Build a Real-Time Streaming Data Visualization System with Amazon Kinesis Analytics

Allan MacInnis
Solutions Architect, AWS

What to Expect from the Session

- Streaming data overview
- Amazon Kinesis platform review
- Amazon Kinesis Analytics Overview
- Amazon Kinesis Analytics patterns
- Streaming data end-to-end example and walk-through
- Amazon Kinesis Analytics Best Practices

Most data is produced continuously

Mobile Apps

Web Clickstream

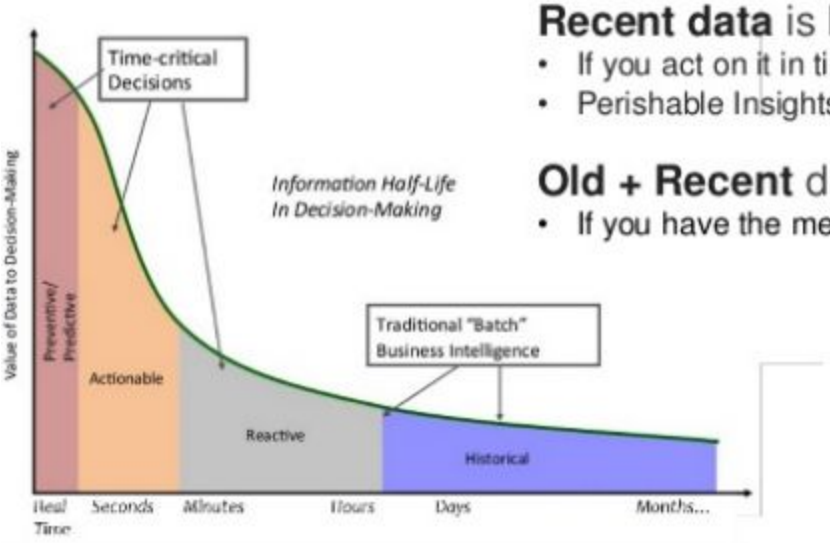
Application Logs

Metering Records

IoT Sensors

Smart Buildings

The diminishing value of data



Recent data is highly valuable

- If you act on it in time
- Perishable Insights (M. Gualtieri, Forrester)

Old + Recent data is more valuable

- If you have the means to combine them

Processing real-time, streaming data

What are the key requirements?

- Durable
- Continuous
- Fast
- Correct
- Reactive
- Reliable

Amazon Kinesis makes it easy to work with real-time streaming data

Amazon Kinesis Streams

- For Technical Developers
- Collect and stream data for ordered, replayable, real-time processing

Amazon Kinesis Firehose

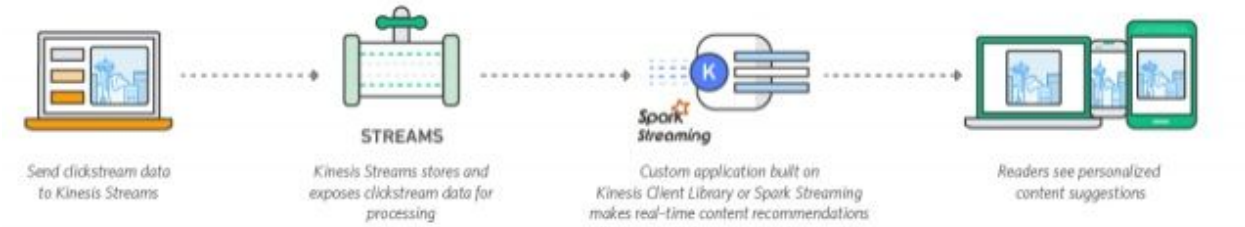
- For all developers, data scientists
- Easily load massive volumes of streaming data into Amazon S3, Redshift, ElasticSearch

Amazon Kinesis Analytics

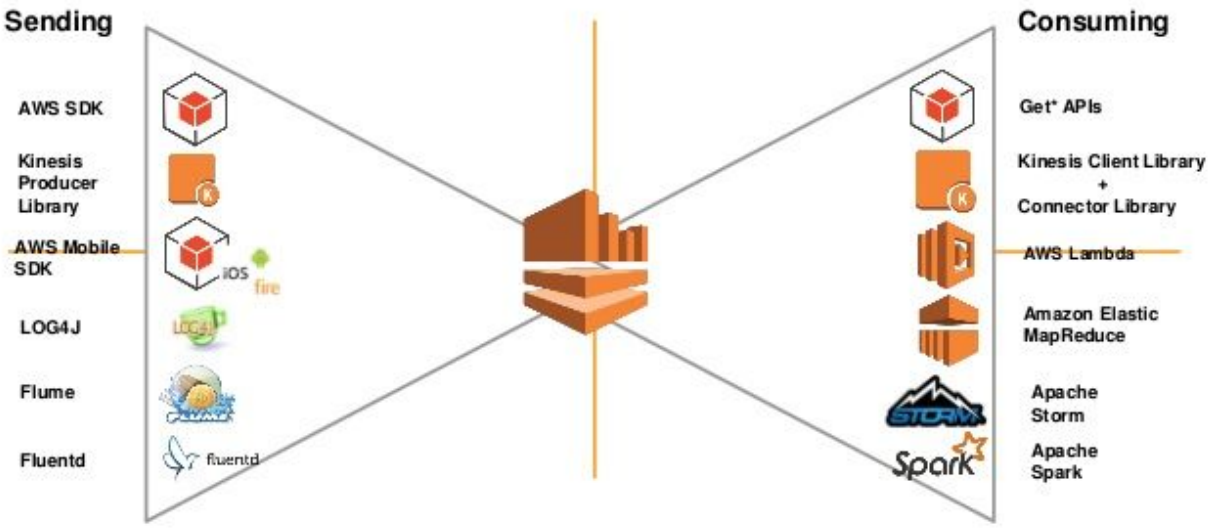
- For all developers, data scientists
- Easily analyze data streams using standard SQL queries

Amazon Kinesis Streams

- Reliably ingest and durably store streaming data at low cost
- Build custom real-time applications to process streaming data



Sending & Reading Data from Kinesis Streams



Amazon Kinesis Firehose

- Reliably ingest and deliver batched, compressed, and encrypted data to S3, Redshift, and Elasticsearch
- Point and click setup with zero administration and seamless elasticity

Amazon Kinesis Analytics

- Interact with streaming data in real-time using SQL
- Build fully managed and elastic stream processing applications that process data for real-time visualizations and alarms

Amazon Kinesis Analytics: Service Overview

Kinesis Analytics

Easy to use

Automatic elasticity

Real-time processing

Pay for only what you use

Standard SQL for analytics

Use SQL to build real-time applications

Connect to streaming source

Easily write SQL code to process streaming data

Continuously deliver SQL results

Connect to streaming source

- Streaming data sources include Kinesis Firehose or Kinesis Streams
- Input formats include JSON, .csv, variable column, unstructured text
- Each input has a schema; schema is inferred, but you can edit
- Reference data sources (S3) for data enrichment

Write SQL code

- Build streaming applications with one-to-many SQL statements
- Robust SQL support and advanced analytic functions
- Extensions to the SQL standard to work seamlessly with streaming data
- Support for at-least-once processing semantics

Continuously deliver SQL results

- Send processed data to multiple destinations
  - S3, Amazon Redshift, Amazon ES (through Firehose)
  - Streams (with AWS Lambda integration for custom destinations)
- End-to-end processing speed as low as sub-second
- Separation of processing and data delivery

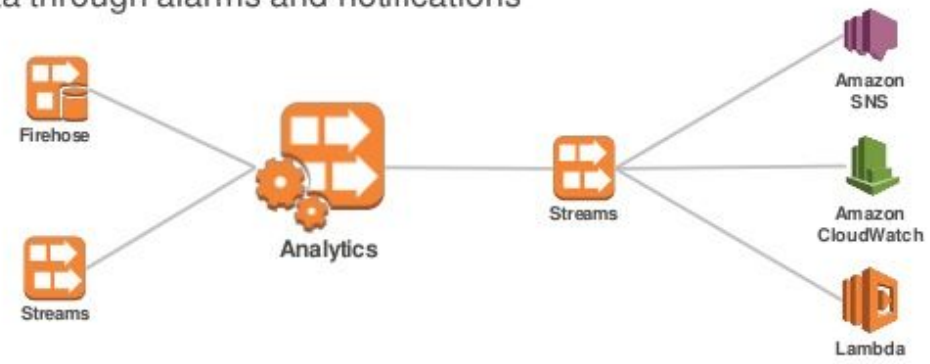
What are common uses for Kinesis Analytics?

