National Archives Catalog (NAC)

**and**

**Description and Authority Services (DAS)**

System Architecture Document (SAD)

**For the DAS and NAC Modernization**

**Preliminary Version**

Version 1.0

September 25, 2017

Prepared for:

National Archives and Records Administration (NARA)

Catalog Task NAMA-16-G-005, DME 16 Task 005

Prepared by:



1760 Old Meadow Road

McLean, VA 22102

This document contains Proprietary Information provided by PPC – A DSA Company,

Handle in Accordance with Proprietary and Confidential Restrictions

***Acknowledgements:***

***PPC would like to acknowledge the significant technical contributions made by the following staff:***

* ***Dr. Urmi Majumder***
* ***John Henson***
* ***Edwin Piedmont***
* ***Matthew Mariano***
* ***Fawad Shaikh***

Contents

[1 Introduction 7](#_Toc494124331)

[1.1 Document Scope 7](#_Toc494124332)

[1.2 Background and Motivation 7](#_Toc494124333)

[1.2.1 Description and Authority Services Background 7](#_Toc494124334)

[1.2.2 National Archives Catalog Background 8](#_Toc494124335)

[1.2.3 System Components in Current NAC and DAS Applications 9](#_Toc494124336)

[1.2.4 Problems with the Existing Setup in DAS and NAC 11](#_Toc494124337)

[1.3 Future Components Overview 15](#_Toc494124338)

[1.3.1 DAS System Components: Design, Implementation and Integration 15](#_Toc494124339)

[1.3.2 NAC System Components: Design, Implementation and Integration 17](#_Toc494124340)

[1.4 Assumptions 18](#_Toc494124341)

[2 High-Level Modernization Architecture 19](#_Toc494124342)

[2.1 High-Level Architectural Diagram 19](#_Toc494124343)

[2.1.1 Components 20](#_Toc494124344)

[2.1.2 Interfaces 22](#_Toc494124345)

[2.2 Data and Data Flow 22](#_Toc494124346)

[2.2.1 Initial Migration 23](#_Toc494124347)

[3 Supporting Infrastructure 25](#_Toc494124348)

[3.1 Amazon Web Services Resources 25](#_Toc494124349)

[3.1.1 Simple Storage Service 25](#_Toc494124350)

[3.1.2 Simple Queueing Service 25](#_Toc494124351)

[3.1.3 Route 53 25](#_Toc494124352)

[3.1.4 Identity and Access Management 25](#_Toc494124353)

[3.2 Automation 25](#_Toc494124354)

[3.2.1 Orchestration System 25](#_Toc494124355)

[4 System Components 26](#_Toc494124356)

[4.1 Ingest Component 26](#_Toc494124357)

[4.1.1 Physical Design Diagram 26](#_Toc494124358)

[4.1.2 Infrastructure and Resources 27](#_Toc494124359)

[4.1.3 Design Aspects and Technical Considerations 27](#_Toc494124360)

[4.2 Unified Data Store 28](#_Toc494124361)

[4.2.1 Physical Design Diagram 28](#_Toc494124362)

[4.2.2 Infrastructure and Resources 29](#_Toc494124363)

[4.2.3 Storage 29](#_Toc494124364)

[4.2.4 Design Aspects and Technical Considerations 30](#_Toc494124365)

[4.3 Unified Search Index 31](#_Toc494124366)

[4.3.1 Physical Design Diagram 31](#_Toc494124367)

[4.3.2 Infrastructure and Resources 32](#_Toc494124368)

[4.3.3 Storage 32](#_Toc494124369)

[4.3.4 Design Aspects and Technical Considerations 33](#_Toc494124370)

[4.4 Description and Authority Services Application Programming Interface 34](#_Toc494124371)

[4.4.1 Infrastructure and Resources 34](#_Toc494124372)

[4.4.2 Design Aspects and Technical Considerations 34](#_Toc494124373)

[4.5 Unified Cache Store 35](#_Toc494124374)

[4.5.1 Physical Design Diagram 35](#_Toc494124375)

[4.5.2 Infrastructure and Resources 35](#_Toc494124376)

[4.5.3 Design Aspects and Technical Considerations 35](#_Toc494124377)

[4.6 Reference Application 35](#_Toc494124378)

[4.6.1 Functional Component Diagram 36](#_Toc494124379)

[4.6.2 Infrastructure and Resources 36](#_Toc494124380)

[4.6.3 Functional Interfaces 36](#_Toc494124381)

[4.6.4 Design Aspects and Technical Considerations 36](#_Toc494124382)

List of Figures

[Figure 1: Existing DAS Components and their Interactions 10](#_Toc494124321)

[Figure 2: Existing NAC Components and their Interactions 11](#_Toc494124322)

[Figure 3: Future DAS System Components and their Interactions 16](#_Toc494124323)

[Figure 4: Future NAC System Components and their Interactions 18](#_Toc494124324)

[Figure 5: DAS/NAC Architectural Diagram 20](#_Toc494124325)

[Figure 6: Ingest Component Diagram 26](#_Toc494124326)

[Figure 7: Amazon RDS Aurora DB Cluster 29](#_Toc494124327)

[Figure 8: DAS/NAC Physical Design 32](#_Toc494124328)

List of Tables

[Table 1: DAS API Component Interfaces 22](#_Toc494124310)

[Table 2: Amazon Aurora PostgreSQL Performance by the Numbers 30](#_Toc494124311)

**Version Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author/Reviewer** | **Summary Description** |
| 0.9 | 9/25/2017 | John Henson | Preliminary Version |
| 1.0 | 9/25/2017 | Aurora Rischak | QC review |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Document Scope

This document describes the major and minor components of the Description and Authority Services (DAS) and National Archives Catalog (NAC) modernization architecture, their relationships, and the data within the context of the National Archives and Records Administration (NARA) DAS/NAC modernization effort. For purposes of context and illustration, certain low-level elements may be referenced but should not be considered definitive. For low-level implementation details of these components, refer to the NARA Modernization Software Design Document (SDD).

Furthermore, this is a preliminary document that represents the tentative state of the architecture as agreed upon by Project Performance Corporation (PPC) engineers and NARA systems engineering leadership at the time of writing. Certain sections are left incomplete, and in those cases PPC has provided as much information as possible with the expectation that those topics will be further developed in later releases of this document. Details including auto scaling policies and CloudWatch alarm configuration may not be determined now.

## Background and Motivation

The current DAS and NAC systems are isolated systems that share the same information. Since they are isolated from each other, the information that originates from DAS must be transferred on a regular basis to NAC. The transfer process is lengthy and cumbersome. DAS exports can take many hours to generate and tie up system resources that would otherwise be used to perform DAS-specific operations. NAC ingestion of that data is error-prone, has in the past taken days if not weeks to complete, and relies on third-party proprietary software and libraries. With the growing size of the data and the expected acceleration of that growth, the process can’t continue to support the NARA’s mission of providing public access to archival holdings.

The primary purpose of this effort is to develop a design that achieves two major goals: cannot create a single location for DAS Description and Authority data, and Consolidate the underlying search technology. The first goal eliminates the weekly data migration activity and reduces the time it takes for a record to become publicly accessible. The second goal greatly improves the search functionality for both DAS and NAC in terms of performance and reliability.

[some words about how not doing the modernization will affect users.]

### Description and Authority Services Background

The NARA DAS is an archival repository built by PPC to replace Archival Research Catalog (ARC). DAS provides data-entry capabilities to NARA’s staff, and this data is made available to the public via the NAC system. DAS allows NARA’s staff to describe and control archival holdings at appropriate descriptive levels for all media, as well as to create and maintain Authority records that provide access points to the Description data. DAS has about 600 users (as of September 2017,) throughout the United States. As DAS is responsible for the collection of metadata related to the publicly accessible information about NARA’s holdings, this data must go through a rigorous workflow process that ensures data integrity prior to becoming publicly available. The application ensures this integrity through the implementation of business rules that must be met before descriptions and authorities can be published into the approved domain. Additionally, DAS supports a comprehensive role- and group-based security model and offers users sophisticated search capabilities, workflow, record editing and global operations. From an implementation perspective, DAS is a three-tier application, hosted in the Amazon Web Services (AWS) Cloud, with a .NET front-end, Java middle tier and an Oracle data store.

