Big Data Architectural Patterns and Best Practices on AWS

Big Data Montréal (BDM52)

Antoine Généreux, Solutions Architect, AWS

April 4th, 2017
What to Expect from the Session

Big data challenges
Architectural principles
How to simplify big data processing
What technologies should you use?
  • Why?
  • How?
Reference architecture
Design patterns
Ever-Increasing Big Data

- Volume
- Velocity
- Variety
Big Data Evolution

Batch processing

Stream processing

Machine learning
Cloud Services Evolution

Virtual machines

Managed services

Serverless
Plethora of Tools

- hadoop
- Hive
- Spark
- EMR
- S3
- DynamoDB
- SQS
- Presto
- Cassandra
- MongoDB
- AWS Redshift
- AWS Glacier
- RDS
- ElastiCache
- Apache HBase
- Apache Kafka
- Apache Storm
- Amazon Kinesis
- Amazon Kinesis Streams app
- Data Pipeline
- Amazon ElasticSearch Service
- Kibana
- Logstash
- Elasticsearch
- Apache Zeppelin
- Jupyter
- Lambda
- Amazon ML
- DynamoDB Streams
- Amazon Kinesis
Big Data Challenges

Why?

How?

What tools should I use?

Is there a reference architecture?
Architectural Principles

Build decoupled systems
  • Data → Store → Process → Store → Analyze → Answers

Use the right tool for the job
  • Data structure, latency, throughput, access patterns

Leverage AWS managed services
  • Scalable/elastic, available, reliable, secure, no/low admin

Use log-centric design patterns
  • Immutable logs, materialized views

Be cost-conscious
  • Big data ≠ big cost
Simplify Big Data Processing

COLLECT → STORE → PROCESS/ANALYZE → CONSUME

Time to answer (Latency)
Throughput
Cost

data → answers
COLLECT
Types of Data

- In-memory data structures
- Database records
- Search documents
- Log files
- Messages
- Data streams

Transactions

Files

Events
What Is the Temperature of Your Data?
# Data Characteristics: Hot, Warm, Cold

<table>
<thead>
<tr>
<th></th>
<th>Hot</th>
<th>Warm</th>
<th>Cold</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Volume</strong></td>
<td>MB–GB</td>
<td>GB–TB</td>
<td>PB–EB</td>
</tr>
<tr>
<td><strong>Item size</strong></td>
<td>B–KB</td>
<td>KB–MB</td>
<td>KB–TB</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>ms</td>
<td>ms, sec</td>
<td>min, hrs</td>
</tr>
<tr>
<td><strong>Durability</strong></td>
<td>Low–high</td>
<td>High</td>
<td>Very high</td>
</tr>
<tr>
<td><strong>Request rate</strong></td>
<td>Very high</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Cost/GB</strong></td>
<td>$$-$$</td>
<td>$$-$$-$$-$$ $</td>
<td>$</td>
</tr>
</tbody>
</table>
Store
Types of Data Stores

- In-memory: Caches, data structure servers
- Database: SQL & NoSQL databases
- Search: Search engines
- File store: File systems
- Queue: Message queues
- Stream storage: Pub/sub message queues
Message & Stream Storage

**Amazon SQS**
- Managed message queue service

**Apache Kafka**
- High throughput distributed streaming platform

**Amazon Kinesis Streams**
- Managed stream storage + processing

**Amazon Kinesis Firehose**
- Managed data delivery

**Amazon DynamoDB**
- Managed NoSQL database
- Tables can be stream-enabled

---

COLLECT

<table>
<thead>
<tr>
<th>Applications</th>
<th>In-memory</th>
<th>Database</th>
<th>Search</th>
<th>File store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web apps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile apps</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data centers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Logging**

<table>
<thead>
<tr>
<th>Applications</th>
<th>In-memory</th>
<th>Database</th>
<th>Search</th>
<th>File store</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS CloudTrail</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Transport**

<table>
<thead>
<tr>
<th>Applications</th>
<th>In-memory</th>
<th>Database</th>
<th>Search</th>
<th>File store</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Import/Export</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snowball</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IoT**

