GAM401

# AWS re:INVENT

## Designing for the Future: Building a Flexible Event-based Analytics Architecture for Games

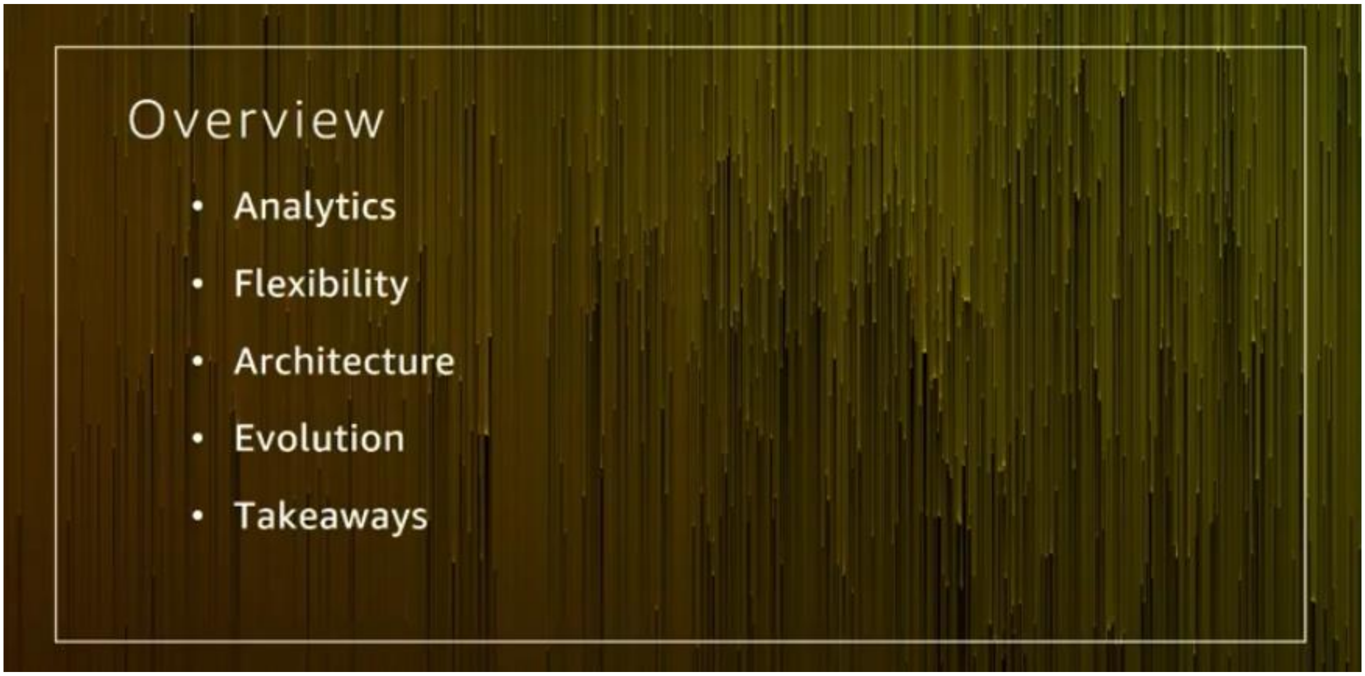
Brent Nash
Senior Software Engineer
Amazon Game Studios

November 27, 2017

re:Invent

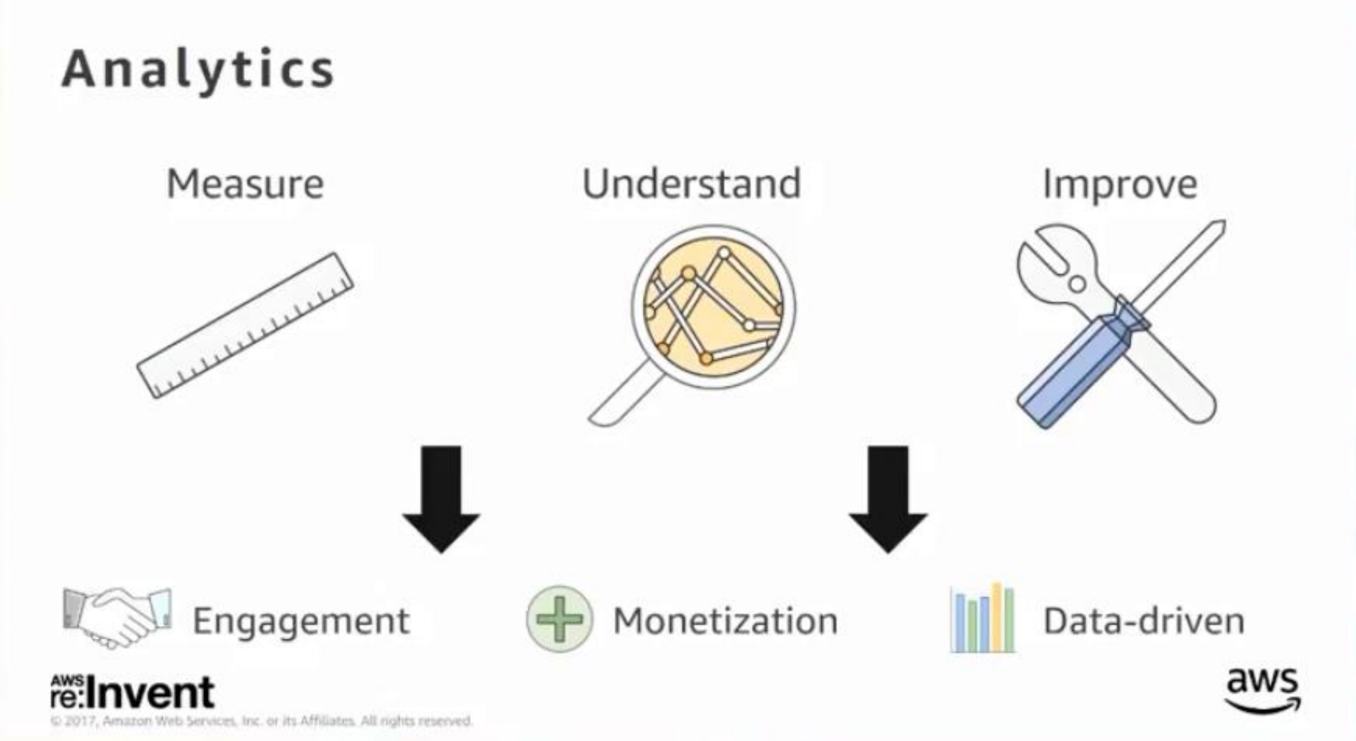
aws

The pace of technology innovation is relentless, especially at AWS. Designing and building new system architectures is a balancing act between using established, production-ready technologies while maintaining the ability to evolve and take advantage of new features and innovations as they become available. In this session, learn how Amazon Game Studios built a flexible analytics pipeline on AWS for their team battle sport game, Breakaway, that provided value on day one, but was built with the future in mind. We will discuss the challenges we faced and the solution we built for ingesting, storing and analyzing gameplay telemetry and dive deep into the technical architecture using AWS many services including Amazon Kinesis, Amazon S3, and Amazon Redshift. This session will focus on game analytics as a specific use case, but will emphasize an overarching focus on designing an architectural flexibility that is relevant to any system.



## Overview

- Analytics
- Flexibility
- Architecture
- Evolution
- Takeaways



## Analytics

Measure

Understand

Improve

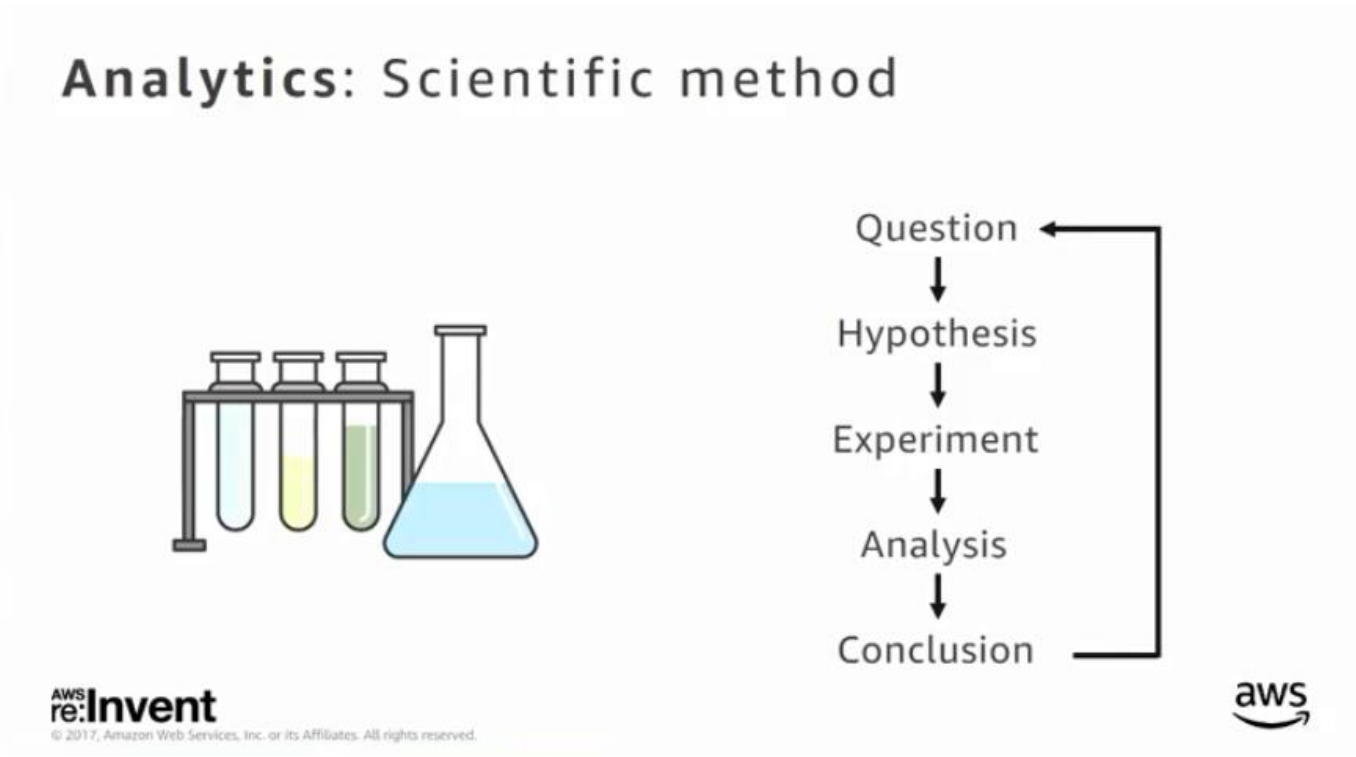
Engagement

Monetization

Data-driven

AWS re:Invent

aws



## Analytics: Scientific method

Question

Hypothesis

Experiment

Analysis

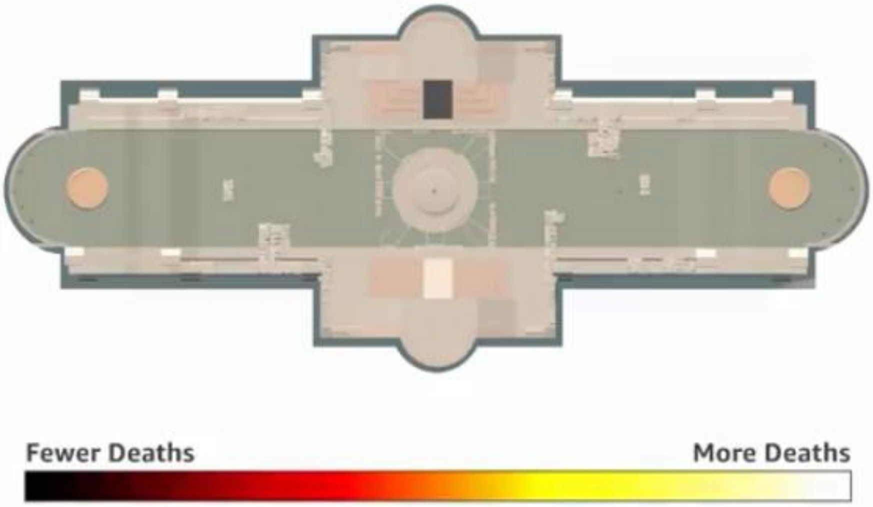
Conclusion

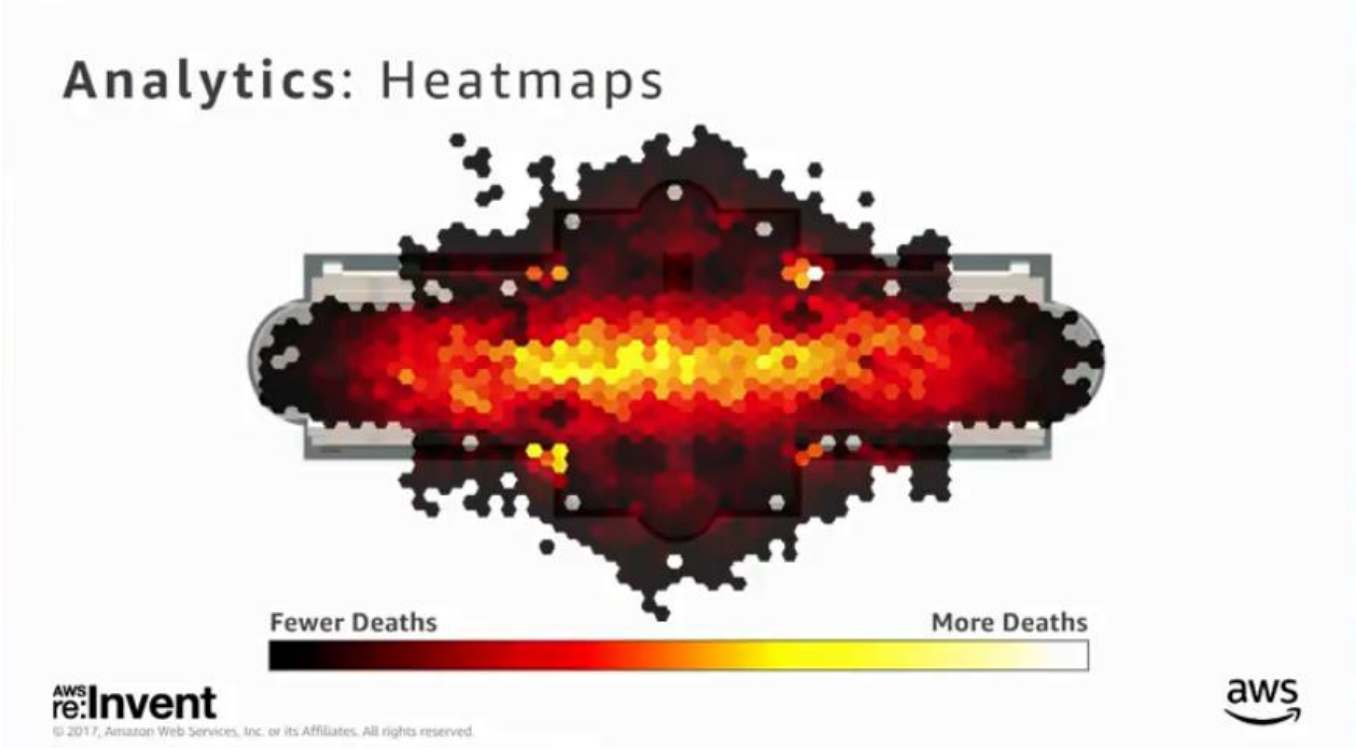
AWS re:Invent

aws

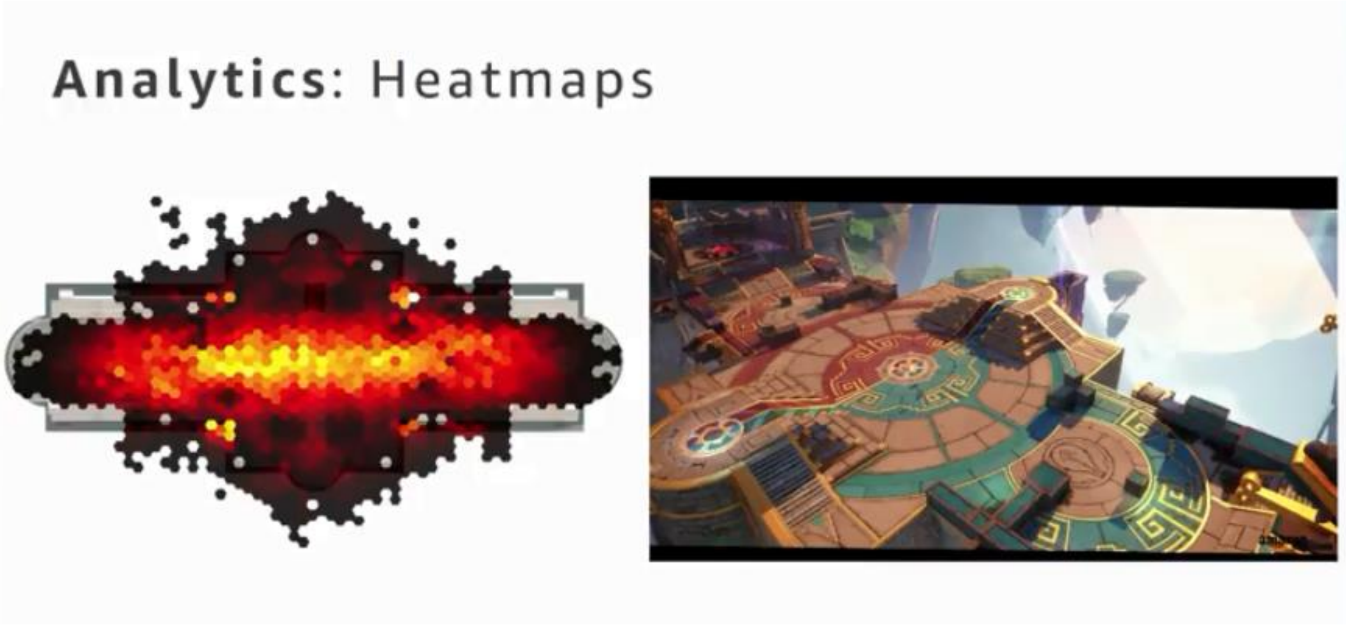
## Analytics: Level design

## Analytics: Heatmaps



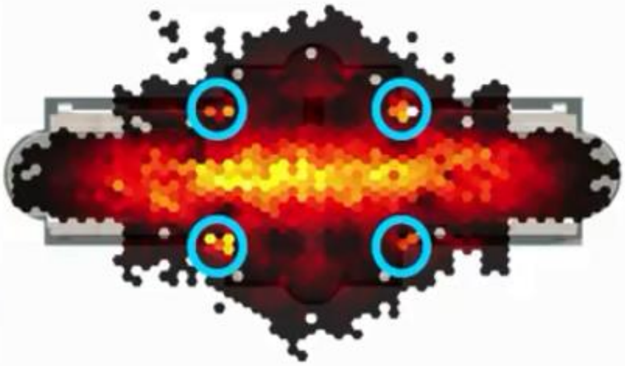


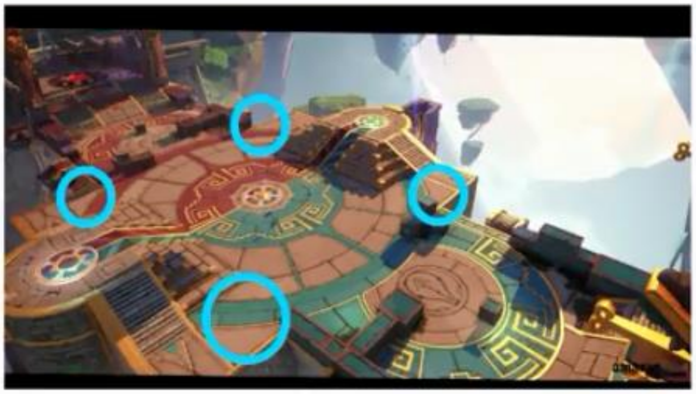
## Analytics: Heatmaps



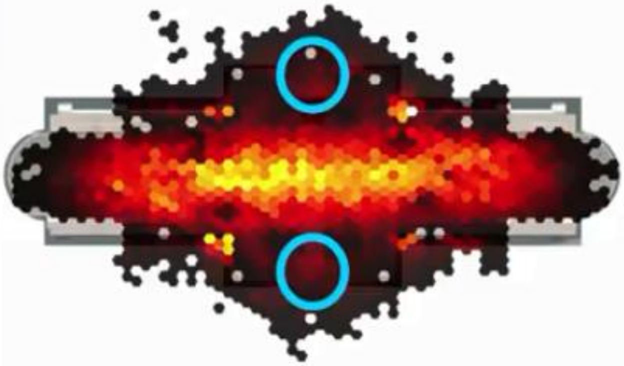
## Analytics: Heatmaps

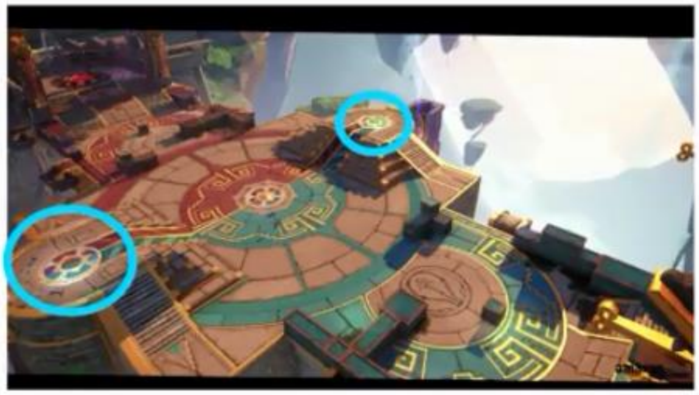
## Analytics: Heatmaps





## Analytics: Heatmaps

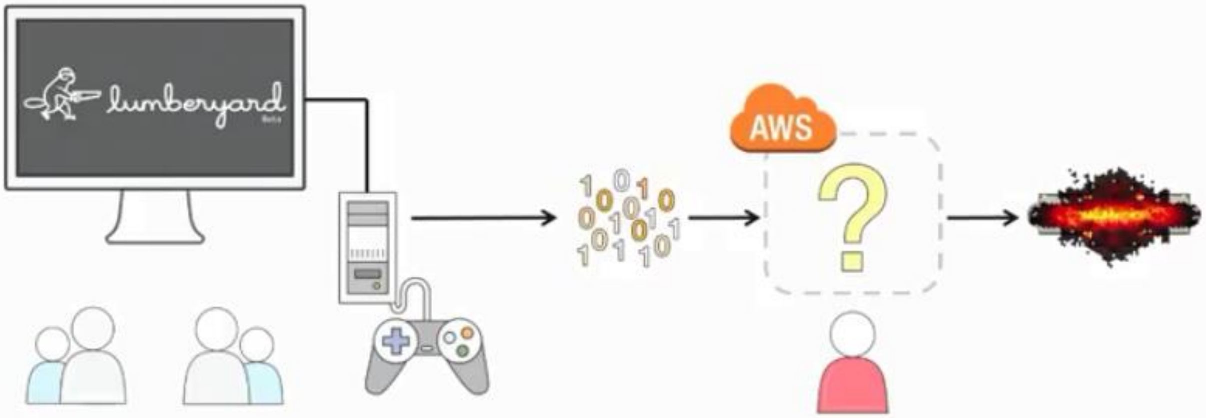




## Flexibility: Background

**Back in 2015...**

## Flexibility: Background



Game prototyping

Analytics prototyping

## Flexibility: Motivation

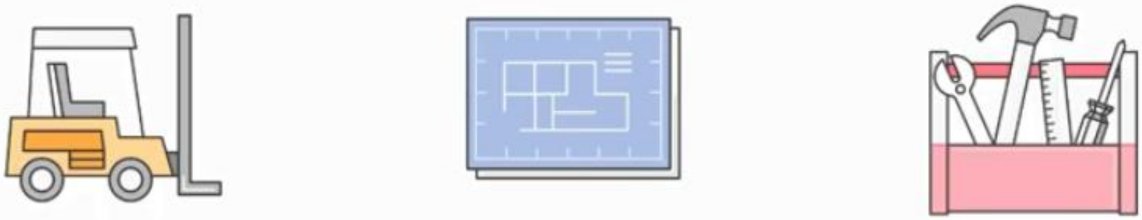
**flex·i·bil·i·ty** **(\ˌflek-sə-ˈbi-lə-tē\)** ***(noun)***

***"Characterized by a ready capability to adapt to new, different, or changing requirements"***

- Ambiguous requirements
- Evolving tech landscape
- Changing requirements
- The "awesome prototype conundrum"

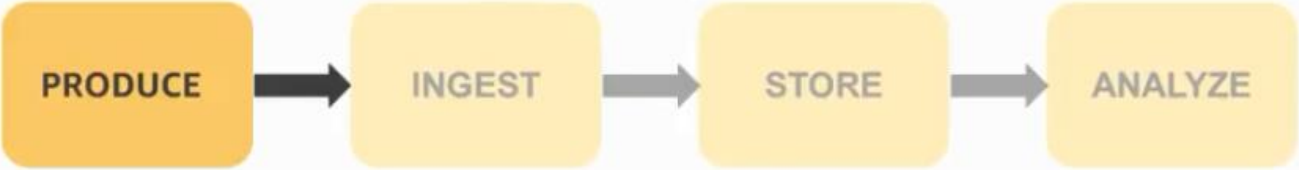
AWS re:Invent

# Architecture

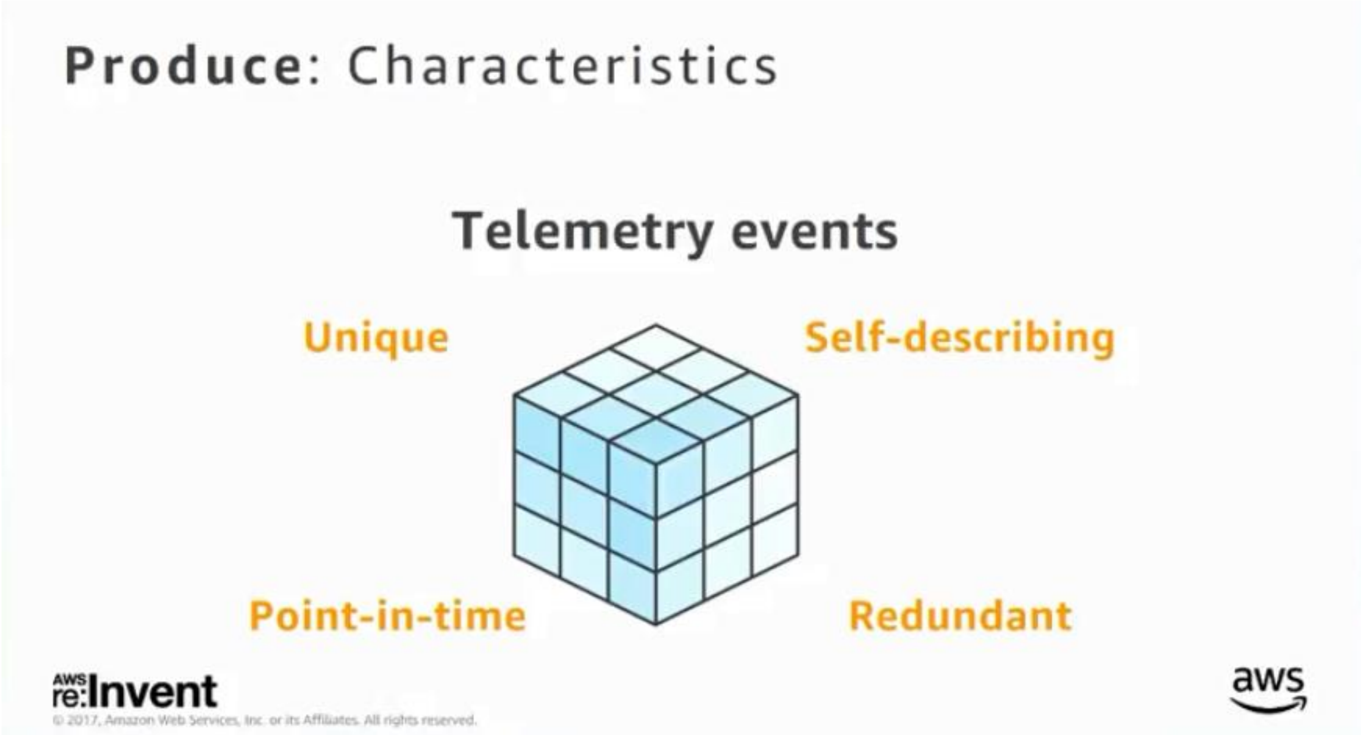


Onward to the architecture...

