OpenSearch Logging

1. Summary

An OpenSearch cluster that will aggregate application logs from account registration from adam services (Arches, AIS etc.) into an easily queryable place and a programmable API that will query this system to retrieve the logs for a given registration attempt (i.e. provide an entityid and return all logs associated with it).

With this service, the team would be able to use this going forward to simplify and speed up on-call ticket resolution. Many of our current tickets are related to a failed registration of one form or another, like an entity key that wasn't put in the right database or an ID that wasn't created properly. Being able to get all logs associated with a registration in one click will enable on-calls to quickly find the cause of a problem and solve it.

2. Background

ADAM-Galaxy (and parts of SHIELD) has ~7 different services involved in registering an advertising account, depending on the type of account and ingress point. Tracking the status of an account registration can be very difficult, since the logs for the above services are stored in different places, different formats, and different systems entirely. For example:

- GRS and GAMS logs are stored in CloudWatch, and can only be queried (at least most easily) through the AWS Console. Different accounts store the logs for each stage/region of each service, and while there is a product to query them cross-account, it hasn't been launched to production yet and won't work across regions.
- Arches, AmsIdentityService, AdvertiserRegistration, and VssAdvertisingService logs are stored in Timber, but they're stored compressed (making querying slower) and logs are only pushed to Timber every hour off the Apollo hosts.
- AdvertiserUserManagementService logs are also stored in Timber, but on a different fleet (owned by SHIELD) and in a different format (zst instead of gz).

It's difficult and time-consuming to query all of these databases, but all of them will frequently contain critical information about an account registration.

3. Terminology

- **Admin Tool** — Amazon's advertising tools website service
- **API** — Application Programming Interface
- **Arches** — ArchesService
- **AIS** — AmsIdentityService
- **CDK** — Cloud Development Kit
- **entityid** — The primary key for the Entity, an object representing an advertising account in AmsIdentityService
- **GAMS** — Global Account Management Service
- **GRS** — Global Registration Service
- **Logstash** — a tool for managing events and logs. You can use it to collect logs, parse them, and store them for later use (like, for searching). It is fully free and fully open source and can dynamically unify data from disparate sources and normalize the data into destinations of your choice.
- **NAWS** — Native AWS
- **POC** — Proof of Concept

- **SHIELD** — AD
- **UI** — User Interface
- **VPC** — Virtual Private Cloud

4. In Scope / Out of Scope

In Scope:

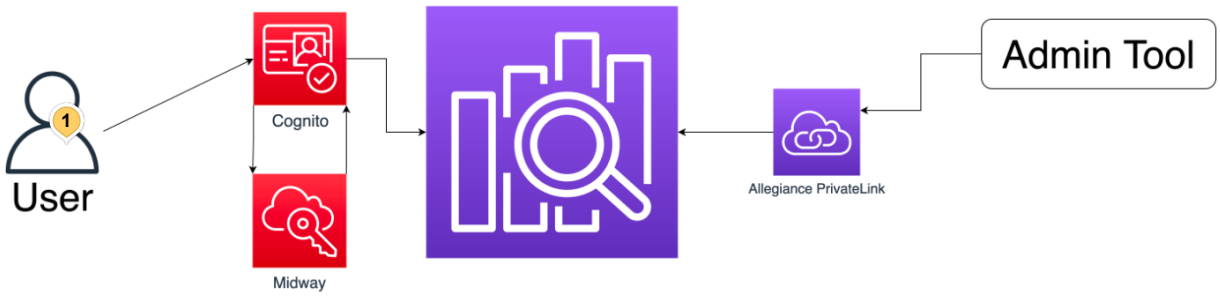
- Productionize POC
- Expand POC to other services: AmsIdentityService, GRS, GAMS, Arches
- Expand above services into taking prod logs
- Implementing new API

Out of Scope:

- Building a UI for the service
- Security for POC

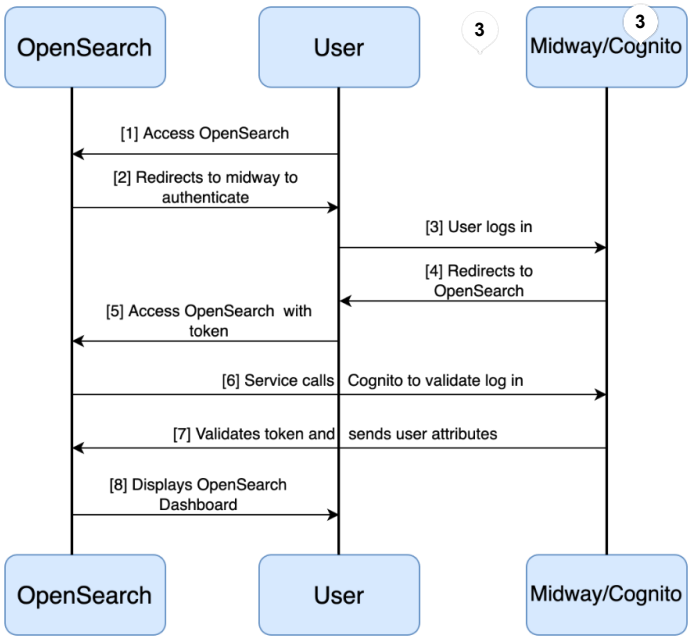
5. Current Architecture

Below is a low level design of the current architecture for the proof of concept. The lines in red signify aspects of the diagram that have not yet been implemented in the proof of concept. The cluster, along with all the aspects of security with allegiance, cognito and midway have been set up. The only adam service that has been onboarded to the OpenSearch cluster is Admin Tool through the allegiance private link. The existing pipeline in currently in admin tool beta.



With the current service, a user goes through cognito and midway (for authentication). It sets up pool of users using federate system to allow only authorized users (sspa-adam) to be able to log into service with federate and midway. Other services access the cluster through Allegiance.

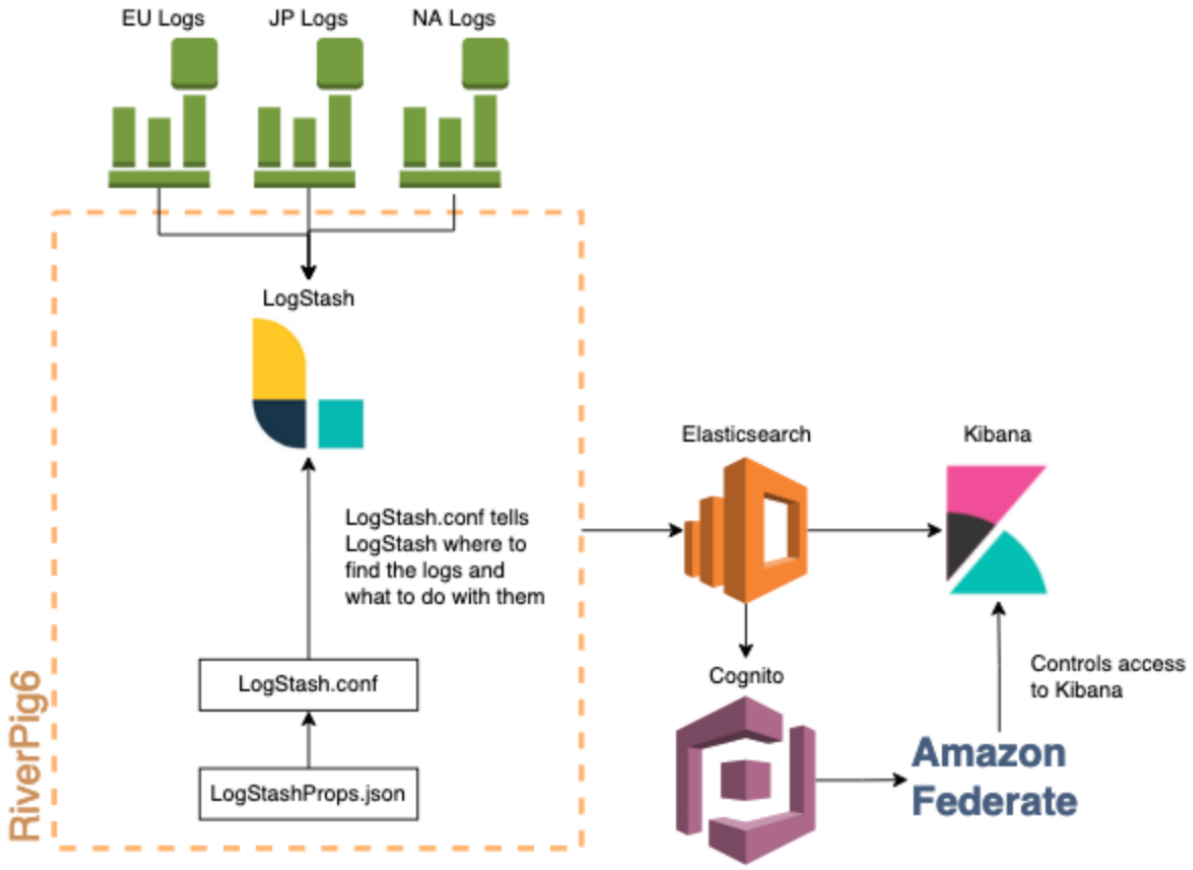
Below is an architecture diagram showing how the security is currently set up for accessing the OpenSearch cluster.