The primary business driver for this modernization effort is for PPC to build a system that can keep up with the ever-increasing volume of archival Descriptions. As of September 2017, we have almost 19 million Descriptions in DAS stored as structured Extensible Markup Language (XML) and 37 million links to digitized materials, such as:

* American Standard Code for Information Interchange (ASCII) text
* Image formats, e.g. JPEG, Tagged Image file (TIF), Graphics Interchange Format (GIF), Portable Network Graphics (PNG), bitmap (BMP)
* Audio/visual formats, e.g., Audio Video Interleave (AVI), MOV, Moving Pictures Export Group (MPEG)-4 (MP4), and Windows Media Video (WMV)
* Microsoft (MS) Office formats such as Excel, PowerPoint, and Word
* Sound formats including MP3 and Waveform Audio File Format (WAV)
* Adobe Portable Document Format (PDF) format
* Hypertext Markup Language (HTML) pages

The application must be able to scale (without underlay impacting system performance) to support billions of archival descriptions that are needed to describe billions of records generated by the federal government each year. It also needs to be reliable and consistently available during normal NARA work hours.

### National Archives Catalog Background

NAC, originally called Online Public Access (OPA), is the online public portal to NARA’s records. NAC is key to maintaining NARA’s commitment to open government through public access to high-value records. Public access to government records strengthens democracy by allowing Americans to claim their rights of citizenship, hold their government accountable, and understand their history so that they can participate more effectively in their government. NARA has stated “We will reach beyond the traditional role of making records available for others to discover and will make access happen by providing flexible tools and accessible resources that promote public participation.” NAC is the end-product of this goal to make all records available to the public in digital form to ensure that anyone can explore, discover, and learn from NARA holdings.

The catalog currently provides access to records created or received by organizations of the executive, legislative and judicial branches of the Federal Government. Some examples of frequently requested records available to public through NAC include documents about the Trans-Mississippi West (1804-1912), Holocaust-Era assets, records about the US House of Representatives, 1789-1989 and US Senate, 1789-1989. Most of the records are textual, coming from DAS, but NAC also provides access to other forms of media such as microfilm, maps and charts, architectural and engineering plans, aerial photographs, motion picture reels, video recordings, sound recordings, computer data sets, still pictures and others totaling almost 19 million electronic records (as of September 2017) currently not available elsewhere online. Besides content from DAS, NAC searches all web pages on archives.gov, blogs.archives.org, nara.gov, ofr.gov, presidentialtimeline.org, and many others and presents those pages in the search results along with any catalog records, thus enabling researchers to work more efficiently.

Apart from searching the content via their website, NAC allows the public to download this content, as well as access the same programmatically via application programming interfaces (APIs). Furthermore, NAC allows citizen archivists to transcribe some of the millions of digitized pages of records, thus improving search relevancy and accessibility to historical federal records. Additionally, researchers and the public alike can tag, translate and comment on records, thus allowing NARA to add contributions from external sources to its own repository. The transformation of NAC into a social catalog enables ongoing contributions and scalability of crowdsourcing to all records within the catalog.

### System Components in Current NAC and DAS Applications

#### Existing DAS System Components

The current DAS application as shown in Figure 1 has 3 distinct interfaces for interacting with the common Oracle database:

* A desktop based user interface (UI) leveraging the capabilities of the Windows Presentation Foundation (WPF) client platform
* A web portal for batch ingestion
* A REprepresentational State Transfer (REST) API interface for partner agencies and other applications at NARA to interact with DAS.

Each of the interfaces has its own corresponding application tier. For instance, the desktop UI interacts with the service tier using JBoss Enterprise Service Bus (ESB). Essentially, the Java based service tier hosts a set of SOAP web 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. The Presentation tier communicates with the Service 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, implemented as a Java Message Service (JMS)-compliant queue using the open-source Apache ActiveMQ product.

The web portal for batch ingestion is supported by Windows scheduler services that splits large input files into smaller chunks and processes each Description before saving it in Oracle. On the other hand, for the .NET based Web API interface, the request type and the size of the result set decides whether the response would be generated synchronously or the request would be proxied over to a background Windows service that processes the request and posts the result set to the consumer endpoint.

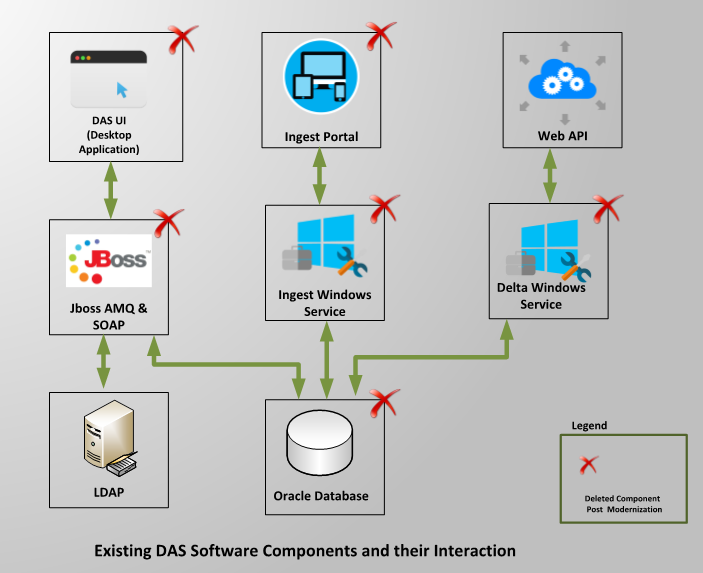


Figure : Existing DAS Components and their Interactions

#### Existing NAC System Components

NAC has the following primary components (see Figure 2):

* The Content Processing Service (CPS) module is responsible for receiving new documents from DAS, modifying them as necessary and finally publishing them to the search and access sub-system for eventual dissemination to the public. This module is implemented using the Aspire framework and all the content generated in each step is stored in Amazon Simple Storage Service (S3) buckets.
* The NAC search engine consumes output from the content processing module (indexing) and responds to requests from the application server (searching). The search engine is implemented in Apache Solr Cloud for distributed indexing and searching support, thus allowing the NAC search engine to be highly scalable and fault-tolerant.
* OPA Storage is the long-term content storage for all publicly available NARA data and is implemented using AWS S3 buckets. It contains digital objects, alternative renditions of digital objects (thumbnails, image tiles, alternative resolutions), sub-documents, tables, and record-structured data.
* Annotations and Registration Database, implemented in MySQL, contains tags, comments and transcriptions as well as bookkeeping information for all annotations. It also holds information on all registered users, including profile information and encrypted passwords.
* NAC Application Services, hosted on the NAC Application Server, is responsible for handling all end-user requests for search and access, and authorized user requests for creating and deleting annotations.
* Catalog Browser and APIs: This refers to the public website, written in JavaScript and HTML5, where end-users can search, access and download records. The documents can also be accessed programmatically via search and access APIs.

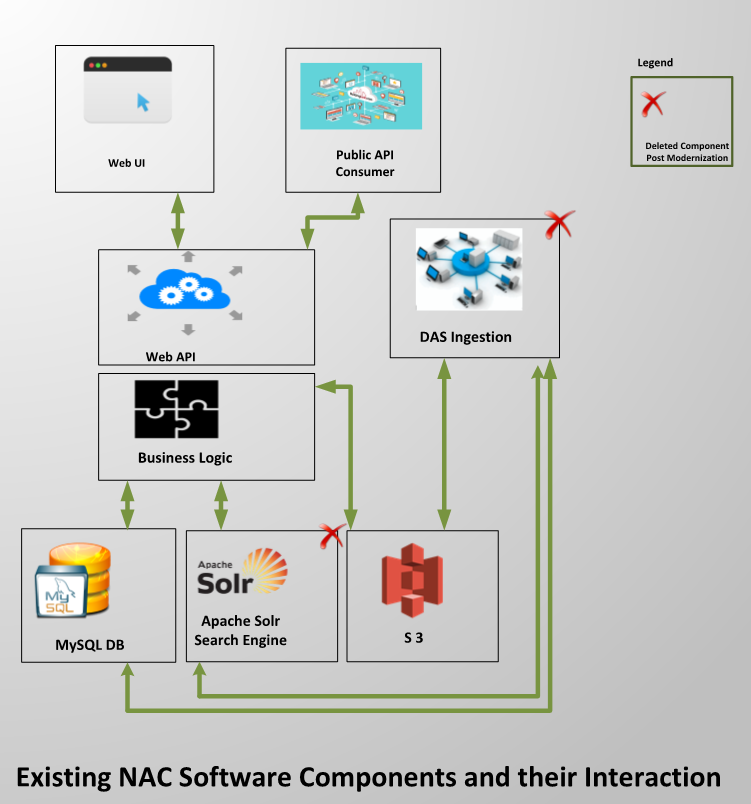


Figure : Existing NAC Components and their Interactions

### Problems with the Existing Setup in DAS and NAC

This section describes the problems with the existing Architecture.

