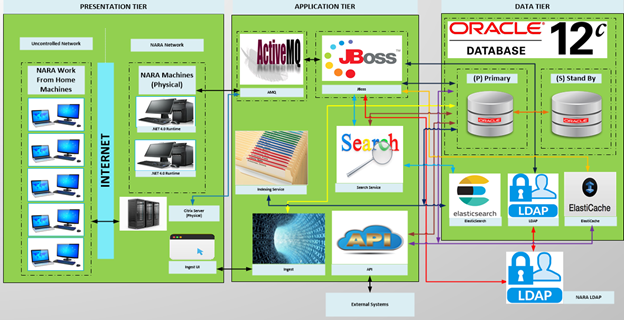
# 1. DAS Architecture

This section provides a high-level architecture for the DAS System. As illustrated in Figure 1, DAS has a three-tier architecture consisting of:

* Presentation
* Application, and
* Data tiers



**Figure 1 - High Level DAS System Architecture**

## 1.1 DAS Presentation Tier

The Presentation Tier includes the desktops or laptops NARA Staff use to access the DAS system. Computers located within the NARA network access the DAS system directly over the NARA Local Area Network (LAN). External computers (e.g., users working remotely) first authenticate and gain access to the NARA LAN through a remote access Citrix server. Installation of the .NET 4.0 Framework (full version) is required to operate the user interface (UI) portion of the system. This includes computers within the NARA network as well as the Citrix server. External computers do not need the .NET 4.0 Framework installed, as these components are installed on the Citrix server. The DAS system application UI components are deployed to the JBoss web container hosted in a Tomcat instance and sent to the user’s machine as necessary. Ad Hoc and scheduled reports are generated via Logi Analytics in the presentation tier. Highlights of the Presentation tier are as follows:

* The Presentation tier provides a rich interactive user experience leveraging the capabilities of the Windows Presentation Foundation (WPF) client platform.
* The Presentation tier implements SOA 2.0 concepts and is rules, security, event-driven, and messaging aware. A messaging aware UI can receive notifications directly from the Application Tier.
* The Presentation Tier interacts with the Application Tier asynchronously by sending and receiving messages on the DAS message queue. The JBoss Enterprise Service Bus (ESB), which is responsible for Application Tier service orchestrations, also leverages the capabilities of the DAS message queue to communicate with the Presentation Tier and for internal Application Tier messaging.

## 1.2 DAS Application Tier

The Application Tier hosts the set of services which, when combined, implement the DAS System methods and expose data. The services are designed to stand alone without reference to the Presentation tier or other external UI components. The Presentation tier communicates with the Application tier by way of the JBoss ESB. The ESB exposes all capabilities offered by the DAS system. The Presentation Tier interacts with the ESB by sending and receiving messages on the DAS message queue. The DAS message queue is implemented as a Java Message Service (JMS)-compliant queue using the open-source Apache Active MQ product. An enterprise infrastructure element, the ESB exposes functionality available to other “external” NARA systems as web services. The ESB performs orchestrations, as necessary, to map underlying services to desired capability. The Application tier consists of the following major application platforms:

* The JBoss Platform (hosting the JBoss Application Server, JBoss ESB, JBoss jBPM, and the Apache ActiveMQ message service)
* AWS Elasticsearch Service: AWS managed service as the DAS search engine; Elasticsearch version 6.3 is utilized
* Search Service: Consumes the REST APIs that Elasticsearch 6.3 exposes to search content and also serves as the producer to support the search functions for both description and authorities in DAS SOA module
* Indexing Service: Background scheduler task that scans an Oracle log table to find new/updated/deleted descriptions, digital objects and, authorities; and then index them into the search engine
* Logi Analytics Service: Integrates with WPF to support reports generated from client grid information

## 1.3 DAS Data Tier

The data tier contains all data held and controlled by the DAS system using Oracle 12c XML database. In May 2019, to address DAS system performance and capacity limitations, NARA re-engineered the DAS application and data tiers to use AWS Elasticsearch Service for fulfilling various search requests from the DAS UI and DAS API instead of Oracle. This has addressed many of the problems related to poor performance that NARA users experienced when performing DAS searches. Also located in the Data Tier is the 389 LDAP Directory Server cluster to store DAS-specific information about users provisioned in the DAS system.

For a holistic view and understanding of the DAS system, please see GFI Item “DAS Detailed Design Document” also listed in Section 10.3.2 of the PWS.