Generate time series analytics

- Compute key performance indicators over time periods
- Combine with static or historical data in S3 or Amazon Redshift

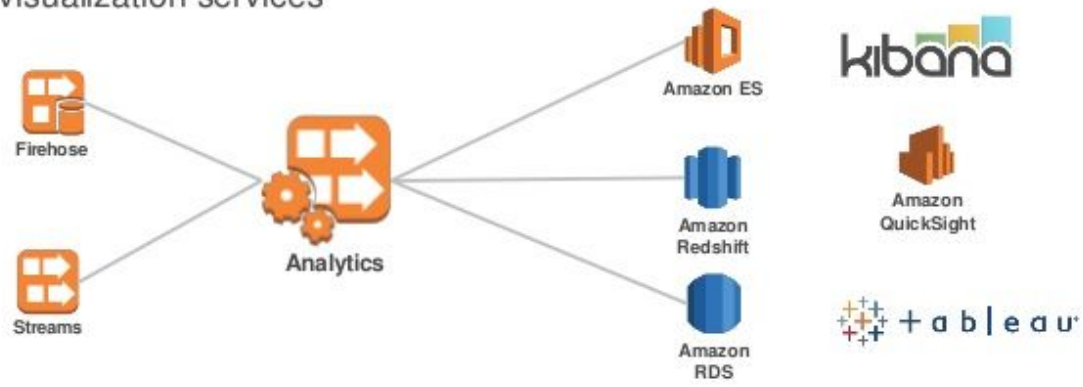
Create real-time alarms and notifications

- Build sequences of events from the stream, like user sessions in a clickstream or app behavior through logs
- Identify events (or a series of events) of interest, and react to the data through alarms and notifications



Feed real-time dashboards

- Validate and transform raw data, and then process to calculate meaningful statistics
- Send processed data downstream for visualization in BI and visualization services



Example: Real-time Dashboard

Example Scenario Requirements

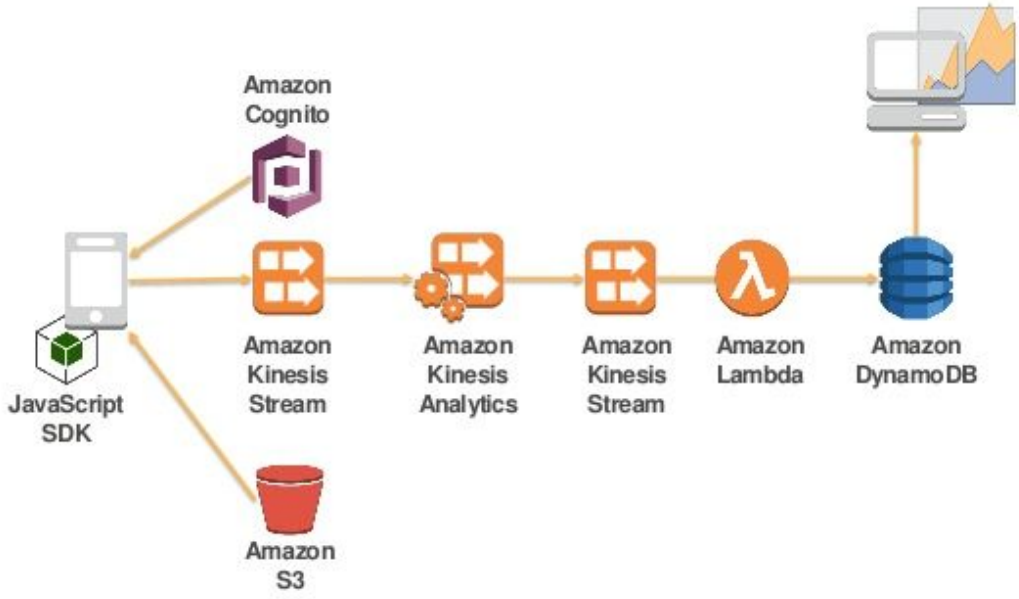
Data to capture every second:

- Total distinct users
- Number of users for each Operating System
- Number of users in each quadrant

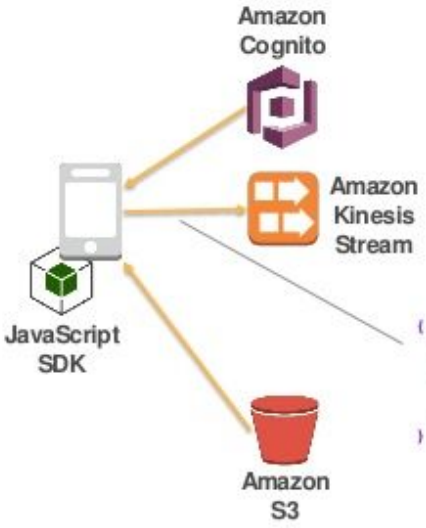
Output Requirements

- Update DynamoDB table every second, with each aggregate values

End-to-End Architecture



Data Input



Source JSON Data

- Once per second, using JavaScript SDK:
  - Unique Cognito ID (anonymous user)
  - OS
  - Quadrant
  - Data sent to Kinesis Stream


```
{
"recordTime": 1486505943.204,
"cognitoId": "us-east-1:3626e211-d2a3-447b-8231-e1f4e0486f44",
"os": "Android",
"quadrant": "A"
}
```

How is raw data mapped to a schema?


```
{
  "recordTime": 1486505943.204,
  "cognitoId": "us-east-1:<guid>",
  "os": "Android",
  "quadrant": "A"
}
```

cognitoID	os	quadrant
<guid1>	Android	A
<guid2>	iOS	B

Source Data for Kinesis Analytics

How is streaming data accessed with SQL?

STREAM

- Analogous to a TABLE
- Represents continuous data flow

```
CREATE OR REPLACE STREAM DISTINCT_USER_STREAM(
    COGNITO_ID VARCHAR(64),
    DEVICE VARCHAR(32),
    OS VARCHAR(32),
    QUADRANT char(1),
    DT TIMESTAMP);
```

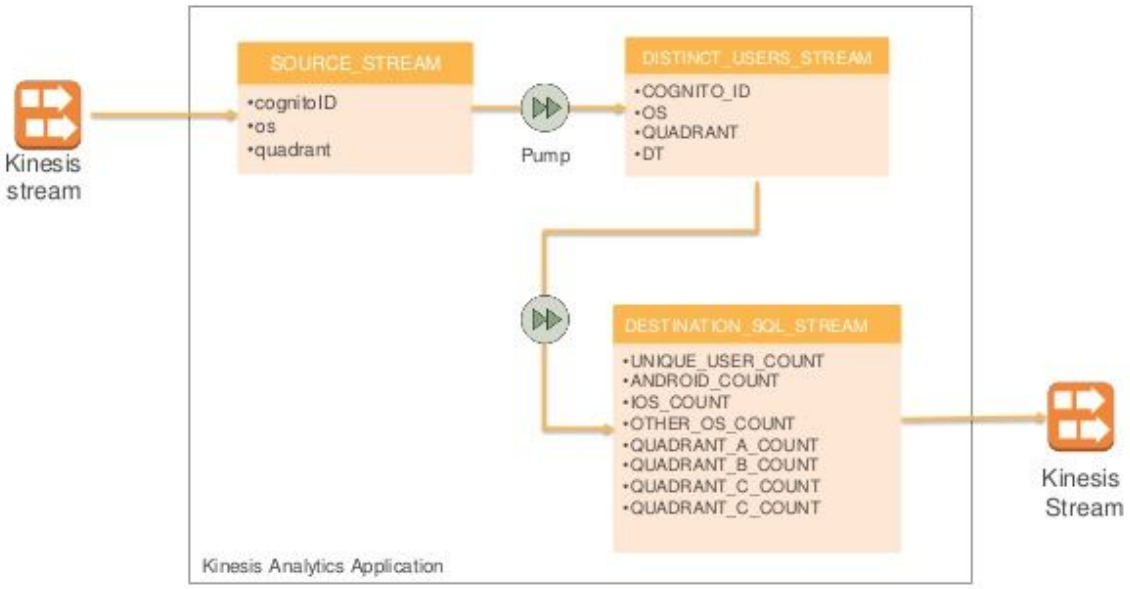
How is streaming data accessed with SQL?

PUMP

- Continuous INSERT query
- Inserts data from one in-application stream to another

```
CREATE OR REPLACE PUMP "DISTINCT_USER_PUMP" AS
    INSERT INTO "DISTINCT_USER_STREAM"
        SELECT STREAM DISTINCT
            "cognitoId",
              ...
```

How do we model our data?



How do we get distinct user records?