# Architecture: High-level



PRODUCE → INGEST → STORE → ANALYZE



## Produce: Characteristics

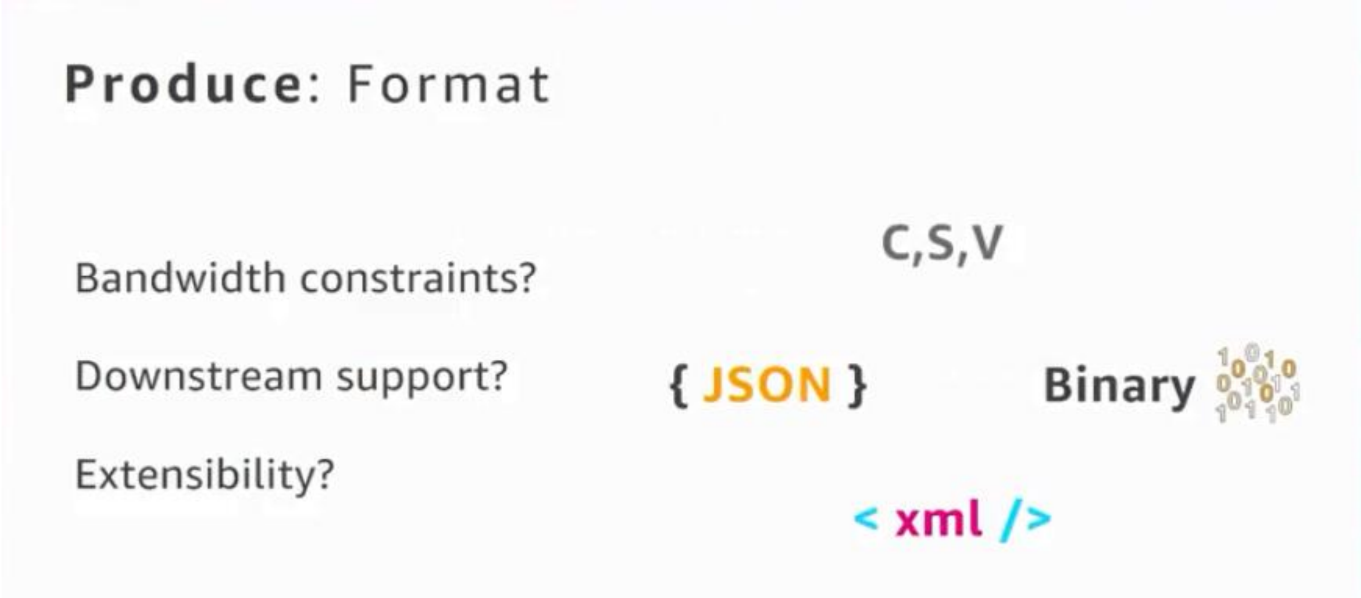
### Telemetry events

Unique

Self-describing

Point-in-time

Redundant



## Produce: Format

Bandwidth constraints?

Downstream support?

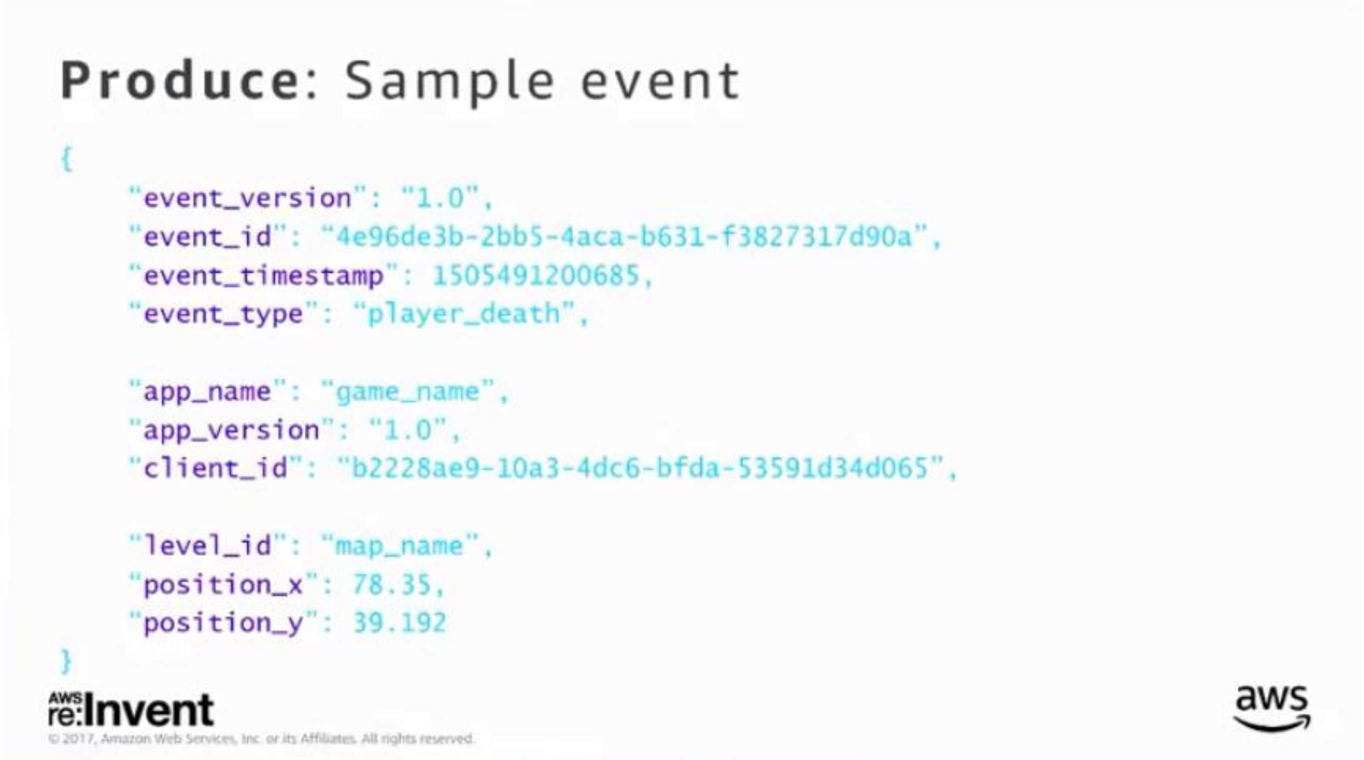
Extensibility?

C,S,V

{ JSON }

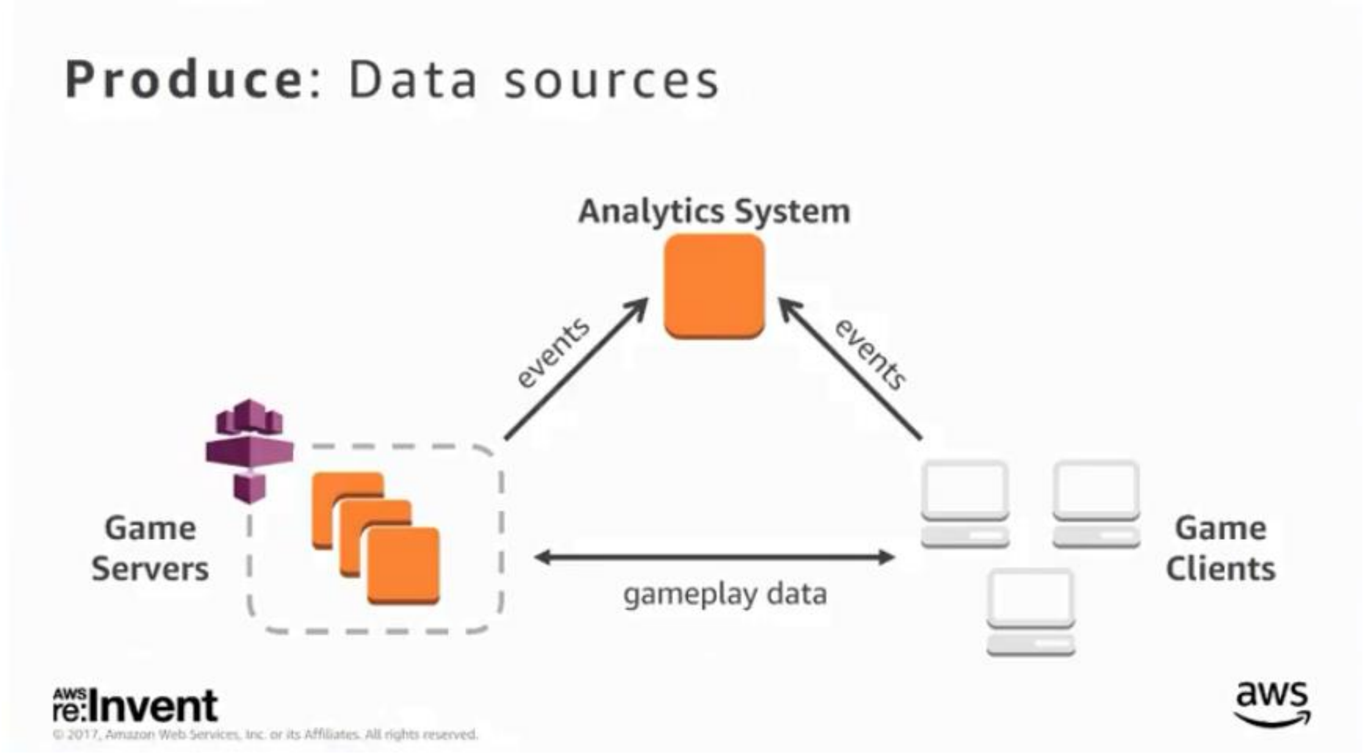
Binary

< xml />

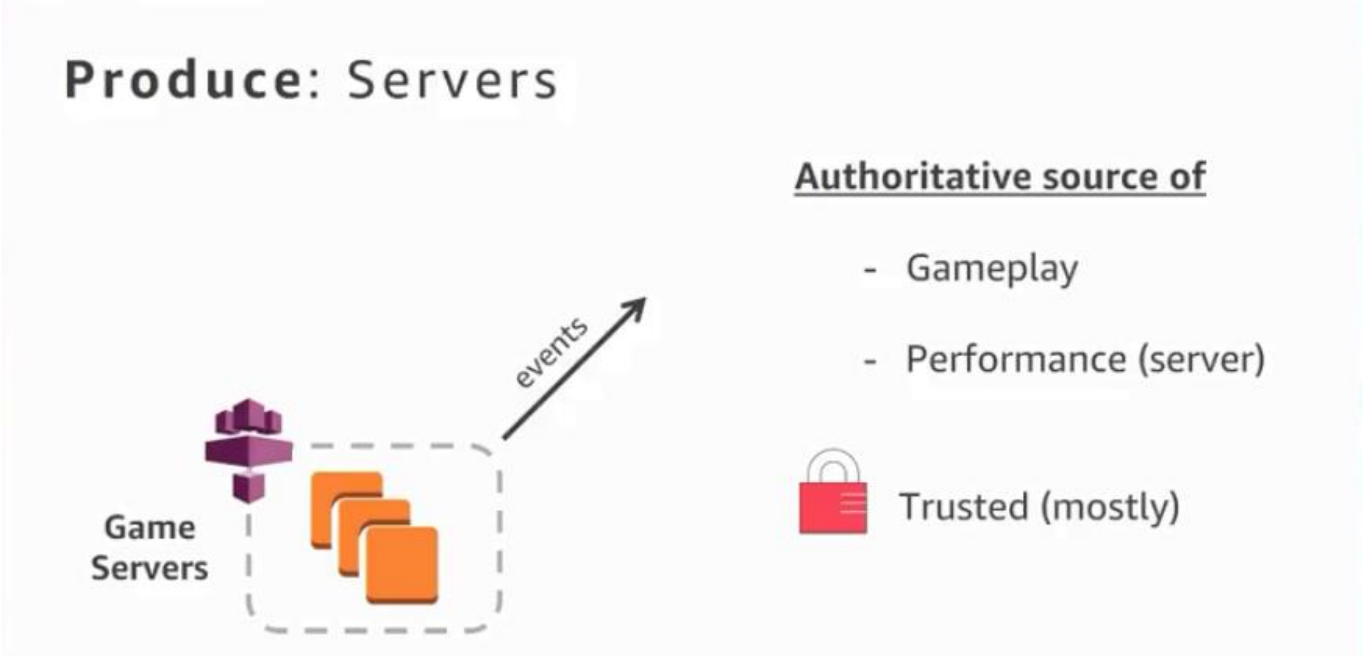


## Produce: Sample event

```
{
    "event_version": "1.0",
    "event_id": "4e96de3b-2bb5-4aca-b631-f3827317d90a",
    "event_timestamp": 1505491200685,
    "event_type": "player_death",

    "app_name": "game_name",
    "app_version": "1.0",
    "client_id": "b2228ae9-10a3-4dc6-bfda-53591d34d065",

    "level_id": "map_name",
    "position_x": 78.35,
    "position_y": 39.192
}
```



## Produce: Data sources

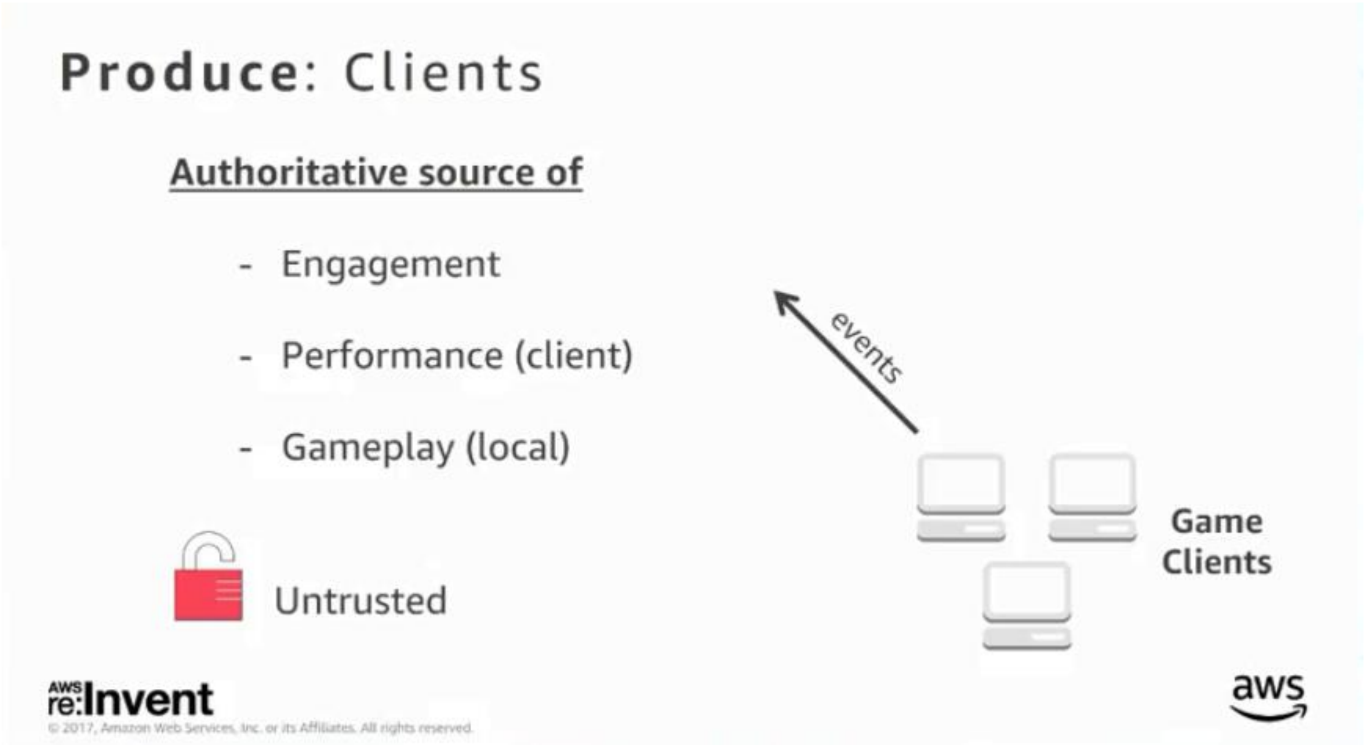


## Produce: Servers

**Authoritative source of**

- Gameplay
- Performance (server)

🔒 Trusted (mostly)



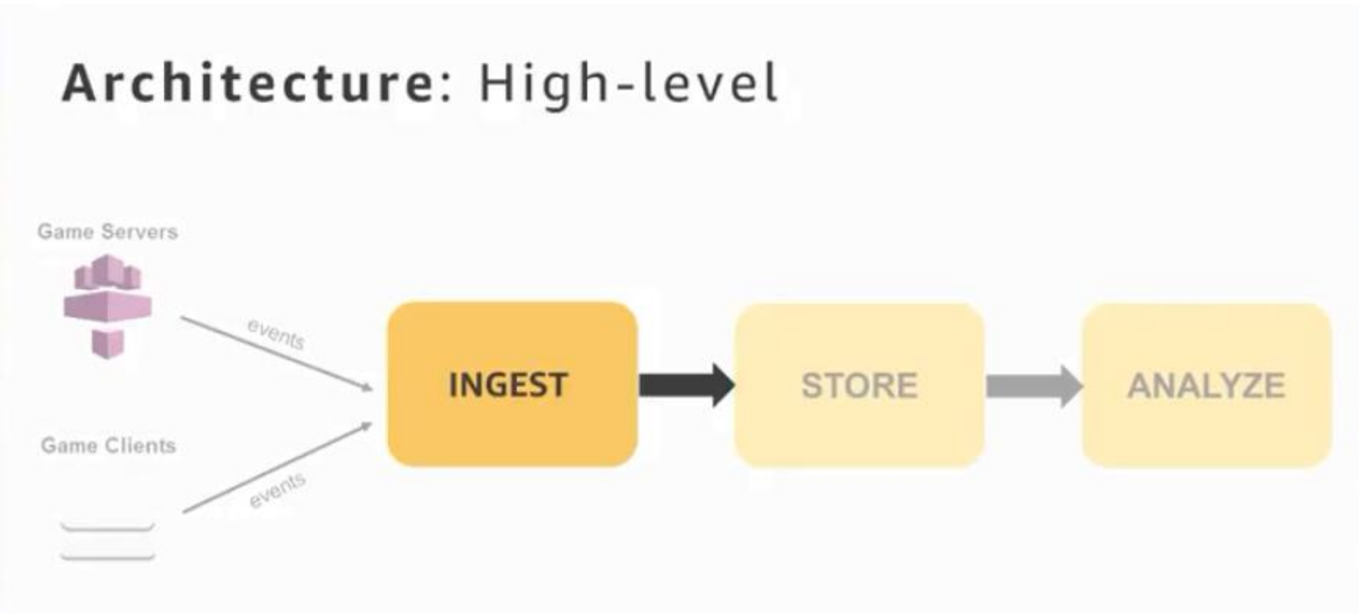
## Produce: Clients

**Authoritative source of**

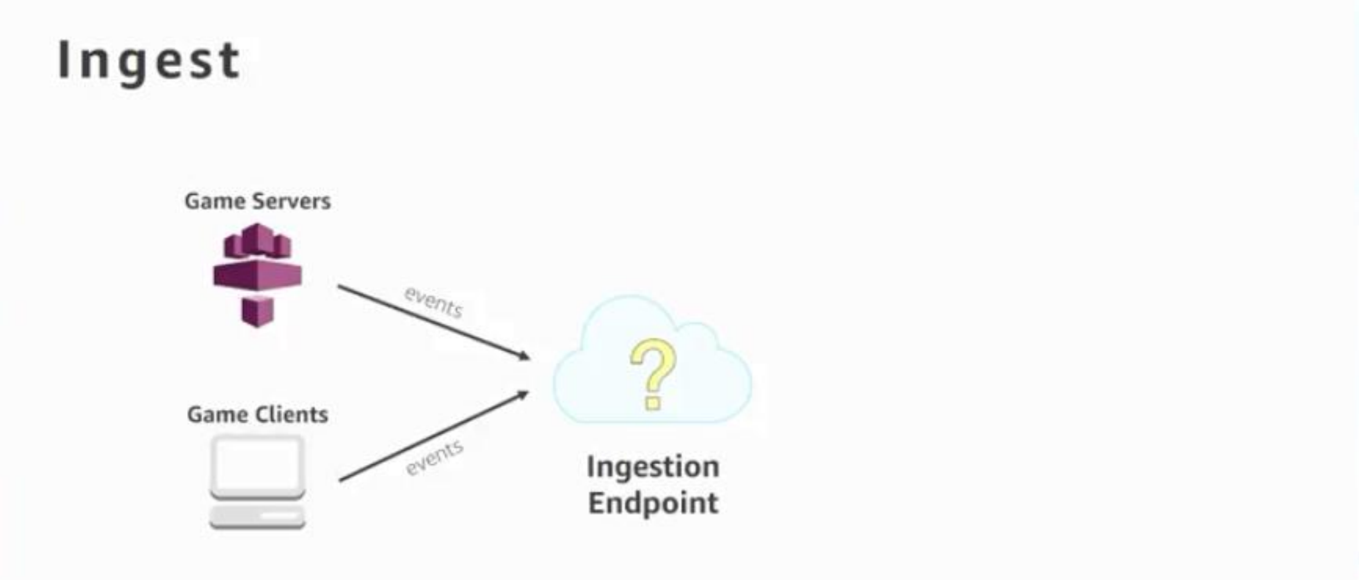
- Engagement
- Performance (client)
- Gameplay (local)