#### Scalability Limitations of DAS

DAS was originally designed with an emphasis on implementing a flexible schema. That way, new properties can be easily added to a Description or an Authority type without having to make a cascading set of changes on dependent entities. Consequently, all links, independent of whether they are parent links or Authorities for a Description, are persisted in the same table in the database, along with its position information in the hierarchical XML structure in the Description document. As of September 2017, with 19 million descriptions, the link table has 251 million records. To prevent degradation of performance of description persistence and retrievals, we must continuously partition the related tables. With the DAS Ingest module in place since summer of 2015, the size of the link table has tripled. For instance, during the month of May in 2017, 1.5 million descriptions with 3.7 million digital objects were created in DAS. NARA is looking to ingest 6 billion descriptions and 25 billion digital objects by the end of this decade. With the current architecture, that would imply 330 billion records in the link table, and performance of basic saves and gets for a description entity is bound to suffer.

Besides using Oracle as a data store, the XML DB features of Oracle, including support for XQuery, Structured Query Language (SQL)/XML and Extensible Stylesheet Language Transformation (XSLT), are used to serve keyword- and field-based searches in DAS. Oracle enables special indexing on XML, including Oracle Text indexes, to accelerate such searches. Using text indexes speeds up searches when compared to performing string matches in a character large object (CLOB) column. However, it cannot compete with the performance of a standard B-Tree column index. Additionally, in Oracle 12c, such text indexes were converted into two-level indexes under the feature name “near-real-time indexes.” This allows the database to have a small, fragmented index containing all the recent updates to the table, that fits comfortably in memory without having to modify the large “standard” index. Unfortunately, this technique is not at all suitable for the Description table with 19 million records as of September 2017 that see numerous updates via global operations or inserts via ingest each day. Moreover, Oracle 12c introduced a new job DR$BGOptJob which runs every 2 minutes and moves index data from “in memory” $G table to “on disk” $I table. However, when records are added to $G faster than they are removed, DR$BGOptJob runs continuously, taking away valuable central processing units (CPUs) and memory resources from other database operations, such as complex searches and bulk inserts and updates. To get around this issue, today we manually rebuild the index on the Description table during all maintenance windows. But as the Description table size grows, so does the size of the context index and the time to fully recreate the index. Currently, it takes almost 9 hours to drop and recreate such an index that keeps track of all the metadata fields in a Description XML.

#### Limitations of Using Oracle as the DAS Database

DAS runs on an Oracle 12c database (version 12.1.0.1.0) hosted in the AWS cloud. DAS uses High Storage (HS)1 8xlarge Elastic Compute (EC2) instances for the database server. Para-virtual (PV) Amazon Machine Images (AMIs) are used with Oracle Enterprise Linux (OEL) 6.7 for the operating system. For the Logical Volume Manager (LVM), DAS uses Oracle Automatic Storage Management (ASM) to maximize the input/output (I/O) performance via disk striping. HS1 8x large instances have 24x2 terabyte (TB) local solid state drives (SSDs), which are used for Oracle ASM with external redundancy for maximizing the database read/write performance. There is an identical instance established as the standby production database with a Data Guard broker to keep the data synchronized between the primary and secondary databases. The database instances cannot be stopped and restarted as NARA data lives in the associated ephemeral storage. In fact, as the instance grows older, the likelihood of disk failures increases. During the summer of 2015, two ASM disks failed in the primary production database. DAS had to be switched over to use the standby database for the primary production database. Then ASM was rebuilt in the old primary database (with normal redundancy using 22 out of the 24 available disks) and converted to be the new standby database. In early 2016, a similar disk failure in the user acceptance testing (UAT) database prevented the scheduling of daily Recovery Manager (RMAN) backups in the instance. Using a RAID 0 configuration with Elastic Block Storage (EBS) Provisioned Input/Output Operations Per Second (PIOPS) volumes with at least 10,000 IOPS, instead of ephemeral storage as the ASM, is possible but cost prohibitive. For instance, with even 10 such volumes, the monthly AWS cost would be around 9,000 U.S. dollars (USD). This does not include the monthly licensing fees that NARA pays to obtain enterprise support from Oracle.

#### Loss of Enterprise Support for the JBoss SOA-P Platform

The middle tier in DAS is implemented using JBoss Service-Oriented Architecture Platform (SOA-P) 5.3.1, platform which lost maintenance support from RedHat on 11/30/2016 and will lose extended life support on 05/31/2018. Hence, the current application must be upgraded to either more recent versions of the products used today, or migrated to a new architecture. The components of SOA-P 5.3.1 that DAS primarily uses include JBoss ESB 4.11.1, jBPM 3.2.14, Drools 5.3.1 and Enterprise Application Platform (EAP) 5.2. The replacement products for JBoss ESB is Fuse 6.2.1 which includes Apache ActiveMQ, Apache Camel, Apache CXF. For jBPM 3, it is BPMS 6.2, which includes Drools 6.3 and jBPM 6.3. EAP 5.2 must be upgraded EAP 6 to serve as a container for Fuse, business process management software (BPMS) and other web services that DAS currently has but are not affected by the change. With the modernization effort, however, we intend to move away from a service bus based architecture to a REST API based architecture that can be uniformly consumed by DAS UI as well as other applications at NARA and their partners and only use RedHat BPMS module for implementing DAS workflows.

#### Limitations of Desktop User Interface in DAS

DAS UI is a desktop application based on WPF, part of Microsoft .NET Framework 4.0. It provides a rich interactive user experience, leveraging the capabilities of the WPF client platform. However, it requires computers to be located within the NARA network to access the DAS System directly over the NARA local area network (LAN). For external computers (e.g., users working from home), users need to first authenticate and gain access to the NARA remote access Citrix server before they can access the DAS System. Additionally, installation of the .NET 4.0 Framework (full version) is required to operate the DAS UI. For each monthly release of DAS, separate builds are created for Citrix and non-Citrix users. However, much coordination is needed with NARA information technology (IT) to install the latest build of DAS UI to the Citrix servers which, in turn, means DAS Citrix users cannot benefit from the new feature releases or defect fixes in DAS as early as the DAS users who have access to NARA desktops.

Furthermore, third party libraries such as DevExpress Windows Presentation Foundation (WPF) Control Libraries have been used in DAS UI to implement most of the UI controls. Besides prohibitive licensing fees, some features cannot be implemented in DAS UI because of its use of DevExpress libraries. One such example is the focus issue in DAS UI. After populating one or more fields in the UI, the Save button is not enabled unless the user clicks somewhere in the UI.

#### Heterogenous Modules in DAS Middle Tier

Prior to the deployment of .NET based DAS Ingest module to the production environment in the summer of 2015, bulk ingestion would use the existing ESB/SOA based middle tier. However, because of the index related issues mentioned in section 1.2.4.1, whenever such an import job would kick off in the background, index syncing would utilize enough CPU cycles to noticeably degrade usability of DAS. Additionally, the original bulk ingestion module in DAS was Oracle based and single-threaded and, hence, limited in the number of Descriptions that can be ingested at a time. What the .NET Ingest module did was to use multi-threading to massively parallelize the ingestion process. Additionally, it stores the processed descriptions in staging tables and only migrated them to the primary tables at nighttime when the CPU load on DAS is minimal, thus bypassing the need for the index to continuously synchronize and burn CPU cycles.

#### Duplicate Data Store for DAS and NAC

DAS is the system of record at NARA. Hence, every week NAC ingests all new and updated Authorities and Descriptions created in DAS and pushes them to the search engine so that they can be searched and accessed by the public. After ingestion, the Description XML in NAC looks identical to the one in DAS. Citizen archivists can, however, create annotations such as tags, transcriptions and comments for the same archival descriptions ingested from DAS in NAC. Consequently, to save costs and reduce data duplication, it makes sense for the NAC application tier to use DAS APIs to retrieve Description, Object and Authority metadata from DAS and associate them with their corresponding tags, transcriptions and comments from the annotations database in the catalog.