Use PUMP to insert distinct records into in-app STREAM

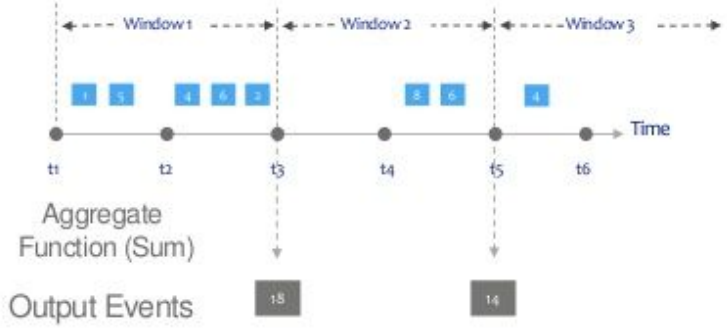
```
CREATE OR REPLACE PUMP "DISTINCT_USER_PUMP" AS
    INSERT INTO "DISTINCT_USER_STREAM"
        SELECT STREAM DISTINCT
            "cognitoId",
            "device",
            "os",
            "quadrant",
            FLOOR(s.ROWTIME TO SECOND)
        FROM "SOURCE_SQL_STREAM_001" s;
```

How do we aggregate streaming data?

- A common requirement in streaming analytics is to perform set-based operation(s) (**`count`**, **`average`**, **`max`**, **`min`**,..) over events that arrive within a specified period of time
- Cannot simply aggregate over an entire table like typical static database
- How do we define a subset in a potentially infinite stream?
- Windowing functions!

Windowing Concepts

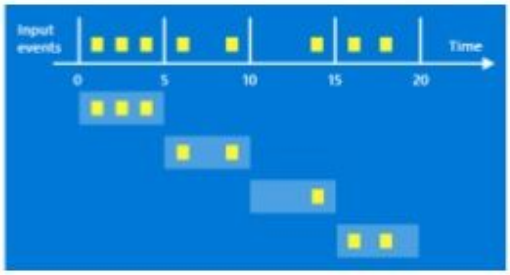
- Windows can be **tumbling** or **sliding**
- Windows are fixed length



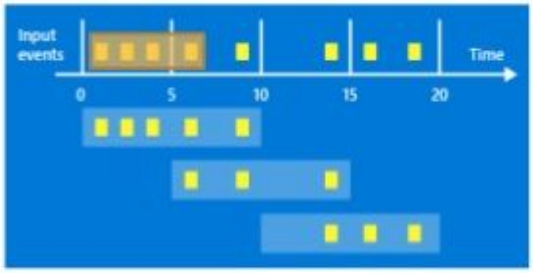
Output record will have the timestamp of the end of the window

Comparing Types of Windows

- Output created at the end of the window
- The output of the window will be single event based on the aggregate function used



Tumbling window
Aggregate per time interval



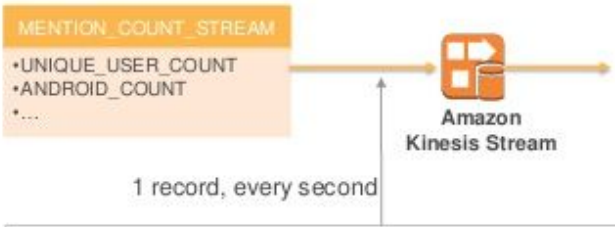
Sliding window
Windows constantly re-evaluated

How do we aggregate per second?

- Tumbling window, group by time period

```
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
        SELECT STREAM
            COUNT(dus.COGNITO_ID) AS UNIQUE_USER_COUNT,
            COUNT((CASE WHEN dus.OS = 'Android' THEN COGNITO_ID ELSE null END)) AS ANDROID_COUNT,
            COUNT((CASE WHEN dus.OS = 'iOS' THEN COGNITO_ID ELSE null END)) AS IOS_COUNT,
            COUNT((CASE WHEN dus.OS = 'Windows Phone' THEN COGNITO_ID ELSE null END)) AS WINDOWS_PHONE_COUNT,
            COUNT((CASE WHEN dus.OS = 'other' THEN COGNITO_ID ELSE null END)) AS OTHER_OS_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'A' THEN COGNITO_ID ELSE null END)) AS QUADRANT_A_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'B' THEN COGNITO_ID ELSE null END)) AS QUADRANT_B_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'C' THEN COGNITO_ID ELSE null END)) AS QUADRANT_C_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'D' THEN COGNITO_ID ELSE null END)) AS QUADRANT_D_COUNT,
            ROWTIME
        FROM "DISTINCT_USER_STREAM" dus
        GROUP BY
            FLOOR(dus.ROWTIME TO SECOND);
```

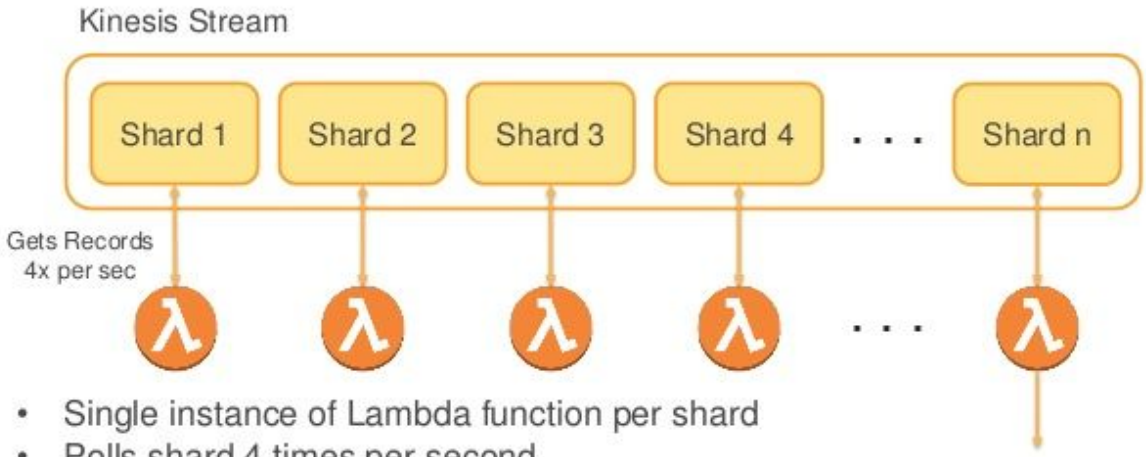
Output to Kinesis Stream



1 record, every second

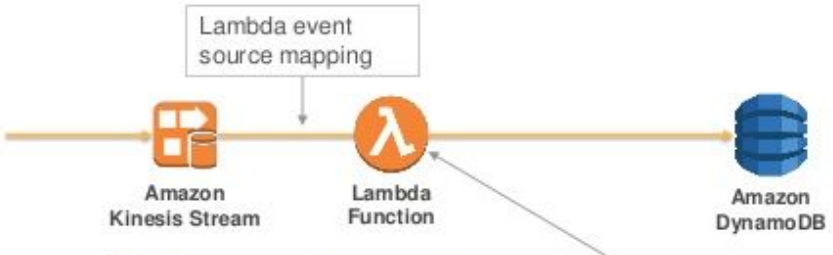
```
{
    "unique_user_count": 96,
    "android_count": 50,
    "ios_count": 46,
    "android_count": 50,
    "quadrant_a_count": 80,
    "quadrant_b_count ": 10,
    "quadrant_c_count ": 3,
    "quadrant_d_count ": 3
}
```

Processing a Kinesis Streams with AWS Lambda



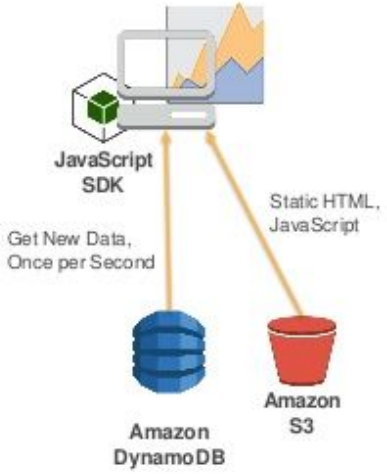
- Single instance of Lambda function per shard
- Polls shard 4 times per second
- Lambda function instances created and removed automatically as stream is scaled

Persist aggregated data in DynamoDB




```
event.Records.forEach((record) => {
    const payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
    var docClient = new AWS.DynamoDB.DocumentClient();
    var table = "user-quadrant-data";
    var data = JSON.parse(payload);
    var params = {
        TableName: table,
        Item:{
            "dataType": "quadrantRollup",
            "windowtime": (new Date(data.WINDOW_TIME)).getTime(),
            "userCount": data.UNIQUE_USER_COUNT,
            "quadrantA": data.QUADRANT_A_COUNT,
            "quadrantB": data.QUADRANT_B_COUNT, ...
        }
    };
    docClient.put(params, function(err, data) {  ...
```

Render Dashboard



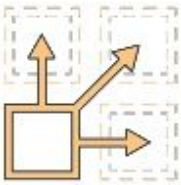
Kinesis Analytics Best Practices

Managing Applications

Set up Cloudwatch Alarms

- `MillisBehindLatest` metric tracks how far behind the application is from the source
- Alarm on `MillisBehindLatest` metric. Consider triggering when 1-hour behind, on a 1-minute average. Adjust accordingly for applications with lower end-to-end processing needs.