🔓 Untrusted

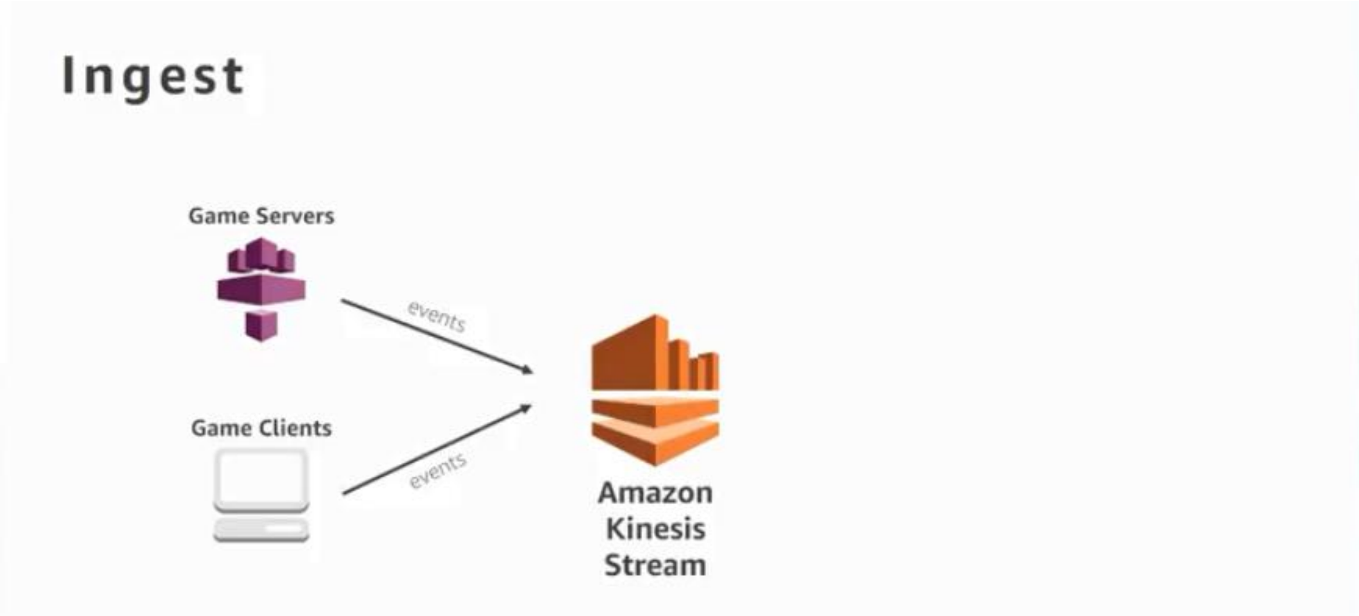
AWS re:Invent



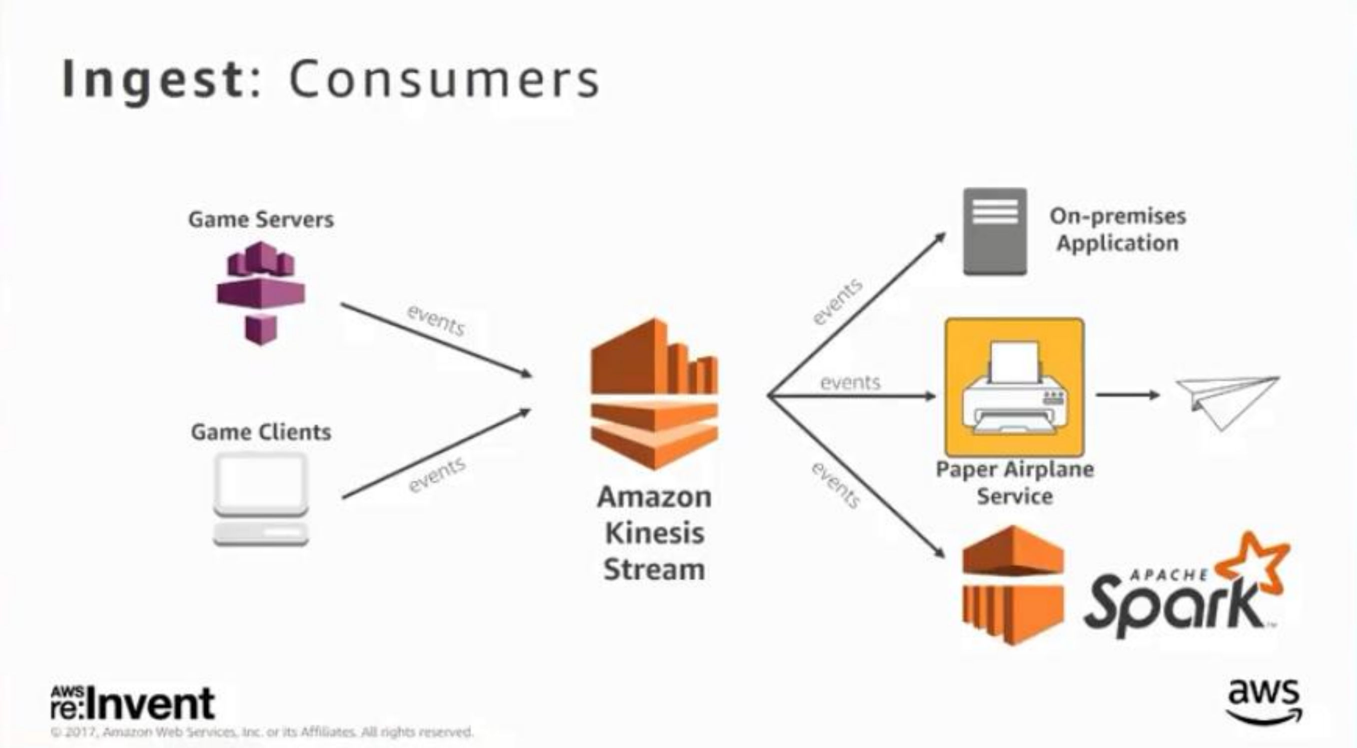
# Architecture: High-level



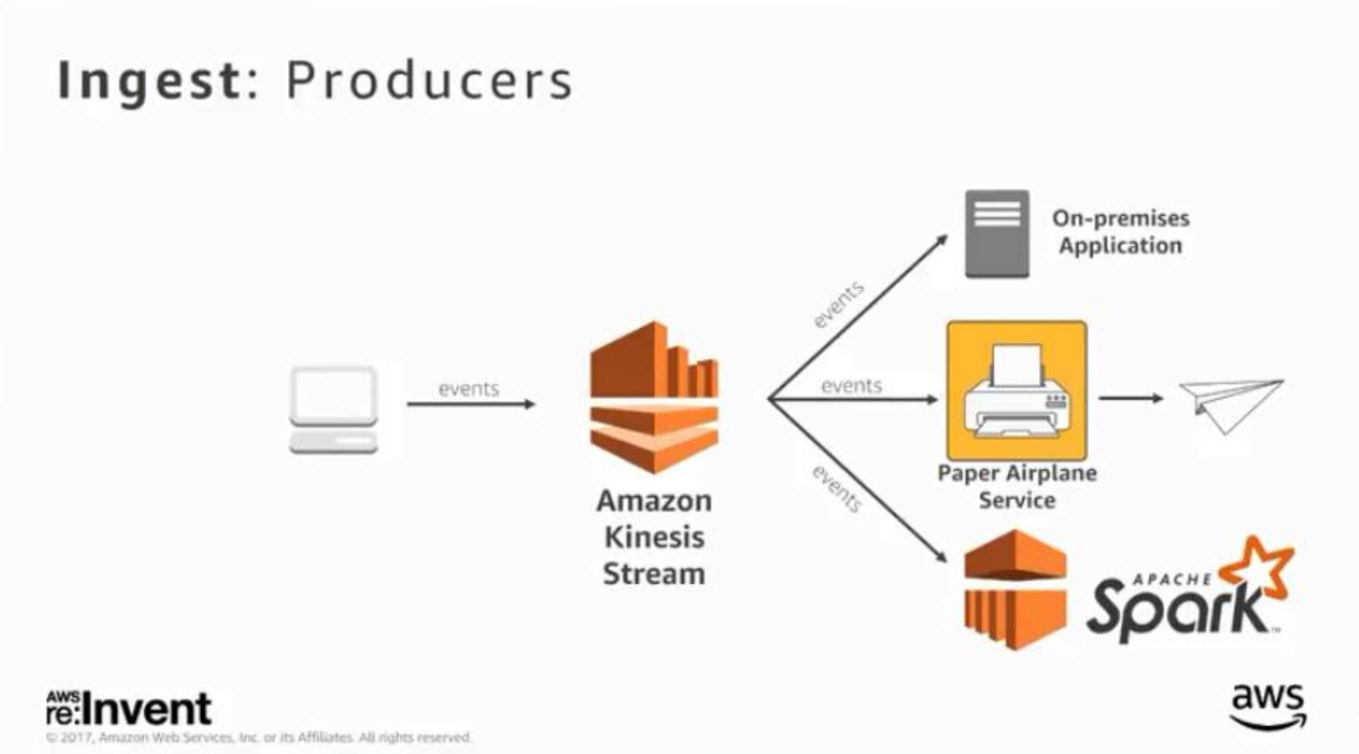
# Ingest



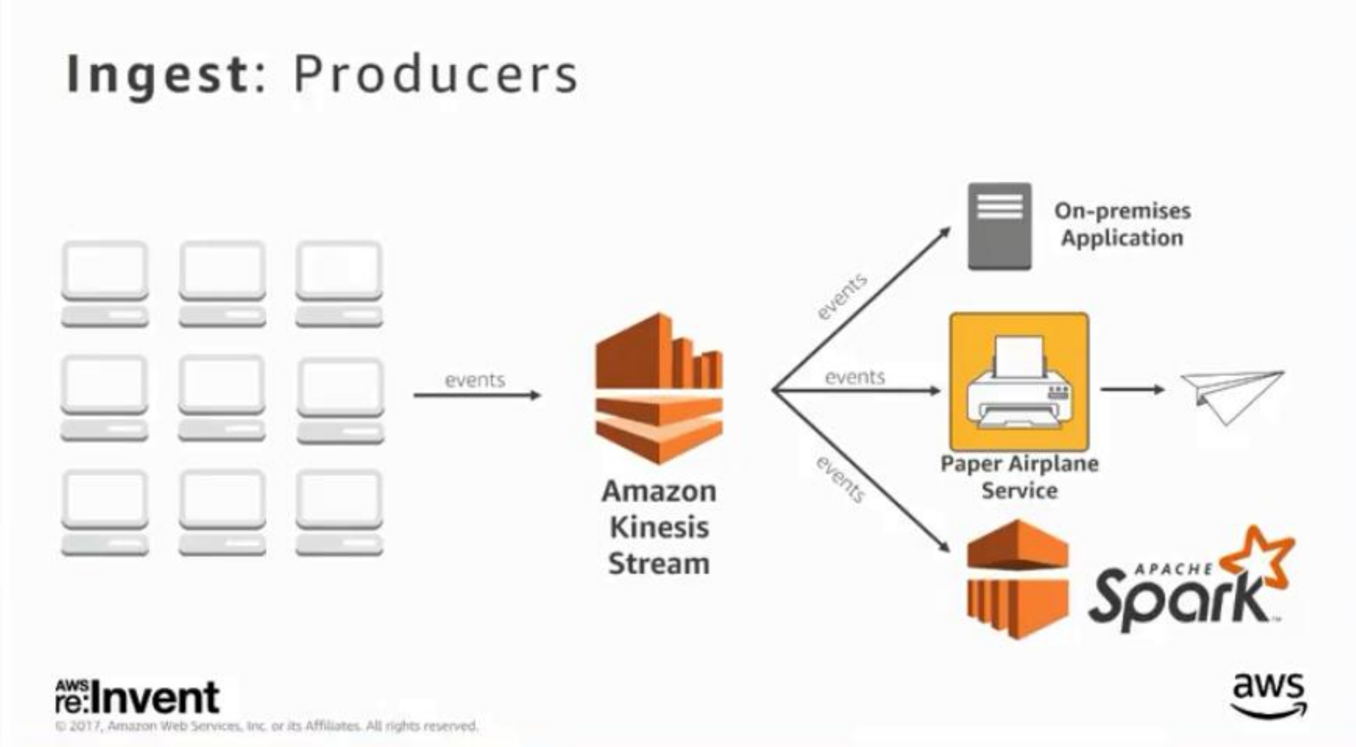
# Ingest



## Ingest: Consumers



## Ingest: Producers



# Ingest: Producers

events

Amazon Kinesis Stream

events

events

events

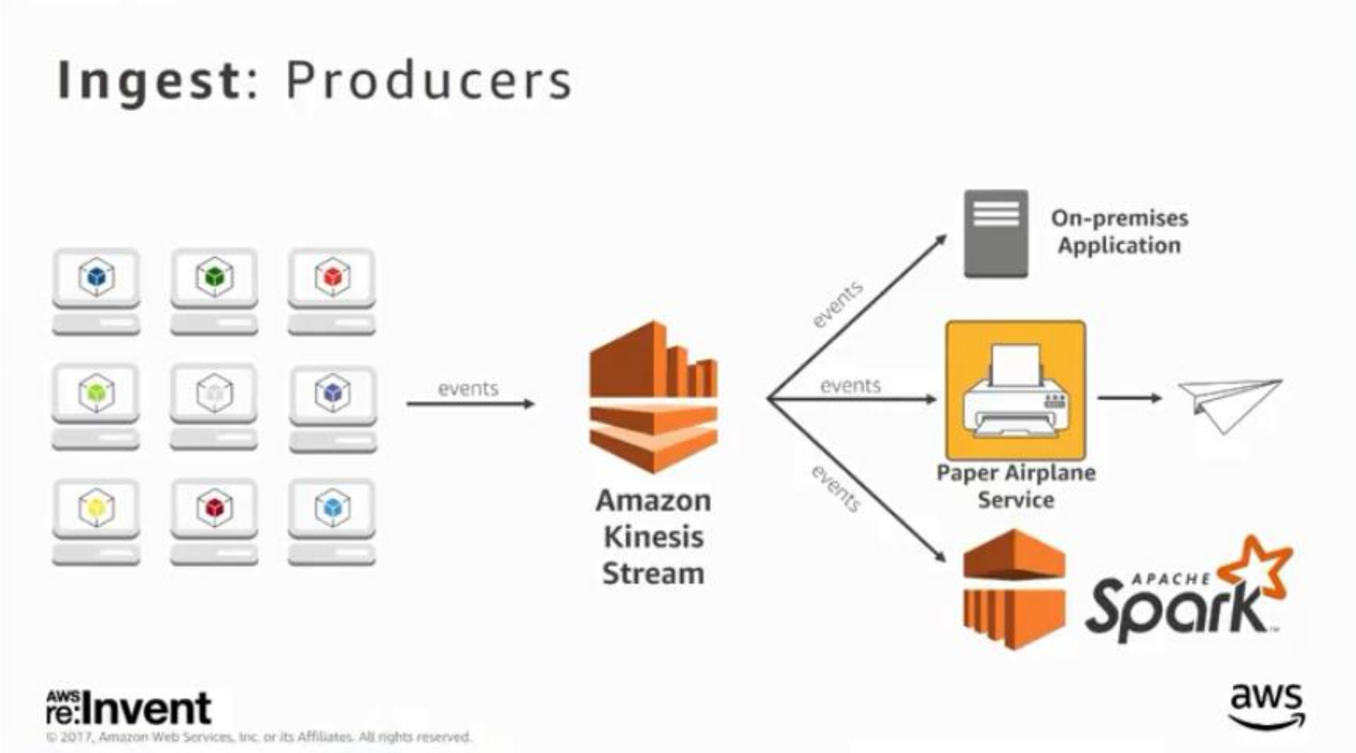
On-premises Application

Paper Airplane Service

APACHE Spark

re:Invent

aws



# Ingest: Producers

events

Amazon Kinesis Stream

events

events

events

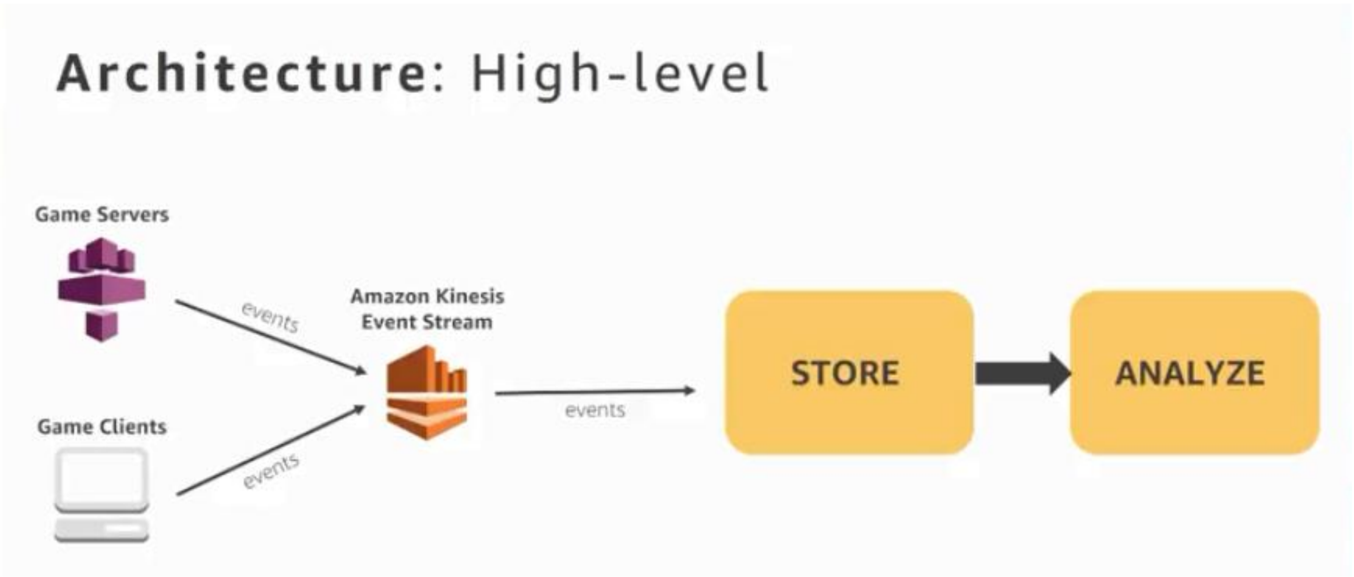
On-premises Application

Paper Airplane Service

APACHE Spark

re:Invent

aws



## Architecture: High-level

Game Servers

events

Game Clients

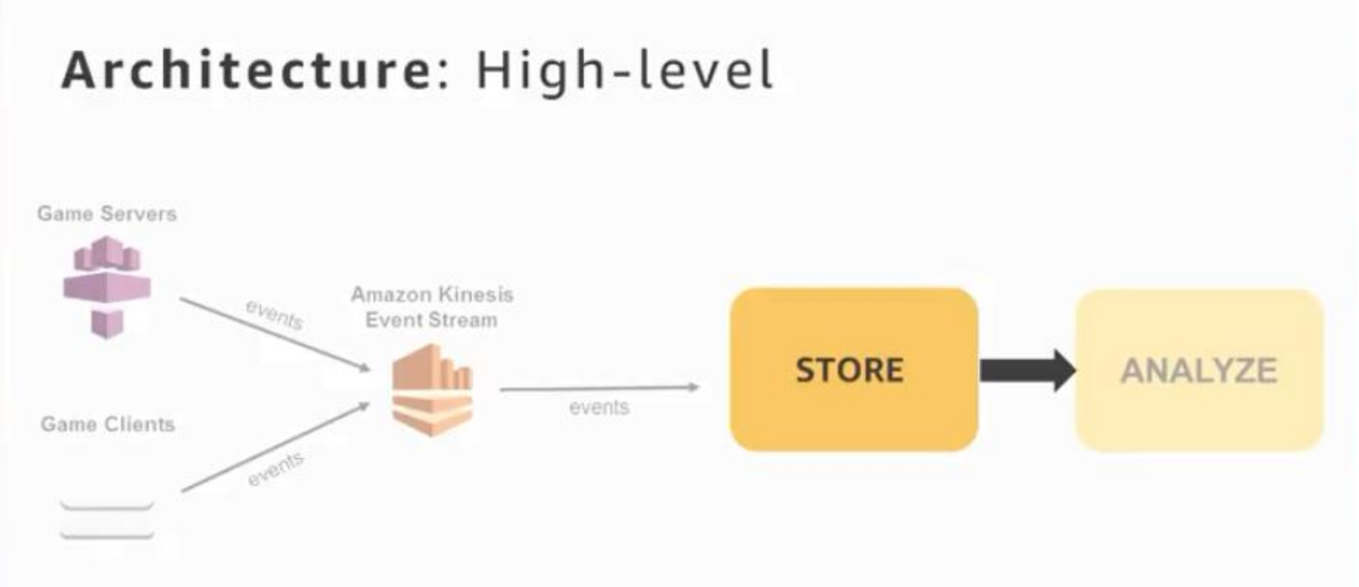
events

Amazon Kinesis Event Stream

events

STORE

ANALYZE



## Architecture: High-level

Game Servers

events

Game Clients

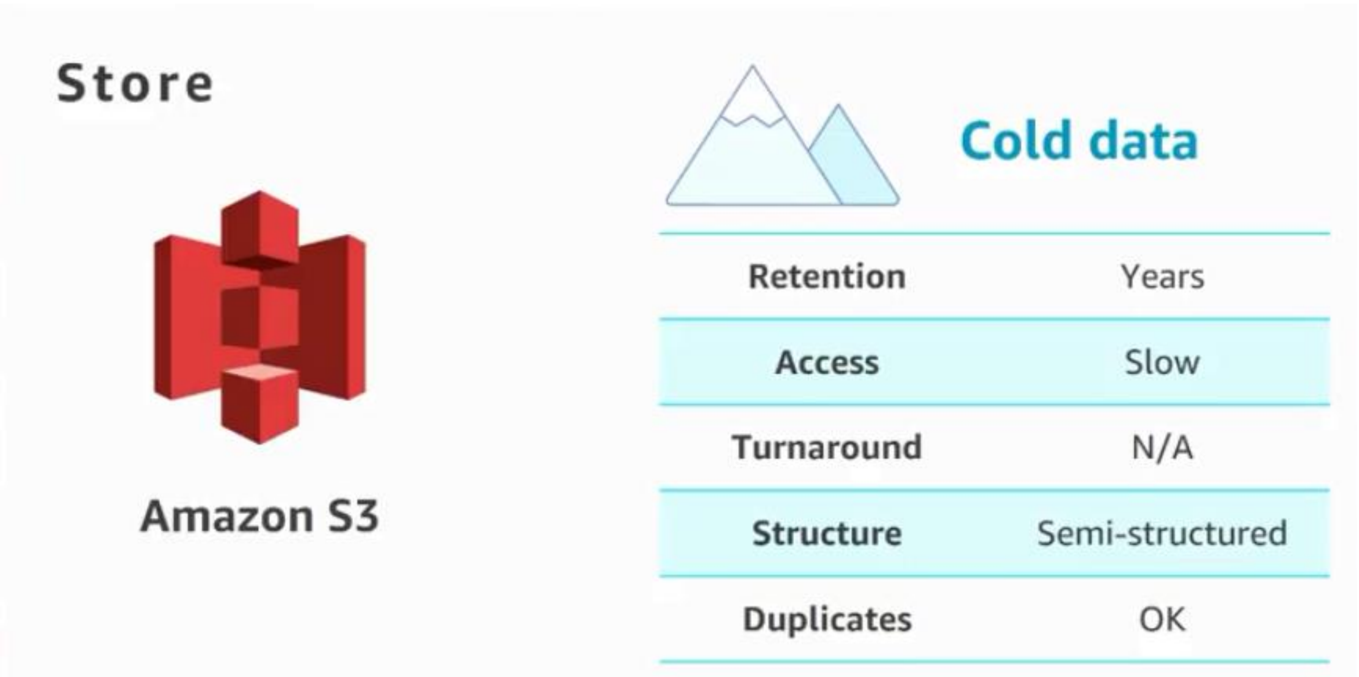
events

Amazon Kinesis Event Stream

events

STORE

ANALYZE

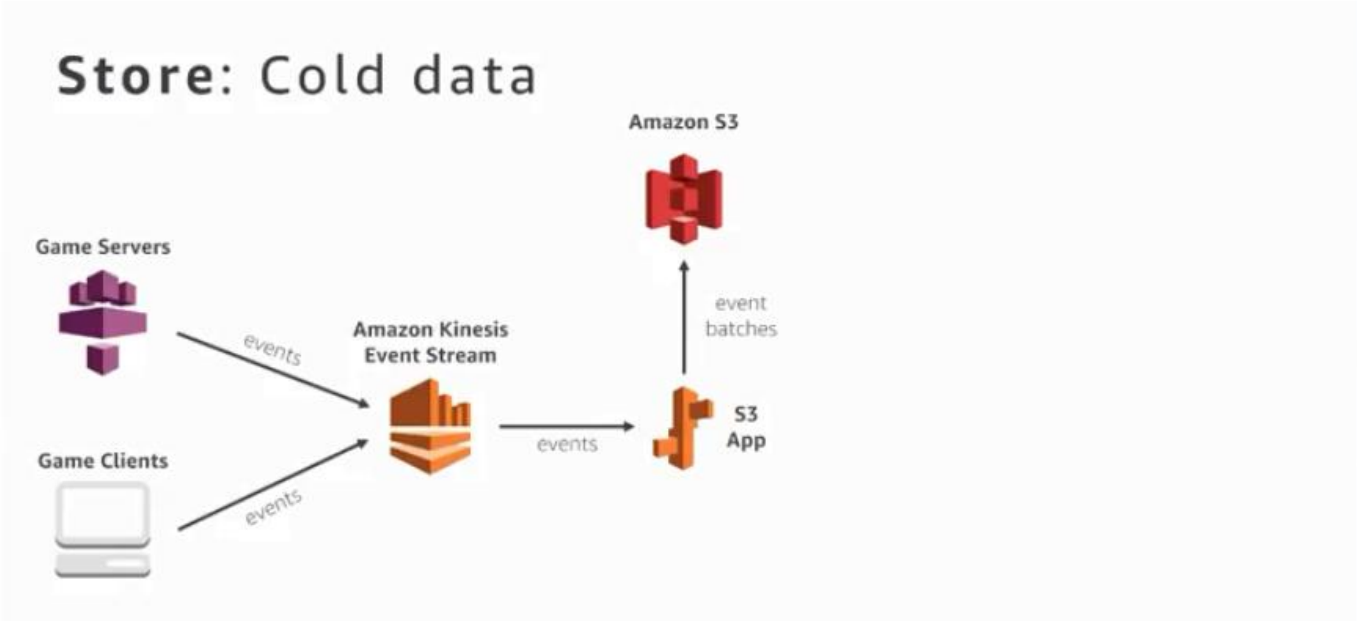


## Store

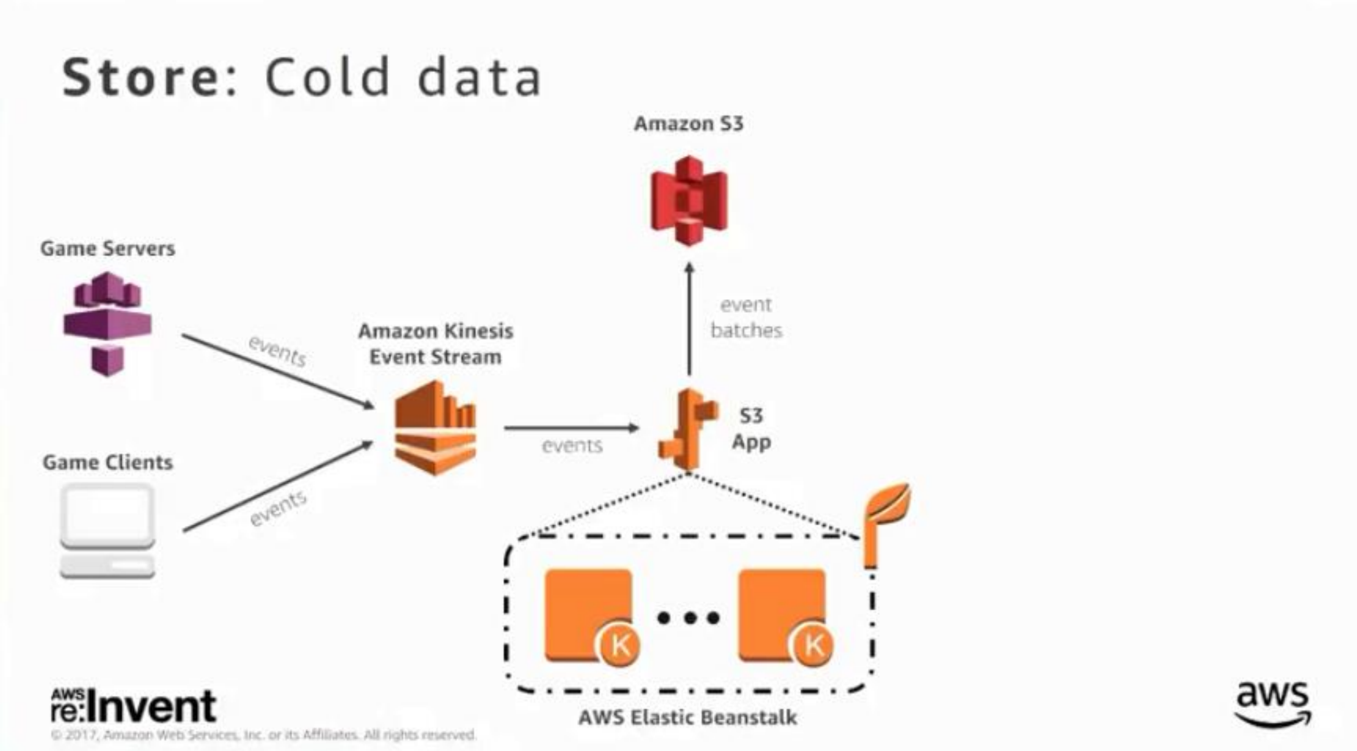
Amazon S3

Cold data

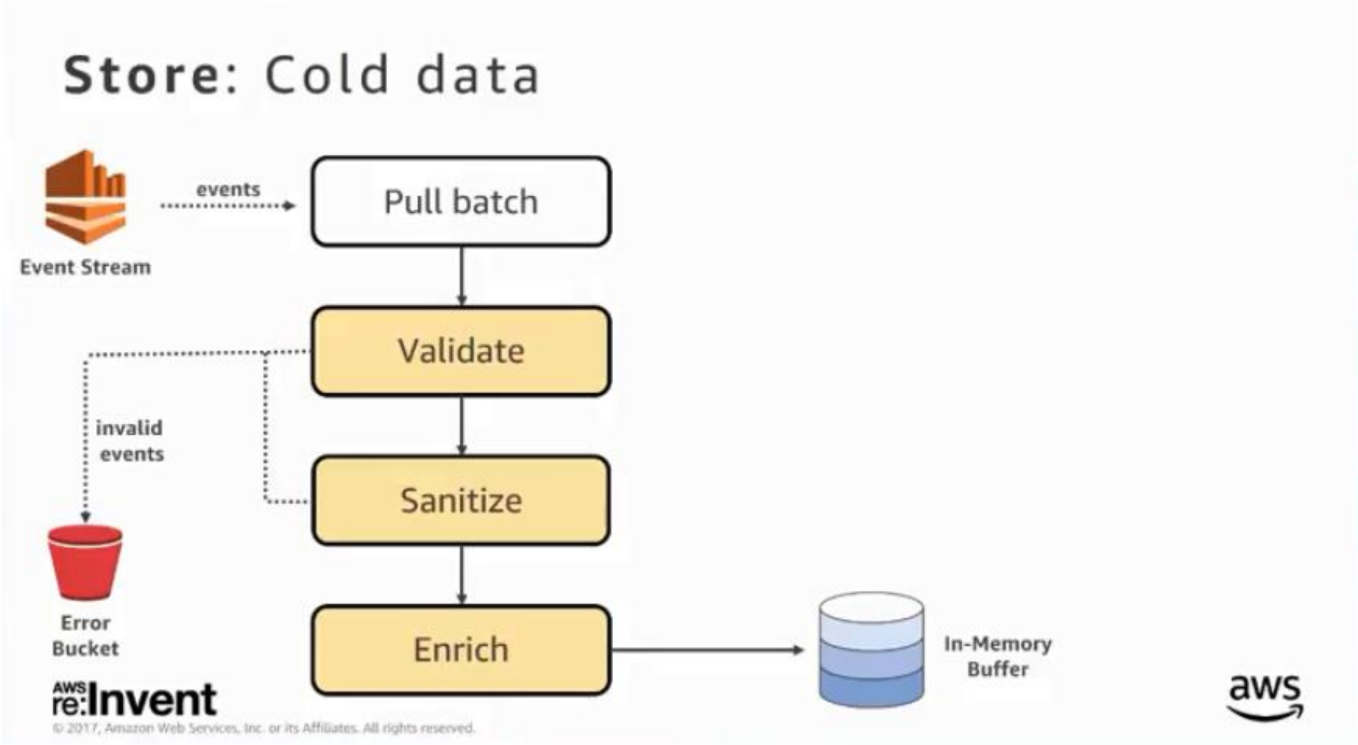
| | |
|---|---|
| **Retention** | Years |
| **Access** | Slow |
| **Turnaround** | N/A |
| **Structure** | Semi-structured |
| **Duplicates** | OK |