<table>
<thead>
<tr>
<th>Applications</th>
<th>In-memory</th>
<th>Database</th>
<th>Search</th>
<th>File store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Devices</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensors &amp; IoT platforms</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Messaging**

<table>
<thead>
<tr>
<th>Applications</th>
<th>In-memory</th>
<th>Database</th>
<th>Search</th>
<th>File store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon SQS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache Kafka</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Kinesis Streams</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon Kinesis Firehose</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amazon DynamoDB Streams</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Devices**

<table>
<thead>
<tr>
<th>Applications</th>
<th>In-memory</th>
<th>Database</th>
<th>Search</th>
<th>File store</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS IoT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Sensors & IoT platforms**

<table>
<thead>
<tr>
<th>Applications</th>
<th>In-memory</th>
<th>Database</th>
<th>Search</th>
<th>File store</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS IoT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---
Why Stream Storage?

- Decouple producers & consumers
- Persistent buffer
- Collect multiple streams
- Preserve client ordering
- Parallel consumption
- Streaming MapReduce

Producer 1: 4 3 2 1  
Key = red

Producer 2: 4 3 2 1  
Key = green

Producer 3: 4 3 2 1  
Key = blue

Producer n: 4 3 2 1  
Key = violet

DynamoDB stream
Amazon Kinesis stream
Kafka topic

Consumer 1:
- Count of red = 4
- Count of violet = 4

Consumer 2:
- Count of blue = 4
- Count of green = 4

shard 1 / partition 1
shard 2 / partition 2
What About Amazon SQS?

- Decouple producers & consumers
- Persistent buffer
- Collect multiple streams
- **No** client ordering (Standard)
  - FIFO queue preserves client ordering
- **No** streaming MapReduce
- **No** parallel consumption
  - Amazon SNS can publish to multiple SNS subscribers (queues or λ functions)
## Which Stream/Message Storage Should I Use?

<table>
<thead>
<tr>
<th></th>
<th>Amazon DynamoDB Streams</th>
<th>Amazon Kinesis Streams</th>
<th>Amazon Kinesis Firehose</th>
<th>Apache Kafka</th>
<th>Amazon SQS (Standard)</th>
<th>Amazon SQS (FIFO)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AWS managed</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Guaranteed ordering</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Delivery (deduping)</strong></td>
<td>Exactly-once</td>
<td>At-least-once</td>
<td>At-least-once</td>
<td>At-least-once</td>
<td>At-least-once</td>
<td>Exactly-once</td>
</tr>
<tr>
<td><strong>Data retention period</strong></td>
<td>24 hours</td>
<td>7 days</td>
<td>N/A</td>
<td>Configurable</td>
<td>14 days</td>
<td>14 days</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>3 AZ</td>
<td>3 AZ</td>
<td>3 AZ</td>
<td>Configurable</td>
<td>3 AZ</td>
<td>3 AZ</td>
</tr>
<tr>
<td><strong>Scale / throughput</strong></td>
<td>No limit / ~ table IOPS</td>
<td>No limit / ~ shards</td>
<td>No limit / automatic</td>
<td>No limit / automatic</td>
<td>No limits / automatic</td>
<td>300 TPS / queue</td>
</tr>
<tr>
<td><strong>Parallel consumption</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Stream MapReduce</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Row/object size</strong></td>
<td>400 KB</td>
<td>1 MB</td>
<td>Destination row/object size</td>
<td>Configurable</td>
<td>256 KB</td>
<td>256 KB</td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Higher (table cost)</td>
<td>Low</td>
<td>Low</td>
<td>Low (+admin)</td>
<td>Low-medium</td>
<td>Low-medium</td>
</tr>
</tbody>
</table>

**New**
Amazon S3 is highlighted as the file storage solution in the diagram.
Why Is Amazon S3 Good for Big Data?

- Natively supported by big data frameworks (Spark, Hive, Presto, etc.)
- No need to run compute clusters for storage (unlike HDFS)
- Can run transient Hadoop clusters & Amazon EC2 Spot Instances
- Multiple & heterogeneous analysis clusters can use the same data
- Unlimited number of objects and volume of data
- Very high bandwidth – no aggregate throughput limit
- Designed for 99.99% availability – can tolerate zone failure
- Designed for 99.999999999% durability
- No need to pay for data replication
- Native support for versioning
- Tiered-storage (Standard, IA, Amazon Glacier) via life-cycle policies
- Secure – SSL, client/server-side encryption at rest
- Low cost
What About HDFS & Data Tiering?