A user calls OpenSearch then OpenSearch redirects the user to federate midway to authenticate (have them log in with their credentials), the user is then redirected back to OpenSearch after a successful login. The service then calls cognito to validate that the user has actually logged in successfully, cognito then validates the user's log in by sending a token with the user's attributes back to OpenSearch. Finally the user gains access to the OpenSearch dashboard.

Lastly, in this section we will look at how the logs are transferred to OpenSearch from the services. This process happens through the use of the ELK stack - Elasticsearch, Logstash and Kibana.

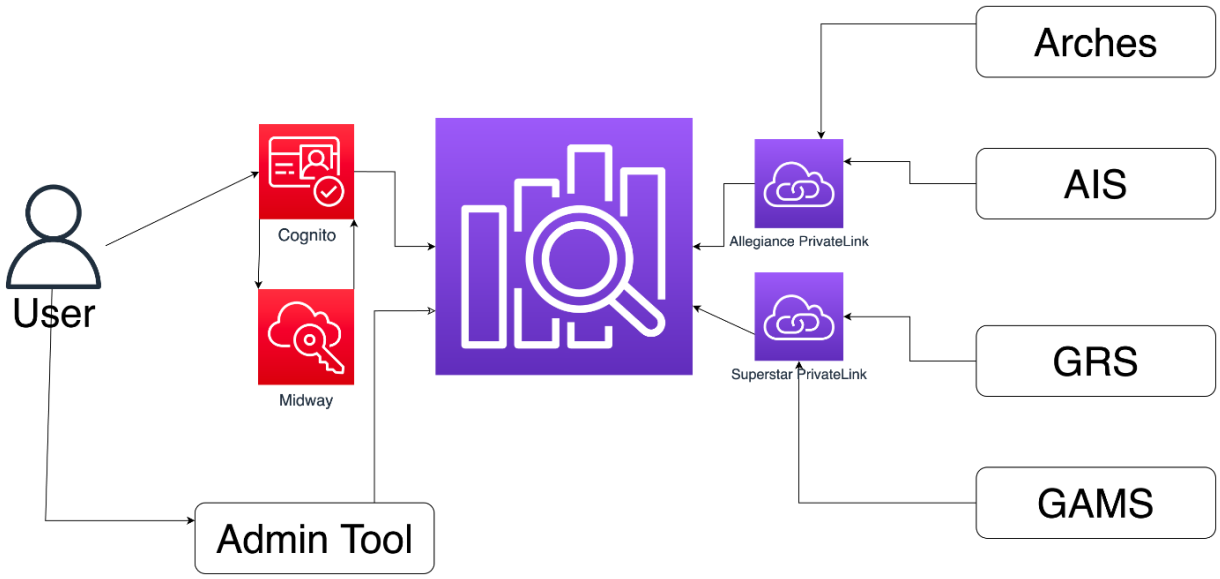
Using Logstash you can parse and push the entries in one or more processes log files to Elasticsearch. Logstash does most of the heavy lifting, we need only specify the Elasticsearch domain and the log files we wish to push and how to parse them. The parsing splits the log entries up into separate records and attributes to be pushed to Elasticsearch. Logstash will by default, at the time of writing, push log entries every 50 ms to Elasticsearch or every 125 log entries. Kibana gives you a powerful web interface with search, filter, and statistical analysis functionality driven by Elasticsearch. Using this stack we can have real-time access to all log entries from all hosts running a service. This is much more powerful and less time consuming than grepping through logs of individual hosts. Below is an architecture diagram of a typical ELK stack flow:



For this project we are using OpenSearch, which is a fork of the Elasticsearch and Kibana code base. OpenSearch is an easier way to run the ELK stack as the Amazon OpenSearch service delivers Elasticsearch with built-in Kibana and it supports all the standard Logstash input plugins.

6. Proposed Architecture

The goal is to have the AIS and Arches services (AIS, GRS, GAMS, Arches, etc.) onboarded to the cluster and as a stretch, have an API that can query the system located in admin tool and having GAMS and GRS onboarded through the Superstar private link. Below is a high level diagram of the planned architecture.