# Store: Cold data

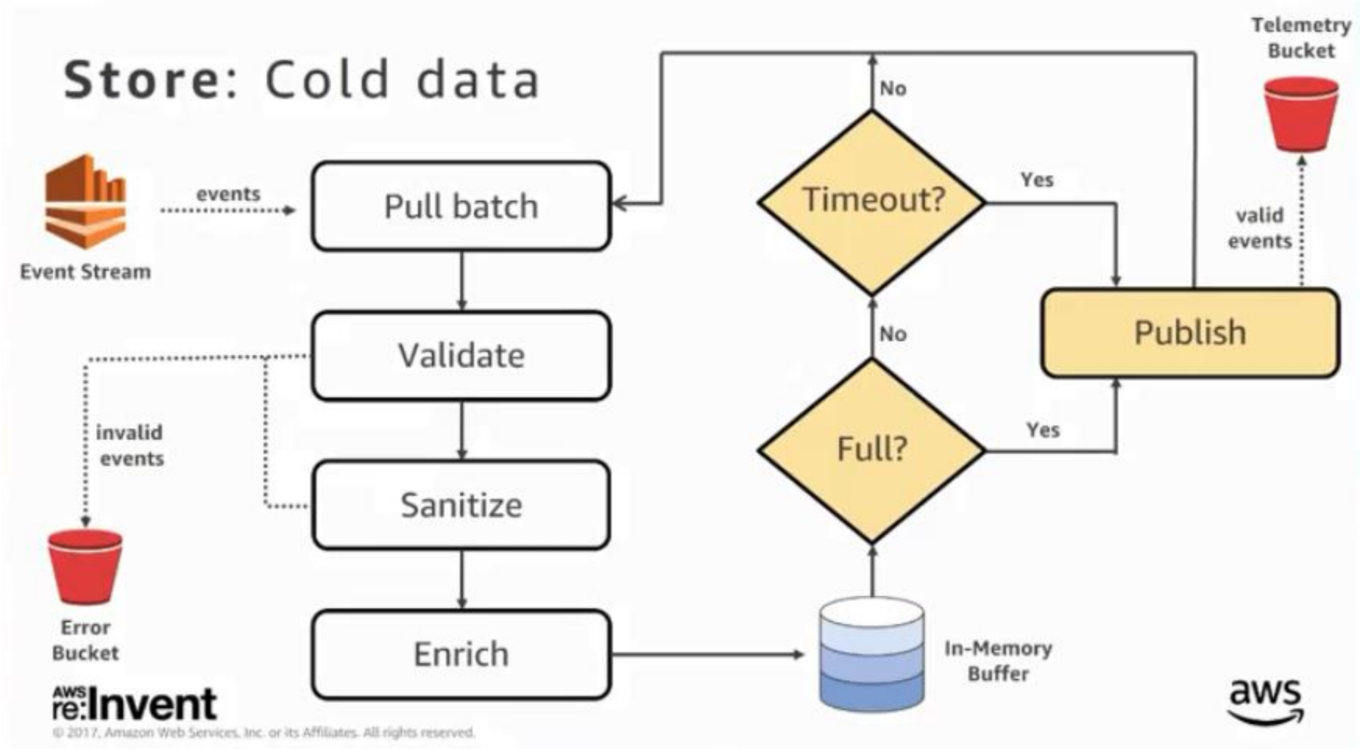


# Store: Cold data



## Store: Cold data

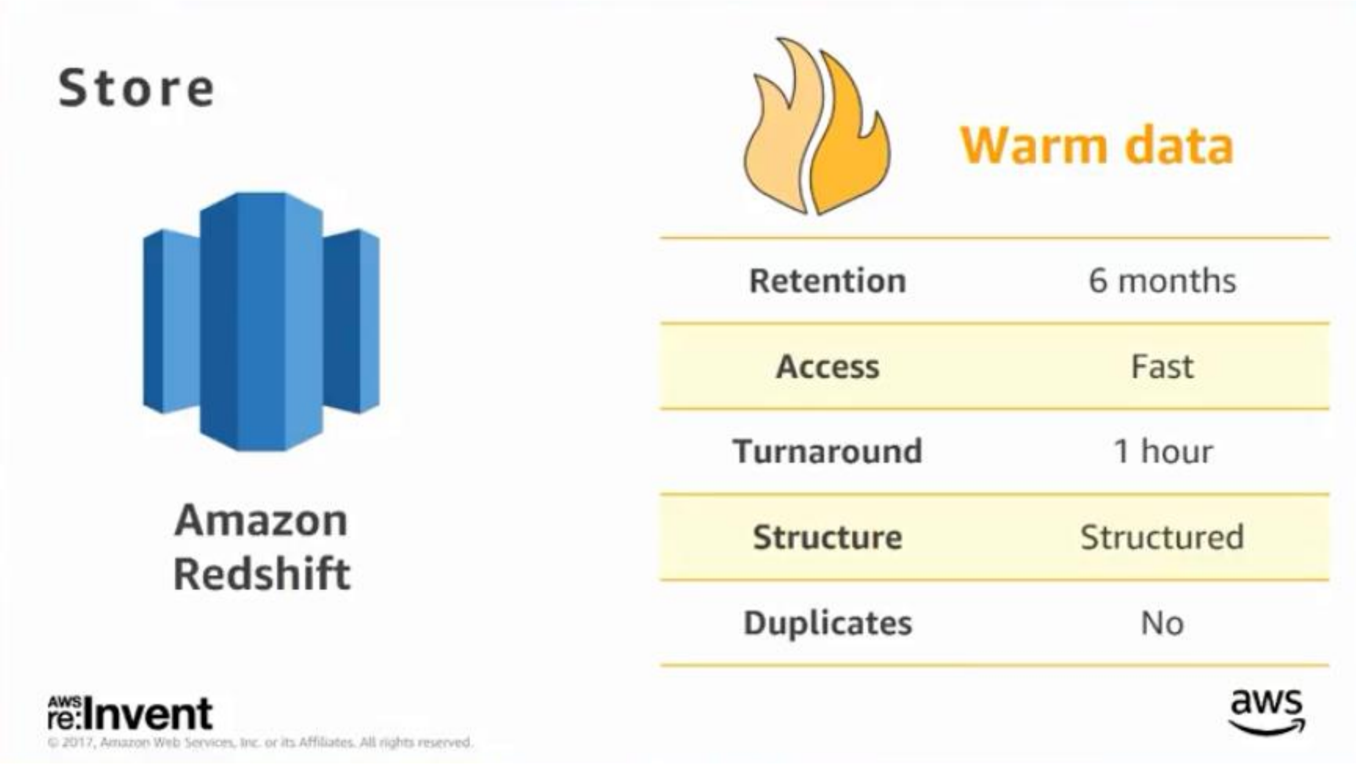
This is what the S3 App is doing.



## Store: Cold data

We are using 100MB buffer size to batch up the data into a single JSON file and store it up in S3

## Store: Cold data



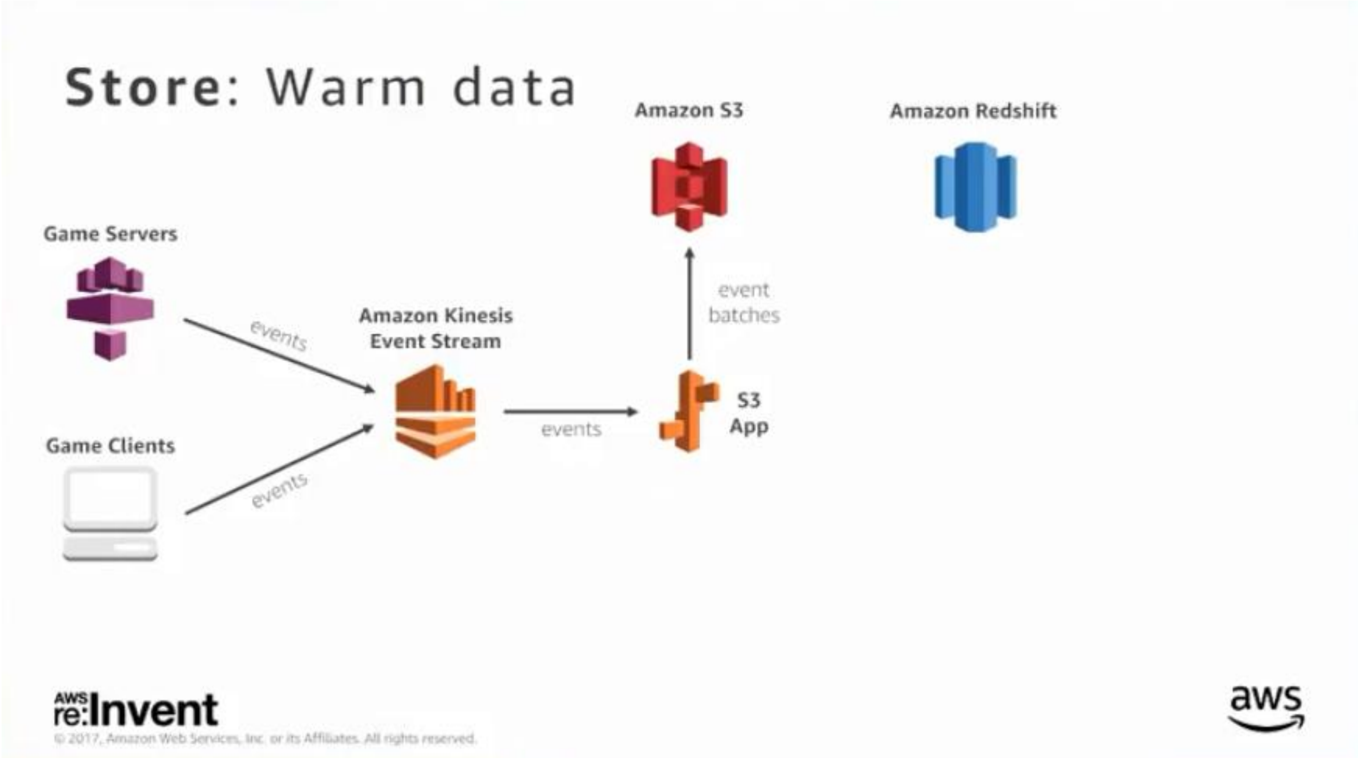
## Store

Amazon Redshift

Warm data

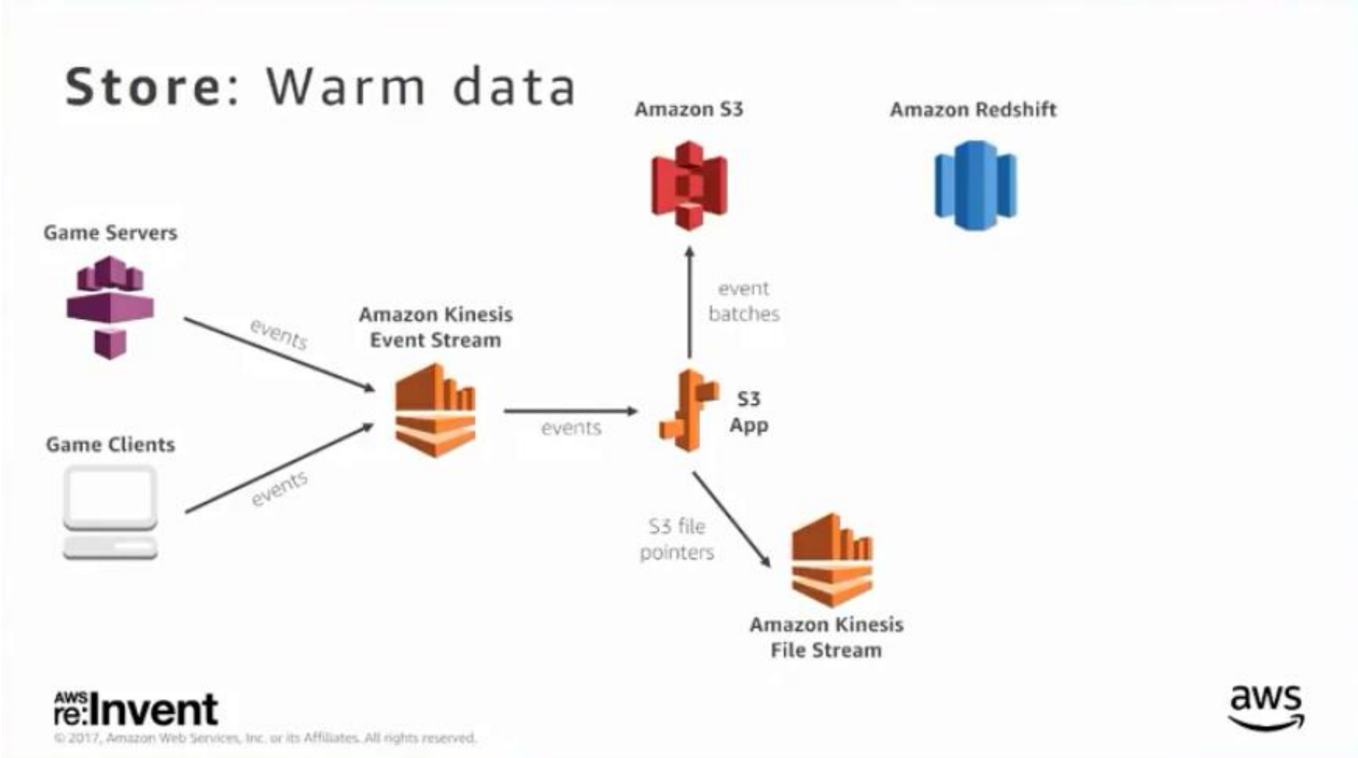
| | |
|---|---|
| Retention | 6 months |
| Access | Fast |
| Turnaround | 1 hour |
| Structure | Structured |
| Duplicates | No |

Newer data like the last 6 months is more relevant to our gaming scenario, we also want structured data, filtering out duplicates. We use Redshift also because it is SQL compatible



## Store: Warm data

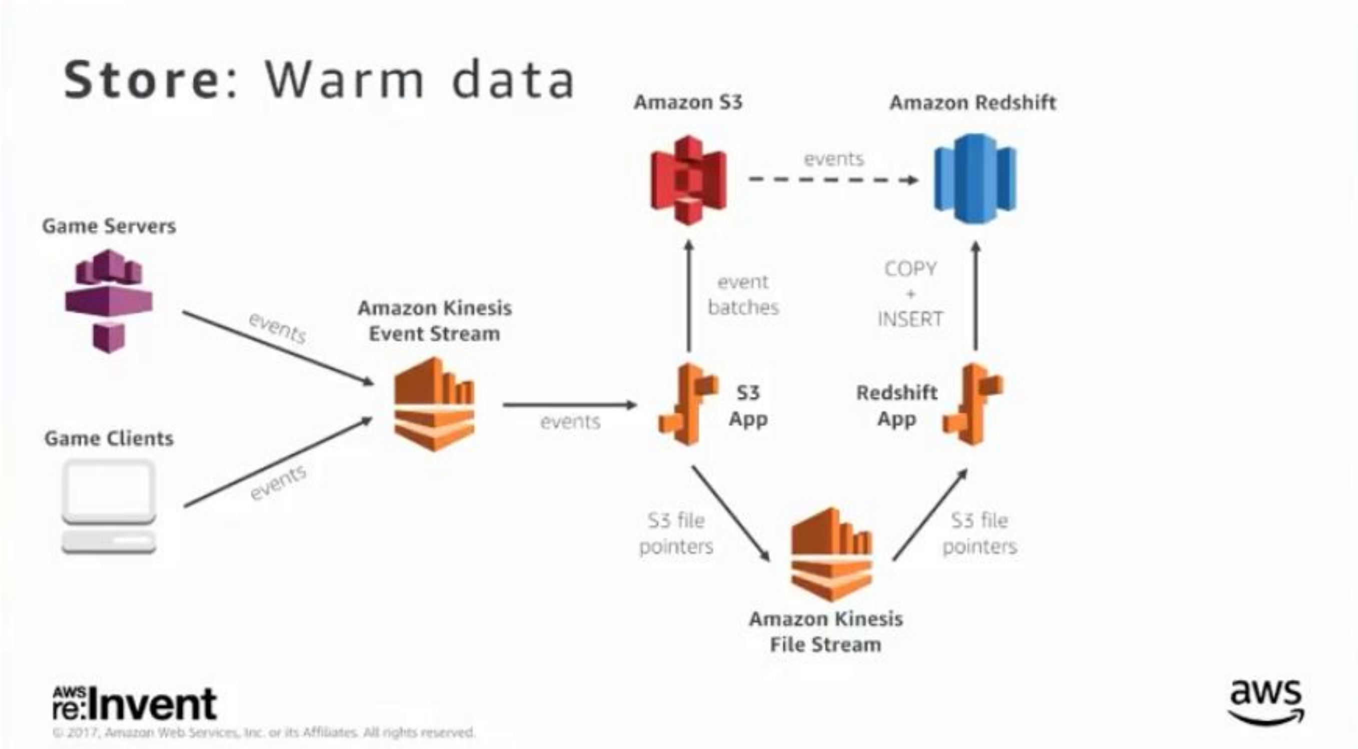
re:Invent



## Store: Warm data

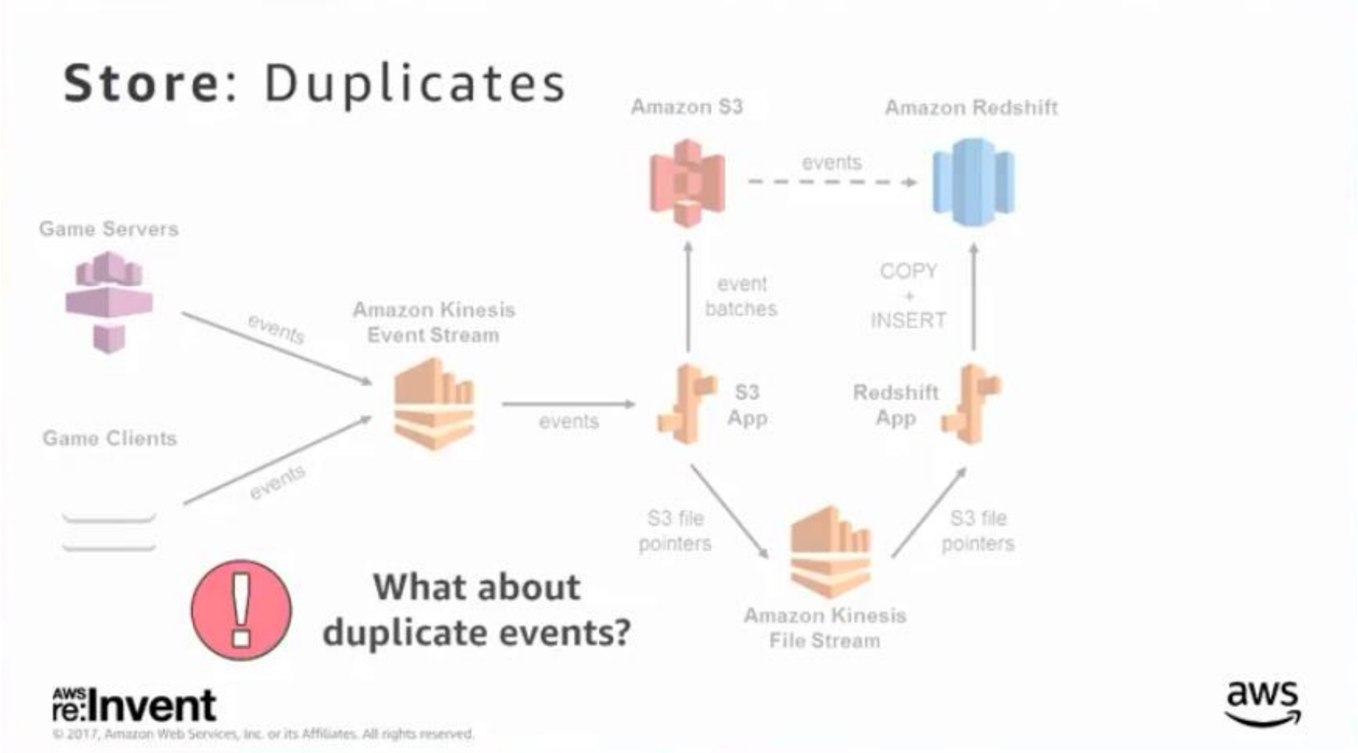
re:Invent

Every time we send one of the 100MB batched JSON file to S3, we also send a secondary file pointer data to another Kinesis stream



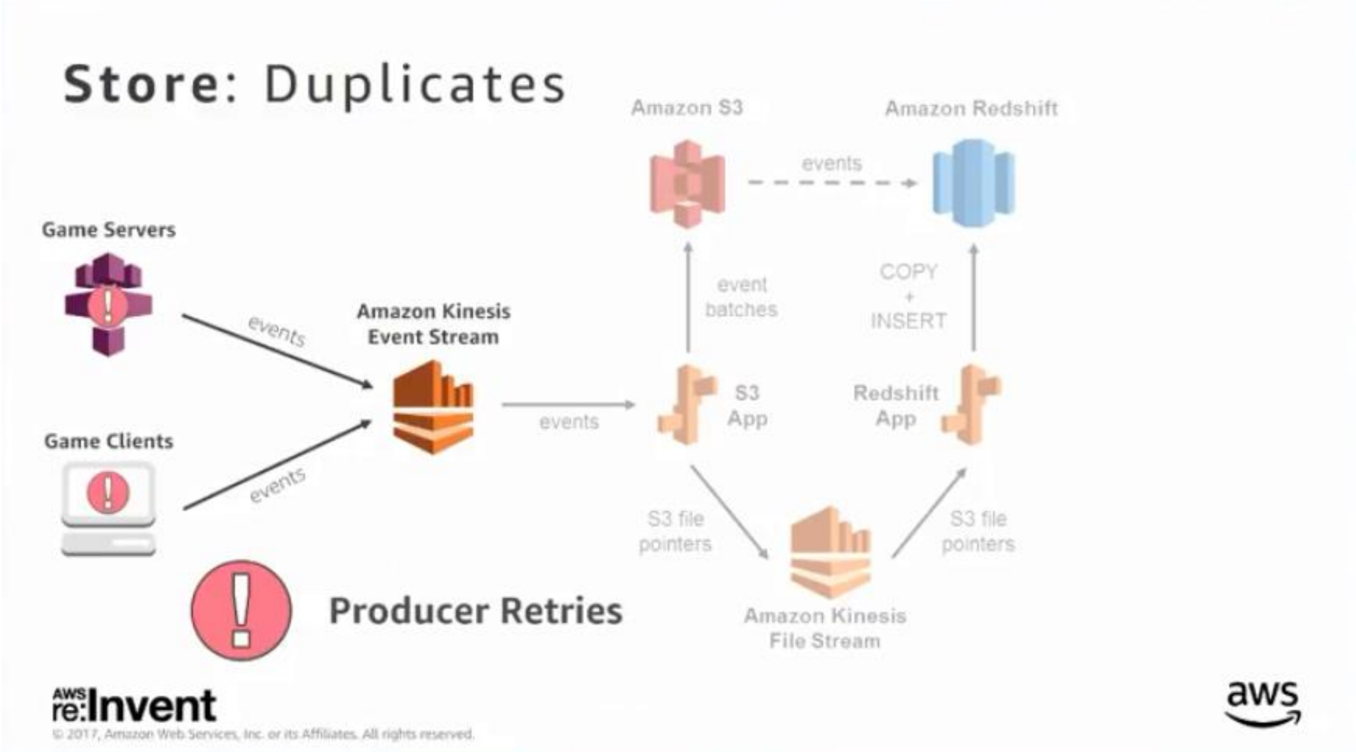
Store: Warm data

We then have another Elastic Beanstalk application that collects data from the 2nd Kinesis stream, buffer the file pointers data up in-memory, when it deems the data large enough, it will push the data into our Redshift cluster using the Redshift COPY command to copy all the data up from S3 using the file pointers.



Store: Duplicates

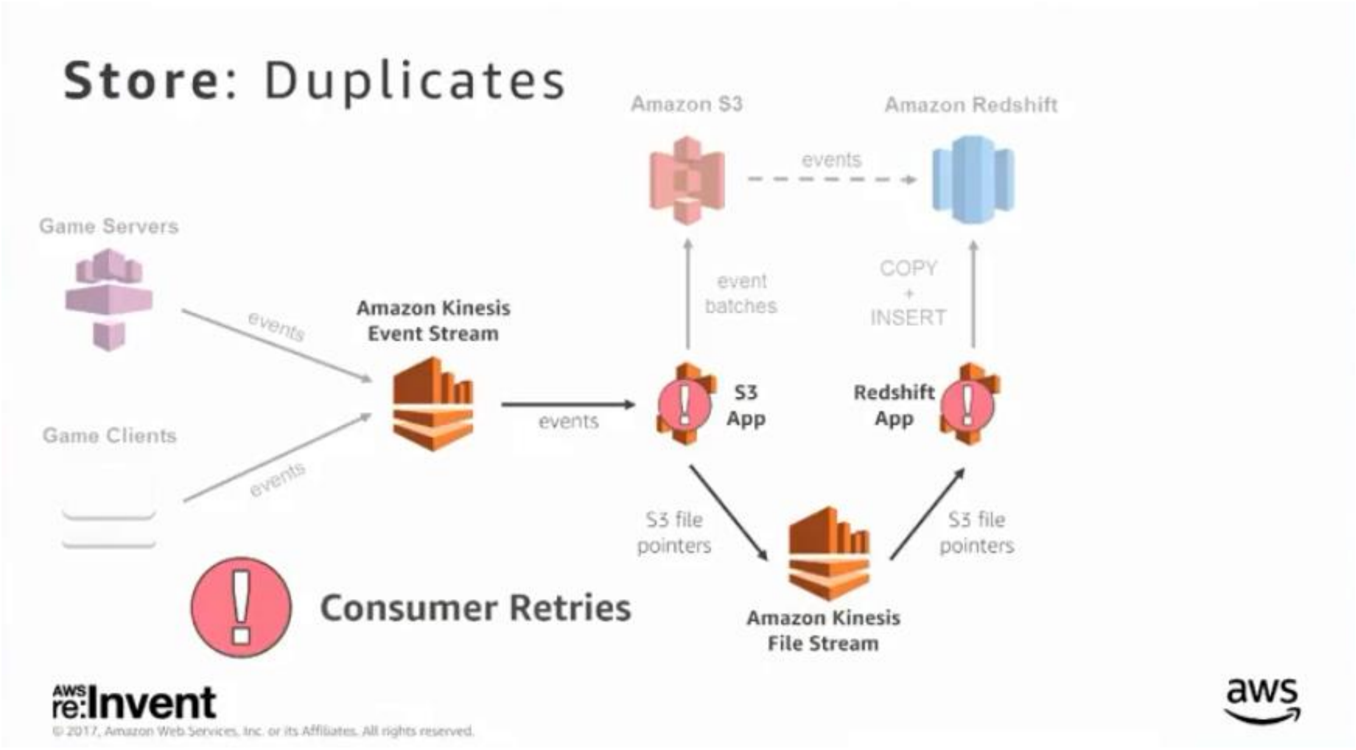
What about duplicate events?