• Use HDFS for very frequently accessed (hot) data
• Use Amazon S3 Standard for frequently accessed data
• Use Amazon S3 Standard – IA for less frequently accessed data
• Use Amazon Glacier for archiving cold data
Anti-Pattern

Applications

Data Tier

RDBMS
Best Practice: Use the Right Tool for the Job

Data Tier

Applications

In-memory
Amazon ElastiCache
Redis
Memcached

NoSQL
Amazon DynamoDB
Cassandra
HBase
MongoDB

SQL
Amazon Aurora
Amazon RDS
MySQL
PostgreSQL
Oracle
SQL Server

Search
Amazon Elasticsearch Service
Amazon ElastiCache
- Managed Memcached or Redis service

Amazon DynamoDB
- Managed NoSQL database service

Amazon RDS
- Managed relational database service

Amazon Elasticsearch Service
- Managed Elasticsearch service
Which Data Store Should I Use?

Data structure → Fixed schema, JSON, key-value

Access patterns → Store data in the format you will access it

Data characteristics → Hot, warm, cold

Cost → Right cost
### Data Structure and Access Patterns

<table>
<thead>
<tr>
<th>Access Patterns</th>
<th>What to use?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put/Get (key, value)</td>
<td>In-memory, NoSQL</td>
</tr>
<tr>
<td>Simple relationships $\rightarrow$ 1:N, M:N</td>
<td>NoSQL</td>
</tr>
<tr>
<td>Multi-table joins, transaction, SQL</td>
<td>SQL</td>
</tr>
<tr>
<td>Faceting, search</td>
<td>Search</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>What to use?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed schema</td>
<td>SQL, NoSQL</td>
</tr>
<tr>
<td>Schema-free (JSON) (Key, value)</td>
<td>NoSQL, Search</td>
</tr>
<tr>
<td>(Key, value)</td>
<td>In-memory, NoSQL</td>
</tr>
</tbody>
</table>
In-memory SQL Request rate
High Low
Cost/GB
High Low
Latency
Low High
Data volume
Low High

Amazon Glacier
S3
Search
NoSQL
In-memory
SQL
Hot data
Warm data
Cold data

Structure
High
Low
## Which Data Store Should I Use?

<table>
<thead>
<tr>
<th></th>
<th>Amazon ElastiCache</th>
<th>Amazon DynamoDB</th>
<th>Amazon RDS/Aurora</th>
<th>Amazon ES</th>
<th>Amazon S3</th>
<th>Amazon Glacier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Average latency</strong></td>
<td>ms</td>
<td>ms</td>
<td>ms, sec</td>
<td>ms,sec</td>
<td>ms, sec, min (~ size)</td>
<td>hrs</td>
</tr>
<tr>
<td><strong>Typical data stored</strong></td>
<td>GB (no limit)</td>
<td>GB–TBs (no limit)</td>
<td>GB–TB (64 TB max)</td>
<td>GB–TB</td>
<td>MB–PB (no limit)</td>
<td>GB–PB (no limit)</td>
</tr>
<tr>
<td><strong>Typical item size</strong></td>
<td>B-KB (400 KB max)</td>
<td>KB (64 KB max)</td>
<td>KB (2 GB max)</td>
<td>B-KB (2 GB max)</td>
<td>KB-TB (5 TB max)</td>
<td>GB (40 TB max)</td>
</tr>
<tr>
<td><strong>Request Rate</strong></td>
<td>High – very high</td>
<td>Very high (no limit)</td>
<td>High</td>
<td>High</td>
<td>Low – high (no limit)</td>
<td>Very low</td>
</tr>
<tr>
<td><strong>Storage cost GB/month</strong></td>
<td>$$</td>
<td>¥¢¢</td>
<td>¥¢¢</td>
<td>¥¢¢</td>
<td>¥</td>
<td>¥4/10</td>
</tr>
<tr>
<td><strong>Durability</strong></td>
<td>Low - moderate</td>
<td>Very high</td>
<td>Very high</td>
<td>High</td>
<td>Very high</td>
<td>Very high</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>High 2 AZ</td>
<td>Very high 3 AZ</td>
<td>Very high 3 AZ</td>
<td>High 2 AZ</td>
<td>Very high 3 AZ</td>
<td>Very high 3 AZ</td>
</tr>
</tbody>
</table>