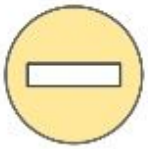
Managing Applications



Increase input parallelism to improve performance

- By default, a single source in-application stream is created
- If application is not keeping up with input stream, consider increasing input parallelism to create multiple source in-application streams

Managing Applications



Limit number of applications reading from same source

- Avoid `ReadProvisionedThroughputExceeded` exceptions
- For an Amazon Kinesis Streams source, limit to 2 total applications
- For an Amazon Kinesis Firehose source, limit to 1 application

Defining Input Schema

- Review and adequately test inferred input schema
- Manually update schema to handle nested JSON with greater than 2 levels of depth
- Use SQL functions in your application for unstructured data

Authoring Application Code

- Avoid time-based windows greater than one hour
- Keep window sizes small during development
- Use smaller SQL queries, with multiple in-application streams, rather than a single, large query

Limits

- Maximum row size in an in-application stream is 50 KB
- Maximum input parallelism is 10 in-application streams.
- Each application supports one streaming source, and one reference data source. The reference data source can be no larger than 1 GB in size.

Pricing

- Pay only for what you use.
- Charged an hourly rate, based on the average number of Kinesis Processing Units (KPU) used to run your application.
- A single KPU provides one vCPU, and 4 GB of memory.
- $0.11 per KPU-hour (US East).

Deep Dive – Log Analytics with Elasticsearch Service

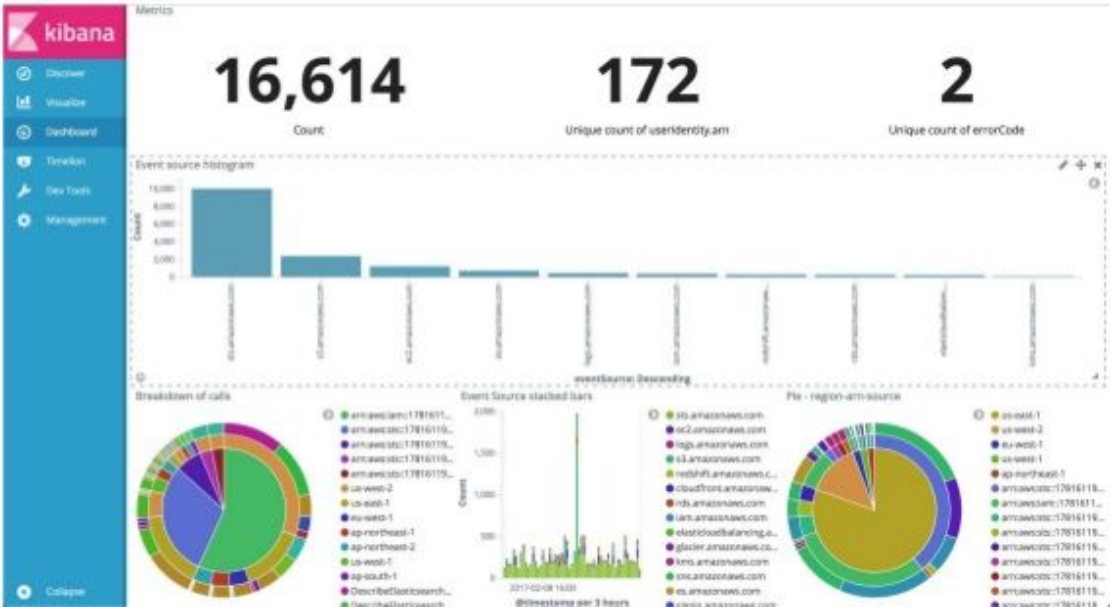
Jon Handler, Prinicipal Solutions Architect

April 5, 2017

What to do with a terabyte of logs?

```
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0" 200 4179
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/countdown.html HTTP/1.0" 200 3985
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
129.94.144.152 - - [01/Jul/1995:00:00:13 -0400] "GET / HTTP/1.0" 200 7074
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200 1204
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200 1204
129.94.144.152 - - [01/Jul/1995:00:00:17 -0400] "GET /images/ksclogo-medium.gif HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:17 -0400] "GET /images/launch-logo.gif HTTP/1.0" 200 1713
ppptky391.asahi-net.or.jp - - [01/Jul/1995:00:00:18 -0400] "GET /facts/about_ksc.html HTTP/1.0" 200 3977
net-1-141.eden.com - - [01/Jul/1995:00:00:19 -0400] "GET /shuttle/missions/sts-71/images/KSC-95EC-0916.jpg HTTP/1.0" 200 34029
```

Visualize it with Kibana!



Amazon Elasticsearch Service

Amazon Elasticsearch Service is a cost-effective managed service that makes it easy to deploy, manage, and scale open source Elasticsearch for log analytics, full-text search and more.

elasticsearch

kibana

Log analytics architecture

Amazon Elasticsearch Service Benefits

Easy to Use

Deploy a production-ready Elasticsearch cluster in minutes

Simplifies time-consuming management tasks such as software patching, failure recovery, backups, and monitoring

Open

Get direct access to the Elasticsearch open-source API

Fully compatible with the open source Elasticsearch API, for all code and applications

Secure

Secure Elasticsearch clusters with AWS Identity and Access Management (IAM) policies with fine-grained access control access for users and endpoints

Automatically applies security patches without disruption, keeping Elasticsearch environments secure

Available

Provides high availability using Zone Awareness, which replicates data between two Availability Zones

Monitors the health of clusters and automatically replaces failed nodes, without service disruption

AWS Integrated

Integrates with Amazon Kinesis Firehose, AWS IOT, and Amazon CloudWatch Logs for seamless data ingestion

AWS CloudTrail for auditing, AWS Identity and Access Management (IAM) for security, and AWS CloudFormation for cloud orchestration

Scalable

Scale clusters from a single node up to 20 nodes

Configure clusters to meet performance requirements by selecting from a range of instance types and storage options including SSD-powered EBS volumes

Amazon Elasticsearch Service Leading Use Cases

Log Analytics & Operational Monitoring

- Monitor the performance of applications, web servers, and hardware
- Easy to use, powerful data visualization tools to detect issues quickly
- Dig into logs in an intuitive, fine-grained way
- Kibana provides fast, easy visualization

Search

- Application or website provides search capabilities over diverse documents
- Tasked with making this knowledge base searchable and accessible
- Text matching, faceting, filtering, fuzzy search, auto complete, highlighting, and other search features
- Query API to support application search

Leading enterprises trust Amazon Elasticsearch Service for their search and analytics applications

Media & Entertainment

mlbam

FT FINANCIAL TIMES

COMCAST

SCOPELY

Online Services

Pinterest

lyft

amazon alexa

Zillow

Expedia

Technology

AUTODESK

Adobe

canvas

CITRIX

Mc Graw Hill Education

invision

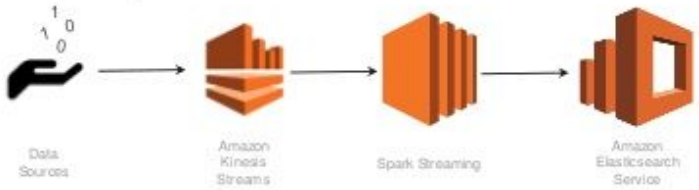
Other

British Gas

NORDSTROM

Bristol-Myers Squibb

Adobe Developer Platform (Adobe I/O)



PROBLEM

- Cost effective monitor for XL amount of log data
- Over 200,000 API calls per second at peak - destinations, response times, bandwidth
- Integrate seamlessly with other components of AWS eco-system

SOLUTION

- Log data is routed with Amazon Kinesis to Amazon Elasticsearch Service, then displayed using AES Kibana
- Adobe team can easily see traffic patterns and error rates, quickly identifying anomalies and potential challenges

BENEFITS

- Management and operational simplicity
- Flexibility to try out different cluster config during dev and test

McGraw Hill Education

PROBLEM

- Supporting a wide catalog across multiple services in multiple jurisdictions
- Over 100 million learning events each month
- Tests, quizzes, learning modules begun / completed / abandoned