#### Problems with the Solr-Based Search Engine in NAC

The search engine in NAC uses Search Technologies’ proprietary query parsing and processing framework called Query Processing Language (QPL). This framework is a Solr query parser plug-in that is added to the standard Solr installation. Per Search Technologies, the original contractor for NAC, there are several basic query functions which are not supported by the out-of-the-box Solr query parsers such as XML Searching such as use of “between” operator to search between XML tags in the archival descriptions, Authority records, and technical and content metadata or range searches within XML metadata. This was, however, addressed as part of Task Order 4 (TO4) for the catalog that recently went into production. After TO4 development, catalog users can also perform exact-match text searches, field exists and not exists searches. To allow such searches, new fields were configured in Solr using a special Solr Tokenizer that not only indexes the value of the leaf nodes but also the paths to those leaf nodes.

#### Limitations of the NAC Ingestion Module

Every week, NAC ingests all new and updated authorities and descriptions created in DAS and pushes them to the search engine so that they can be searched and accessed by the public. The content processing system is based on Search Technologies’ proprietary Aspire Content Processing system “data flow pipelines,” which takes series of re-usable components and configures them and connects them together into sequential pipelines which process the data, step-by-step. Just as with Oracle in DAS, use of Aspire Content Processing libraries in NAC require hefty licensing fees to be paid every year.

## Future Components Overview

This section discusses the overall technical approach and high-level components for the combined NAC/DAS systems, and provides some context for the detailed DAS REST API Framework design, data model and database design, search schema and engine design, and the bulk ingestion module design that are described in the SDD.

### DAS System Components: Design, Implementation and Integration

A high-level view of the future DAS system components and their interaction is presented in Figure 3; it summarizes the design vision for the modernized DAS application.

The modernized DAS application is envisioned as a three-tier system consisting of the Presentation, Service, and Data tiers:

* Presentation Tier—The presentation tier software will be written in JavaScript and HTML-5 and will run inside the user’s web browser. It will handle the following:
* Page presentation for user work trays, searches and editing descriptions and authorities
* All Cascading Style Sheets (CSS) for all styles
* All clicks and control flow from page to page
* All control interactivity which does not require a full-page refresh (for example, page previews, fly-outs, tabs, etc.)
* Service Tier—This tier will be implemented using REST API framework and will handle API requests from end-users, external applications and batch processing services. The service tier will have sub-systems for:
* Handling all API requests for end-users and consumer applications
* Managing user work trays and description and authority workflows
* Managing create, read, update, and delete (CRUD) operations on descriptions and authorities, using Amazon Aurora as the managed database service
* Sending searches to AWS Elasticsearch service and formatting responses
* Managing DAS users, groups and roles using NARA Lightweight Directory Access Protocol (LDAP)
* Batch processing descriptions and authorities
* Global operations on descriptions and authorities
* Processing bulk deletes and approvals on descriptions and authorities
* Bulk exports
* Ad hoc and canned reports for descriptions, digital objects, authorities and proposals
* Data Tier—The data tier comprises of five distinct components:
* PostgreSQL-compatible Amazon Aurora to store descriptions and its related annotations, authorities and user workflows
* Amazon Elasticsearch Service to handle basic and advanced searches on under-edit and approved descriptions, authorities and user work tray items
* Amazon Simple Storage Service (S3) as a temporary store for input description and authority files meant for batch processing
* Amazon ElastiCache to allow fast retrieval of frequently accessed descriptions and authorities
* NARA LDAP to store NARA users, groups and roles

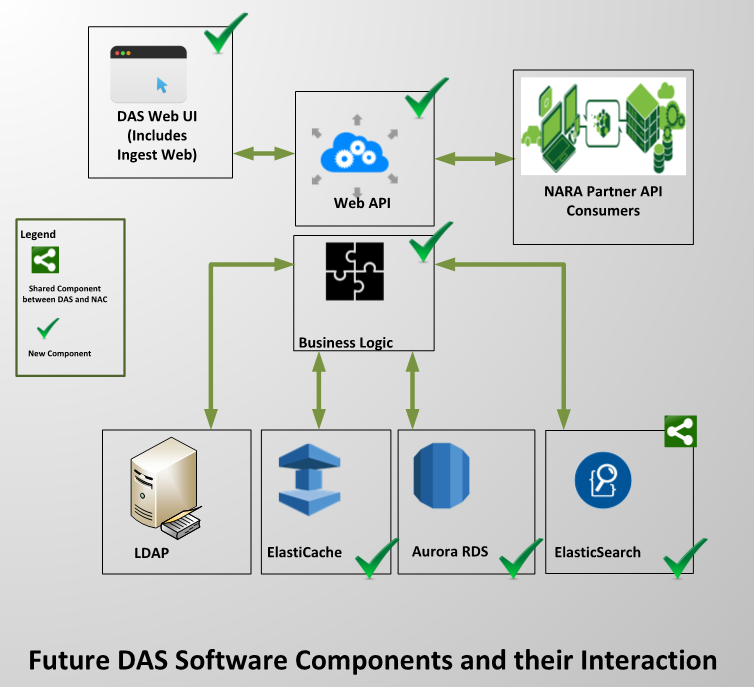


Figure 3: Future DAS System Components and their Interactions

### NAC System Components: Design, Implementation and Integration

A high-level view of the future NAC components is presented in Figure 4; it summarizes the design vision for the modernized NAC system. The modernized NAC system is envisioned to replace or significantly update several of the modules mentioned in the previous section. For instance, the Content Processing module will be eliminated completely, as there will no need for a weekly export from DAS to be processed.

Instead, NAC will use the DAS Description and Authority web services to access Authority and Description records that were previously stored and accessed from the Apache Solr search index. The services that currently crawl the presidential web pages for new content or periodically check the annotations database for additions, deletions and updates will be refactored and bundled with other scheduler tasks in NAC. Similarly, the Apache Solr based search engine will be replaced with Amazon Elasticsearch Service and NAC Application Services will be refactored to communicate with Amazon Elasticsearch services instead of self-managed Solr Cloud servers. DAS-specific contents such as description and authorities and NAC-specific contents such as tags, comments and transcriptions will be stored in separate search indexes. For search requests that require output from both, the NAC application services would merge the result set and send it back to the end-user. Finally, the content of OPA storage would reduce dramatically because only digital object related content such as the original image and its alternative renditions (thumbnails, image tiles, alternative resolutions) that are created by a Lambda function upon upload will be stored in S3 going forward.

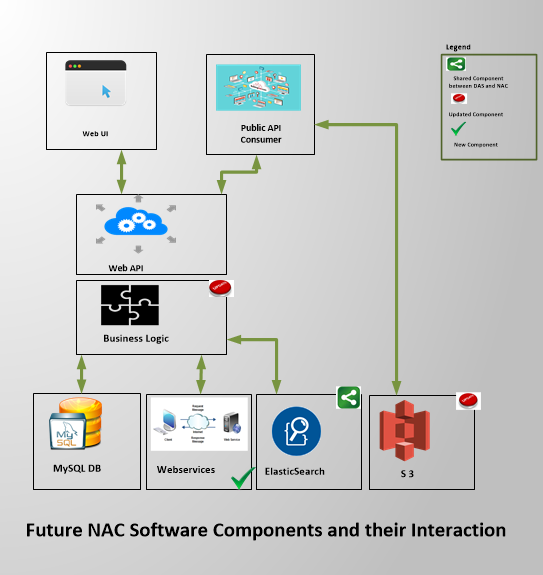


Figure 4: Future NAC System Components and their Interactions

## Assumptions

The logical DAS data structure between the two DAS and NAC systems is presumed to be consistent between the current existing production and modernization architectures.

# High-Level Modernization Architecture

Like the current systems, all the server-side resources are provisioned in the AWS cloud infrastructure. The only component of the system not running in the AWS cloud is the DAS desktop client application that runs on NARA workstations and in Citrix desktop images.

The system leverages AWS managed services to provide a scalable and fault-tolerant architecture. Additionally, the design takes advantage of the zonally distributed design of the AWS infrastructure, minimizing downtime and providing high availability.

Both DAS and NAC share common storage and search components, which are implemented using AWS Relational Database Service (RDS) Aurora PostgreSQL, and AWS Elasticsearch Service respectively. The advantages to using these services is that AWS takes care of the low-level concerns of the systems, and provides routine backups at regularly scheduled intervals. The work required to perform these essential tasks is built into these services.