### Hot data

- Amazon ElastiCache
- Amazon DynamoDB
- Amazon RDS/Aurora
- Amazon ES
- Amazon S3
- Amazon Glacier

### Warm data

- Amazon ElastiCache
- Amazon DynamoDB
- Amazon RDS/Aurora
- Amazon ES
- Amazon S3
- Amazon Glacier

### Cold data

- Amazon ElastiCache
- Amazon DynamoDB
- Amazon RDS/Aurora
- Amazon ES
- Amazon S3
- Amazon Glacier
Cost-Conscious Design

Example: Should I use Amazon S3 or Amazon DynamoDB?

“I’m currently scoping out a project. The design calls for many small files, perhaps up to a billion during peak. The total size would be on the order of 1.5 TB per month…”

<table>
<thead>
<tr>
<th>Request rate (Writes/sec)</th>
<th>Object size (Bytes)</th>
<th>Total size (GB/month)</th>
<th>Objects per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>2048</td>
<td>1483</td>
<td>777,600,000</td>
</tr>
</tbody>
</table>
Cost-Conscious Design

Example: Should I use Amazon S3 or Amazon DynamoDB?

https://calculator.s3.amazonaws.com/index.html
Amazon S3 or DynamoDB?

<table>
<thead>
<tr>
<th>Request rate (Writes/sec)</th>
<th>Object size (Bytes)</th>
<th>Total size (GB/month)</th>
<th>Objects per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>2,048</td>
<td>1,483</td>
<td>777,600,000</td>
</tr>
</tbody>
</table>

Amazon DynamoDB is a high performance non-relational database service that is easy to set up, operate, and scale. It is designed to address the core problems of database management, performance, scalability, and reliability. It also provides predictable high performance and low latency at scale.

Amazon S3 is storage for the Internet. It is designed to make web-scale computing easier for developers.

Storage:
- Storage: 1,483 GB
- Reduced Redundancy Storage: 0 GB

Requests:
- PUT/COPY/POST/LIST Requests: 7,776,000 Requests
- GET and Other Requests: 0 Requests

Amazon S3 Service (US-East) charges:
- Storage: $44.27
- Put/List Requests: $3,888.00

Amazon DynamoDB Service (US-East) charges:
- Provisioned Throughput Capacity: $261.69
- Indexed Data Storage: $382.61
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Request rate ( Writes/sec )</th>
<th>Object size (Bytes)</th>
<th>Total size (GB/month)</th>
<th>Objects per month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>300</td>
<td>2,048</td>
<td>1,483</td>
<td>777,600,000</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>300</td>
<td>32,768</td>
<td>23,730</td>
<td>777,600,000</td>
</tr>
</tbody>
</table>

Amazon S3:
- Storage: $700.55
- Put/List Requests: $3888.00
- Amazon DynamoDB Service (US-East): $10131.40
- Provisioned Throughput Capacity: $4187.04
- Indexed Data Storage: $5944.36
- DynamoDB Streams: $0.00

Amazon DynamoDB:
- Storage: $44.27
- Put/List Requests: $3888.00
- Amazon DynamoDB Service (US-East): $644.30
- Provisioned Throughput Capacity: $261.69
- Indexed Data Storage: $382.61
- DynamoDB Streams: $0.00