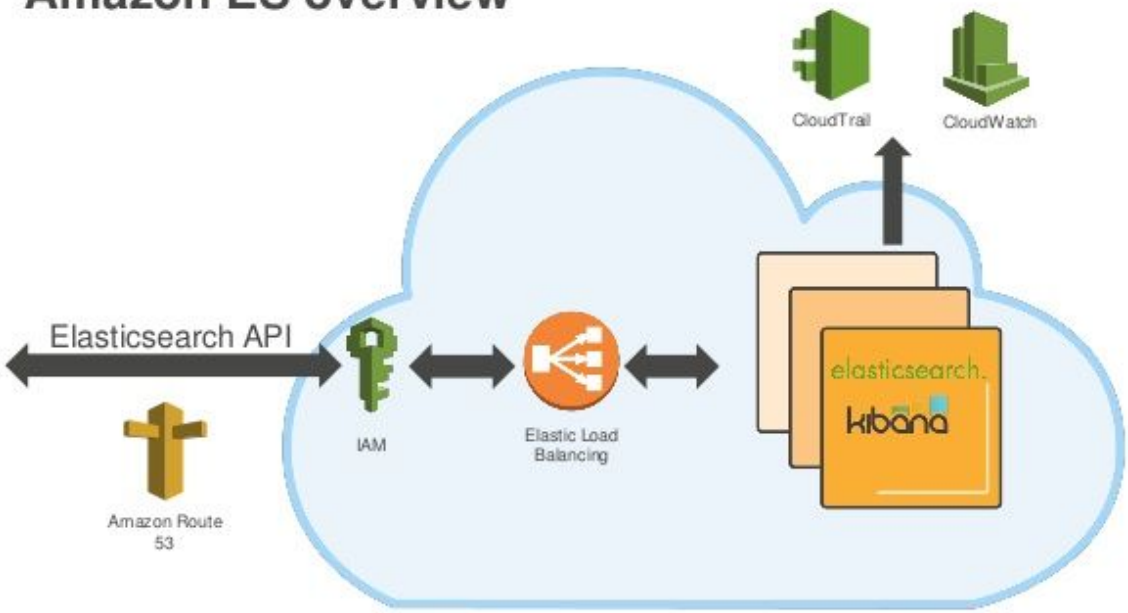
SOLUTION

- Search and analyze test results, student/teacher interaction, teacher effectiveness, student progress
- Analytics of applications and infrastructure are now integrated to understand operations in real time

BENEFITS

- Confidence to scale throughout the school year. From 0 to 32TB in 9 months
- Focus on their business, not their infrastructure

Amazon ES overview



Create Elasticsearch domain

Step 1: Define domain
Step 2: Configure cluster
Step 3: Set up access policy
Step 4: Review

Configure cluster

Configure the instance and storage settings for your cluster based on the traffic, data, and availability requirements of your application. A cluster is a collection of one or more data nodes, optional dedicated master nodes, and storage required to run Elasticsearch and operate your domain.

Node configuration

Selecting the correct instance type and instance count depends on the compute, memory, and storage needs of your application. Take into account the size of the Elasticsearch indices, shards, and replicas you intend to create, the types of queries you will run, and the amount of storage you will need as you decide these settings. If you have a large volume of data to upload or anticipate a large amount of query traffic to your domain, you can preconfigure the cluster to handle this requirement.

Instance count 1

Instance type m3.medium.elasticsearch (def...

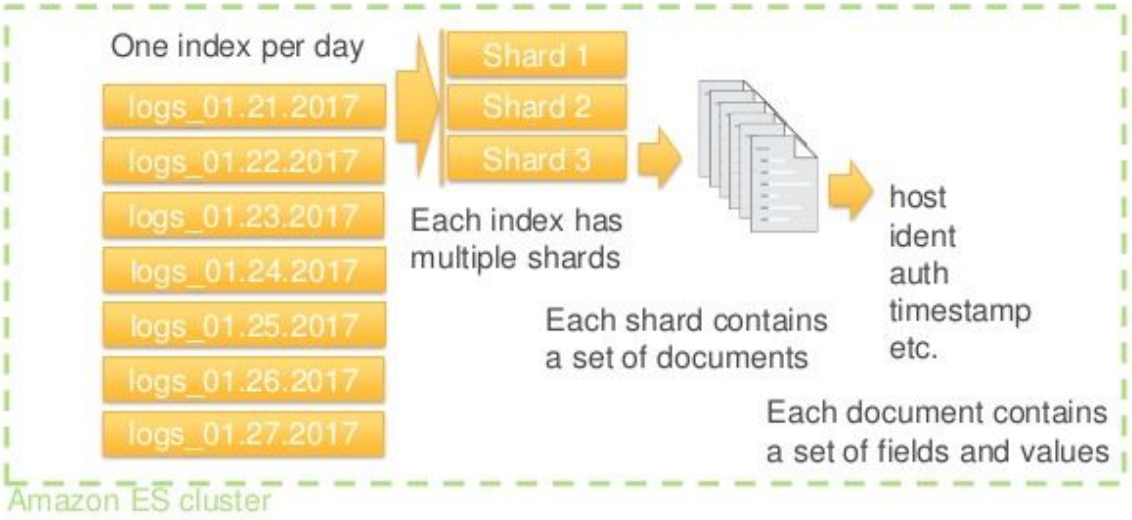
Enable dedicated master

Enable zone awareness

Storage configuration

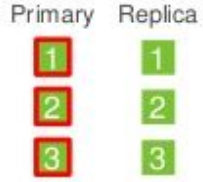
Choose a storage type for your data nodes. If you choose the EBS storage type, you will need to specify the EBS volume type and EBS volume size for the cluster. The EBS volume size setting is configured per instance. Multiply the volume size by the number of data nodes in your cluster for the total storage size available in your cluster. Take into account size of indices, shards, and replicas you intend to create in your cluster when configuring storage settings. Storage settings do not apply to any dedicated master nodes in the cluster.

Data pattern

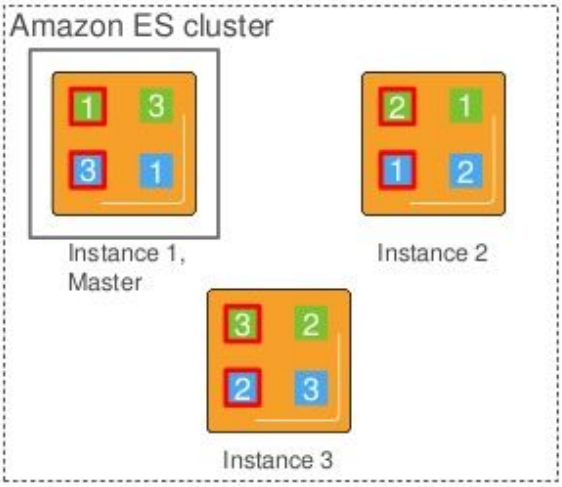


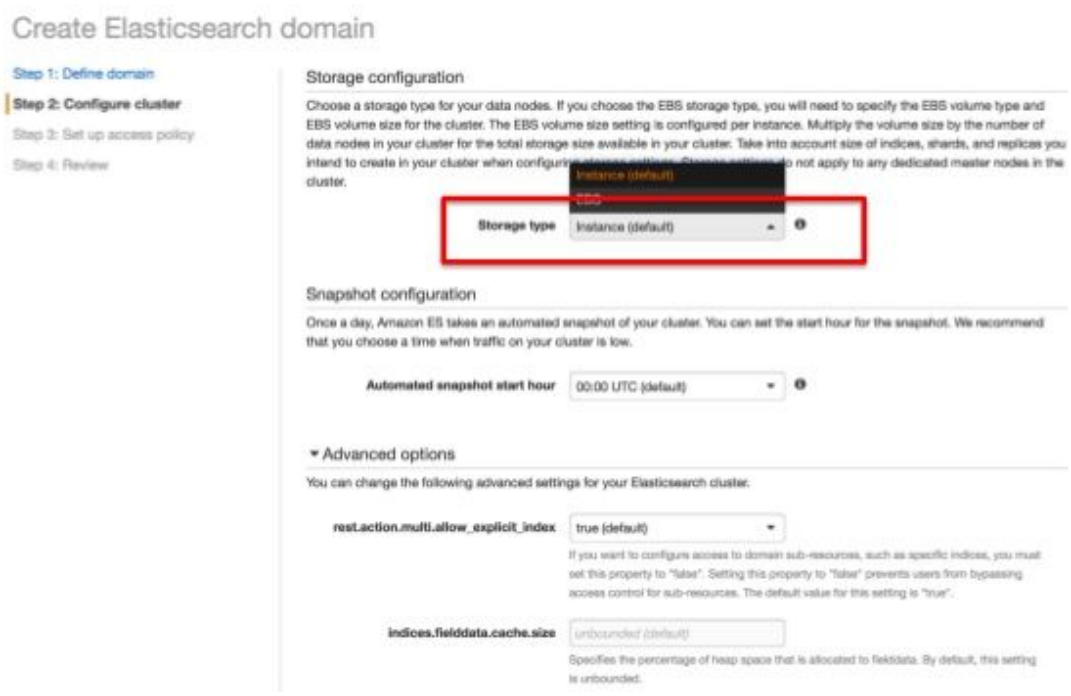
Deployment of indices to a cluster

- Index 1
  - Shard 1
  - Shard 2
  - Shard 3



- Index 2
  - Shard 1
  - Shard 2
  - Shard 3





How many instances?

