING
  data STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.ql.io.parquet.serde.ParquetHiveSerDe'
LOCATION 's3://bucket-name/'
tblproperties ("parquet.compress"="SNAPPY")
```

TINYINT, SMALLINT, INT, BIGINT

BOOLEAN

DOUBLE, DECIMAL

STRING

BINARY

TIMESTAMP

DATE (not supported for Parquet)

VARCHAR

ARRAY, MAP, STRUCT

```
{
    "metadata": {
        "client_id": "21353253123",
        "timestamp": 1497996200,
        "category_id": "1"
    },
    "payload": "g3ng0943g93490gn3094"
}
```

```
CREATE EXTERNAL TABLE events (
  metadata struct<client_id:string,
                  timestamp:timestamp,
                  category_id:string
                >,
  payload string
)
ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
LOCATION 's3://events/'
```

Query language

Presto SQL is used for querying data:

```
SELECT [ ALL | DISTINCT ] select_expression [, ...]
[ FROM from_item [, ...] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
[ HAVING condition ]
[ UNION [ ALL | DISTINCT ] union_query ]
[ ORDER BY expression [ ASC | DESC ] [ NULLS FIRST | NULLS LAST] [, ...] ]
[ LIMIT [ count | ALL ] ]
```

SELECT data FROM events where headers.user_id = 123 order by headers.timestamp limit 10;

SELECT os, COUNT(*) count FROM cloudfront_logs WHERE date BETWEEN date '2014-07-05' AND date '2014-08-05' GROUP BY os;

SELECT customer.c_name, lineitem.l_quantity, orders.o_totalprice FROM lineitem, orders, customer WHERE lineitem.l_orderkey = orders.o_orderkey AND customer.c_custkey = orders.o_custkey;

Best practices

Why?

Good performance → Low cost!

Dataset	Size on Amazon S3	Query Run time	Data Scanned	Cost
Logs stored as Text files	1 TB	237 seconds	1.15TB	$5.75
Logs stored in Apache Parquet format*	130 GB	5.13 seconds	2.69 GB	$0.013
Savings	**87% less with Parquet**	**34x faster**	**99% less data scanned**	**99.7% cheaper**

Partitioning

- Find good partitioning field like a date, version, user, etc.
- Update Athena with partitioning schema (use PARTITIONED BY in DDL) and metadata
- You can create partitions manually or let Athena handle them (but that requires certain structure)
- But there is no magic! You have to use partitioning fields in queries (like regular fields), otherwise no partitioning is applied

Partitioning

CREATE EXTERNAL TABLE events ...
PARTITIONED BY (year string, month string, day string)

1) SELECT data FROM events WHERE event_id = '98632765';

2) SELECT data FROM events WHERE event_id = '98632765' AND year = '2017' AND month = '06' AND day = '21';

Partitioning

- s3://events/2017/06/20/1.parquet
- s3://events/2017/06/20/2.parquet
- s3://events/2017/06/20/3.parquet
- s3://events/2017/06/21/1.parquet
- s3://events/2017/06/21/2.parquet
- s3://events/2017/06/21/3.parquet
- s3://events/2017/06/22/1.parquet
- ...

Manual partitioning:

- ALTER TABLE events ADD PARTITION (date='2017-06-20') location 's3://events/2017/06/20/'
- ...

Partitioning

- s3://events/date=2017-06-20/1.parquet
- s3://events/date=2017-06-20/2.parquet
- s3://events/date=2017-06-20/3.parquet
- s3://events/date=2017-06-21/1.parquet
- s3://events/date=2017-06-21/2.parquet
- s3://events/date=2017-06-21/3.parquet
- s3://events/date=2017-06-22/1.parquet
- ...

Automatic partitioning:

- MSCK REPAIR TABLE events

Performance tips

- **Use binary formats like Parquet!**
- Don't forget about compression
- Only include the columns that you need
- LIMIT is amazing!
- For more SQL optimizations look at Presto best practices
- **Avoid a lot of small files:**

Query	Number of files	Run time
`SELECT count(*) FROM lineitem`	5000 files	8.4 seconds
`SELECT count(*) FROM lineitem`	1 file	2.31 seconds
Speedup		72% faster

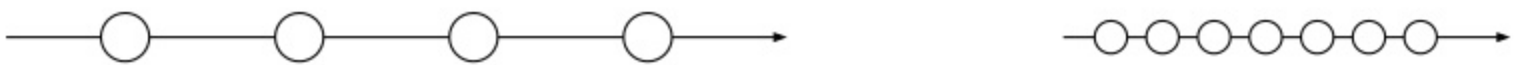
The dilemma

Volume of data ↓ Number and size of files (buffering) ↑ Time to index ↓

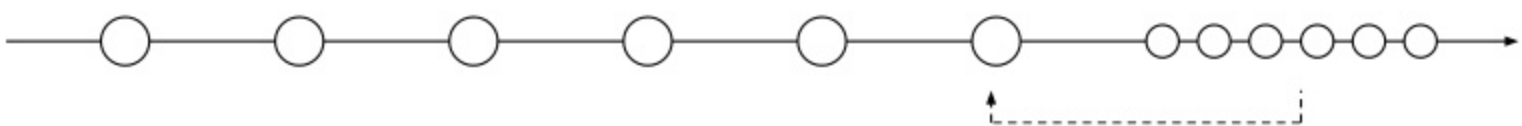
Given certain data volume you want the number of files as less as possible with file sizes as large as possible appear in S3 as soon as possible. It's really hard. You have to give up something.

Possible solutions?

- Don't give up anything! Have two separate pipelines, one with long retention (bigger files) and another one with short retention (smaller files, fast time to index). Cons? Double on size.



- Give up on number of files and size. But! Periodically merge small files in background. Cons? Lots of moving parts and slower queries against fresh data.



Demo

Amazon Product Reviews

http://jmcauley.ucsd.edu/data/amazon/

Summary

- AWS Athena is great, right?!
- Think about the file structure, formats, compression, etc.
- Streaming data to S3 is probably the hardest task
- Don't forget to optimize – use partitioning, look at Presto SQL optimization tricks, etc.
- Good performance means low cost

Questions?

@sap1ens

https://www.demonware.net/careers