An AWS Virtual Private Cloud (VPC) contains the Elastic Compute Cloud (EC2) and RDS instances that host the major components of the modernization architecture. A bastion host, or Jump Host, and a Web Host, are the only instances accessible from the Internet, and are further protected by limiting the associated Security Group to NARA and vendor-specific network Internet Protocol (IP) address ranges. Access to other instances within the VPC is only available through the Jump Host. AWS ElasticSearch instances are provisioned in a service-controlled VPC that is managed by AWS.

Using auto scaling groups, the major components can automatically instantiate additional compute nodes to increase processing throughput based on data volume as indicated by request load or queue length. The Chunking, and Processing Service module-specific auto scaling groups adjust based on the length of the Job Queue, and Single Description Queue, respectively. The DAS application programming interface (API) component auto scaling group is set behind an Elastic Load Balancer that distributes requests across the set of instances, and adjusts the size of the group by monitoring request load. The Ingestion Component modules do not have listening interfaces, so they do not require an Elastic Load Balancer. Additionally, the auto scaling groups are configured to ensure that instances are distributed across multiple availability zones (Multi-AZ) for redundancy.

## High-Level Architectural Diagram

The preliminary modernization architecture leverages multiple availability zones and data replication to provide high-availability. Auto scaling groups at multiple levels allow the system to scale as data volume changes while also keeping costs low. See Figure 5.

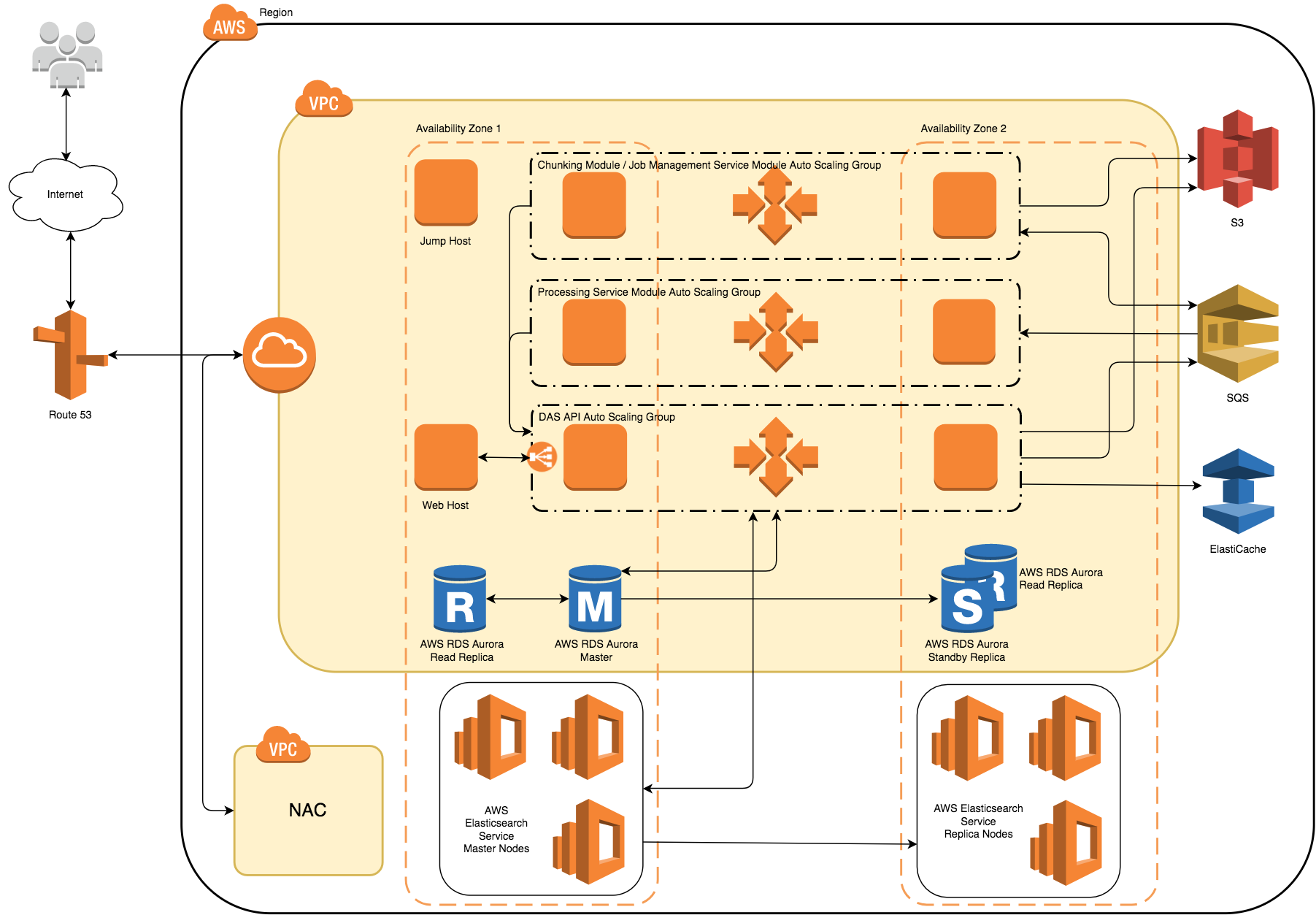


Figure 5: DAS/NAC Architectural Diagram

### Components

The major components of the planned modernization architecture are detailed in the following subsections.

#### Ingestion Component

DAS data submitted in bulk is processed by the Ingestion Component, which is comprised of two primary modules. A distributed, queue-based data processing approach permits virtually limitless scalability, ensuring that NARA meets its increasing data and performance demands. Each EC2 instance in the module operates independently on discrete, independent datasets. The Ingest Component’s three modules are responsible for the handling of bulk Description Ingest: the Chunking module, Job Management Service module, and the Processing Service module. The Job Management Service module software runs alongside the Chunking module software in each node in the auto scaling group. It is a utility module designed to maintain the status of jobs as they move through the workflow.

##### Chunking Module

In this module, large XML files containing the XML representation of many Description records are broken out to be processed individually. Chunking module servers independently read messages from the Job Queue as they become available. Messages on the queue contain the location of the bulk XML file. Individual DAS Description XML records in the bulk XML files are placed on Single Description Queue for individual record processing. The Chunking module is horizontally scalable on demand, providing a proportionate increase in overall processing throughput.

##### Job Management Service Module

The Job Management Service module performs basic housekeeping tasks relating to the progress of jobs and their artifacts. As a job moves throughout the system, the Job Management Service makes calls to the API to update the status of the corresponding database records. The Job Management Service also provides visibility into job status allowing the end-user to query the API and track updates to a job.

##### Processing Service Module

The Processing module servers pull messages from a Simple Queue Service (SQS) queue and submit them to the DAS API for insertion into the AWS RDS database for persistent storage, and for indexing in the search index.

#### DAS API

The DAS API component is the centerpiece of the application tier of the modernization architecture. All operations requiring read and write access to the Unified Data Store and Search Index are proxied through the DAS API’s RESTful interface. The DAS API is horizontally scalable as demand increases.

#### Primary Data Store

AWS RDS Aurora (PostgreSQL) has been chosen for its optimized performance in AWS, and simplified scalability and redundancy features. The primary data store contains full Description and Authority record information as well as any DAS-related housekeeping data.

#### Search Index

The AWS ElasticSearch service is designed to index all Description, Authority, and other DAS- and NAC-related metadata and information. The index is separated by data type and disposition to mitigate the risk of having restricted and unrestricted information stored together.

#### Cache

For global-level record caching across components and within auto scaling groups, AWS ElastiCache will store frequently accessed records for improved retrieval performance and decreased load on the write-heavy database. Use of the cache is coordinated by the DAS API between both the consolidated data store and search index.