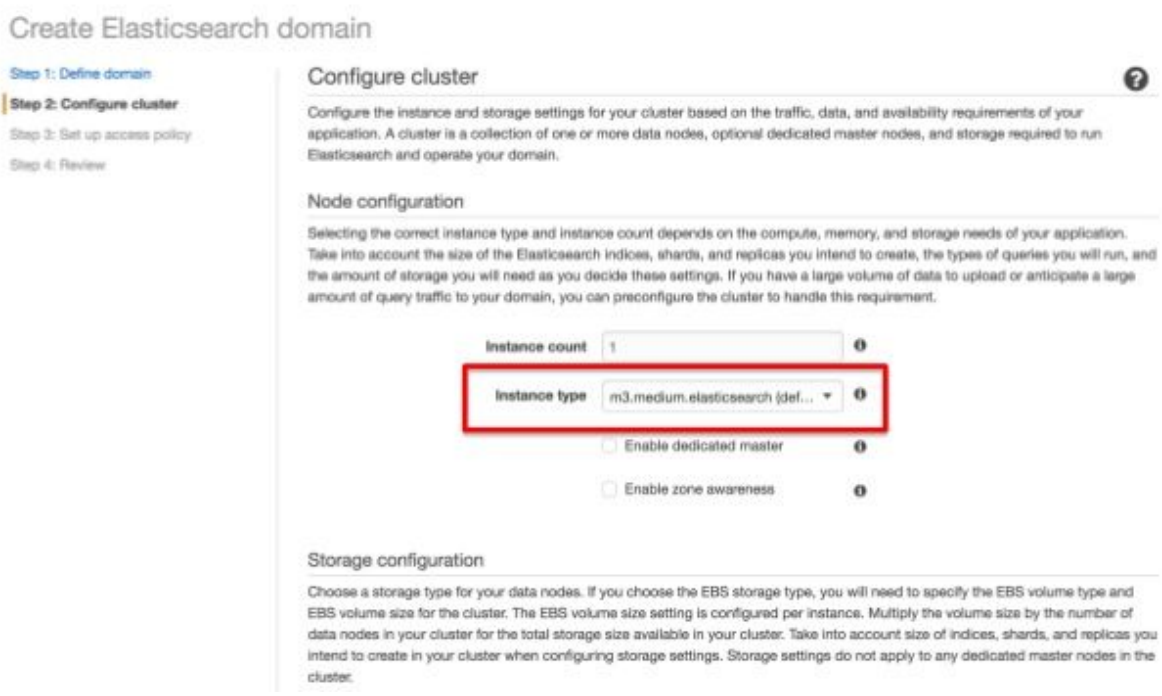
The index size will be about the same as the corpus of source documents

- **Double this** if you are deploying an index replica

Size based on storage requirements

- Either local storage or up to 1.5 TB of Amazon Elastic Block Store (EBS) per instance

- Example: 2 TB corpus will need 4 instances
  - Assuming a replica and using EBS
  - With i2.2xlarge nodes using 1.6 TB ephemeral storage



