PUTTING DATA TO USE

Big Data and Analytics with AWS

Karan Desai
Solutions Architect
AWS
speaker:~ $ whoami
> Solutions Architect at AWS since 2016
> Previously Akamai
> Previously Ericsson
> MS EE Virginia Tech
> San Francisco Bay Area resident
> Likes the cloud, airplanes, photography
What to Expect from the Session

• Big data challenges
• Architectural principles
• Big data processing steps
• What technologies should you use?
  • Why?
  • How?
• Reference architecture
• Design patterns
What is Big Data?

Volume

Variety

Velocity

Veracity
Big Data Evolution

Batch processing

Stream processing

Artificial intelligence
Cloud Services Evolution

Virtual machines

Managed services

Serverless
Plethora of Tools

Amazon EMR  Amazon S3  Amazon DynamoDB  Amazon SQS
Amazon Redshift  Amazon Glacier  Amazon RDS  ElastiCache
Amazon Kinesis  Amazon QuickSight  AWS Glue  Amazon ES
Lambda  Amazon ML  Amazon DynamoDB Streams  Amazon Kinesis Analytics
Big Data Challenges

- Why?
- How?
- What tools should I use?
- Is there a reference architecture?
Architectural Principles
Architectural Principles

• Build **decoupled** systems

Data →
  → Store →
  → Process →
  → Analyze →
  → **Answers**
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 **managed** and **serverless** services
  • Scalable/elastic, available, reliable, secure, no/low admin

• Use **log-centric** design patterns
  • Immutable logs (data lake), materialized views

• Be **cost-conscious**
  • Big data ≠ big cost

• Enable your applications to use **AI/ML**
What Is the **Temperature** of Your Data?
## Characterize your data: 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>

**Heatmap:**
- Hot data
- Warm data
- Cold data
Big Data Processing Steps

Data -> COLLECT -> STORE -> PROCESS/ANALYZE -> CONSUME -> Answers

Time to answer (Latency)
Throughput
Cost
Choosing the Right Tools
Type of Data

Data structures
Database records
Media files
Log files
Data streams
Transactions
Files
Events

COLLECT

Devices
Sensors
IoT platforms
AWS IoT

Web apps
Mobile apps
Data centers
AWS Direct Connect

Logging
AWS CloudTrail
Amazon CloudWatch

Migration
Import/export
Snowball

Data Transport & Logging
Applications

Transactions
Files
Events

Amazon CloudWatch
AWS CloudTrail

Import/export
Snowball

AWS Direct Connect

Web apps
Mobile apps
Data centers
Devices
Sensors
IoT platforms

Files

Data Transport & Logging
Applications

RECORS

FILES

STREAMS
STORE
Type of Data

Data structures
Database records
Media files
Log files
Data streams

COLLECT

<table>
<thead>
<tr>
<th>Applications</th>
<th>Records</th>
<th>STREAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web apps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile apps</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data centers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Devices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IoT platforms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Transport &amp; Logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
</tr>
<tr>
<td>AWS CloudTrail</td>
</tr>
<tr>
<td>Amazon CloudWatch</td>
</tr>
<tr>
<td>Migration</td>
</tr>
<tr>
<td>Import/export</td>
</tr>
<tr>
<td>Snowball</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-memory</td>
</tr>
<tr>
<td>NoSQL</td>
</tr>
<tr>
<td>SQL</td>
</tr>
<tr>
<td>File/object store</td>
</tr>
<tr>
<td>Stream storage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type of Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORE</td>
</tr>
<tr>
<td>In-memory</td>
</tr>
<tr>
<td>NoSQL</td>
</tr>
<tr>
<td>SQL</td>
</tr>
<tr>
<td>File/object store</td>
</tr>
<tr>
<td>Stream storage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensors</td>
</tr>
<tr>
<td>IoT platforms</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web apps</td>
</tr>
<tr>
<td>Mobile apps</td>
</tr>
<tr>
<td>Data centers</td>
</tr>
<tr>
<td>Devices</td>
</tr>
<tr>
<td>Sensors</td>
</tr>
<tr>
<td>IoT platforms</td>
</tr>
</tbody>
</table>
Stream Storage

Apache Kafka
- High throughput distributed streaming platform

Amazon Kinesis Streams
- Managed stream storage

Amazon Kinesis Firehose
- Managed data delivery
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 Lambda functions)
### Which Stream/Message Storage Should I Use?

