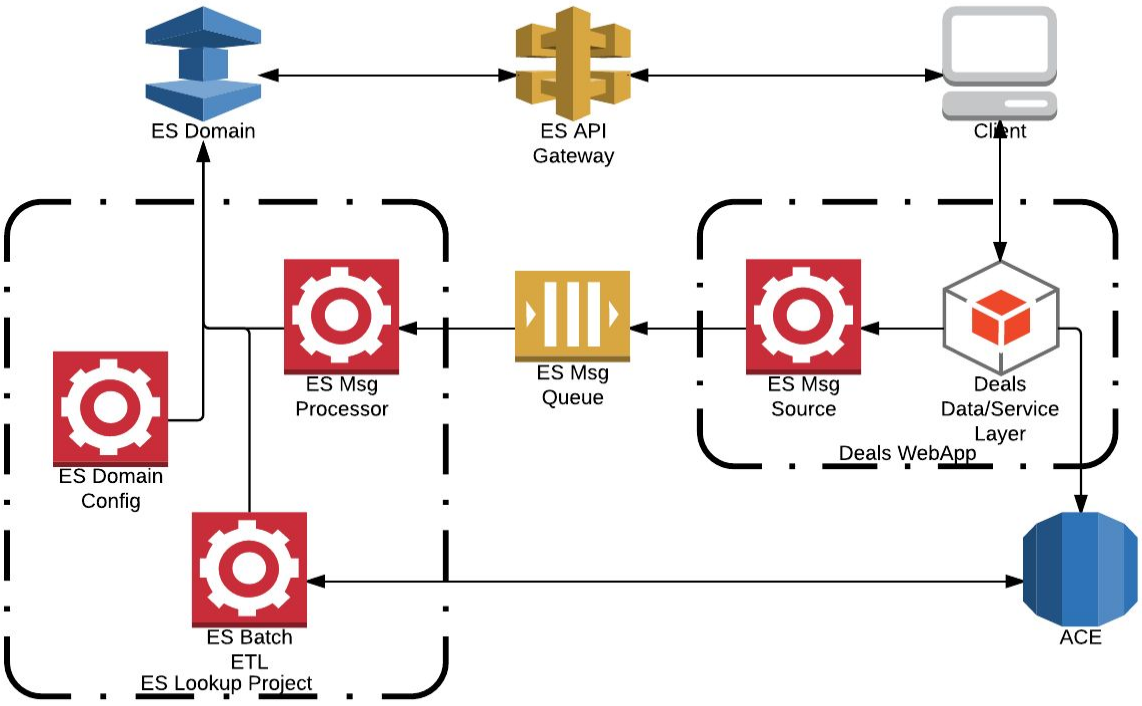
ElasticSearch Lookup Infrastructure

Date	User	Change
17 May 2016	Miguel Cordones	Created

Introduction

The goal is to design a lookup function for deals that respond to partial string matches. The well known use case is 'type ahead' lookups. We will build on top of the AWS infrastructure and existing work. A stretch goal is to refine the design so that it can be replicated where search functionality is required, microservices style. A high level architectural diagram of the system is given below.



System Description

The client (UI in our case) will gather search criteria from users and reach out to the ES Domain. The client can directly hit the ES Domain API, or go via ES API Gateway, or use a custom API in the Deals Data/Service Layer. The ES Domain will return a set of search results to the client, which will include the ACE primary key, to enable us to lookup the entry in the primary data store (ACE).

There are two ways to get data into the ES Domain: First, an ES Batch ETL can query deals from ACE and insert the entries into the ES Domain. The benefits of this approach are simplicity and a full initialization. The drawback is its built in delay to respond to small data changes. For example if a single entry is added or deleted, a full batch ETL is wasteful. A second data ingestion process can remedy these deficiencies. An ES Message Processor will listen to a message queue for inserts, updates and deletes. These messages will be generated by the existing Deals Data/Service Layer from the corresponding inserts, updates and deletes.

Development

The custom code components required for the system (see the diagram) are contained in the two boxes labelled "Deals WebApp" and "ES Lookup Project". Only a small amount of code is required for a POC. The implementation plan below will provide some detail on what code when and iterative refinement.

New Custom Components

This custom code runs (more or less) continuously in production:

ES Batch ETL

Runs a query against ACE, pushes the results to an ES Domain. Could also monitor the ES MQ and respond to 'full reload' requests. Good candidate for AWS Lambda.

Not quite -- the Oracle SDK is a problem for a 'serverless' solution. It seems that we will need a container at least to interface with Oracle. Then Lambdas watching S3 or SQS could load the data into ElasticSearch. There is also a streaming mechanism for ES. Streaming or events would be a good option to avoid moving large files around.

Oracle Rest Data Services is a layer on top of Oracle Relational DB removes the need for the Oracle SDK, so Lambdas would again be feasible.

ES Msg Processor

Reads individual insert, update, delete messages from the ES MQ and updates the ES Domain accordingly. Definitely shares concerns with the ES Batch ETL process. Good candidate for AWS Lambda.

ES Msg Source

Posts the insert, update and delete messages to the ES MQ based on corresponding Deals Data/Service Layer operations. Need to integrate AWS SQS with Deals team java code.

Standard Components and Configuration

Infrequently run configuration and/or deployment scripts:

ES Domain Config

Bash script to create the ES Domain. Integrated into the deployment job?

ES Msg Queue

Bash script to create the MQ. Integrated into the deployment job?

ES API Gateway

(If necessary) Bash script to configure the gateway. Integrated into the deployment job?

Implementation Plan

I have created an ES Domain and made indices for Advertiser and Art Size. I can adapt this existing (python) code to process Deal data into a new index. This can form the basis of the batch ETL. A hackathon is in the planning stages to explore the UX Team's CardView and Search components. I will make the the ES Domain available to the hacking team. One of my goals would be to walk away with the answer questions around API Gateway or custom service work. Resolving those questions, and providing the batch ETL would give a deployable feature (MVF).

Iterative refinements would include:

- The message queueing aspect of the ES data updates
- Performance and monitoring metrics
- Deployment and operational considerations for scaling
- Packaging and documenting the pattern to facilitate more widespread use
-

Open Questions

1. Can the client effectively reach out to the ES Domain directly (most desirable)? If not, we can possibly fall back to AWS API Gateway? or if all else fails, implement custom service endpoints in our service layer (least desirable).

2. Does ES support an atomic 'rename index'? This will allow us to use the copy and swap pattern for full data loads.

3. The whole arena of Lucene query syntax. There may be index tuning (config) required to align search results with expectations.

4. Should we try to make the Data/Service Layer insert, update, delete atomic with the ES Msg Source (probably not possible)? Should we have a fault counter and force a reload at some point?

5. AWS SQS setup and integration

6. AWS lambda error handling

7. AWS API Gateway setup

8. All indices in one domain? Multiple indices per domain? One index per domain?

9. Providing and consuming data across teams:
    a. 1C - Advertisers/Org
    b. ACE - tactics, deals
    c. Art sizes, publishers, …

10. As long as ACE is not in AWS (RDS for example), we will always have to travel a roundabout path to get data into AWS. - The ACE API does this now, and it has added 1.5ms to hit EDWPRD in AOL data center.