Use Amazon DynamoDB and Amazon S3 for web services.
PROCESS / ANALYZE
Analytics Types & Frameworks

Batch
- Takes minutes to hours
- Example: Daily/weekly/monthly reports
- Amazon EMR (MapReduce, Hive, Pig, Spark)

Interactive
- Takes seconds
- Example: Self-service dashboards
- Amazon Redshift, Amazon Athena, Amazon EMR (Presto, Spark)

Message
- Takes milliseconds to seconds
- Example: Message processing
- Amazon SQS applications on Amazon EC2

Stream
- Takes milliseconds to seconds
- Example: Fraud alerts, 1 minute metrics
- Amazon EMR (Spark Streaming), Amazon Kinesis Analytics, KCL, Storm, AWS Lambda

Machine Learning
- Takes milliseconds to minutes
- Example: Fraud detection, forecast demand
- Amazon ML, Amazon EMR (Spark ML)
<table>
<thead>
<tr>
<th>Which Stream &amp; Message Processing Technology Should I Use?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Amazon EMR (Spark Streaming)</strong></td>
</tr>
<tr>
<td><strong>AWS managed</strong></td>
</tr>
<tr>
<td><strong>Serverless</strong></td>
</tr>
<tr>
<td><strong>Scale / throughput</strong></td>
</tr>
<tr>
<td><strong>Availability</strong></td>
</tr>
<tr>
<td><strong>Programming languages</strong></td>
</tr>
<tr>
<td><strong>Uses</strong></td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
</tr>
<tr>
<td>Use case</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>Use case</td>
</tr>
<tr>
<td>Scale/throughput</td>
</tr>
<tr>
<td>AWS Managed Service</td>
</tr>
<tr>
<td>Storage</td>
</tr>
<tr>
<td>Optimization</td>
</tr>
<tr>
<td>Metadata</td>
</tr>
<tr>
<td>BI tools supports</td>
</tr>
<tr>
<td>Access controls</td>
</tr>
<tr>
<td>UDF support</td>
</tr>
</tbody>
</table>
What About ETL?

Data Integration Partners
Reduce the effort to move, cleanse, synchronize, manage, and automatize data related processes.

AWS Glue (Preview)
AWS Glue is a fully managed ETL service that makes it easy to understand your data sources, prepare the data, and move it reliably between data stores.

https://aws.amazon.com/big-data/partner-solutions/
CONSUME
Applications & API

Analysis and visualization

Notebooks

IDE
Putting It All Together
Design Patterns
Primitive: Decoupled Data Bus

Storage decoupled from processing
Multiple stages
Primitive: Pub/Sub

Parallel stream consumption/processing

Data

- Amazon Kinesis
- Amazon Kinesis Connector Library
- AWS Lambda
- Apache Spark

process
store
Analysis framework reads from or writes to multiple data stores

Data flow diagram:
- Data flows from Amazon Kinesis to Amazon Kinesis Connector Library, then to AWS Lambda.
- AWS Lambda flows to Amazon S3.
- Amazon S3 flows to Amazon DynamoDB.
- Amazon DynamoDB flows to Amazon EMR, which is connected to Spark SQL and Spark Streaming.
- Answers are generated from Amazon EMR.
Summary

Build decoupled systems
- Data → Store → Process → Store → Analyze → Answers

Use the right tool for the job
- Data structure, latency, throughput, access patterns

Leverage AWS managed services
- Scalable/elastic, available, reliable, secure, no/low admin

Use log-centric design patterns
- Immutable log, batch, interactive & real-time views

Be cost-conscious
- Big data ≠ big cost
AWS Free Tier (aws.amazon.com/free)

**Compute**
- **750 Hours** per month
  - Amazon EC2:Resizable compute capacity in the Cloud
    - Learn more about Amazon EC2 »

**Storage & Content Delivery**
- **5GB** of standard storage
  - Amazon S3: Secure, durable, and scalable object storage infrastructure
    - Learn more about Amazon S3 »