<table>
<thead>
<tr>
<th>Feature</th>
<th>Amazon Kinesis Streams</th>
<th>Amazon Kinesis Firehose</th>
<th>Apache Kafka (on Amazon EC2)</th>
<th>Amazon SQS (Standard)</th>
<th>Amazon SQS (FIFO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS managed</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Guaranteed ordering</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Delivery (deduping)</td>
<td>At least once</td>
<td>At least once</td>
<td>At least/At most/exactly once</td>
<td>At least once</td>
<td>Exactly once</td>
</tr>
<tr>
<td>Data retention period</td>
<td>7 days</td>
<td>N/A</td>
<td>Configurable</td>
<td>14 days</td>
<td>14 days</td>
</tr>
<tr>
<td>Availability</td>
<td>3 AZ</td>
<td>3 AZ</td>
<td>Configurable</td>
<td>3 AZ</td>
<td>3 AZ</td>
</tr>
<tr>
<td>Scale / throughput</td>
<td>No limit / ~ shards</td>
<td>No limit / automatic</td>
<td>No limit / ~ nodes</td>
<td>No limits / automatic</td>
<td>300 TPS / queue</td>
</tr>
<tr>
<td>Parallel consumption</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Stream MapReduce</td>
<td>Yes</td>
<td>N/A</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Row/object size</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>Cost</td>
<td>Low</td>
<td>Low</td>
<td>Low (+admin)</td>
<td>Low-medium</td>
<td>Low-medium</td>
</tr>
</tbody>
</table>
File/Object Storage

Amazon S3
Managed object storage service built to store and retrieve any amount of data
Use Amazon S3 as Your Persistent File Store

- Natively supported by big data frameworks (Spark, Hive, Presto, etc.)
- **Decouple** storage and compute
  - No need to run compute clusters for storage (unlike HDFS)
  - Can run transient Amazon EMR clusters with Amazon EC2 Spot Instances
  - Multiple & heterogeneous analysis clusters and services can use the same data
- Designed for **99.999999999% durability**
- No need to pay for data replication within a region
- **Secure**: SSL, client/server-side encryption at rest
- Low cost
What About HDFS & Data Tiering?

- Use **HDFS** for **hottest** datasets (e.g. iterative read on the same datasets)
- 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

Use **S3 Analytics** to optimize tiering strategy
**Cache & Database**

**Amazon ElastiCache**
- Managed Memcached or Redis service

**Amazon DynamoDB Accelerator (DAX)**
- Managed in-memory cache for DynamoDB

**Amazon DynamoDB**
- Managed NoSQL database service

**Amazon RDS**
- Managed relational database service
Anti-Pattern

FRIENDS DON'T LET REAL FRIENDS USE RELATIONAL DATABASES
Best Practice: Use the Right Tool for the Job

Applications

Database Tier

In-memory
- Amazon Elasticache
- SAP HANA

GraphDB
- Amazon Neptune
- JanusGraph
- neo4j

NoSQL
- Amazon DynamoDB
- Apache HBase
- cassandra
- MongoDB

SQL
- Amazon RDS/Aurora
- MySQL
- ORACLE DATABASE
- SQL Server 2017

Search
- Amazon ES
- Amazon CloudSearch
- elasticsearch
Which Data Store Should I Use?

Depends on-

• **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>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)</td>
<td>NoSQL, Search</td>
</tr>
<tr>
<td>Key/Value</td>
<td>In-memory, NoSQL</td>
</tr>
<tr>
<td>Graph</td>
<td>GraphDB</td>
</tr>
</tbody>
</table>

### 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>
<tr>
<td>Graph traversal</td>
<td>GraphDB</td>
</tr>
</tbody>
</table>
# Which Data Store Should I Use?