### Interfaces

While the Software Design Document will explain these interfaces in greater detail, this section provides a brief rundown of the primary connections between the major components and AWS services. The most common protocol implemented between components is Hypertext Transfer Protocol (HTTP) and Hypertext Transfer Protocol Secure (HTTPS). The DAS API exposes a REST interface used by all other components to perform most operations. The DAS API, a Java application, uses the Java Database Connectivity (JDBC) API and drivers to connect natively to the AWS RDS database at the prescribed endpoint provided by AWS. See Table 1 for a depiction of which components interface with which protocols.

| Table : DAS API Component Interfaces | | |  |
| --- | --- | --- | --- |
| **Component** | **Interfaces With** | **Protocol** | **Port** |
| Chunking Module | AWS SQS | HTTP/S | 443 |
| DAS API | HTTP/S | 443 |
| AWS S3 | HTTP/S | 443 |
| Processing Service Module | SQS | HTTP/S | 443 |
| DAS API | HTTP/S | 443 |
| Job Management Service Module | DAS API | HTTP/S | 443 |
| DAS API | AWS S3 | HTTP/S | 443 |
| AWS SQS | HTTP/S | 443 |
| AWS Elasticsearch | HTTP/S | 443 |
| AWS RDS | TCP | 5432 |
| AWS ElastiCache | HTTP/S | 443 |
| Reference Application | DAS API | HTTP/S | 443 |

An Internet Gateway provides a connection between the Internet and the system VPC. For security, route tables are configured to limit access to the VPC to NARA- and vendor-specific IP address ranges. Additionally, any subnets within the VPC will have Internet access.

## Data and Data Flow

System data originates from three different sources: NARA DAS Client users, DAS Bulk Ingest, and NAC Annotations from NARA staff and public users. While there may be cases where information is ingested from other sources, those situations are handled on an ad hoc basis.

The dataset is comprised mostly of hierarchical Description and Authority records, generated and updated by NARA staff in either an individual or bulk fashion. Individual Description and Authority records are manipulated through the DAS client, a Windows .NET desktop application which interreacts with the DAS API. Bulk import operations are performed through a web-based application that accepts an XML-formatted document containing the definitions of many Description records. The bulk import application coordinates the processing of this data and kicks off the chain of events that result in the records becoming available in the DAS/NAC consolidated data store and being indexed by the consolidated search index. Catalog annotations are submitted through the NAC web UI and NAC Public API, stored in the NAC annotations MySQL database, and indexed in a consolidated search index.

The normal representation of the DAS records is a well-defined XML format, and it is assumed that this format will be maintained as the official format throughout and beyond the modernization effort. This is an important aspect because the NAC public API output and UI conform to the existing data format, and it’s expected that these features would remain the same throughout the modernization effort.

Depending on the type and origin, data flowing through the system can take one or more paths, and be transformed in several ways during the process. The primary data processing pipeline is comprised of independent discrete processing units that scale horizontally to proportionately accommodate a desired throughput. Description data is subsequently broken into its constituent parts and transformed into SQL queries to add to or update the consolidated data store.

Consumers of the system include NARA employees that use the DAS client to manipulate Description and Authority records, NARA employees and archivists that use the Catalog UI to annotate records, non-NARA research organizations and institutions like universities and libraries that use the Catalog Public API to retrieve large sets of raw record data, and the public, who uses the Catalog API and web-based interface to search the publicly available holdings.

Before normal system operations can begin, a one-time migration must be performed. The duration of the migration activity is dependent on the size of the dataset being migrated.

Bulk Ingest operations are initiated through the DAS Ingest Web interface. The Ingest Web Interface first uploads the bulk XML file to Amazon S3. Next, Ingest Web Interface calls the DAS API Ingest endpoint with a link to the file in S3. The DAS API subsequently submits a corresponding message to the Job Queue. This message is read by the Chunking module, which retrieves the XML file from S3 for distillation into individual DAS Description and Authority XML records. These records are placed onto a second SQS “Single Description Queue” which is read by the Processing Service module. The Single Description Queue is configured to be a First In, First Out (FIFO) queue to maintain order of entry. The Chunking and Processing Service modules continually poll these queues and process data as it arrives.

### Initial Migration

Existing DAS Descriptions and Authorities reside in an Oracle database supported by a proprietary Oracle XML datatype. The easiest way to migrate the data into the new system is to first export this information into an already developed XML format. The processes for performing this step are mature and in use operationally.

A custom extract, transform, and load (ETL) process will be implemented to transform the exported XML data into the necessary SQL statements for insertion into the new Unified Data Store (in this case, AWS RDS Aurora PostgreSQL) Because DAS Description records depend heavily on DAS Authority records, the Authority records are loaded separately into the Unified Data Store prior to loading the Description records.

# Supporting Infrastructure

## Amazon Web Services Resources

### Simple Storage Service

Not including the digital objects and related artifacts, all the bulk XML data processed through the Ingestion Component workflow will reside in Amazon S3. S3 is a high durability, high availability storage solution.

### Simple Queueing Service

AWS SQS is a fully-managed, reliable scalable queuing service. The new architecture leverages the AWS SQS to support a decoupled, and fault-tolerant data processing workflow.

### Route 53

AWS Route 53 is a highly available and scalable Domain Name System (DNS) service. Where necessary, host endpoints are mapped to Domain Name Servers (DNS) entries maintained in Amazon’s DNS service. This service enables load balancing across changing numbers of hosts in an auto scaling group. This is a particularly useful for exposed APIs, such as the DAS API in the case of the modernization architecture. It is not necessary for any consumer of this interface to know anything about the number of hosts in the auto scaling group or their individual IP addresses.

### Identity and Access Management

AWS Identity and Access Management (IAM) is used to manage administrative user accounts, set access policies and roles for EC2 instances, and other AWS Services, including AWS Elasticsearch Service and AWS RDS.

## Automation

### Orchestration System

Eventually, all resources will be described by a set of standard scripts that can be replayed to recreate most, if not all, of the infrastructure described in this document, and ideally, it would be vendor-agnostic.

# System Components

## Ingest Component

This section describes the elements responsible for taking NARA raw Description and Authority information and storing and indexing it for retrieval and searching. Bulk Description data is submitted in XML format through the Ingest Web web-based application responsible for managing all submitted files and their status. These large, multi-record XML data files are validated through the DAS API, and upon successful validation, are stored in AWS S3. Simultaneously, a message is placed on an AWS SQS queue with information about the XML file including its location in S3. Once on the queue, the Chunking module picks the message up and separates each individual DAS record into a single message which is then placed onto an SQS FIFO queue.

### Physical Design Diagram

Figure 6 displays the DAS ingest component modules and how they relate to one another.

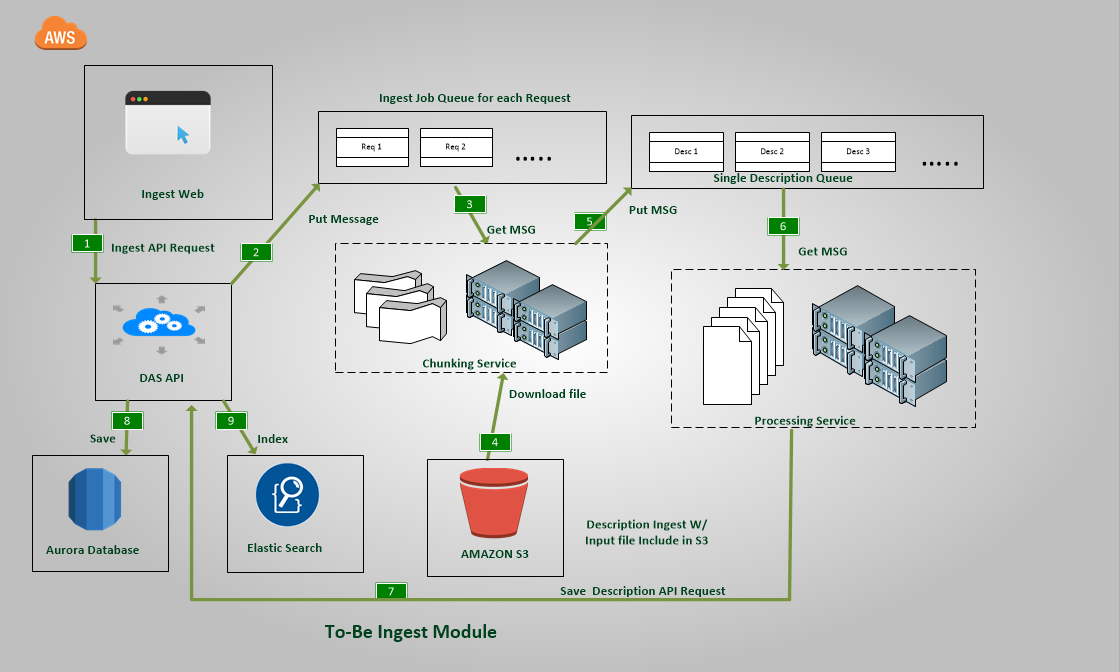


Figure 6: Ingest Component Diagram

### Infrastructure and Resources

This Ingest Component includes two discrete, automatically scalable modules, each responsible for a specific portion of the data intake and transformation process. The system uses a combination of AWS-provided computing infrastructure and managed services.