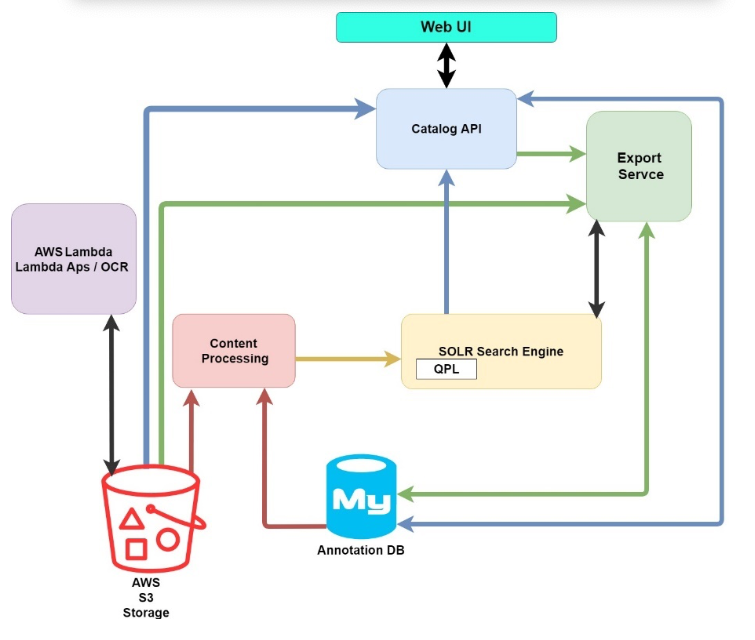
# 2. NAC Architecture

This section provides a high-level architecture of the National Archives Catalog (NAC) System, and its corresponding APIs - API v1 (Internal), API v1 (Public).

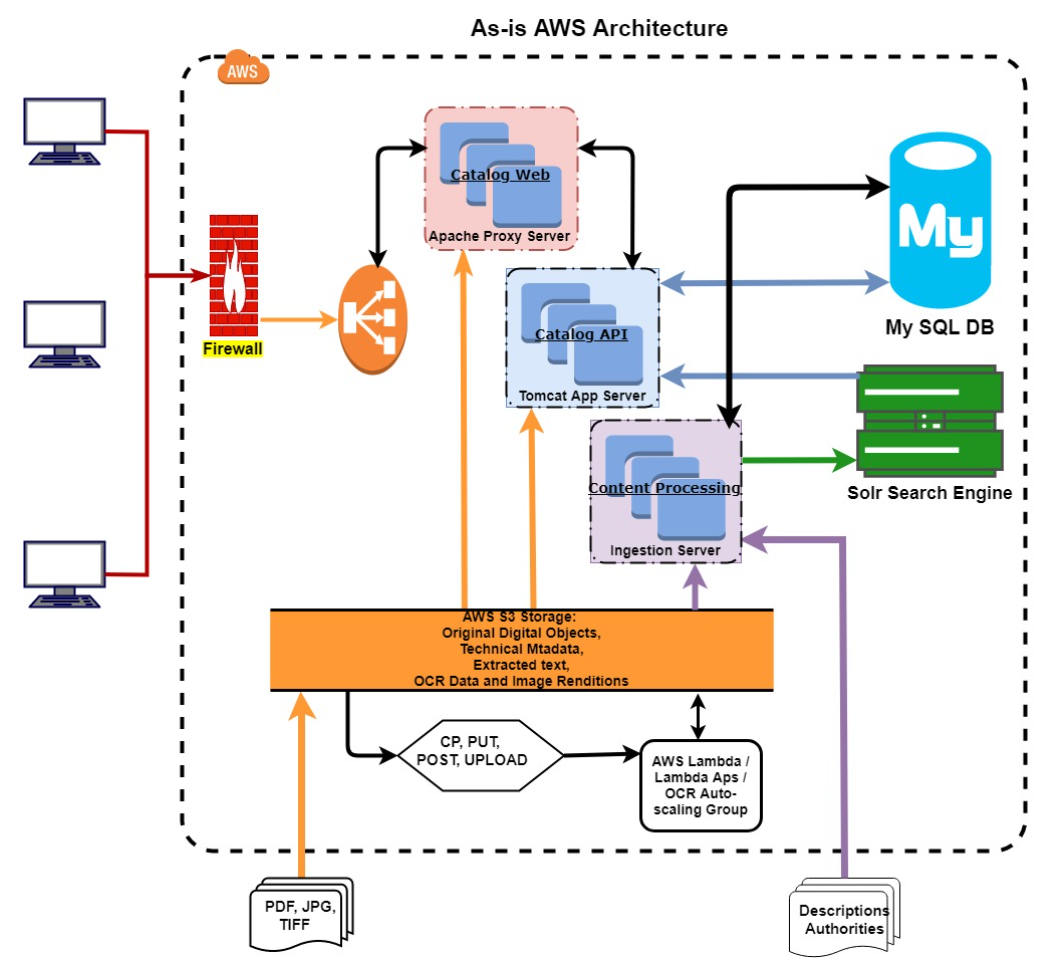
### **4.2.1 NAC System Component and AWS Architecture**