<table>
<thead>
<tr>
<th></th>
<th>Amazon ElastiCache</th>
<th>Amazon DAX</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>µs-ms</td>
<td>µs-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 Volume</strong></td>
<td>GB</td>
<td>GB</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 (400 KB max)</td>
<td>KB (64 KB max)</td>
<td>KB (2 GB max)</td>
<td>KB-TB (5 TB max)</td>
<td>GB (40 TB max)</td>
<td></td>
</tr>
<tr>
<td><strong>Request Rate</strong></td>
<td>High – very high</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>$c$</td>
<td>$c$</td>
<td>$c$</td>
<td>$c$</td>
<td>$c$4/10</td>
</tr>
<tr>
<td><strong>Durability</strong></td>
<td>Low - moderate</td>
<td>NA</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>High 3 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>
Cost-Conscious Design

Example:

• “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>

Should I use Amazon S3 or Amazon DynamoDB?
## Amazon S3 or Amazon 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>77,760,000</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.

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

### Scenario 1

<table>
<thead>
<tr>
<th></th>
<th>Amazon S3 Standard</th>
<th>Amazon DynamoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>$34</td>
<td>$273</td>
</tr>
<tr>
<td>Put/list requests</td>
<td>$3,888</td>
<td>$383</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$3,922</strong></td>
<td><strong>$656</strong></td>
</tr>
</tbody>
</table>

<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>

### Scenario 2

<table>
<thead>
<tr>
<th></th>
<th>Amazon S3 Standard</th>
<th>Amazon DynamoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>$545</td>
<td>$4,556</td>
</tr>
<tr>
<td>Put/List Requests</td>
<td>$3,888</td>
<td>$5,944</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>$4,433</strong></td>
<td><strong>$10,500</strong></td>
</tr>
</tbody>
</table>

<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>32,768</td>
<td>23,730</td>
<td>777,600,000</td>
</tr>
</tbody>
</table>
PROCESS / ANALYZE
Predictive Analytics

- **API-driven Services**
  - Amazon Lex – Deep-learning based chatbots
  - Amazon Polly – Text to speech
  - Amazon Rekognition – Image and video analysis
  - Amazon Comprehend – NLP, sentiment analysis

- **Managed ML Platforms**
  - Amazon Sagemaker
  - Amazon DeepLens
  - MXNet and TensorFlow on AWS

- **AWS Deep Learning AMI**
  - Pre-installed with MXNet, TensorFlow, Caffe2, Theano, Torch, Microsoft Cognitive Toolkit, and Keras; plus DL tools/drivers
Interactive and Batch Analytics

- Amazon Elasticsearch Service
  - Managed Service for Elasticsearch

- Amazon Redshift and Amazon Redshift Spectrum
  - Managed Data Warehouse
  - Spectrum enables querying Amazon S3

- Amazon Athena
  - Serverless Interactive Query Service

- Amazon EMR
  - Managed Hadoop Framework for running Apache Spark, Flink, Presto, Tez, Hive, Pig, HBase, etc.
Stream/Real-time Analytics

Spark Streaming on Amazon EMR

Amazon Kinesis Analytics
  • Managed Service for running SQL on Streaming data

Amazon Kinesis Client Library

AWS Lambda
  • Run code Serverless (without provisioning or managing servers)
  • Services such as S3 can publish events to Lambda
  • Lambda can pool event from a Kinesis
Which Analytics Should I Use?