Determining instance type

Instance type is workload-dependent

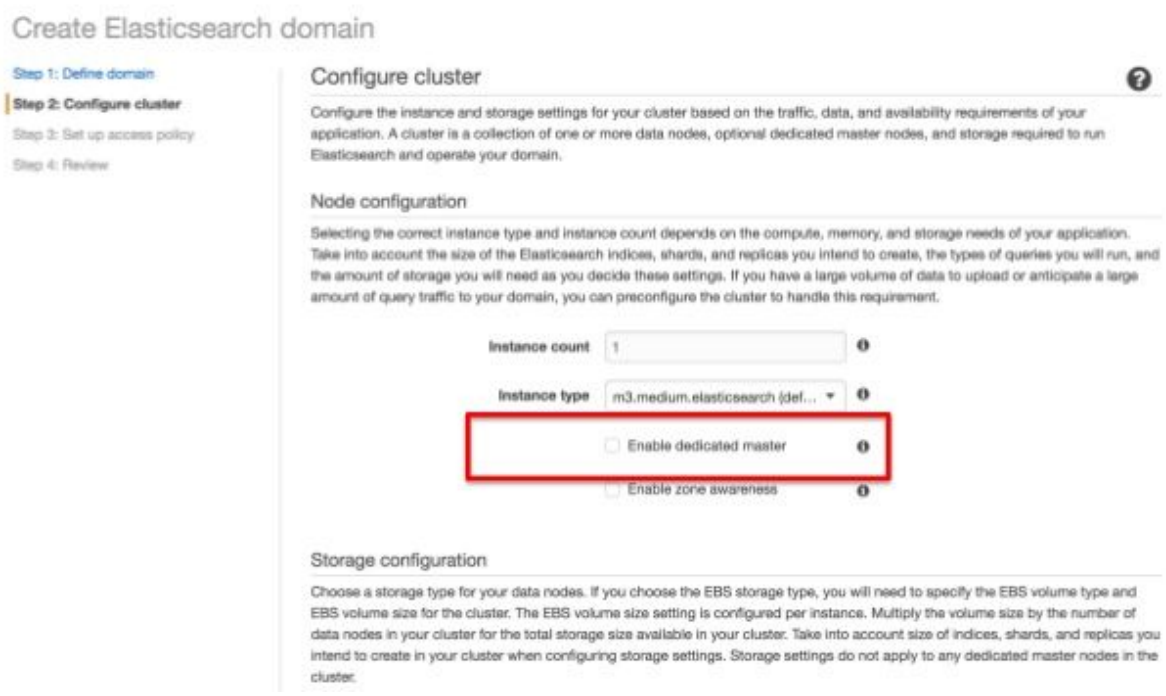
T2: dev, test, QA

M3/M4: solid performance

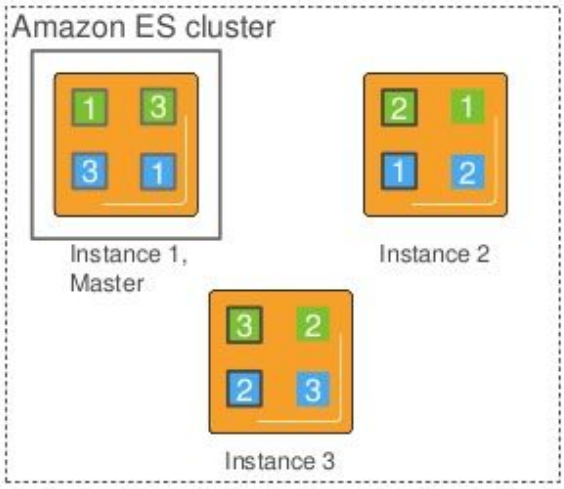
R3/R4: heavier queries, aggregations

C4: High throughput query loads

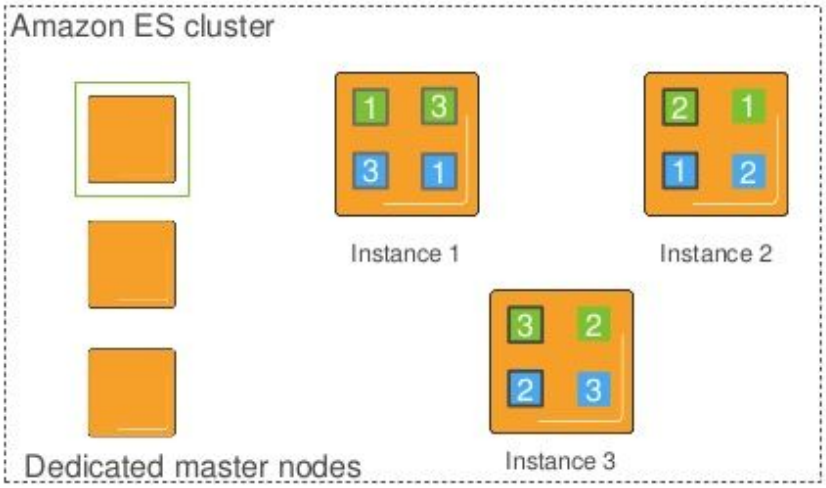
I2: largest storage option



Cluster with no dedicated masters



Cluster with dedicated masters

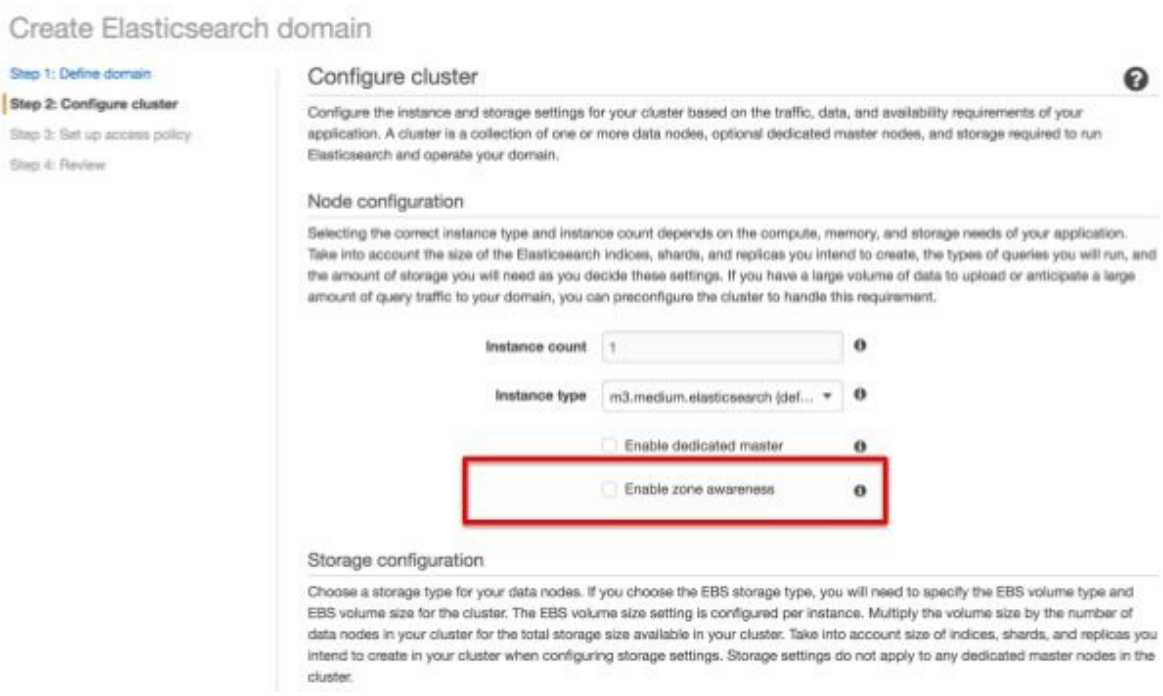


Data nodes: queries and updates

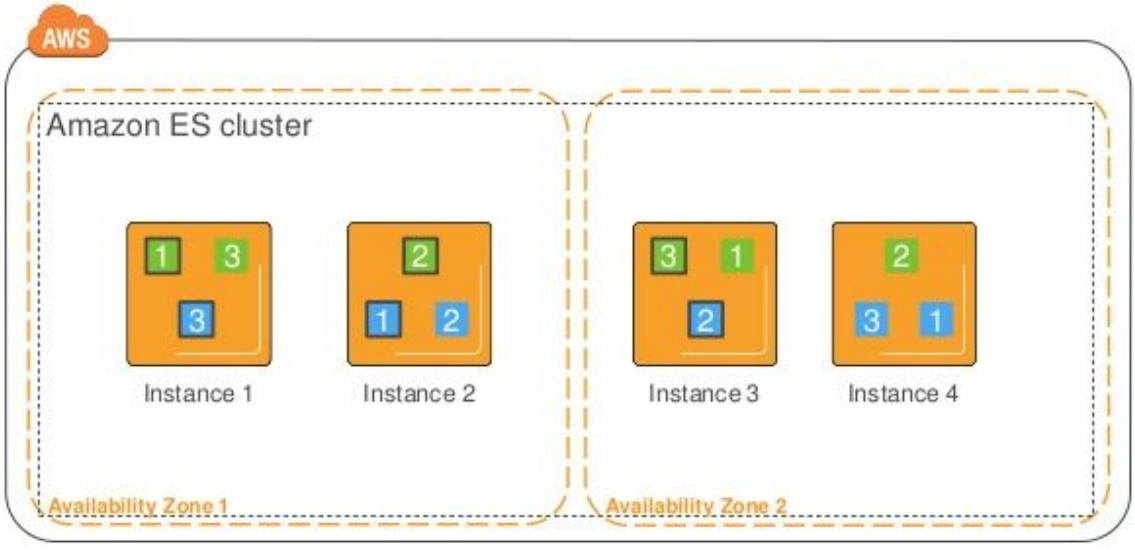
Master node recommendations

Number of data nodes	Master node instance type
< 10	m3.medium+, c4.large+
< 20	m3/4.large+, r3/4.large+
<= 40	c4.xlarge+, m3/4.xlarge+, r4.xlarge+

Always use an odd number of masters, >= 3



Cluster with zone awareness

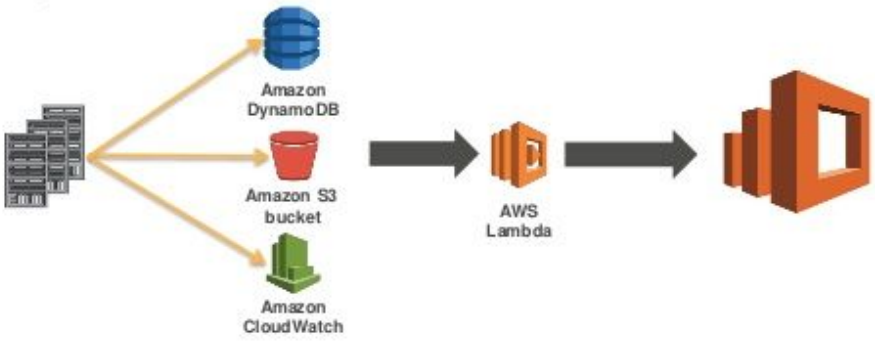


Small use cases

- Logstash co-located on the Application instance
- SigV4 signing via provided output plugin

- Up to 200GB of data
- m3.medium + 100G EBS data nodes
- 3x m3.medium master nodes

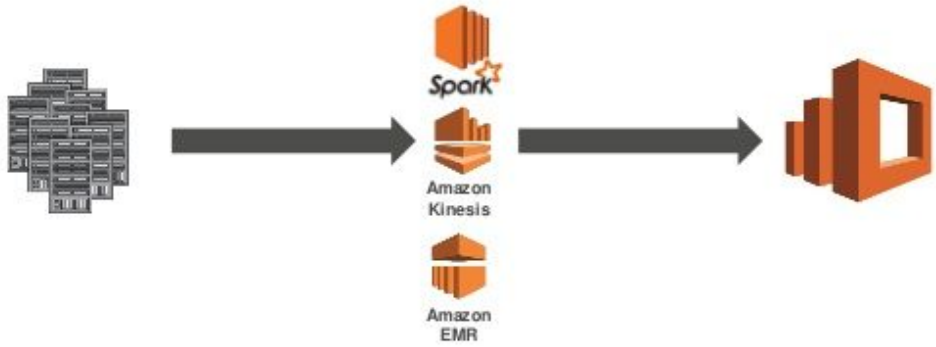
Large use cases



- Data flows from instances and applications via Lambda; CWL is implicit
- SigV4 signing via Lambda/roles

- Up to 5TB of data
- r3.2xlarge + 512GB EBS data nodes
- 3x m3.medium master nodes

XL use cases



- Ingest supported through high-volume technologies like Spark or Kinesis

- Up to 60 TB of data today
- R3.8xlarge + 640GB data nodes
- 3x m3.xlarge master nodes