## Store: Duplicates

Producer Retries

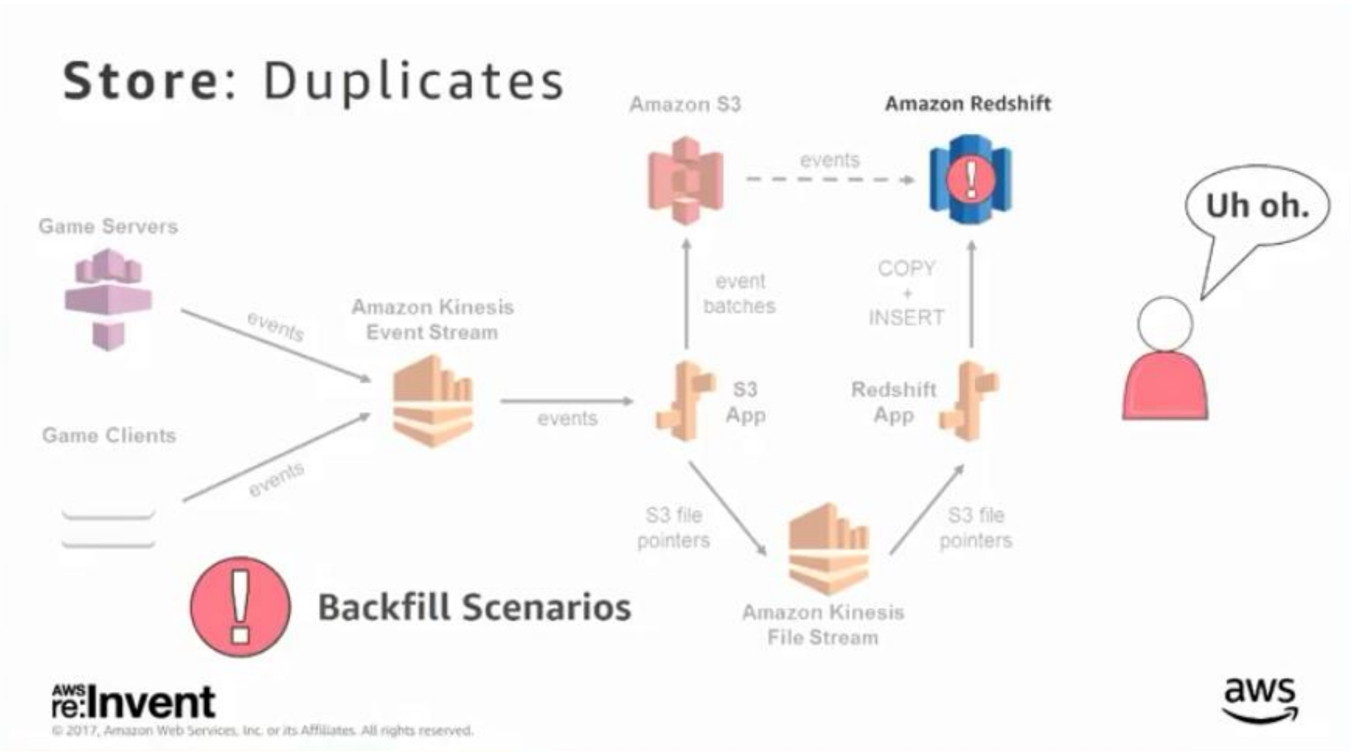
We get duplicate event data from Producer retries when network is flaky



## Store: Duplicates

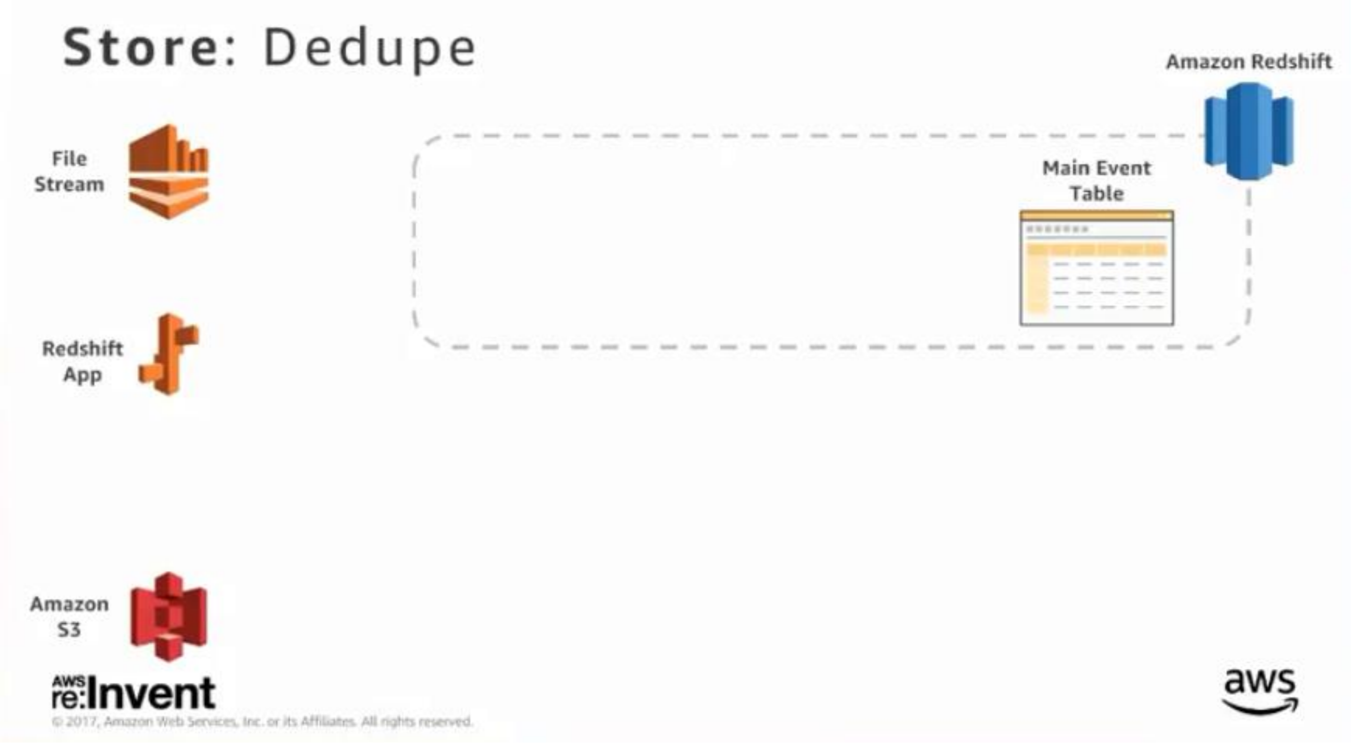
Consumer Retries

Consumer retries can also create duplicate data being generated and stored

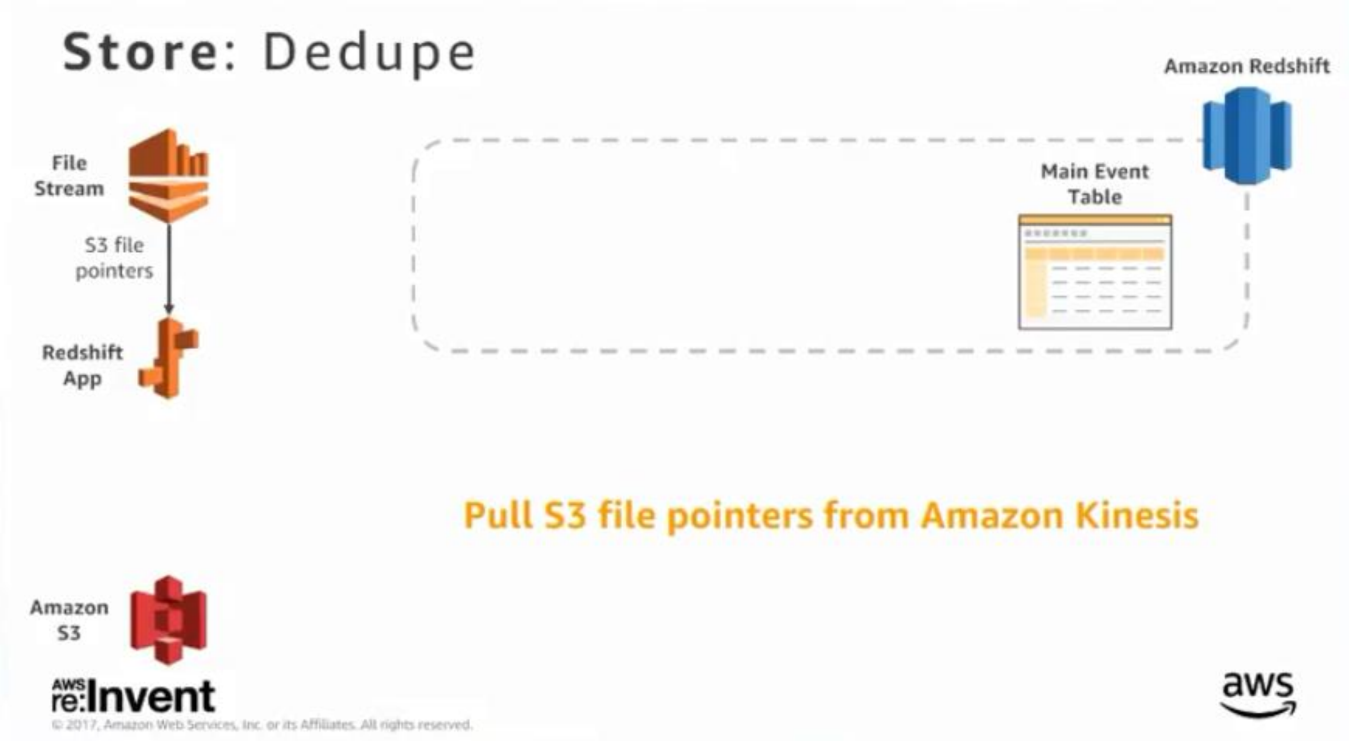


## Store: Duplicates

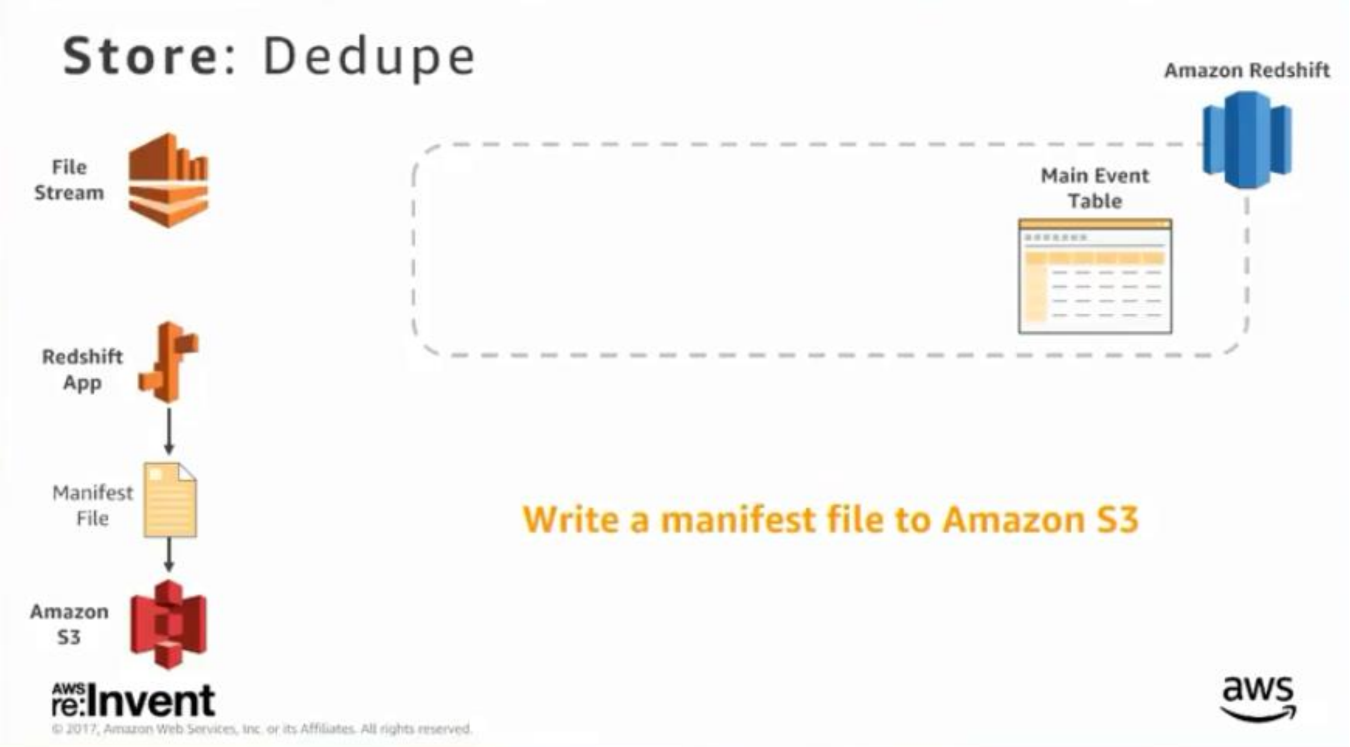
Duplicates can also be caused due to human errors



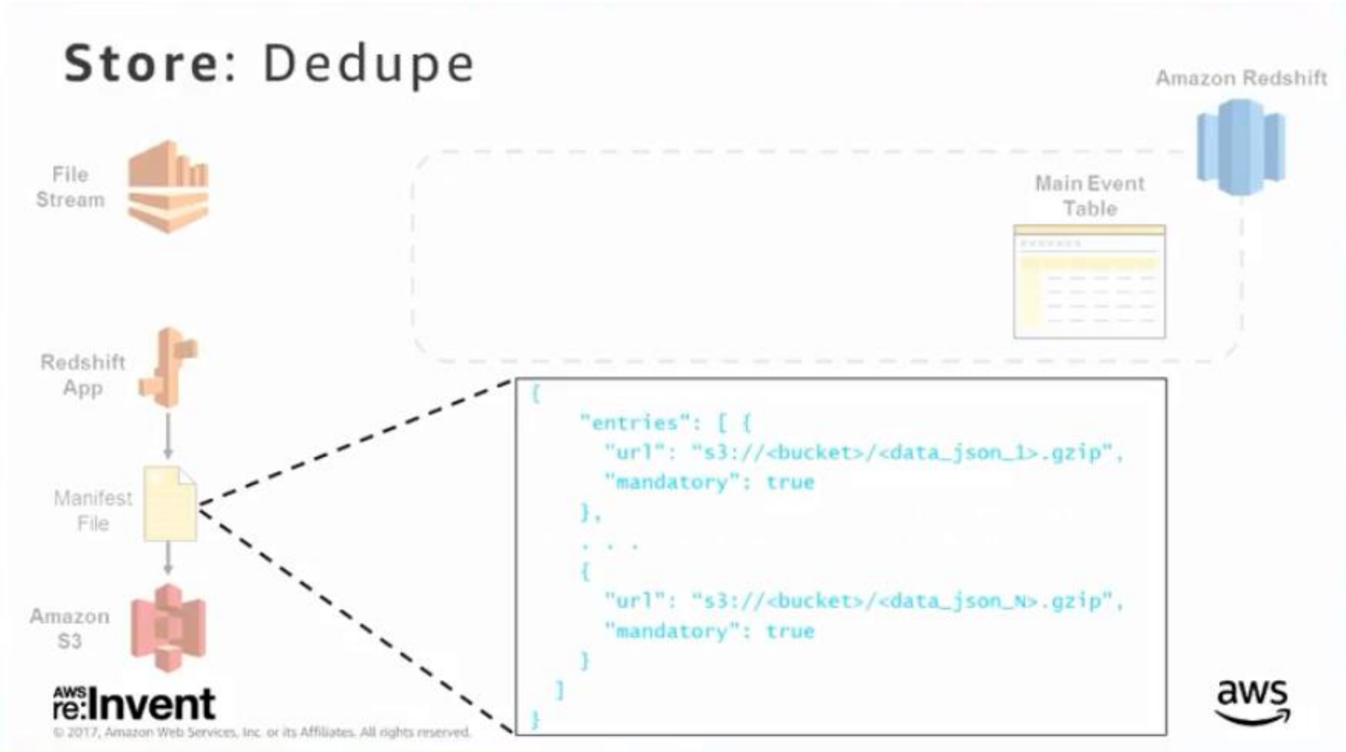
## Store: Dedupe


Store: Dedupe
Amazon Redshift
File Stream
S3 file pointers
Redshift App
Main Event Table
Pull S3 file pointers from Amazon Kinesis
Amazon S3
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

Once the Redshift elastic beanstalk app deems it has enough file pointers batched up in-memory, it’s going to get the files from S3


Store: Dedupe
Amazon Redshift
File Stream
Redshift App
Main Event Table
Manifest File
Write a manifest file to Amazon S3
Amazon S3
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

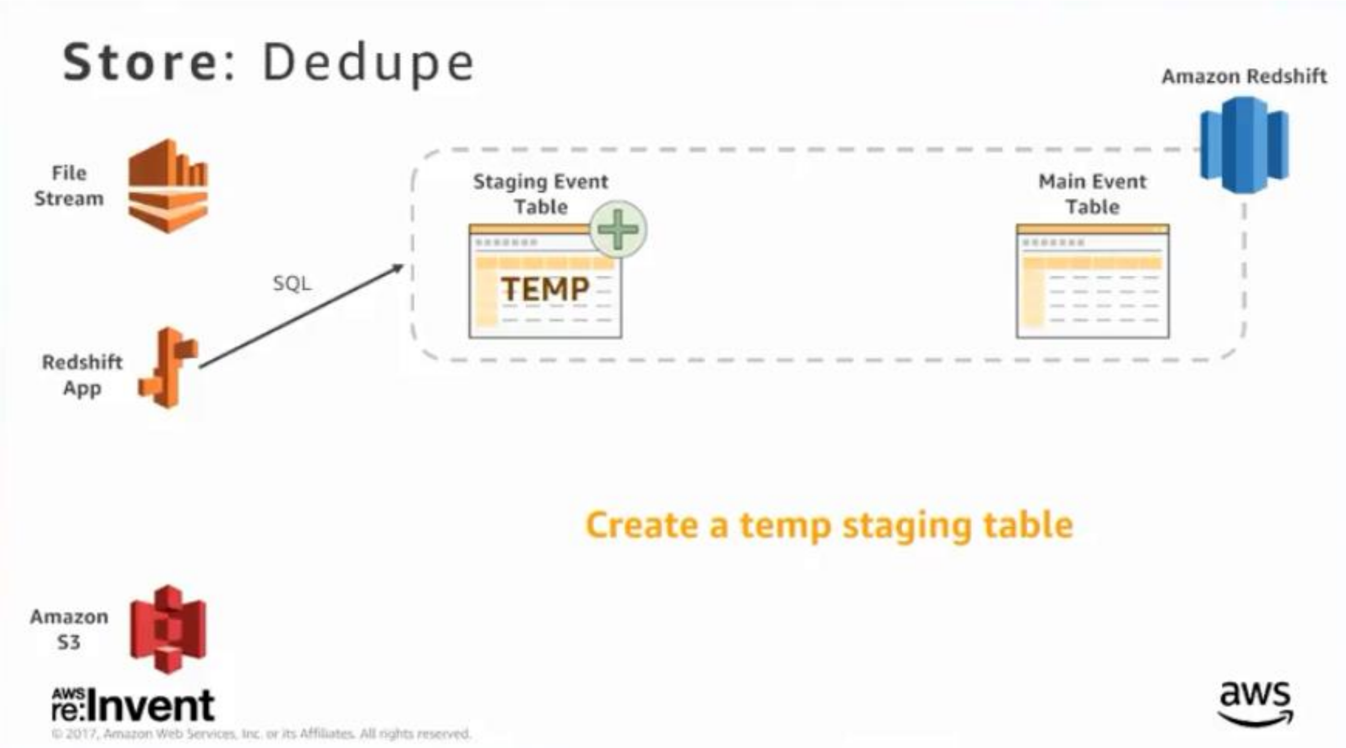
It then writes up all the file pointer data that it has in its memory and push them into a single manifest file



## Store: Dedupe

```
{
    "entries": [ {
      "url": "s3://<bucket>/<data_json_1>.gzip",
      "mandatory": true
    },
    . . .
    {
      "url": "s3://<bucket>/<data_json_N>.gzip",
      "mandatory": true
    }
  ]
}
```

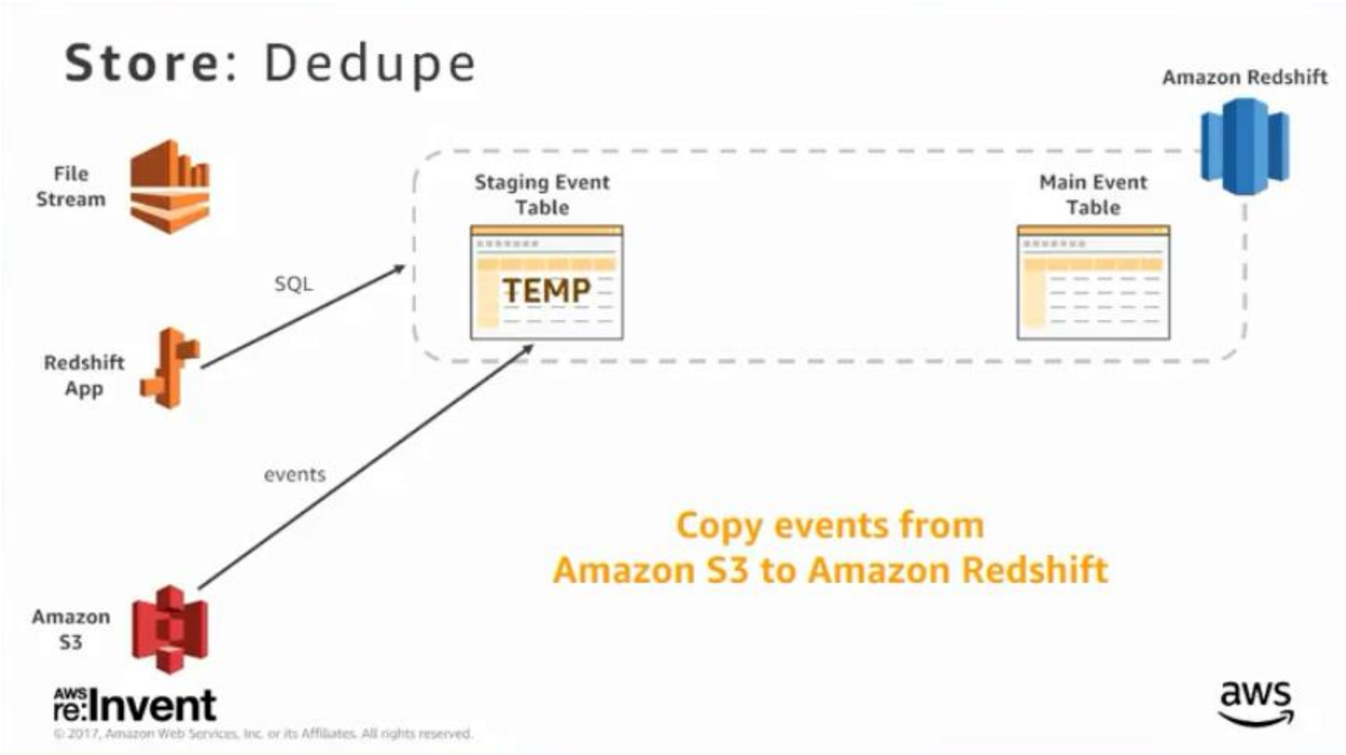
Manifests in Redshift are a set of S3 URIs or pointers combined with a mandatory entry that tells it whether or not this load should fail if that file is not there.



## Store: Dedupe

Create a temp staging table

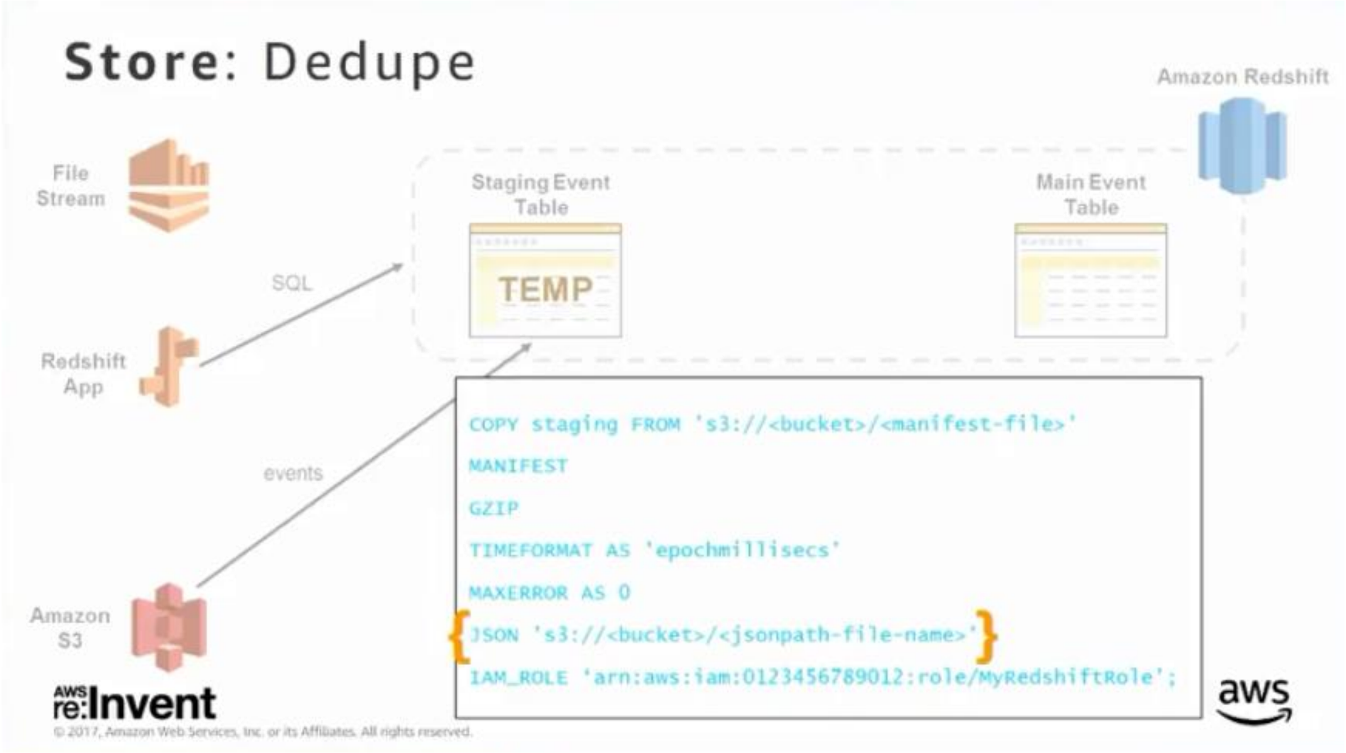
We create an empty temporary table in Redshift to put the data into



Store: Dedupe

Copy events from Amazon S3 to Amazon Redshift

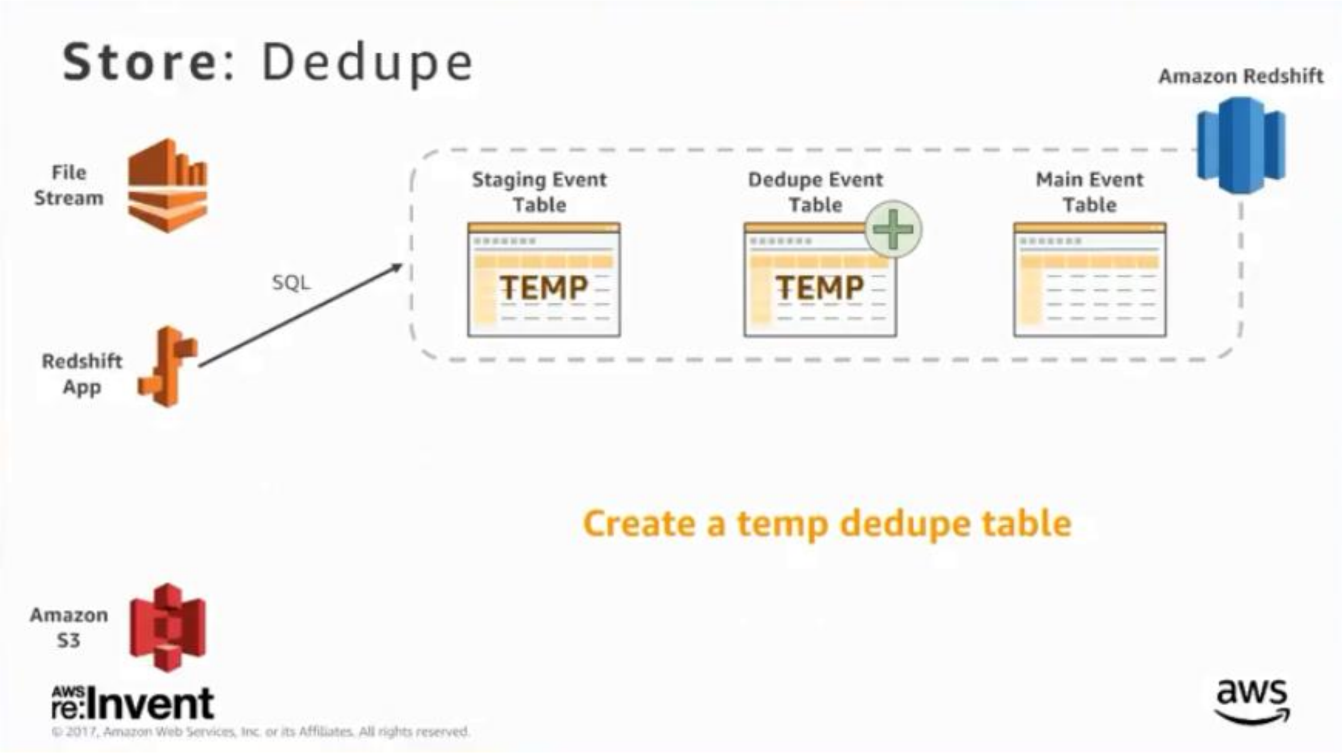
We then use the Redshift COPY command to get all the data out from S3



Store: Dedupe

```
COPY staging FROM 's3://<bucket>/<manifest-file>'
MANIFEST
GZIP
TIMEFORMAT AS 'epochmillisecs'
MAXERROR AS 0
{JSON 's3://<bucket>/<jsonpath-file-name>'}
IAM_ROLE 'arn:aws:iam:0123456789012:role/MyRedshiftRole';
```