<table>
<thead>
<tr>
<th>Type</th>
<th>Processing Time</th>
<th>Example:</th>
<th>Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch</td>
<td>minutes to hours</td>
<td>Daily/weekly/monthly reports</td>
<td>Amazon EMR (MapReduce, Hive, Pig, Spark)</td>
</tr>
<tr>
<td>Interactive</td>
<td>seconds</td>
<td>Self-service dashboards</td>
<td>Amazon Redshift, Amazon Athena, Amazon EMR (Presto, Spark)</td>
</tr>
<tr>
<td>Stream</td>
<td>milliseconds to seconds</td>
<td>Fraud alerts, 1 minute metrics</td>
<td>Amazon EMR (Spark Streaming), Amazon Kinesis Analytics, KCL, AWS Lambda</td>
</tr>
<tr>
<td>Predictive</td>
<td>milliseconds (real-time) to minutes (batch)</td>
<td>Fraud detection, Forecasting demand, Speech recognition</td>
<td>Amazon AI (Lex, Polly, Sagemaker, Amazon Rekognition), Amazon EMR (Spark ML), Deep Learning AMI (MXNet, TensorFlow, Theano, Torch, CNTK, Caffe)</td>
</tr>
<tr>
<td>Managed Service</td>
<td>Yes</td>
<td>No (EC2 + Auto Scaling)</td>
<td>Yes</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
<td>------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Serverless</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Scale/Throughput</td>
<td>No limits / ~ nodes</td>
<td>No limits / ~ nodes</td>
<td>No limits / automatic</td>
</tr>
<tr>
<td>Availability</td>
<td>Single AZ</td>
<td>Multi-AZ</td>
<td>Multi-AZ</td>
</tr>
<tr>
<td>Programming Languages</td>
<td>Java, Python, Scala</td>
<td>Java, others via MultiLangDaemon</td>
<td>ANSI SQL with extensions</td>
</tr>
<tr>
<td>Sliding Window Functions</td>
<td>Build-in</td>
<td>App needs to implement</td>
<td>Built-in</td>
</tr>
<tr>
<td>Reliability</td>
<td>KCL and Spark checkpoints</td>
<td>Managed by KCL</td>
<td>Managed by Amazon Kinesis Analytics</td>
</tr>
</tbody>
</table>

Fast
<table>
<thead>
<tr>
<th>Use case</th>
<th>Amazon Redshift</th>
<th>Amazon Redshift Spectrum</th>
<th>Amazon Athena</th>
<th>Amazon EMR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Use case</strong></td>
<td>Optimized for data warehousing</td>
<td>Query S3 data from Amazon Redshift</td>
<td>Interactive Queries over S3 data</td>
<td>Amazon EMR</td>
</tr>
<tr>
<td><strong>Scale/Throughput</strong></td>
<td>~Nodes</td>
<td>~Nodes</td>
<td>Automatic</td>
<td>~Nodes</td>
</tr>
<tr>
<td><strong>Managed Service</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, Serverless</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Storage</strong></td>
<td>Local storage</td>
<td>Amazon S3</td>
<td>Amazon S3</td>
<td>Amazon S3, HDFS</td>
</tr>
<tr>
<td><strong>Optimization</strong></td>
<td>Columnar storage, data compression, and zone maps</td>
<td>AVRO, PARQUET TEXT, SEQ RCFILE, ORC, etc.</td>
<td>AVRO, PARQUET TEXT, SEQ RCFILE, ORC, etc.</td>
<td>Framework dependent</td>
</tr>
<tr>
<td><strong>Metadata</strong></td>
<td>Amazon Redshift Catalog</td>
<td>Glue Catalog</td>
<td>Glue Catalog</td>
<td>Glue Catalog or Hive Meta-store</td>
</tr>
<tr>
<td><strong>Auth/Access controls</strong></td>
<td>IAM, Users, groups, and access controls</td>
<td>IAM, Users, groups, and access controls</td>
<td>IAM</td>
<td>IAM, LDAP &amp; Kerberos</td>
</tr>
<tr>
<td><strong>UDF support</strong></td>
<td>Yes (Scalar)</td>
<td>Yes (Scalar)</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Amazon EMR</strong></th>
<th>Presto</th>
<th>Spark</th>
<th>Hive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Presto</strong></td>
<td>Interactive Query</td>
<td>General purpose</td>
<td>Batch</td>
</tr>
<tr>
<td><strong>Spark</strong></td>
<td>Interactive Query</td>
<td>General purpose</td>
<td>Batch</td>
</tr>
<tr>
<td><strong>Hive</strong></td>
<td>Interactive Query</td>
<td>General purpose</td>
<td>Batch</td>
</tr>
</tbody>
</table>