**Database**
- **750 Hours** per month of database usage
  - Amazon RDS: Managed Relational Database Service for MySQL, PostgreSQL, MariaDB, Oracle, BYOL, or SQL Server
    - Learn more about Amazon RDS »

**Compute**
- **1 Million** free requests per month
  - AWS Lambda: Compute service that runs your code in response to events and automatically manages the compute resources
    - Learn more about AWS Lambda »

**Analytics**
- **1GB** of SPICE capacity
  - Amazon QuickSight: Fast, easy-to-use, cloud-powered business analytics service at 1/10th the cost of traditional BI solutions
    - Learn more about Amazon QuickSight »

**12 months free and always free products**
AWS Free Tier includes offers that expire 12 months following sign up and others that never expire.

Learn More »
We’re Hiring! (jobs.amazon.com)

Montreal
• Solutions Architect (486946)

Toronto
• Solutions Architect (493577, 493576, 493575, 493574)
• Security Architect (480504)
• Cloud Infrastructure Architect (456134)
• Senior Solutions Architect (416514)
• Technical Account Manager (403410)

Vancouver
• Solutions Architect (493572, 493571)
• SI Solutions Architect (473414)
• Senior Solutions Architect (458043)
Thank you! Questions?

Antoine Généreux, genereux@amazon.com
Resources
Blogs, Whitepapers and Video Sessions

(Blog) Powering Amazon Redshift Analytics with Apache Spark and Amazon Machine Learning

(Blog) Implement Serverless Log Analytics Using Kinesis Analytics

(Blog) Harmonize, Search and Analyze Loosely Coupled Datasets on AWS

(Blog) Build an Event-Based Analytics Pipeline for Amazon Game Studios’ Breakaway

(Whitepaper) Big Data Analytics Options on AWS
https://d0.awsstatic.com/whitepapers/Big_Data_Analytics_Options_on_AWS.pdf

re:Invent 2016 Breakout Sessions | Big Data Mini Con (13 videos)
https://www.youtube.com/playlist?list=PLhr1KZpdzuki1XNNdkDwQqWObl5FDsV

re:Invent 2016 Breakout Sessions | Databases (26 videos)
https://www.youtube.com/playlist?list=PLhr1KZpdzukc97mkL2-TOE-vJtp22c1yc

re:Invent 2016 Breakout Sessions | Machine Learning Mini Con (14 videos)
https://www.youtube.com/playlist?list=PLhr1KZpdzukexYSNcl9iBbmn9iYKu2pu
Code Samples ([https://github.com/awslabs](https://github.com/awslabs))

Harmonize, Search and Analyze Loosely Coupled Datasets on AWS (code related to blog post)
[https://github.com/awslabs/harmonize-search-analyze](https://github.com/awslabs/harmonize-search-analyze)

AWS Data Lake Solution

Serverless MapReduce Reference Architecture
[https://github.com/awslabs/lamba-refarch-mapreduce](https://github.com/awslabs/lamba-refarch-mapreduce)

Streaming Analytics Pipeline
[https://github.com/awslabs/streaming-analytics-pipeline](https://github.com/awslabs/streaming-analytics-pipeline)

Real-time analytics using Kinesis and Spark Streaming
[https://github.com/awslabs/real-time-analytics-spark-streaming](https://github.com/awslabs/real-time-analytics-spark-streaming)

Serverless Reference Architecture for Real-time Stream Processing
[https://github.com/awslabs/lamba-refarch-streamprocessing](https://github.com/awslabs/lamba-refarch-streamprocessing)

Amazon Kinesis Data Generator

Machine Learning Samples
[https://github.com/awslabs/machine-learning-samples](https://github.com/awslabs/machine-learning-samples)

CFN Cluster for Deep Learning AMIs
[https://github.com/awslabs/deeplearning-cfn](https://github.com/awslabs/deeplearning-cfn)

Amazon Simple Beer Service
[https://github.com/awslabs/simplebeerservice](https://github.com/awslabs/simplebeerservice)