Best practices

Data nodes = Storage needed/Storage per node

Use GP2 EBS volumes

Use 3 dedicated master nodes for production deployments

Enable Zone Awareness

Set indices.fielddata.cache.size = 40

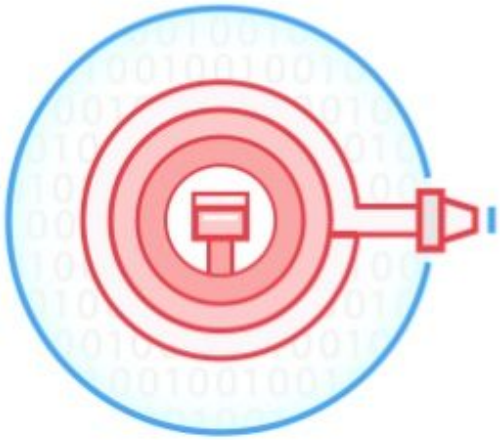
Kinesis Firehose

Kinesis Firehose overview

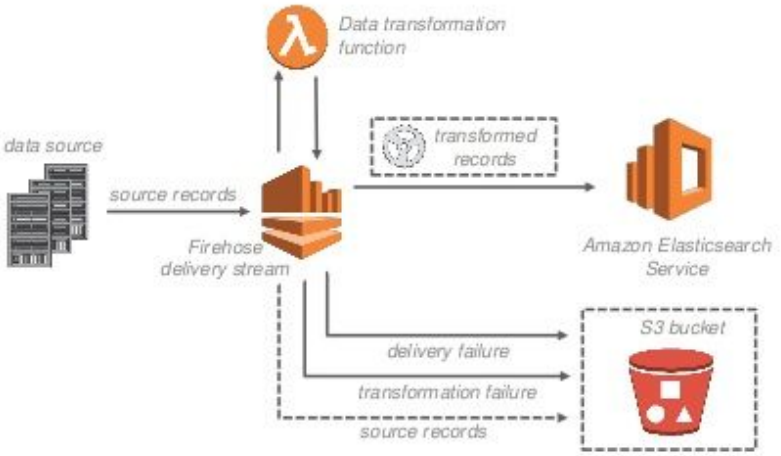
Delivery Stream: Underlying AWS resource

Destination: Amazon ES, Amazon Redshift, or Amazon S3

Record: Put records in streams to deliver to destinations



Kinesis Firehose delivery architecture with transformations



Lambda blueprints for common use cases

▼ (Optional) Create a new Lambda function

Choose a Lambda blueprint below to begin configuring a new Lambda function. Return to this screen once you've finished configuration, refresh the list of available Lambda functions below and select your new function.

Filter

Blueprint name	Description
Kinesis Firehose processing	This blueprint contains the record transformation and status model required by Firehose in order to return transformed records from Lambda to Firehose. Use this blueprint for any custom transformation logic.
Apache Log to JSON	This blueprint parses and converts Apache log lines into JSON objects, with predefined JSON field names.
Apache Log to CSV	This blueprint parses and converts Apache log lines into CSV format.
Syslog to JSON	This blueprint parses and converts Syslog lines into JSON objects, with predefined JSON field names.
Syslog to CSV	This blueprint parses and converts Syslog lines into CSV format

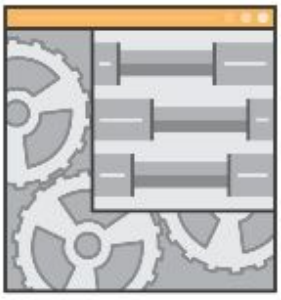
Transform this

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245

To this


```
{
    "verb": "GET",
    "ident": "-",
    "bytes": 6245,
    "@timestamp": "1995-07-01T00:00:01",
    "request": "GET /history/apollo/ HTTP/1.0",
    "host": "199.72.81.55",
    "authuser": "-",
    "@timestamp_utc": "1995-07-01T04:00:01+00:00",
    "timezone": "-0400",
    "response": 200
}
```


Kinesis Firehose features for ingest



Serverless scale

Error handling

S3 Backup

Best practices

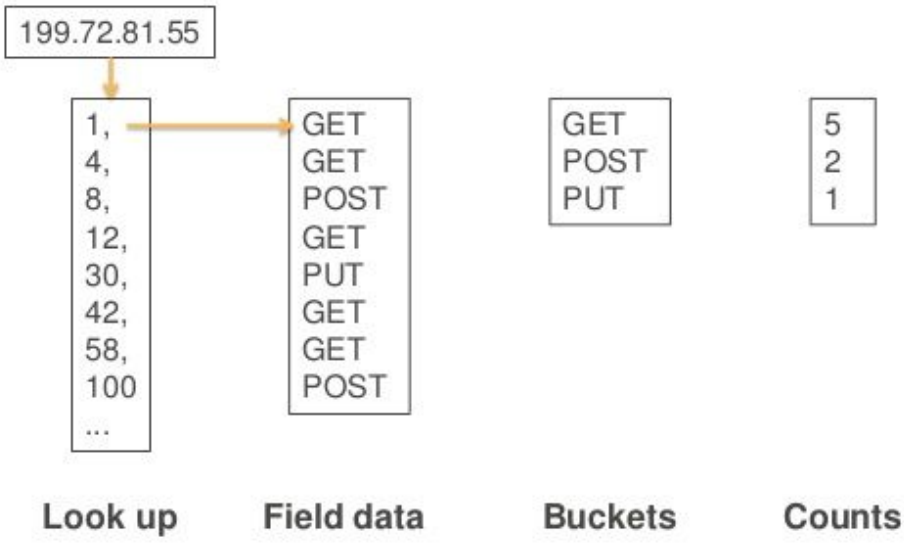
Use smaller buffer sizes to increase throughput, but be careful of concurrency

Use index rotation based on sizing

Default: stream limits: 2,000 transactions/second, 5,000 records/second, and 5 MB/second

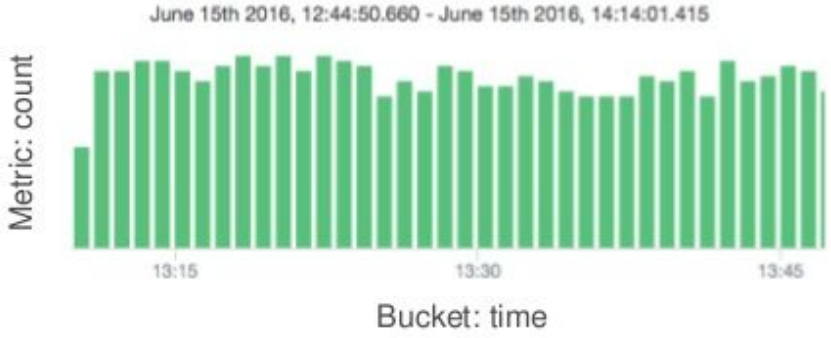
Log analysis with aggregations

host:199.72.81.55 with <histogram of verb>

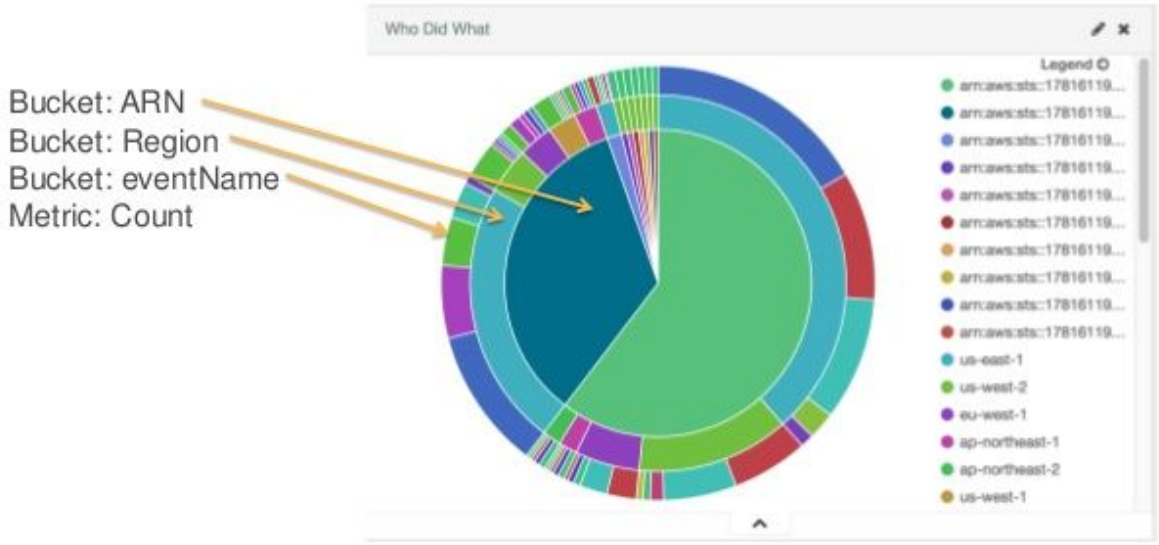


Amazon ES aggregations

Buckets – a collection of documents meeting some criterion
Metrics – calculations on the content of buckets



A more complicated aggregation



Best practices

Elasticsearch provides statistical evaluations based on field data gathered from matching documents

Visualizations are based on buckets/metrics

Use a histogram on the x-axis first, then sub-aggregate

Amazon Elasticsearch Service

Run Elasticsearch in the AWS cloud with Amazon Elasticsearch Service

Use Kinesis Firehose to ingest data simply

Kibana for monitoring, Elasticsearch queries for deeper analysis