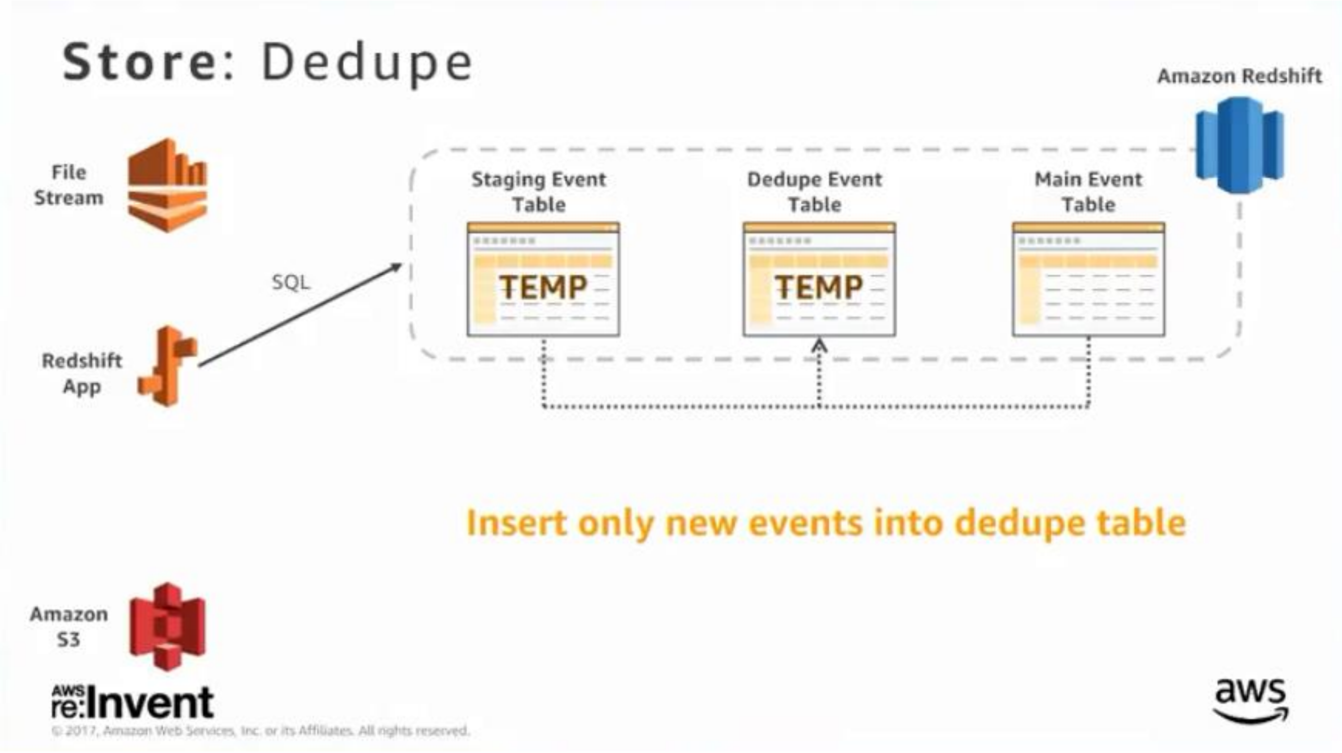
This is what a Redshift COPY command looks like, we tell Redshift we are loading JSON data. The ***jsonpath*** lets you map the fields on your input JSON onto database columns, it also lets you chose and pick only the fields that you care about mapping into database columns and also lets you order the database columns. This helps us decouple our database schema from our Redshift schema



## Store: Dedupe

Create a temp dedupe table

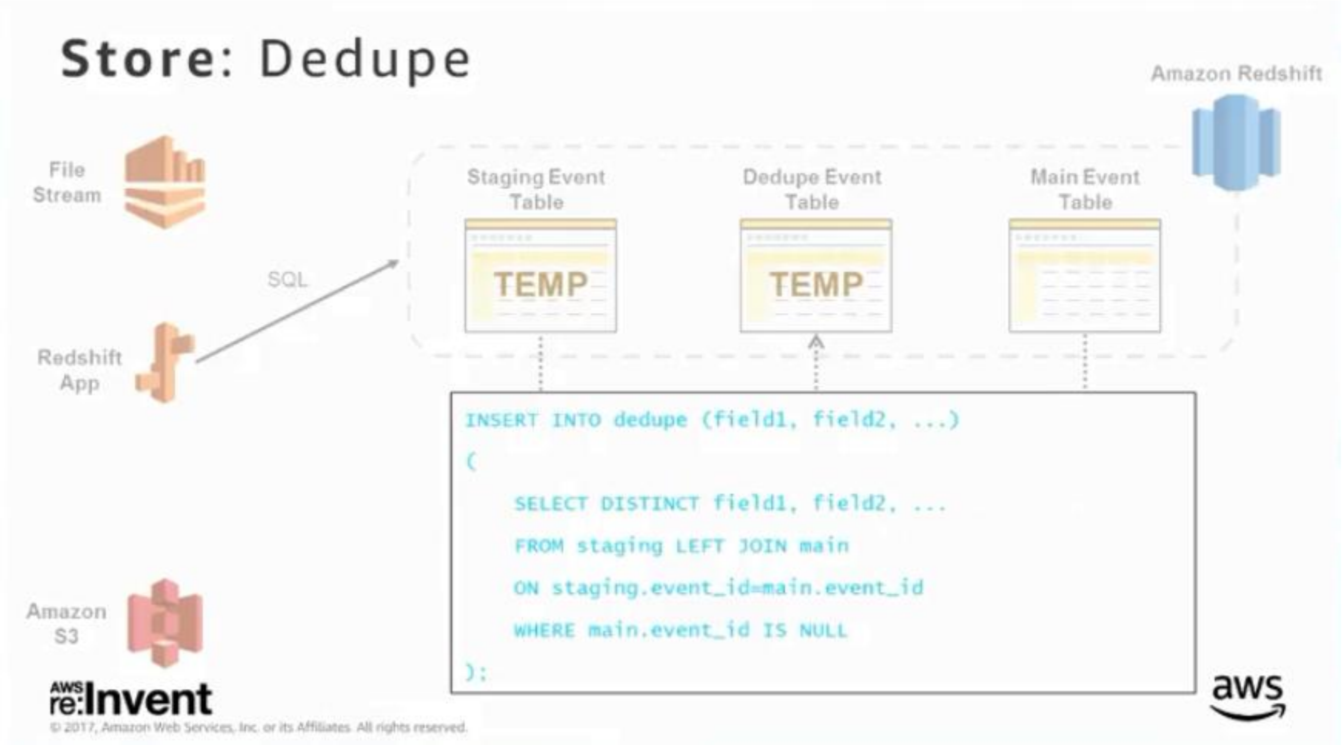
We now have all the events in the staging table including duplicates if present



## Store: Dedupe

Insert only new events into dedupe table

We the create a secondary table called the dedupe table, we now do some dedupe magic by taking everything in the staging table and left join it with our existing set of events in the main table and put only the unique data into the secondary dedupe table



Store: Dedupe

Amazon Redshift

File Stream

SQL

Redshift App

Amazon S3

Staging Event Table

TEMP

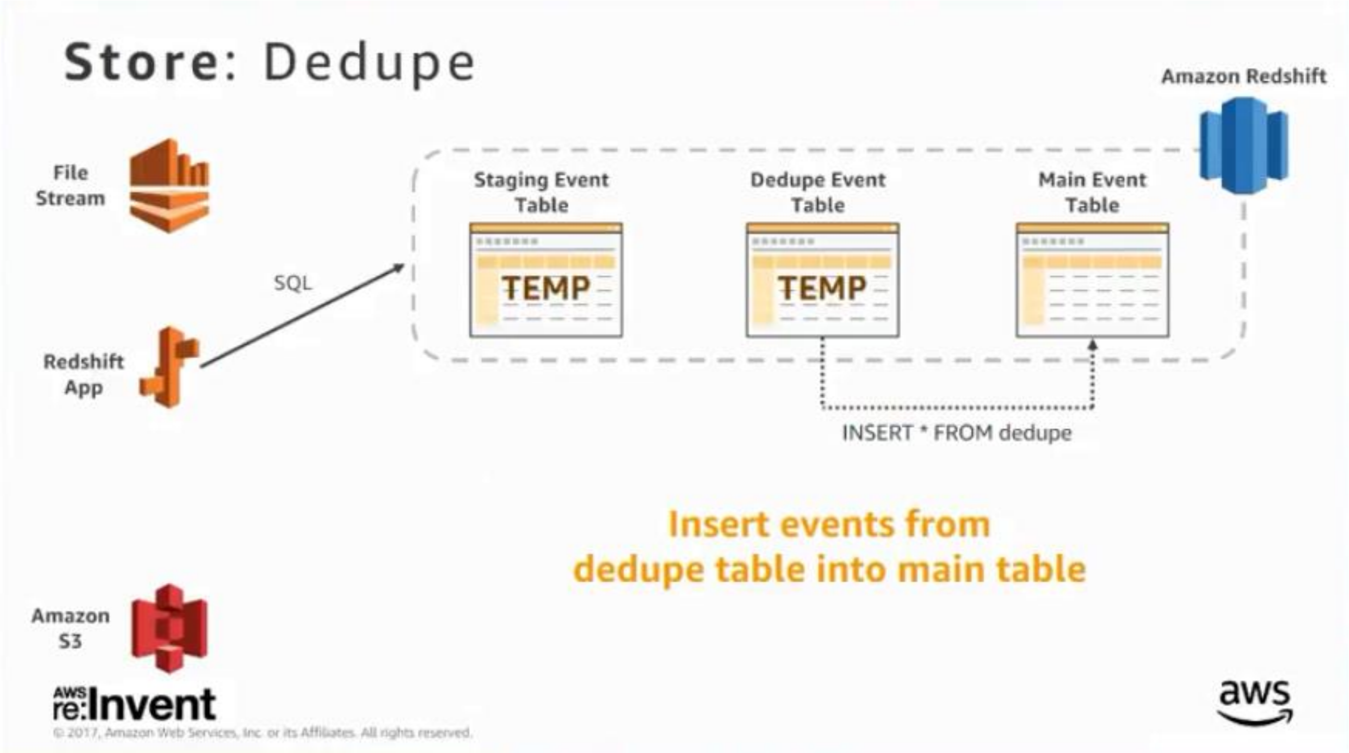
Dedupe Event Table

TEMP

Main Event Table

```
INSERT INTO dedupe (field1, field2, ...)
(
    SELECT DISTINCT field1, field2, ...
    FROM staging LEFT JOIN main
    ON staging.event_id=main.event_id
    WHERE main.event_id IS NULL
);
```

This looks like above



Store: Dedupe

Amazon Redshift

File Stream

SQL

Redshift App

Amazon S3

Staging Event Table

TEMP

Dedupe Event Table

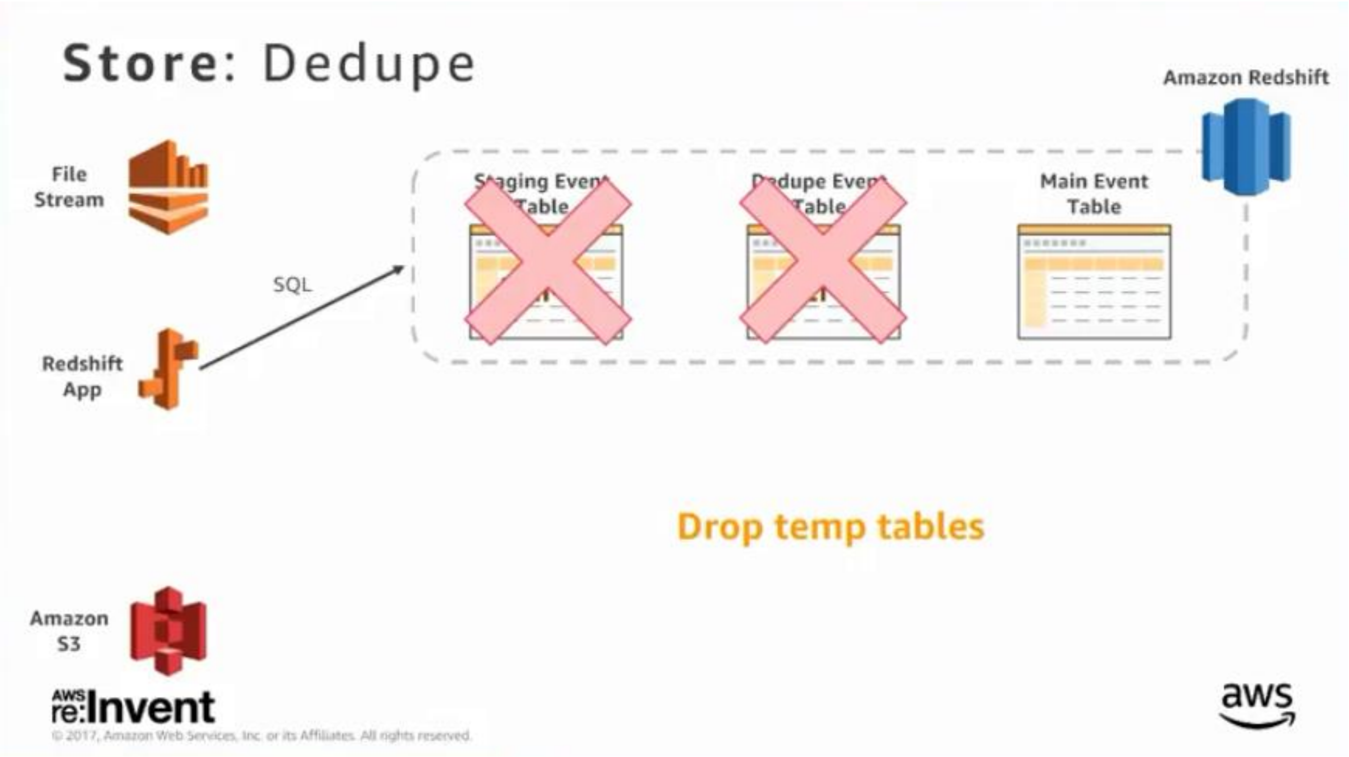
TEMP

Main Event Table

INSERT * FROM dedupe

Insert events from dedupe table into main table

We get unique event data in the dedupe table and then do a simple merge back to the Main table



## Store: Dedupe

File Stream

Redshift App

SQL

Staging Event Table

Dedupe Event Table

Main Event Table

Amazon Redshift

Drop temp tables

Amazon S3

AWS re:Invent

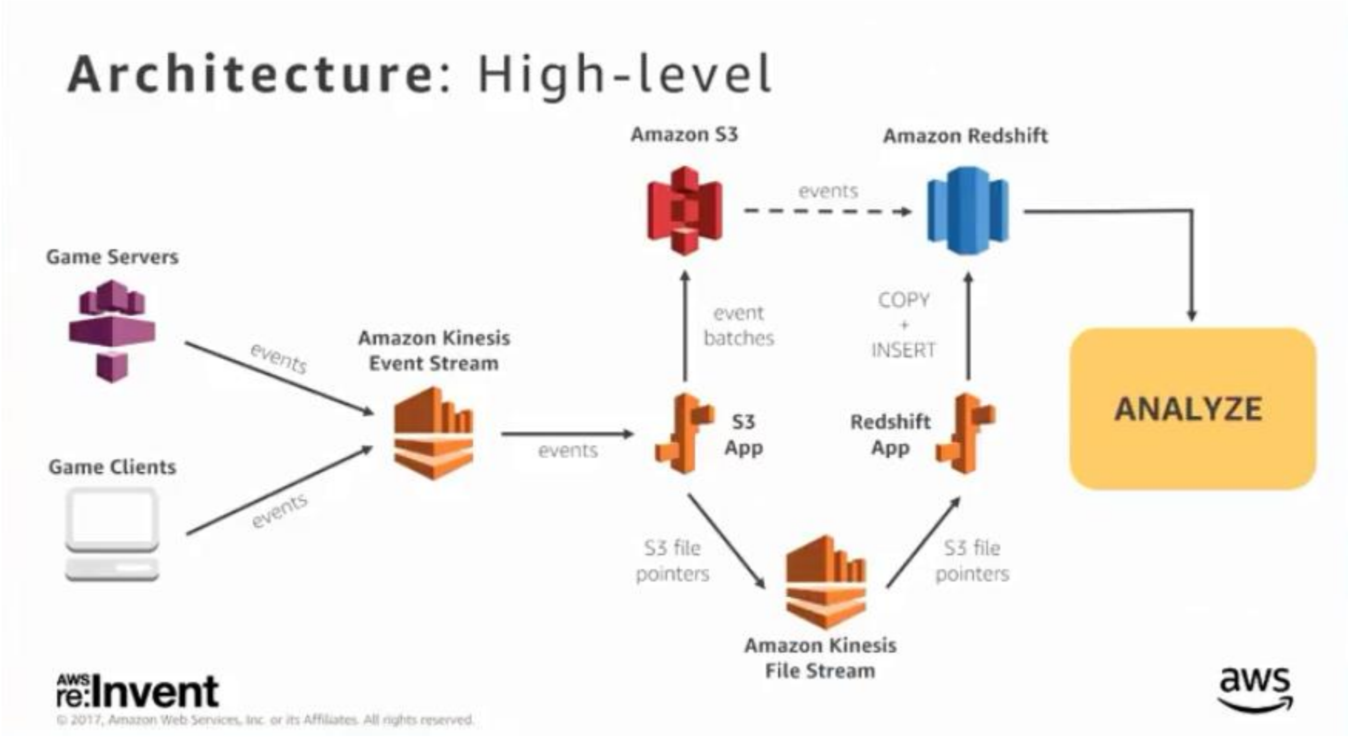
aws

We then drop the temp tables when done

## Store: Warm data

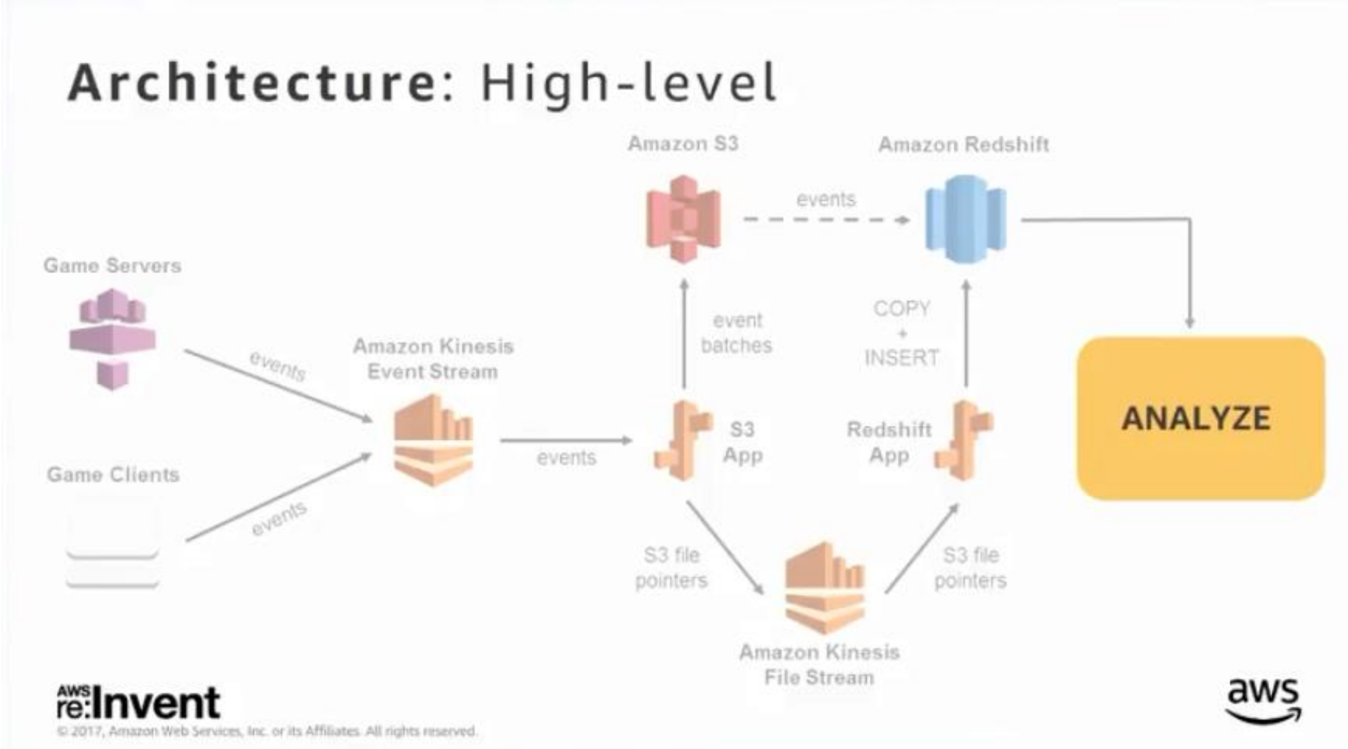
AWS re:Invent

aws



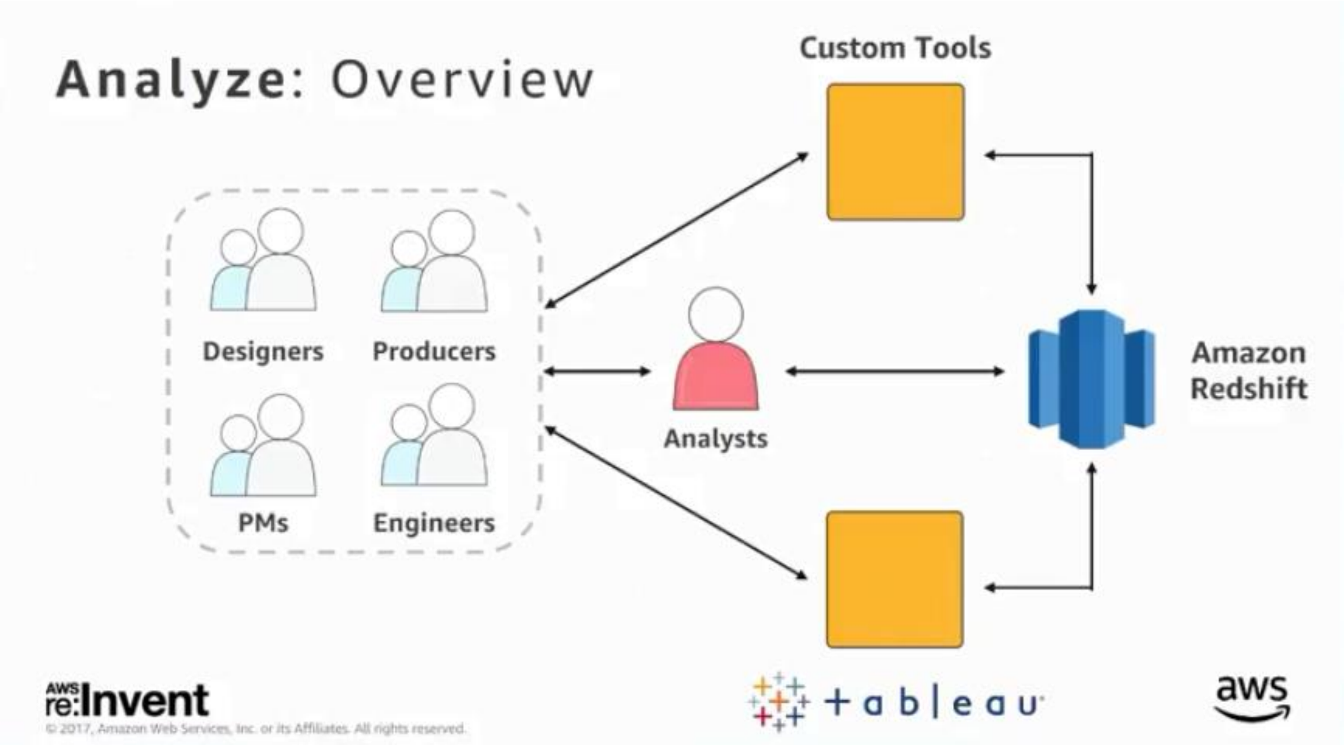
Architecture: High-level

We are back to our architecture now, we have our cold data in S3 and our warm data in Redshift.

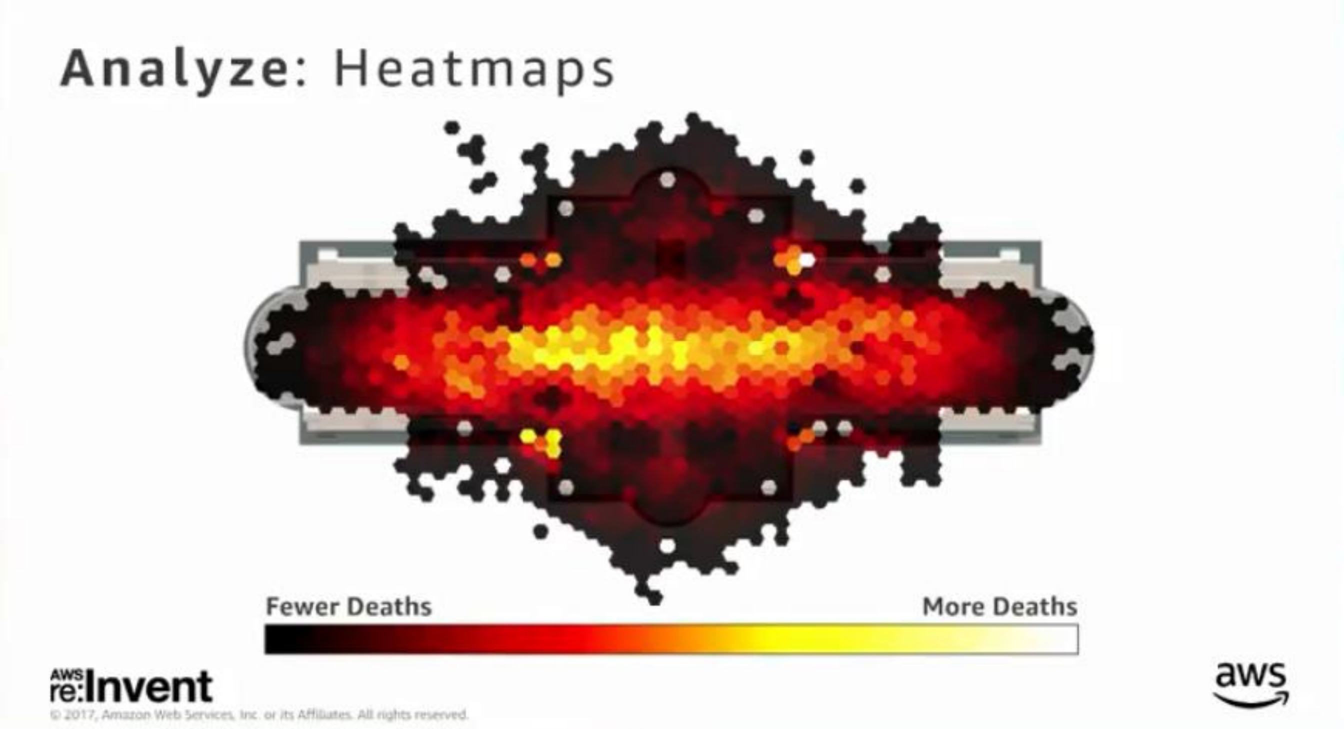


Architecture: High-level

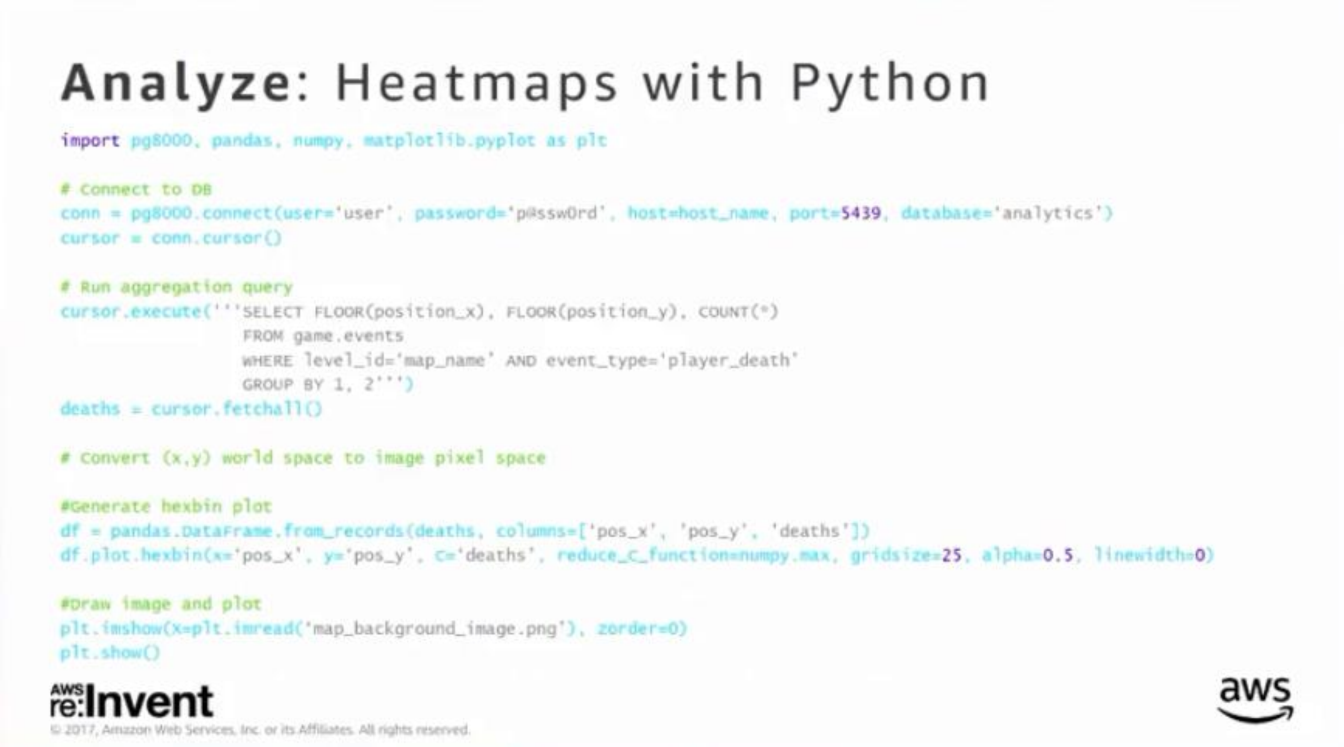
We are now going to analyze the data in Redshift