NAC is a web-based application with a multi-tier client server model, hosted in the Amazon Web Service (AWS) environment. As illustrated in the figures below (Figure 2 represents a logical component view while Figure 3 represents a cloud component view), the primary system components of the current Catalog application are:

* Catalog Web as the Web Application, deployed on Apache HTTP servers
* Catalog API (Internal API v1) as the RESTful API services, deployed on Apache Tomcat servers
* Export Service as the background service for exporting content out of the Catalog, deployed on Apache Tomcat servers as well
* Apache SolrCloud cluster as the search engine
* Content Processing Application to index content into the search engine
* Annotations Database, a MySQL Database in master-slave configuration used for user authorization and authentication as well as a data store for public contributions
* Metadata and VIPS Processing Lambda Functions, OCR and Alternate Processing Servers to generate technical metadata, extracted data, OCR data, and alternate renditions of digital holdings (thumbnails, image tiles, alternative resolutions)
* AWS S3 for storing the original digital objects as well as all the content generated by the NAC Lambda processing pipeline. Only when descriptions and authorities XML data generated by the DAS export services are ingested into NAC will the content processing module combine the XML data from DAS with the metadata and extracted text data from S3 to push to the search engine.

****

**Figure 2. NAC System Component Architecture**

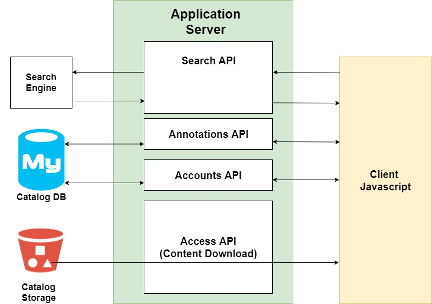
****

**Figure 3. NAC AWS Architecture**

### **4.2.1 NAC API v1**

Apart from searching the content via the website, NAC allows the public to download this content, as well as access the same programmatically via application programming interfaces (APIs). The request-response flow for all APIs in the Catalog (including search) is outlined in Figure 4 below.

### 



**Figure 4. Request-Response Flow for All APIs in the Catalog**

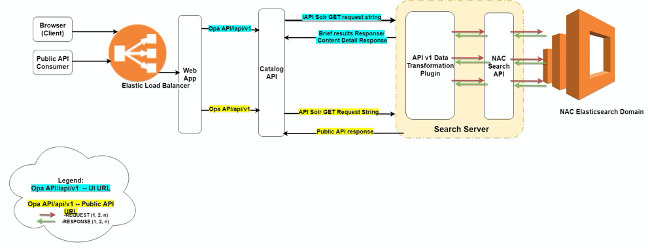
NAC API v1 exists in two flavors - an Internal instance to support the Catalog user interface, referred to as Internal API, and a Public instance for public use, referred to as the Public API. While both the Internal and Public instances provide access to the same data, the primary reason for the Internal and Public APIs to be two separate instances is that the Internal API provides data elements that are specific to the functionality of the Catalog UI, whereas the Public API provides raw access to the core data elements as they exist in the underlying data model.