A user would access the service through cognito which would take them to midway to login with their credentials. Afterwards, they are redirected back to cognito where they can gain access to the OpenSearch cluster and start searching for log data. The users would be able to query the data using an API that will be located in admin tools.

7. Framework Solutions

This section will discuss the options for collecting and querying data and the location of the programmatic API.

7.1 LOG COLLECTION AND QUERYING TOOL

7.1.1 OPENSEARCH

OpenSearch is a distributed, community-driven, Apache 2.0-licensed, 100% open-source search and analytics suite used for a broad set of use cases like real-time application monitoring, log analytics, and website search. OpenSearch provides a highly scalable system for providing fast access and response to large volumes of data with an integrated visualization tool, OpenSearch Dashboards, that makes it easy for users to explore their data.

Pros

- Easy to learn and maintain
- Good community support
- Real time log monitoring
- Powerful tool for querying logs

Cons

- One limitation of this product is that if you create an OpenSearch cluster in a VPC, you won't be able to create indices on the cluster
- Networking is in general more difficult with OpenSearch in AWS

7.1.2 CLOUDWATCH

CloudWatch is a monitoring and observability service that collects monitoring and operational data in the form of logs, metrics, and events, and visualizes it using automated dashboards so you can get a unified view of your AWS resources, applications, and services that run on AWS and on premises. You can visualize the experience of your application end users and validate design choices through experimentation. Correlate your metrics and logs to better understand the health and performance of your resources.

Pros

- CloudWatch offers users the ability to monitor metrics and logs from AWS resources, applications and services, as well as on-premises servers
- Easy to set up alarms and rules
- Application Insights also make it simpler to create dashboards that display alerts and problems

Cons

- Log processing is not real time
- Querying is harder (can only be done using the AWS console)
- Querying and retaining logs at scale also becomes difficult

7.1.3 RECOMMENDATIONS:

After consideration, the decision is to proceed with OpenSearch for the following reasons:

- Though it's simpler to create dashboards using CloudWatch, we don't need dashboards for the scope of this project
- Most adam services aren't in NAWS
- It's more important to be able to query logs powerfully which OpenSearch is better for
- We need realtime log querying which is possible with OpenSearch but not CloudWatch
- There are a lot of logs so using a service that is easier to scale and can retain logs is also important

7.2 PROGRAMMATIC API

This section looks at the proposed benefits of having a programmatic API for querying the logs in OpenSearch. The purpose of the API is to have a user be able to query the service in the cluster to retrieve all logs for a given registration attempt. This would give users the ability to get all logs associated with a registration in one click and will enable on-calls to quickly find the cause of a problem and solve it, simplifying and speeding up on-call ticket resolution.

This API would be located in the one of the adam services (probably Admin Tool) with a UI component where all a user would have to do is put in the entityid for a registration attempt, click a button and have all the related logs for that registration attempt pulled without having to write a query themselves making it a faster approach than querying directly in OpenSearch.

The API is a stretch goal for this project but without it, querying can still be done easily in OpenSearch so having all the logs accessible on that service would still be worthwhile.

8. Roadmap/Task Breakdown

HIGH LEVEL TASKS

- P0
  - Productionizing existing POC
    - [ ] Set up POC on local
    - [ ] Productionize POC (CDK, pipelines, etc.)
    - [ ] Expand to AmsIdentityService
  - Expand to more services
    - [ ] Expand to GRS, GAMS and Arches
    - [ ] Look/adjust configuration as needed
  - Expand to prod
    - [ ] Expand the onboarded services into taking prod logs
- P1
  - API Query for partial logs
    - [ ] Get programmatic call working to OpenSearch beta cluster
    - [ ] Build part 1 of query: getting all logs from a single service for a single request id
    - [ ] For example all request, application, and service logs from AmsIdentityService for a specific PutEntityKey request
  - API Query for marketplace registration logs
    - [ ] Start building part 2 of query: getting all logs for a single marketplace registration in Arches
    - [ ] All Arches logs for the registration
    - [ ] All AIS logs for the calls that the Arches wire logs show

9. Future Considerations

In the future, this project could look to onboard User Interfaces to be used with the service. Onboard GRS and/or GAMS using superstar.

References

- https://broadcast.amazon.com/videos/498735?ref=personal
- https://code.amazon.com/packages/GlobalAccountConstructs/trees/heads/experimental/opensearch-stack/--/lib
- https://code.amazon.com/packages/BASETaskManagerServiceCDK/blobs/mainline/--/lib/constructs/kibana_cognito.ts#L144
- https://code.amazon.com/packages/ADAMTimberPipelineCDK/trees/experimental/opensearch-stack