We have our custom tools that we built like our heatmap generator that can be run as a self-service, we have our warm data in S3, we have our Analysts that run queries against S3, and we use Tableau a lot for visualization from the market store



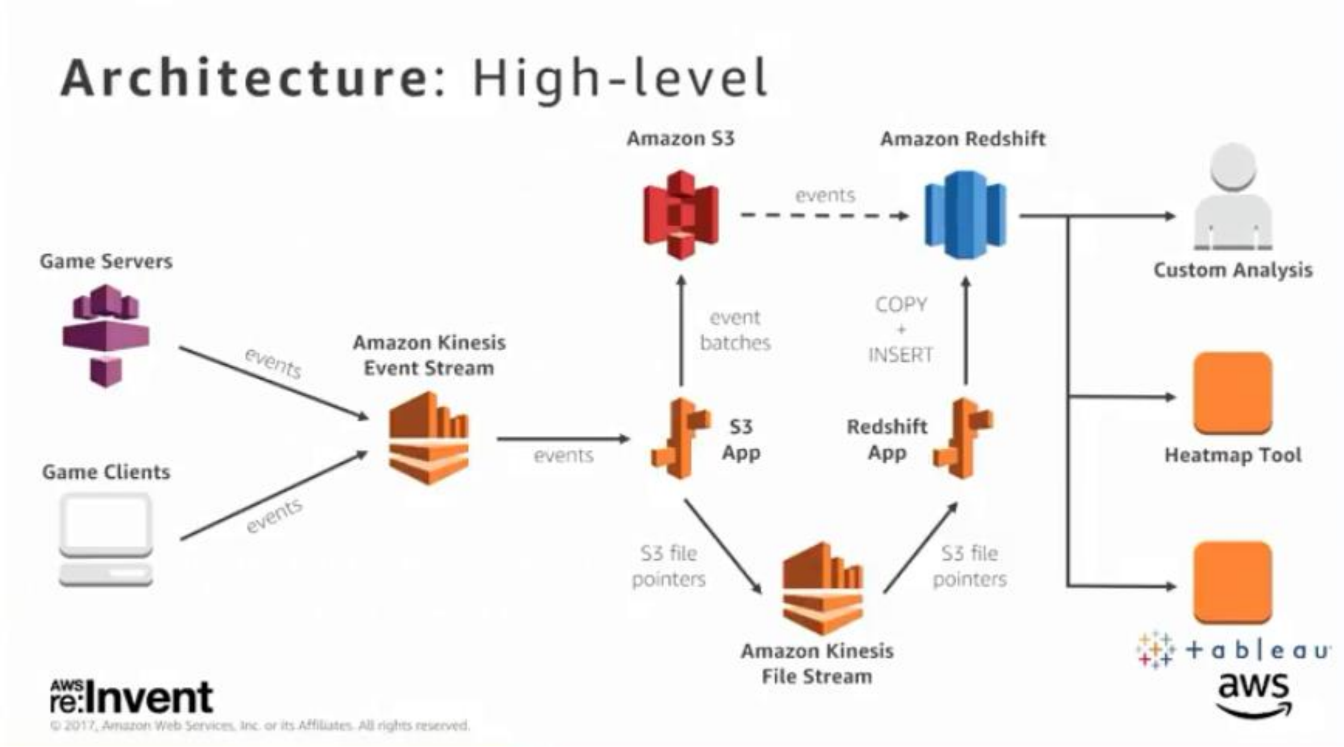
We now want to generate the heatmap using a python script



## Analyze: Heatmaps with Python

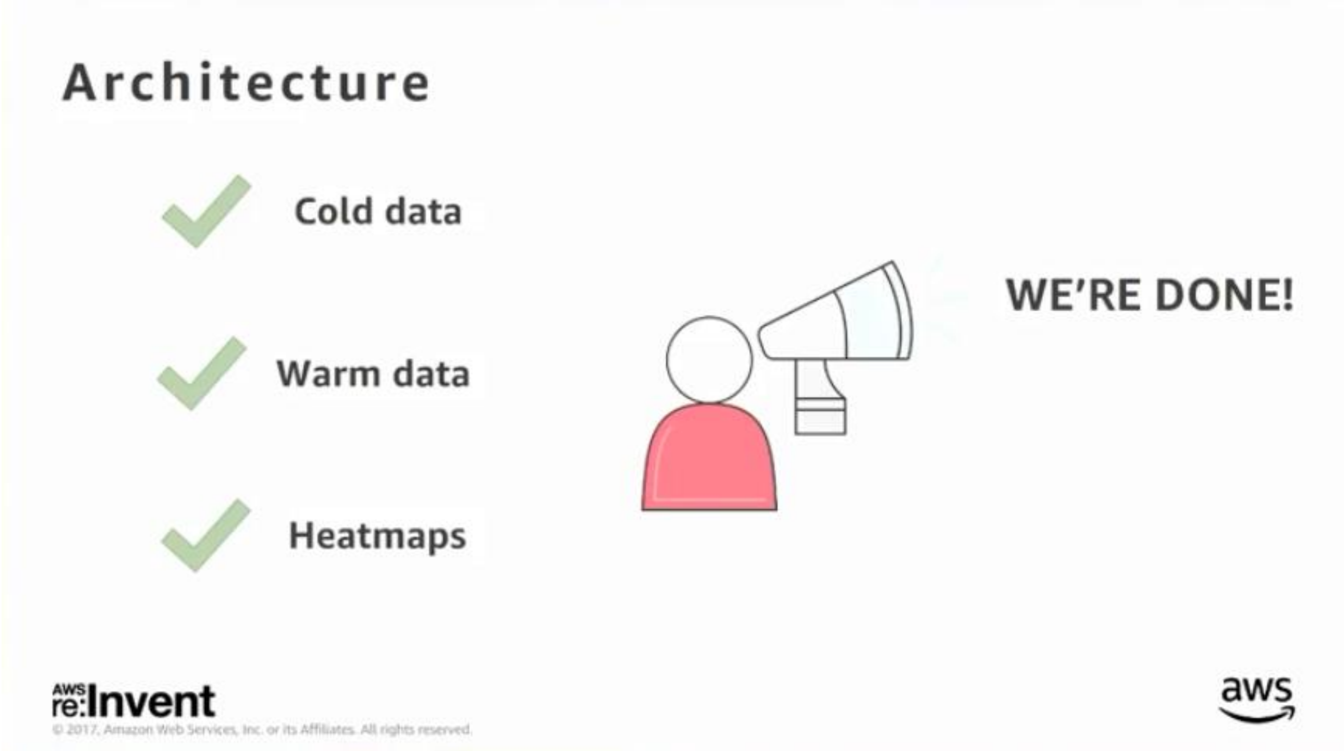
```
import pg8000, pandas, numpy, matplotlib.pyplot as plt

# Connect to DB
conn = pg8000.connect(user='user', password='p@ssw0rd', host=host_name, port=5439, database='analytics')
cursor = conn.cursor()

# Run aggregation query
cursor.execute('''SELECT FLOOR(position_x), FLOOR(position_y), COUNT(*)
                  FROM game.events
                  WHERE level_id='map_name' AND event_type='player_death'
                  GROUP BY 1, 2''')
deaths = cursor.fetchall()

# Convert (x,y) world space to image pixel space

#Generate hexbin plot
df = pandas.DataFrame.from_records(deaths, columns=['pos_x', 'pos_y', 'deaths'])
df.plot.hexbin(x='pos_x', y='pos_y', C='deaths', reduce_C_function=numpy.max, gridsize=25, alpha=0.5, linewidth=0)

#Draw image and plot
plt.imshow(X=plt.imread('map_background_image.png'), zorder=0)
plt.show()
```

We are using a library called pg8000 to connect to the Redshift database. Redshift can crunch data aggregations quickly. We also use the pandas dataframe data format and plot the data



## Architecture: High-level

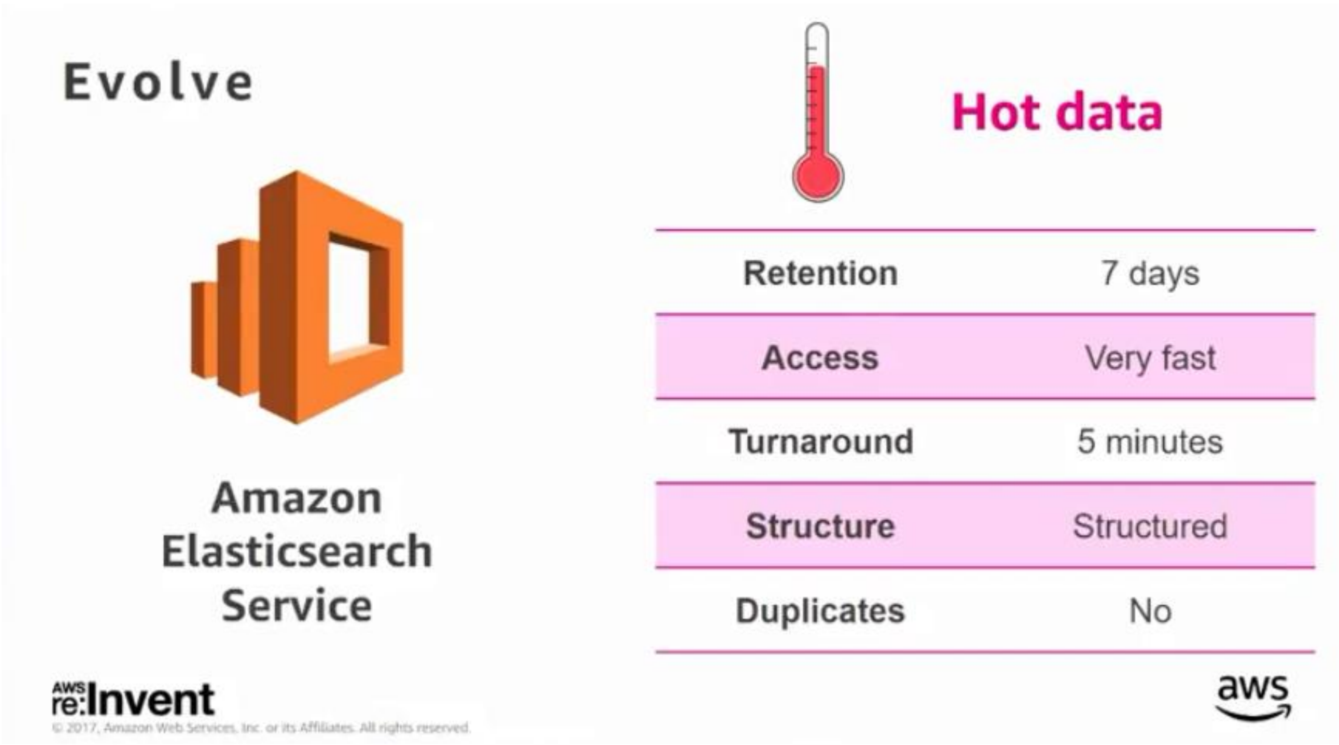
We have now finished our 4-stage architecture



## Architecture

- ✓ Cold data
- ✓ Warm data
- ✓ Heatmaps

WE'RE DONE!



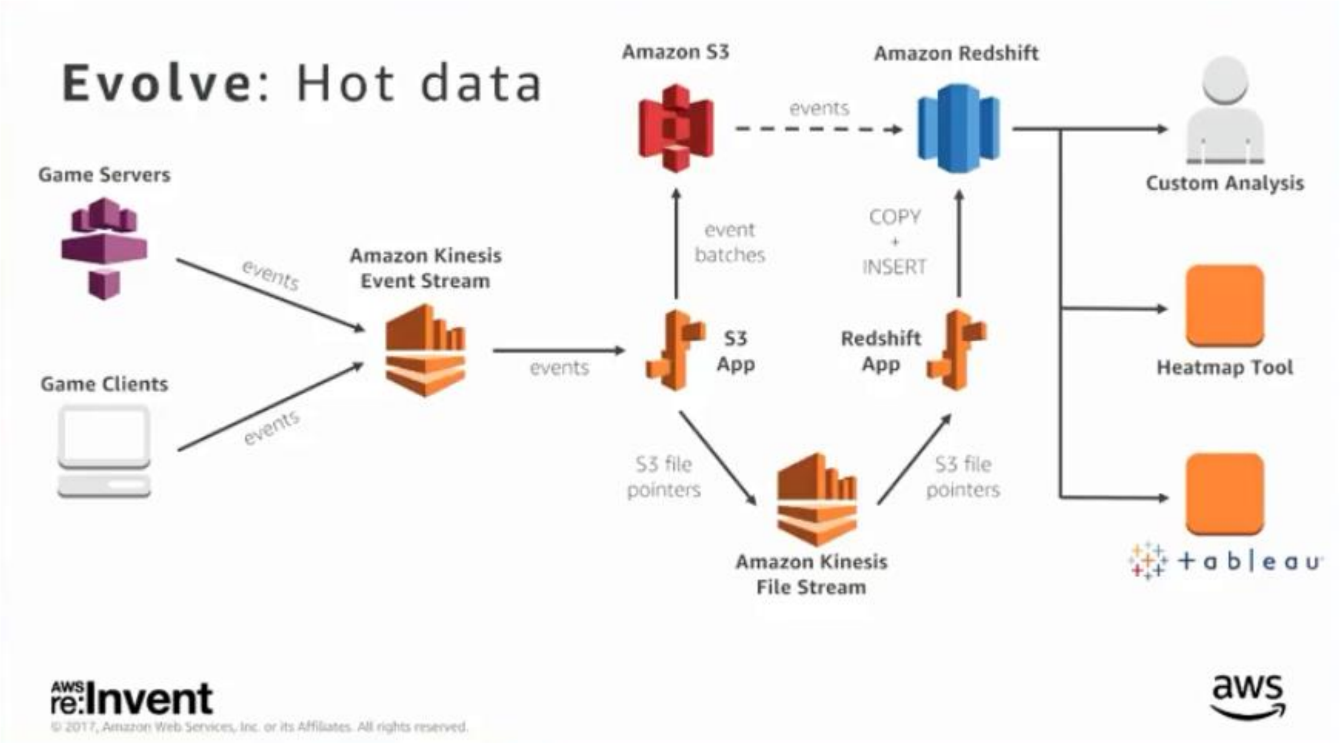
## Evolve

Amazon Elasticsearch Service

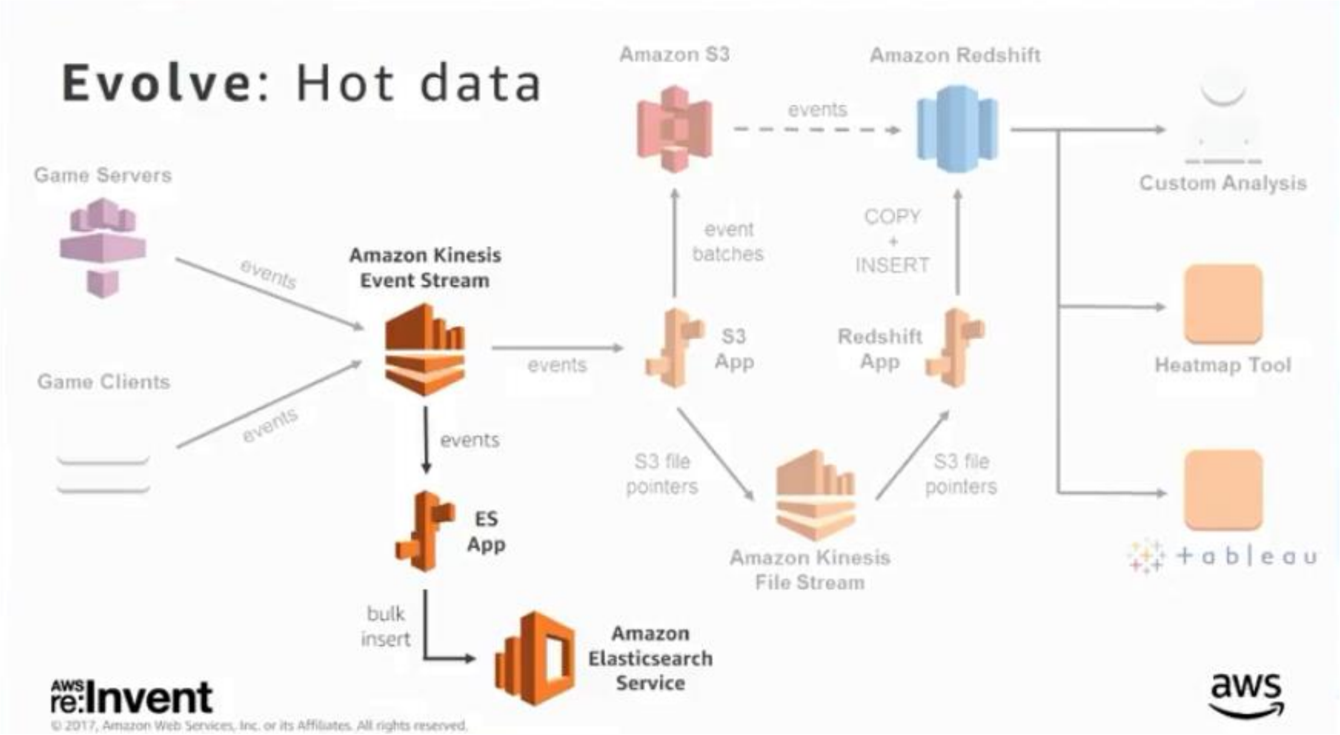
**Hot data**

| | |
|---|---|
| Retention | 7 days |
| Access | Very fast |
| Turnaround | 5 minutes |
| Structure | Structured |
| Duplicates | No |

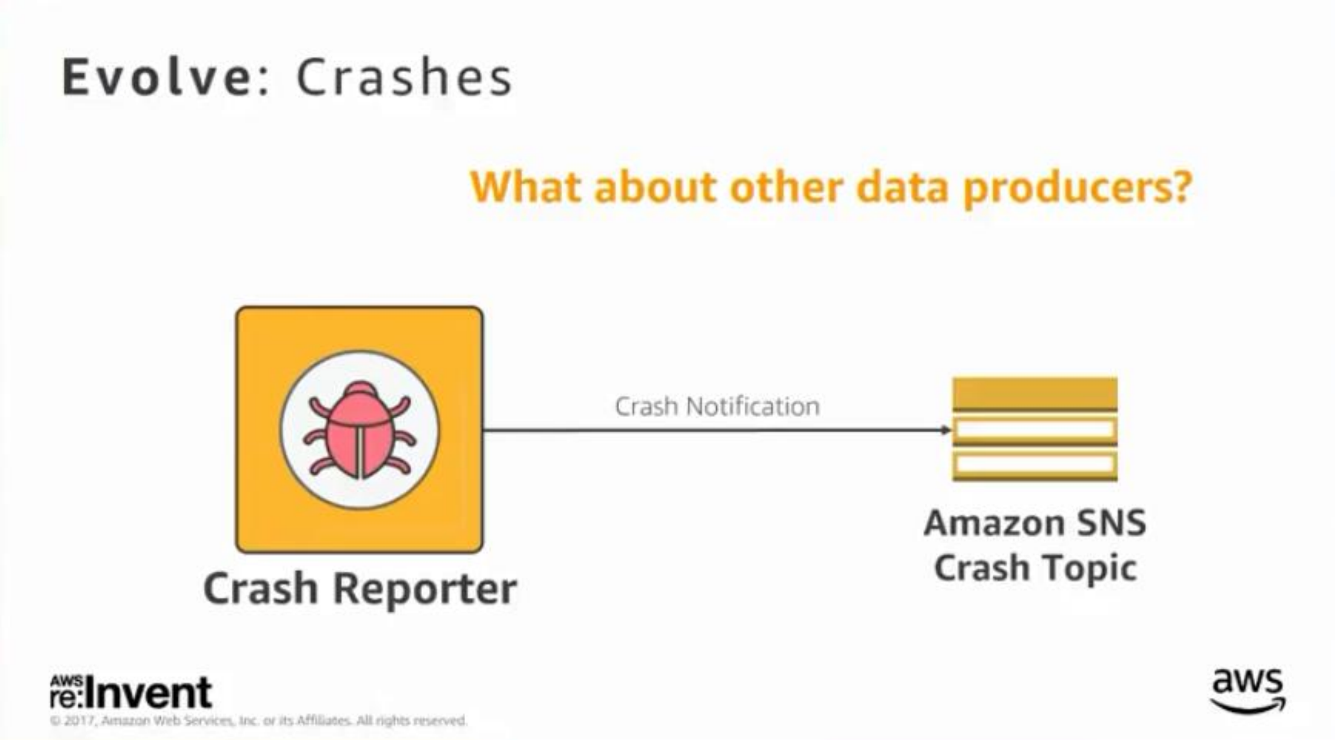
Hot data is because 'most recent is most hot', we are keeping a week data only for hot data and we use Elasticsearch, it is very fast and does cool stuff with time series data, has plugins like kibana



Where do we add Elasticsearch?

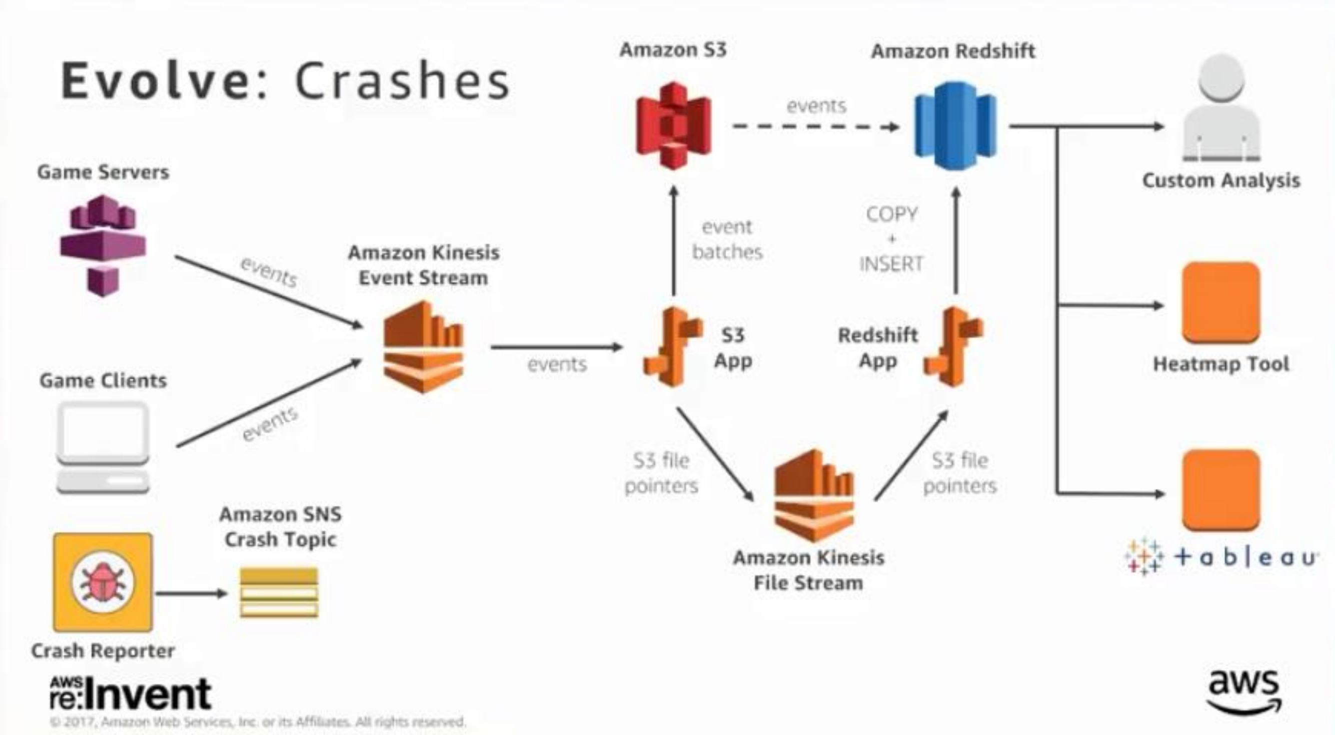


We hooked it up to the Kinesis stream using an Elasticsearch App (ES App) as above. We also use the ES App to drop some events that we don’t want stored in Elasticsearch like combat events



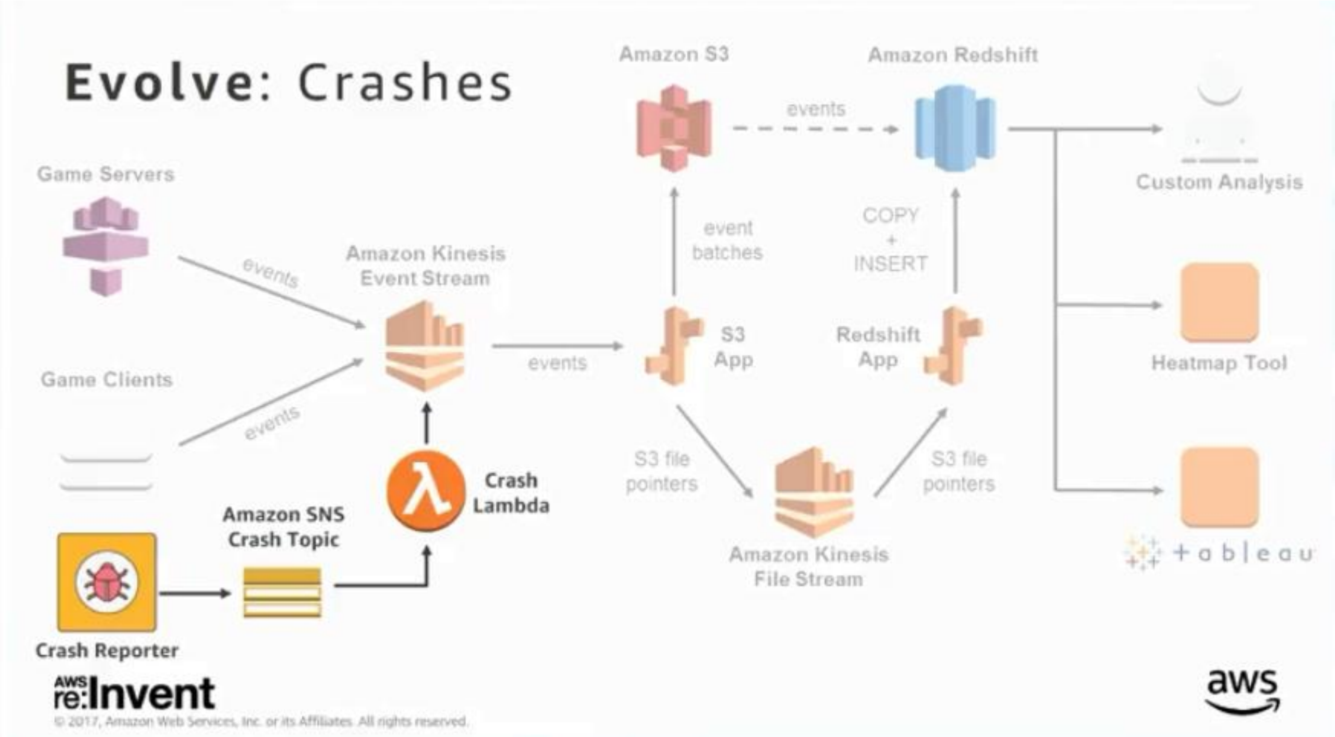
## Evolve: Crashes

What about other data producers?