**Metadata**

- **Amazon Redshift Catalog**
- **Glue Catalog**
- **Glue Catalog or Hive Meta-store**

**Auth/Access controls**

- **IAM, Users, groups, and access controls**
- **IAM**
- **IAM, LDAP & Kerberos**

**UDF support**

- **Yes (Scalar)**
- **No**
- **Yes**

**Optimization**

- **Columnar storage, data compression, and zone maps**
- **AVRO, PARQUET TEXT, SEQ RCFILE, ORC, etc.**
- **Framework dependent**

**Scale/Throughput**

- **~Nodes**
- **Automatic**
- **~Nodes**

**Managed Service**

- **Yes**
- **Yes**
- **Yes**

**Storage**

- **Local storage**
- **Amazon S3**
- **Amazon S3**

**Amazon EMR**

- **Presto**
- **Spark**
- **Hive**

**Use case**

- **Optimized for data warehousing**
- **Query S3 data from Amazon Redshift**
- **Interactive Queries over S3 data**

**Fastest**

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

• Fully managed (serverless) ETL service
• Simple and cost-effective to categorize your data, clean it, enrich it, and move it reliably between various data stores.
CONSUME
• BI/Al Applications
  • Amazon EC2 or ECS/EKS Containers
  • AWS Greengrass

• Data Science
  • Notebooks
  • DS Platforms
  • IDEs

• Analysis and Visualization
  • Amazon QuickSight
  • Tableau
  • ....
Putting It All Together
Reference Architectures
Real-time Analytics

Stream

Amazon Kinesis

AWS Lambda

Amazon EMR

Spark Streaming

KCL app

App state or Materialized View

Log

Real-time prediction

Amazon AI

Fan out

Alerts

Amazon Kinesis

Amazon SNS

Notifications

Downstream

Amazon ElastiCache (Redis)

Amazon DynamoDB

Amazon RDS

Amazon ES

Amazon S3

KPI

Process

Store
Interactive and Batch Analytics

- Amazon Kinesis Firehose
- Amazon S3
- Amazon EMR
- Amazon Redshift
- Amazon Athena
- Amazon ES
- Amazon AI
- Amazon Kinesis Analytics
- Amazon Kinesis Firehose
- Presto
- Spark
- Batch prediction
- Stream
- Files
- Real-time prediction
- Batch
- Consumed
- Process
- Store
What about Metadata?

• **Glue Catalog**
  - Hive Metastore compliant
  - Crawlers - Detect new data, schema, partitions
  - Search - Metadata discovery
  - Amazon Athena, Amazon EMR, and Amazon Redshift Spectrum compatible

• **Hive Metastore** *(Presto, Spark, Hive, Pig)*
  - Can be hosted on Amazon RDS
Security & Governance

- AWS Identity and Access Management (IAM)
- Amazon Cognito
- Amazon CloudWatch & AWS CloudTrail
- AWS KMS and Secrets Manager
- AWS Directory Service
- Apache Ranger
The Challenge

Equipment sensor data from 900 GE Power sites worldwide 500,000 data records per second to collect and process Low-latency automatic alerts when equipment issues detected

Solution

Amazon Kinesis Data Streams ingest 20 billion records per day
Amazon EMR processes data in near-realtime
Analytical data stored on Amazon S3 for archiving

Impact

Better, faster visibility into their power plant operations

“Promptly identified a problem with a customer’s power plant combustion system early enough to schedule equipment shutdown and repaired the problem at a fraction of what a catastrophic failure would have cost. They saved millions of dollars as a result.”
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 and serverless** services
  • Scalable/elastic, available, reliable, secure, no/low admin

• Use **log-centric** design patterns
  • Immutable logs, data lake

• Be **cost-conscious**
  • Big data ≠ Big cost

• Enable your applications to use **AI/ML**
Thank You!

PUTTING DATA TO USE
Big Data and Analytics with AWS

Karan Desai
Solutions Architect
AWS

Email: deskaran@amazon.com
Twitter: @somecloudguy