The Chunking module is comprised of an auto scaling group containing multiple EC2 instances distributed across multiple availability zones. The Chunking module dequeues messages from the Job Queue. Each message points to a bulk XML file stored in Amazon S3. This file is read from S3, and each Description record inside that file is captured in a message and sent to the Description Queue. The auto scaling group can be configured to automatically expand the number of instances in the module based on the size of the queue, and can additionally be locked at a specific size if desired. The Chunking module also hosts a Job Management Service that routinely monitors the state of current ingest jobs and updates their status in the database.

The Processing module is made up of an auto scaling group containing multiple EC2 instances distributed across multiple availability zones. The Processing module dequeues messages from the Description Queue and prepares an HTTP request that is sent to the DAS API for persistence of the DAS Description record.

Two SQS queues are provisioned to support the Ingest Component. The first queue stores messages about bulk XML files ready for the Chunking module to process. The order in which messages are delivered through this queue is not important. Because of the large size of the bulk XML files, messages sent to this queue contain location and other metadata about the bulk XML file, not the actual file data. The second queue stores the individual records pulled from the initial bulk XML records.

### Design Aspects and Technical Considerations

#### Scalability

The use of a queue-based architecture allows the ingest component to process records at rates proportional to the number of servers configured. Overall throughput is a function of the rate at which a single EC2 instance completes processing a unit of data multiplied by the number of total processing instances.

##### Vertical

It is possible to increase the per-server processing rate by increasing the amount of resources associated with a given server. Each EC2 instance has the capability to concurrently process multiple records. The number of possible concurrent processes is dependent on the number of processors and amount of memory allocated to an EC2 instance.

##### Horizontal

Horizontal scaling can be controlled by adjusting the Chunking module and Processing Service module auto scaling group configurations. Automatic and manual ad hoc scaling is available depending on need.

#### Performance

Selection of EC2 instance type will depend on the type of work for the work being performed by the Chunking and Processing modules and the desired overall processing rate, balanced with cost considerations.

#### Security

Physical access to the EC2 instances that make up the Ingest Component is limited to the VPC security group configuration. All hosts in the Chunking and Processing auto scaling groups are not publicly accessible directly from the Internet. Additionally, all host AMIs implement best practices in security configuration, are designed to have the latest recommended security and operating system patches installed, and are compliant with all relevant Center for Internet Security (CIS) benchmarks.

Access to other AWS services is granted with the use of IAM roles. Policies granting access to specific AWS resource and functions are assigned to roles, which are then applied to the appropriate set of EC2 instances in an auto scaling group.

#### Data Integrity

The Chunking module will attempt to submit each Description record it encounters to the Description Queue, but does not guarantee successful processing of each record. Therefore, it is possible for a given bulk ingest job to partially succeed. Any failed record will be identified as such and that information will be made available through a DAS API request.

#### Automatic Failover and Load Balancing

The Chunking module and Processing Service module auto scaling group configurations ensure that for each auto scaling group, any instance that fails will be replaced automatically so that the desired minimum number is always met. The Ingest module is self-load-balancing within both the Chunking module and Processing module as they dequeue messages independently.

## Unified Data Store

The primary physical storage location for all records for DAS and NAC will be stored in AWS RDS Aurora PostgreSQL. All NAC search results will be assembled from the same data used by the DAS system, ensuring consistency between the two systems, and enabling near-real-time updates to publicly-accessible data.

### Physical Design Diagram

The Multi-AZ configuration of the AWS RDS Aurora PostgreSQL cluster is illustrated below in Figure 7.

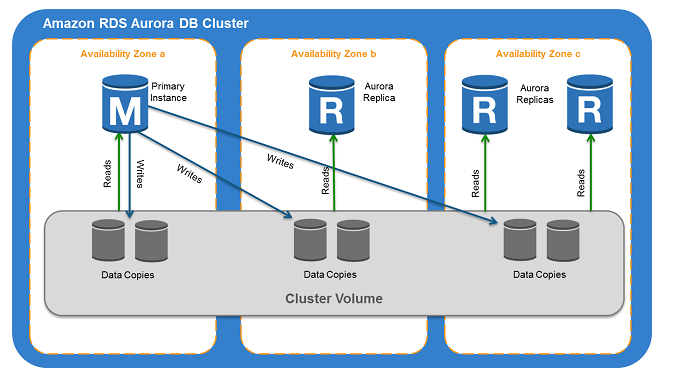


Figure 7: Amazon RDS Aurora DB Cluster

### Infrastructure and Resources

The database will be implemented in AWS RDS Aurora PostgreSQL version 9.6. At least one Read Replica will be provisioned in a different availability zone than the master.

### Storage

#### Source and Target Comparison

The legacy DAS Oracle Database has a total storage footprint of 1.8 TB. At the time of this writing, there is insufficient information available to know how large the current dataset would be in the new data model in AWS RDS Aurora PostgreSQL.

#### Projection

Storage projections will be made available once sufficient data has been collected following the initial data migration activity.

### Design Aspects and Technical Considerations

#### Scalability

##### Vertical

RDS instance types can be scaled vertically on-demand by modifying the existing configuration. Database storage and compute instances are not inherently linked, so instance types can be scaled without concern for storage capacity. Storage size and type (e.g., general purpose vs. provisioned IOPS) can be modified separately.

##### Horizontal

Horizontal scalability of the data storage tier is dependent on data model design and, eventually, implemented partitioning or sharing strategy. In the cases where read queries are performing poorly, it’s possible to create “read replicas” of the master database to improve read performance for those systems that don’t require database write access. The Catalog, for example, could use a read replica to support populating public search query results. The DAS client could similarly use read replicas to support data exports and reporting functionality. Each RDS Master database can have up to fifteen read replicas across different availability zones.

##### Sharing/Partitioning Strategy

The current limit on database size for AWS RDS Aurora is 64 terabytes (TB). When that limit is reached, the database will need to be split into two or more separate databases. In the case of the DAS and NAC modernization effort, the strategy for how this would work has been discussed but only at a high level.

#### Performance

AWS RDS Aurora PostgreSQL offers significant performance improvement compared to the standalone version as illustrated in Table 2:

Table : Amazon Aurora PostgreSQL Performance by the Numbers

|  |  |
| --- | --- |
| **Measurement** | **Result** |
| SysBench Benchmark | 2x-3x faster |
| Data Loading | 3x faster |
| Response Time | >2x faster |
| Throughput at Scale | 3x faster |
| Recovery Speed | Up to 85x faster |

#### Security

Consideration for public and non-public data must be taken into account to avoid the accidental release of potentially sensitive material. Access to the RDS instances will be limited to internal VPC subnet IP address ranges, and best practices regarding PostgreSQL account access will be implemented.

A PostgreSQL service account with appropriate schema authorizations will be created for DAS API programmatic access to the database.

#### Data Integrity

##### Data Redundancy

Data from the Master RDS instance is continuously updated on one or more RDS read replicas. In addition, a hot standby server is synchronously updated to support automatic failover.

#### Automatic Failover and Load Balancing

If the master RDS instance fails, a hot standby in a different availability zone is automatically promoted to master status and begin to serve read and write requests.

## Unified Search Index

The search index component supports the search functionality of the DAS UI, Catalog Web UI and Public API. The AWS Elasticsearch service enables high-performing search capabilities with features including high-availability, redundancy and failover, and automatic backup and restore.

The dual-purpose role of the search engine necessitates a segmented indexing strategy, where records of a type are indexed separately from others. There are seven separate indexes for the following record types: Description, Authority, Digital Object, DAS Workflow, NAC Annotation, Presidential Library Web page, and Archive.gov Web Page. The only records that will be stored in the index are the crawled Web Page records from Presidential Libraries and Archive.gov web sites.

### Physical Design Diagram

The Consolidated DAC/NAS Search Engine planned physical design is depicted in Figure 8. An AWS Elasticsearch Service replica shards are synchronously updated from corresponding master shards.