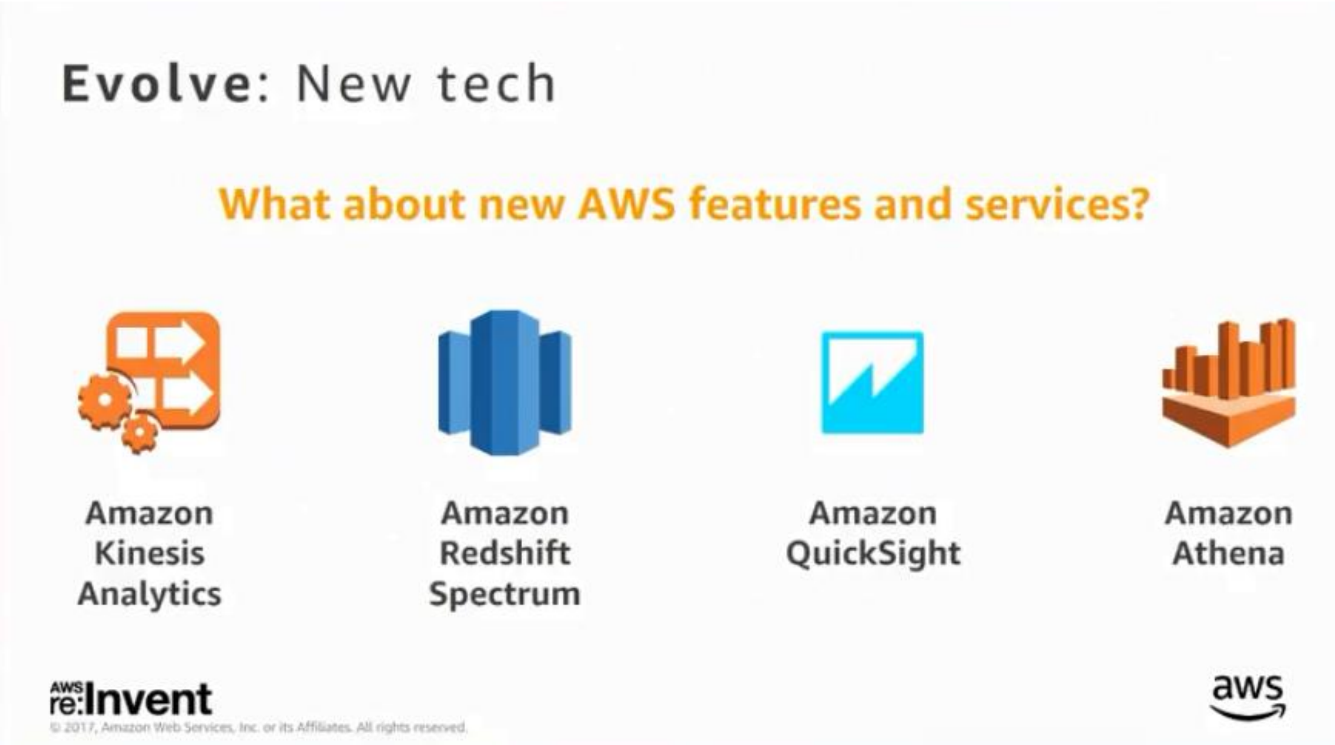
## Evolve: Crashes

## Evolve: Crashes



re:Invent

## Evolve: New tech



**What about new AWS features and services?**

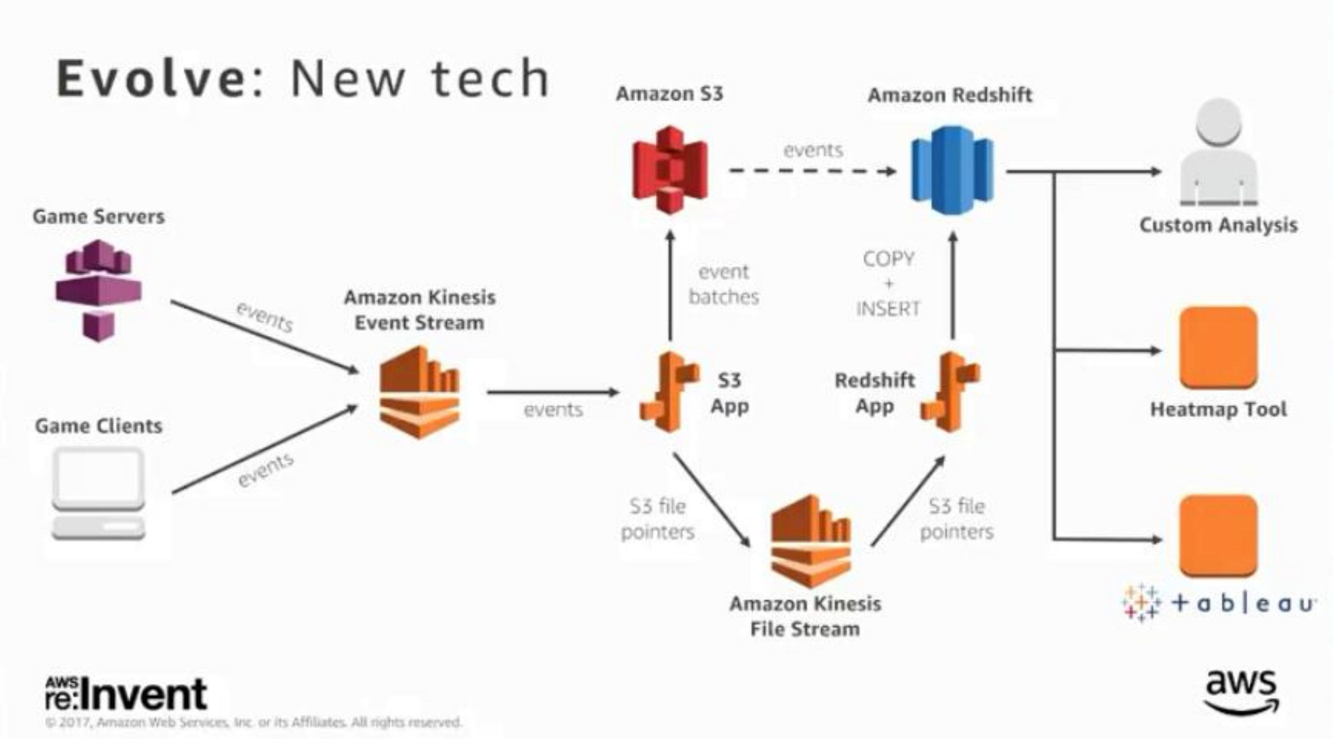
Amazon Kinesis Analytics

Amazon Redshift Spectrum

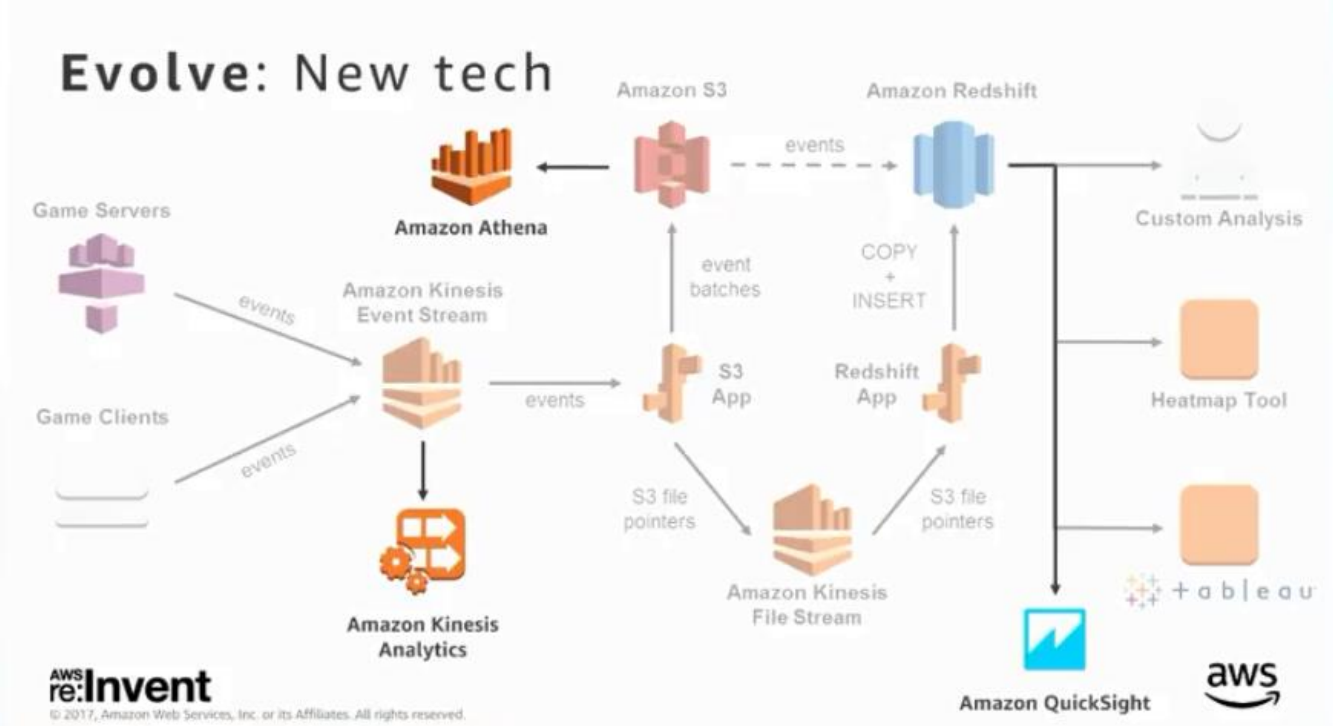
Amazon QuickSight

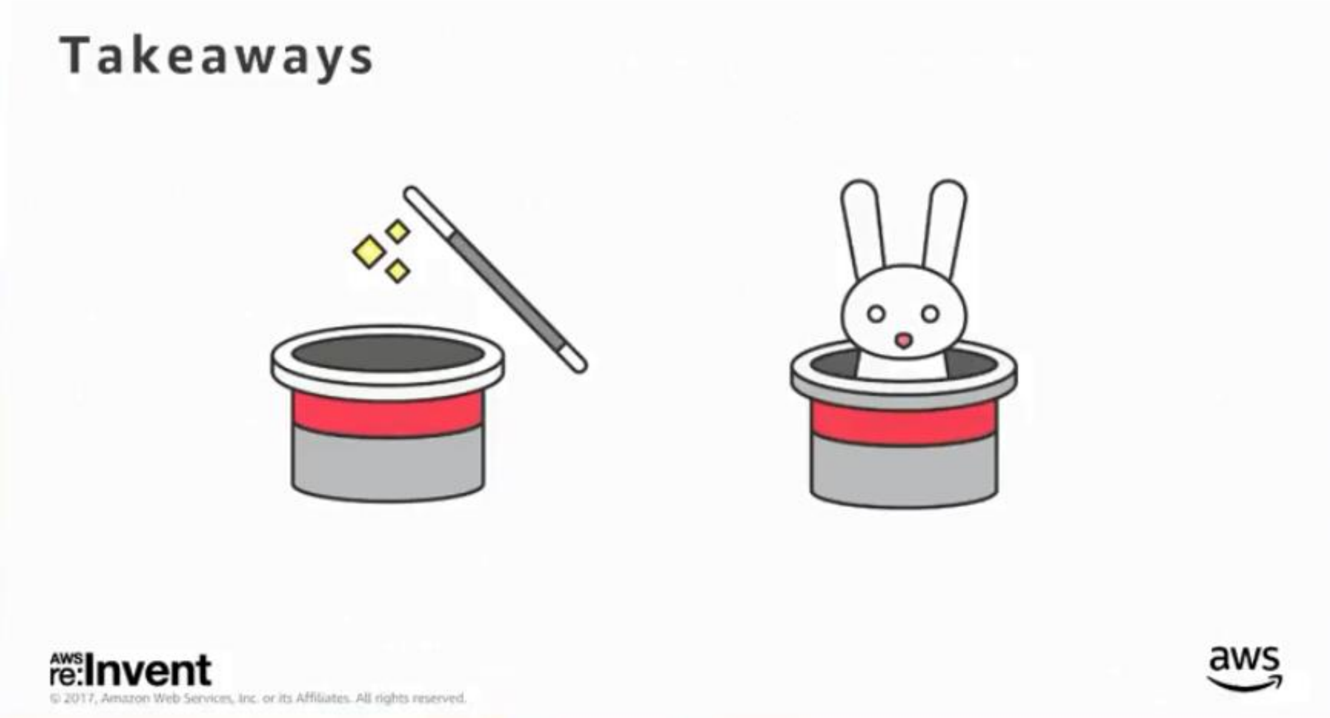
Amazon Athena

re:Invent



We want to evaluate new services now available to refine our architecture

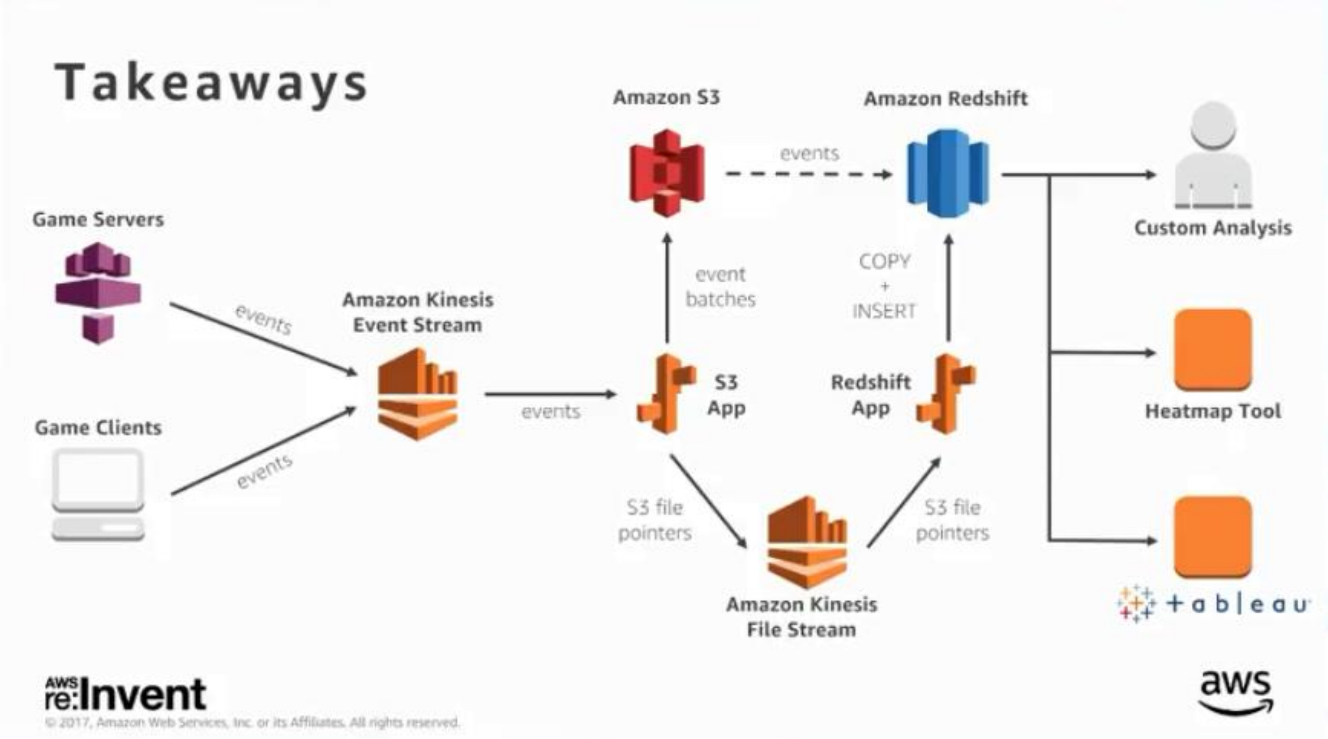




## Takeaways

re:Invent

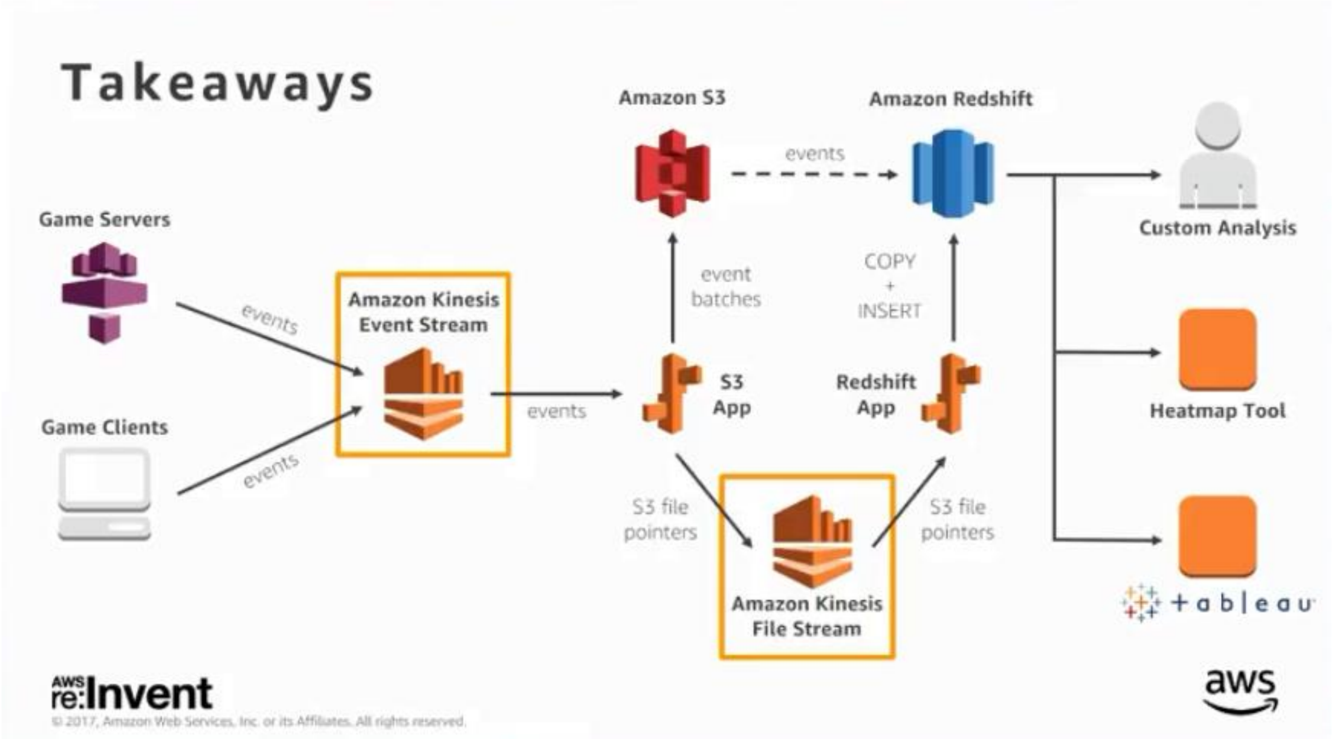
aws



## Takeaways

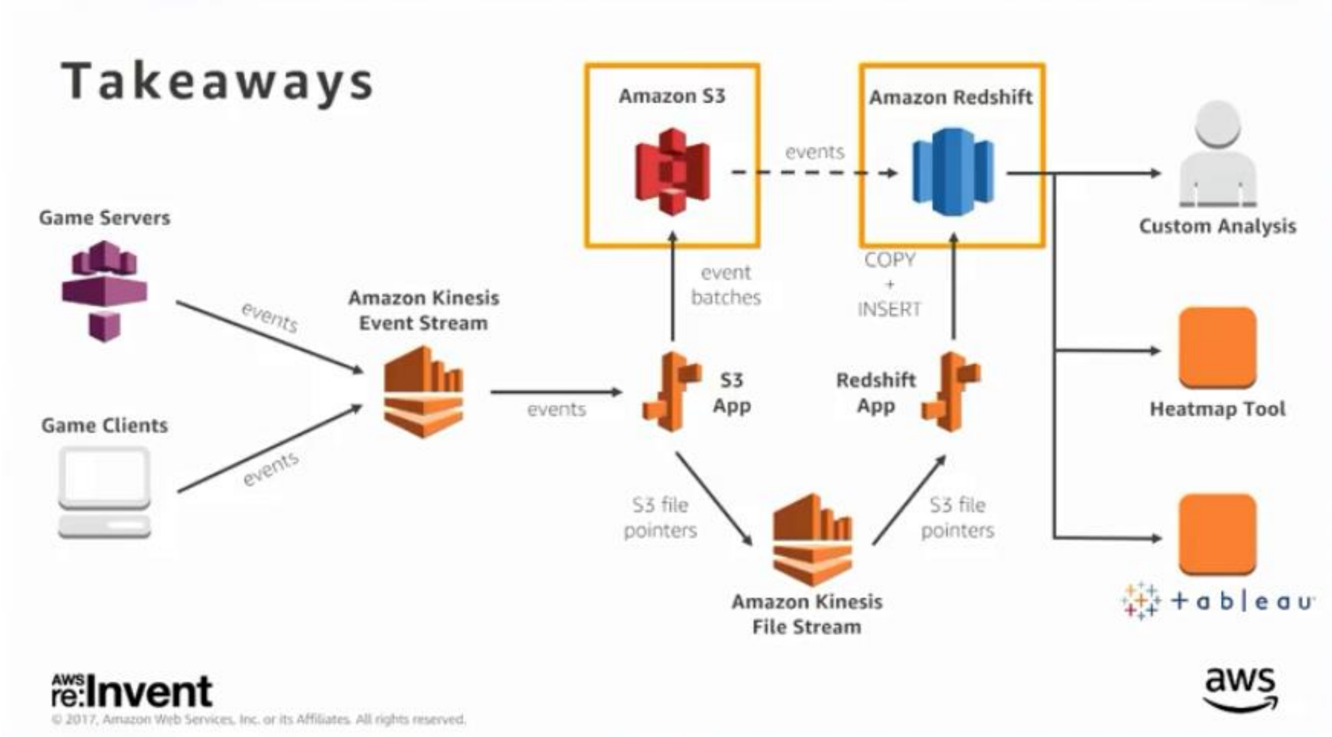
re:Invent

aws



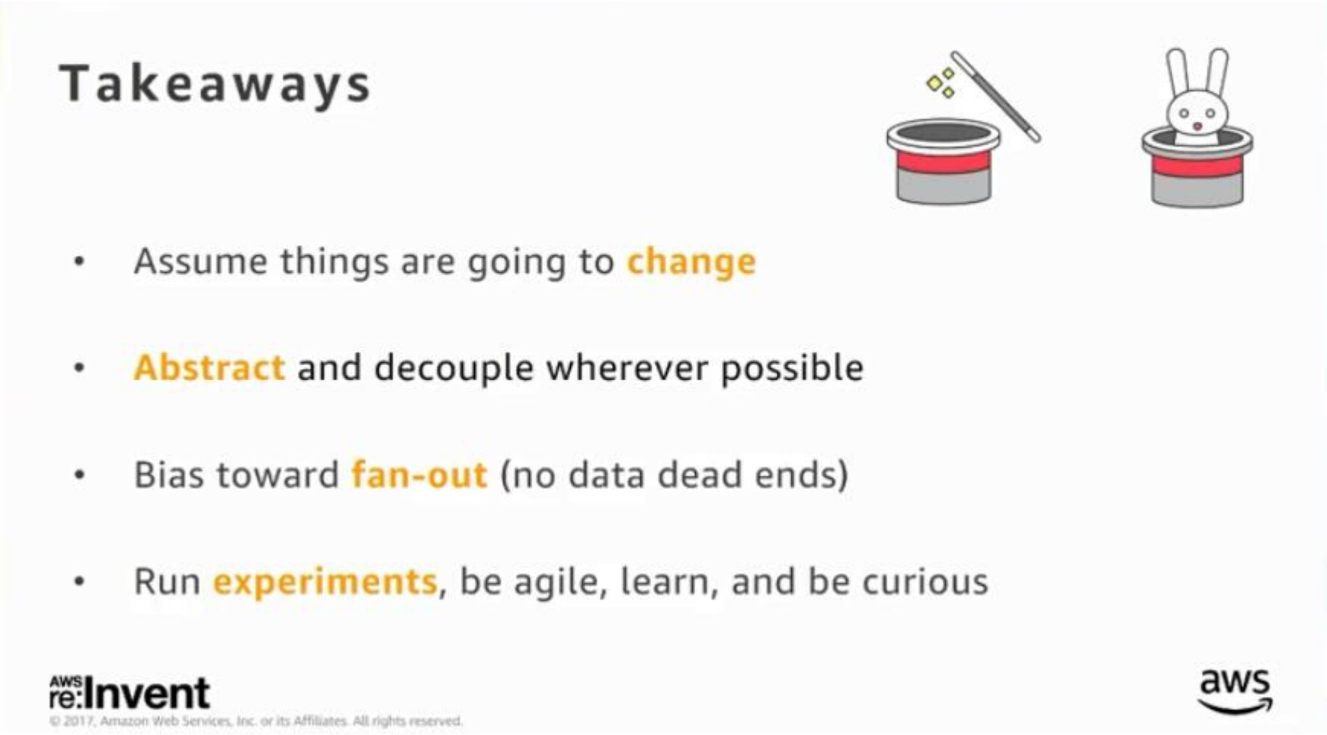
Takeaways

Those Kinesis streams really don't need to be there from a pure functionality point of view, but they help down the line



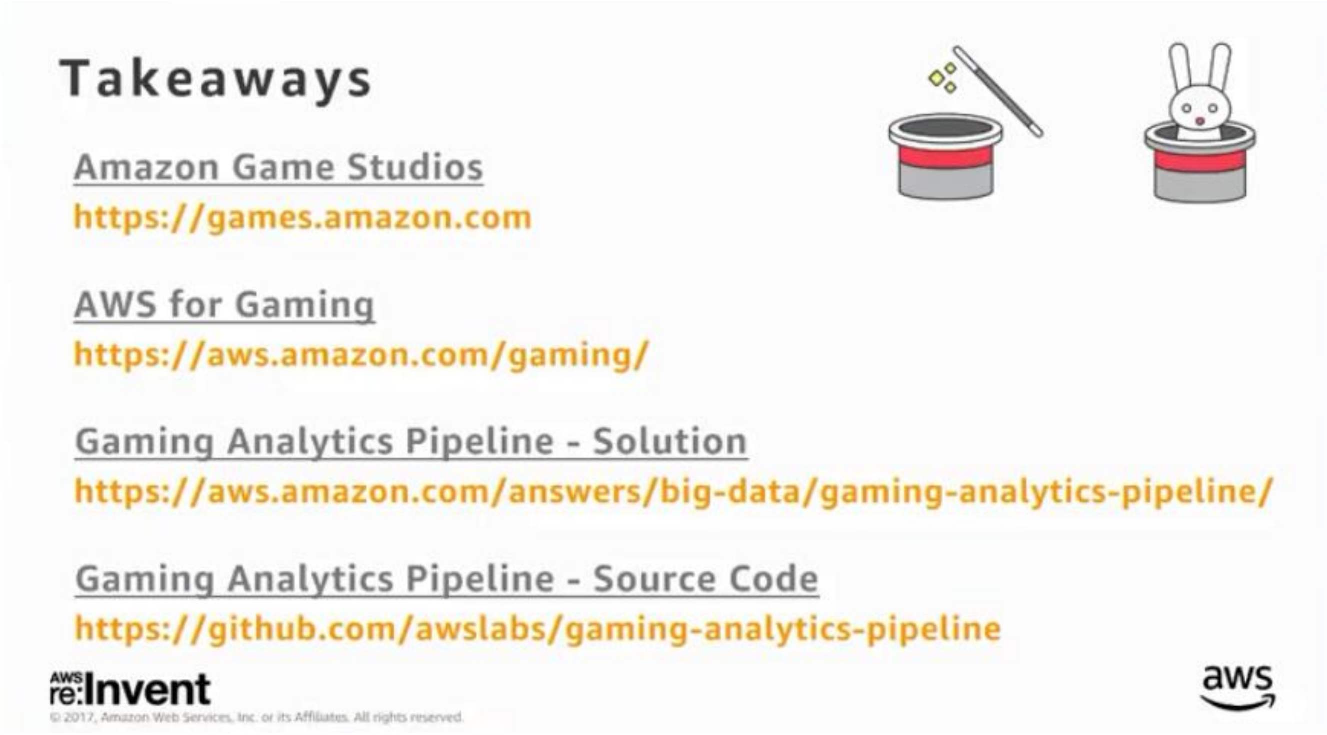
Takeaways

Putting your data into data stores with fan out capability is recommended because they can enable you do many other things in future



## Takeaways

- Assume things are going to **change**
- **Abstract** and decouple wherever possible
- Bias toward **fan-out** (no data dead ends)
- Run **experiments**, be agile, learn, and be curious



## Takeaways

**Amazon Game Studios**
**https://games.amazon.com**

**AWS for Gaming**
**https://aws.amazon.com/gaming/**

**Gaming Analytics Pipeline - Solution**
**https://aws.amazon.com/answers/big-data/gaming-analytics-pipeline/**

**Gaming Analytics Pipeline - Source Code**
**https://github.com/awslabs/gaming-analytics-pipeline**

AWS re:Invent

Thank you!