#### ***4.2.1.1 - Internal API v1***

The internal API supports the Catalog user interface. The primary components of the current Internal API are:

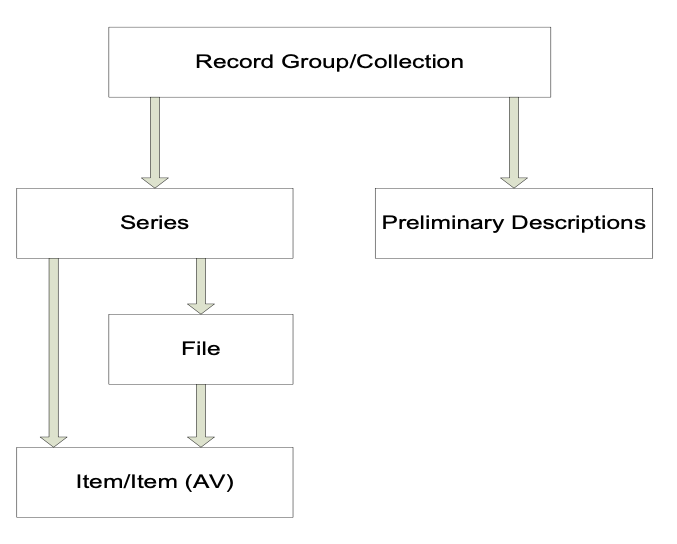
* Access API – For accessing and downloading content by URL from the NARA Catalog
* Search API – Allows searching for content using the search engine, returning brief results. In addition to results, the search API can also do the following:
  + Highlights: These are highlighted search keywords found in the field content.
  + Facets: These are histograms of field metadata values (for certain fields only) which can be used to subset the results (also called “guided navigation”)
  + Save results for bulk export: This allows you to bulk export metadata and content from the Catalog into compressed files (tar.gz) which can be efficiently downloaded (bulk exports are processed in the background)
  + Save results to list: Results can be saved to a list to be individually examined later
  + One step search and process: Allows you to perform any of the access or annotation functions on the first result returned by a search
* Annotations API – For accessing, adding, updating, and deleting tags, transcriptions, discussions, and translations.
* User API – Anything which is user-related, such as information for the user’s home page, user lists, bulk downloads, and viewing user contributions.

The catalog search data flow is displayed in Figure 5 below.

**Figure 5. Catalog Search Data Flow**

#### ***4.2.1.2 - Public API v1***

As referred to above, the Public API is a functional equivalent of the Internal API, but differs in that it provides raw access to the core data elements as they exist in the underlying data model. These data elements are made up of a list of more than 400,000 fields and listed in a document called the NAC “Whitelist” (available in GFI section). While the primary purpose of the Whitelist document - which also exists for DAS - is to provide a list of all the fields that are considered valid for use by the API consumer; it is just as important a document to illustrate the hierarchical nature of the content in the Catalog shown in Figure 6 below.



**Figure 6. Description Relationships**

For example, as it relates to the hierarchy shown in Figure 6 above, the whitelist contains:

* ~ 7K fields at the RecordGroup level as referred to by ‘description.recordGroup.\*’
* ~ 15K fields at the Collection level as referred to by ‘description.collection.\*’
* ~ 3K fields at the Series level as referred to by ‘description.series.\*’
* ~ 29K fields at the FileUnit level as referred to by ‘description.fileUnit.\*’
* ~ 115K fields at the Item level as referred to by ‘description.item.\*’
* ~ 231K fields at the ItemAv level as referred to by ‘description.itemAv.\*’

In sum total, there are about 400,000 such fields that are therefore available for end users to peruse, which can range from a simple keyword query as follows:

<https://catalog.archives.gov/api/v1/?q=armadillo>

To a more complex one, that allows you to, for example, construct a very specific search across all series for records with a "Top Secret" security classification, and the term "Army" in the description's title, as follows:

<https://catalog.archives.gov/api/v1/?description.series.accessRestriction.specificAccessRestrictionArray.specificAccessRestriction.securityClassification.termName=top%20secret&description.series.title=army>

|  |  |
| --- | --- |
| **Example Key from Whitelist** | **Example Value** |
| description.series.title | army |
| description.series.accessRestriction.specificAccessRestrictionArray.specificAccessRestriction.securityClassification.termName | top secret |

While the internal instance of API v1 successfully caters to the needs of the Catalog UI, it is also limited in the sense that it is designed specifically for the Catalog UI. Similarly, while the public instance of API v1 provides raw access to all of the 400,000 or so core data elements, it also has some natural challenges and pain points, namely:

* Data Model is hierarchical, inconsistent and unintuitive
* Two distinct APIs (public and internal) need to be managed
* Performance bottlenecks
* Documentation that is not interactive and is unintuitive to non-archivists
* Lacks engagement tracking

For a holistic view and understanding of the NAC system, please also refer to the:

* Legacy Documents as listed in section 10.3.1 of the PWS
* Current Design and API Documents as listed in section 10.3.2 of the PWS
* In Progress Design and API Documents as listed in section 10.3.3 of the PWS