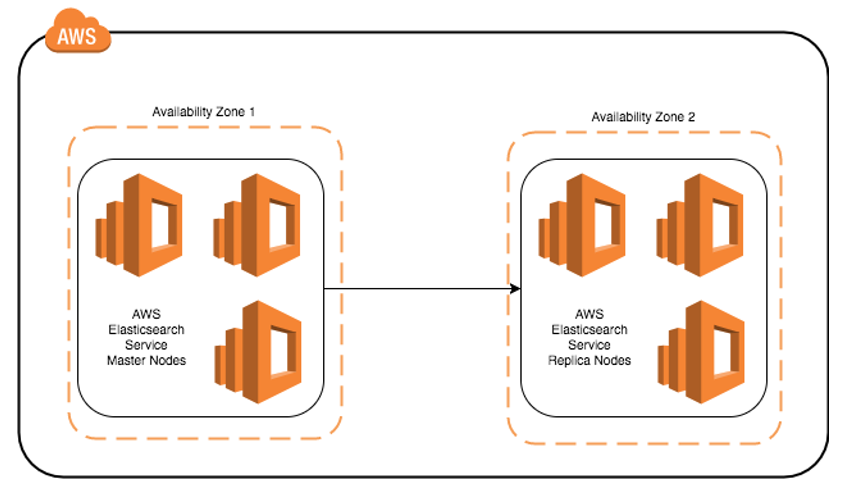


Figure 8: DAS/NAC Physical Design

### Infrastructure and Resources

The Consolidated Search Engine is implemented in the AWS Elasticsearch service running Elasticsearch version 5.5. A series of master nodes are provisioned along with a set of matching replica nodes for redundancy and failover. The nodes are configured to be distributed across multi-AZ.

### Storage

#### Source and Target Comparison

The legacy NAC Solr search engine stores each Description, metadata for each Description Digital Object, and a subset of Authority records in its search index, which on disk is about 340 GB split between two shards.

#### Projection

Compared to the current search index, it is expected that the initial size of the search index will be relatively small since it is no longer necessary to store the contents of each Description record alongside the index. Storage projections will be made as more data is ingested into the indexes.

### Design Aspects and Technical Considerations

The index will be segmented into separate indexes based on data type and intended consumer. This configuration provides clear authorization boundaries, helping to ensure that a minimal combination of public annotation information (e.g., tags, comments, etc.) and non-public information is stored together at the physical level.

#### Scalability

##### Vertical

The search cluster can be scaled vertically by changing the EC2 instance type to add more memory or CPU resources. Storage can also be increased by increasing the size of any EBS volumes.

##### Horizontal

Elasticsearch clusters can easily scale horizontally simply by adding new nodes. Shards are automatically redistributed across the nodes to distribute resource load across the cluster.

###### Sharing/Partitioning Strategy

Each index is split into multiple shards that will be spread across the cluster. This value cannot be changed once it is set, so it is important to consider the long-term consequences of this decision. As more information about the operational needs of the search engine become available, a more accurate estimation of shard count can be made.

#### Performance

AWS Elasticsearch Service offers several instance types with varying amounts of memory and CPU power. Storage type should also be considered for performance reasons. Available storage options include Instance Store, general purpose SSDs, and PIOPS SSD. Instance Store storage offers the greatest performance of the three, while PIOPS SSD EBS volumes are more performant than general purpose SSD EBS volumes. Because the Instance Store storage amount is fixed per Instance Type, Instance Store is ideal if the index is expected to be small, and will not grow very much. If the size of the search index is expected to grow beyond the offered Instance Store sizes, the decision to go with general purpose vs PIOPS is based on performance needs and cost.

#### Security

Access to the AWS Elasticsearch Service domain is enforced by resource-based policies. These policies are stored in AWS IAM, and limit which resources, such as EC2 instances in the DAS API group, can access the AWS Elasticsearch Service API endpoint.

#### Data Integrity

##### Data Redundancy

All index data in the master nodes is copied across the cluster to corresponding replica nodes. Daily snapshots of the Elasticsearch indexes are automatically backed up into S3 in the case of a complete failure.

#### Automatic Failover and Load Balancing

When a cluster node that contains a primary shard fails, AWS Elasticsearch Service promotes a replica shard to a primary shard on another node in a different availability zone.

## Description and Authority Services Application Programming Interface

The DAS API component is the centerpiece of the application tier of the modernization architecture. All functions requiring read and write access to the Unified Data Store and Search Index are proxied through the DAS API’s RESTful interface.

### Infrastructure and Resources

The DAS API component is comprised of an auto scaling group, an AWS Elastic Load Balancer (ELB), and a set of EC2 instances that host the DAS API application.

### Design Aspects and Technical Considerations

#### Scalability

##### Vertical

The DAS API component can be scaled vertically by changing the EC2 instance type to add more memory or CPU resources. Storage can also be increased by increasing the size of any EBS volumes.

##### Horizontal

As demand for the DAS API’s RESTful interface grows, additional EC2 instances may be provisioned to handle the increased load. Scaling is controlled by adjusting the DAS API component auto scaling group configurations. Automatic and manual ad hoc scaling is available depending on requirements and cost factors.

#### Performance

#### Security

Physical access to the EC2 instances that make up the DAS API component is limited by the VPC security group configuration, and an ELB security group configuration. Additionally, all DAS API component host AMIs implement best practices in security configuration, are designed to have the latest recommended security and operating system patches installed, and are compliant with all relevant CIS benchmarks.

Access to other AWS services is granted with the use of IAM roles. Policies granting access to specific AWS resource and functions are assigned to roles, which are then applied to the appropriate set of EC2 instances in the DAS API auto scaling group.

#### Automatic Failover and Load Balancing

The DAS API component auto scaling group configuration ensures that any instance that fails will be replaced automatically so that the desired minimum number is always met. The Ingest module is self-load-balancing within both the Chunking module and Processing module as they dequeue messages independently.

## Unified Cache Store

A centralized global record cache, supported by AWS ElastiCache Service, caches frequently retrieved DAS records from the database.

### Physical Design Diagram

### Infrastructure and Resources

### Design Aspects and Technical Considerations

#### Scalability

##### Vertical

##### Horizontal

#### Performance

#### Security

#### Data Integrity

#### Automatic Failover and Load Balancing

## Reference Application

A Reference application is part of the preliminary discovery phase to demonstrate the basic functionality of the chosen technologies. The application consists of a web server, hosting a simple interface that allows an anonymous user to perform simple keyword searches of the search index, and return matching full DAS Description records. The application also provides an interface for submitting single DAS Description XML for insertion into the database, and for indexing. All operations will be handled through the DAS API implementation through an HTTP RESTful interface.

### Functional Component Diagram

### Infrastructure and Resources

Since it is simple and does not need much in the way of CPU or memory resources, the Reference application is hosted on a modest EC2 instance type.

### Functional Interfaces

All server-side capability of the Reference application is implemented by the DAS API server(s) and software. The DAS API server(s) will expose an HTTP/S RESTful interface with which the Reference application will communicate.

The Reference application UI is an AngularJS Single-Page-Application that supports the basic functions detailed in the following subsections.

#### Insertion and Retrieval

The Reference application leverages the DAS API endpoints for simple insertion and retrieval of full DAS records. Users submit an XML representation of a full DAS record for submission to the DAS Database, and for indexing. The UI also supports the retrieval of single records by the National Association for Information Destruction (NAID).

#### Search

The Reference application will leverage the DAS API endpoints for simple searches of DAS records. Fields searched include top-level Description fields including NAID, Title, Data Control Group, Record Group Number, and Access Restriction Status. For the purposes of demonstrating basic search functionality, all indexed fields are searched using basic keyword search. The UI returns a listing of DAS records that contain the input keyword(s), and allows the user to view the full XML representation of each.

### Design Aspects and Technical Considerations

#### Scalability

The Reference application is meant for simple demonstration purposes, and is not expected to handle a large volume of requests. Therefore, there will be no need to consider scaling of this component.

#### Performance

Due to its simplicity, the resources selected to host the Reference application is expected to be sufficient for its purposes of demonstrating basic functions of the preliminary modernization architecture.

#### Security

There is no built-in user authentication or authorization considered for this application—all interactions are anonymous. Using AWS security groups, access to the application is limited to a specific IP address range or ranges on a specific listening port, probably 443 (the default for HTTP/S).

#### Data Integrity

#### Automatic Failover and Load Balancing

As previously stated, the Reference application is not expected to support heavy loads or need load balancing across replicas.