Systems Integration Plan

Advanced Distributed Learning
Enterprise Course Catalog Initial Operational Capability

8 March 2022

This work was supported by the U.S. Advanced Distributed Learning (ADL) Initiative (FA701421F0184). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ADL Initiative or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
03-08-2022	Report	03/08/2021 - 03/8/2022

4. TITLE AND SUBTITLE
Enterprise Course Catalog Systems Integration Plan

5a. CONTRACT NUMBER
FA701420D007

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
0603769D8Z

6. AUTHOR(S)
Eric Flamer, Sam Schenkman, Michael Son (Deloitte Consulting LLP)

5d. PROJECT NUMBER
N/A

5e. TASK NUMBER
FA701421F0184

5f. WORK UNIT NUMBER
N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Deloitte Consulting LLP

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
OUSD Personnel & Readiness
Advanced Distributed Learning Initiative
13501 Ingenuity Drive, Suite 248
Orlando, Florida 32826

10. SPONSOR/MONITOR'S ACRONYM(S)
OUSD/P&R/FE&T/ADLI

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Distribution A

13. SUPPLEMENTARY NOTES

14. ABSTRACT
The ADL ECC Systems Integration Plan provides a summary overview of the ECC Initial Operational Capability's implementation, guidance on the required interfaces between the ECC and the ADL TLA reference implementation in an AWS Platform One environment, a corresponding integration approach, and stakeholder engagement strategy to support a reference implementation of the ECC IOC.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Sae Schatz
U	U	U	UU	33	19b. TELEPHONE NUMBER *(Include area code)* 571-480-4640

Reset

Systems Integration Plan in Support of:

Advanced Distributed Learning

Enterprise Course Catalog Initial Operational Capability

Review periods. The Government will review and comment on each delivered data item within 10 business days of its receipt. The Contractor shall provide updates or resolutions to all comments within 10 business days of receiving the DIT to close out the deliverable.

Document Control Information

Document Name	Systems Integration Plan
Client	Advanced Distributed Learning (ADL) Initiative
Task Order Name	Enterprise Course Catalog Initial Operating Capability
Document Owner	Eric Flamer, Sam Schenkman, Michael Son (Deloitte Consulting LLP)
Document Version	1.1
Document Status	Final

Document Edit History

Version	Date	Additions/Modifications	Prepared/Revised by
1.0	12/22/2021	Draft submitted to ADL	Deloitte
1.1	1/18/2022	Final submitted to ADL after original feedback on draft	Deloitte

Distribution of Final Document

The following people are designated recipients of the final version of this document:

Name	Organization/Title
Dr. Sae Schatz	Contracting Officer Representative
Steve Faber	Government Technical Point of Contact
Brent Smith	Government Technical Lead
Trey Hayden	Government Solution Engineer

Table of Contents

1.0 Introduction

1.1 Deliverable Purpose

The Enterprise Course Catalog (ECC) is one of the three Enterprise Digital Learning Modernization (EDLM) lines of effort supported by ADL. Today, course catalogs for the Department of Defense (DoD) organizations are stored in disparate locations with inconsistent data schemas which complicates the transport, management, consolidation, and governance of the course catalogs across and within DoD organizations.

The goal of the ECC is to aggregate course catalog across multiple systems and organizations to provide a centralized location for DoD personnel to view and interact with their learning and development data. The aim of this document is to provide a summary overview of the ECC Initial Operational Capability's implementation, guidance on the required interfaces between the ECC and the ADL Total Learning Architecture (TLA) reference implementation in an Amazon Web Services (AWS) Platform One environment, a corresponding integration approach, and stakeholder engagement strategy to support a reference implementation of the ECC Initial Operational Capability (IOC).

The Deloitte Team, comprised of Deloitte practitioners and our Lingatech sub-contractors, will perform system integration activities to successfully deploy ECC on Platform One in preparation for an IOC test and evaluation period, which is expected to occur during the months of June and July 2022. ECC IOC implementation efforts will begin within the DoD Platform One environment starting in December 2021 and concluding in August 2022 after the completed government acceptance testing activities. Beyond August 2022, ADL may continue to mature, test, and harden the implemented ECC IOC in pursuit of the appropriate accreditations required to support additional testing within stakeholder environments using live course data, which may include integration with other DoD systems in preparation for the Full Operational Capability (FOC).

1.2 Deliverable Scope

The scope of the ECC IOC Systems Integration Plan (SIP) is to provide technical guidance for integrating ECC IOC with DoD organizations to extract, transform and load course metadata within the Platform One Cloud environment. The intended audiences for this document are developers or engineers who are familiar with TLA standards and core cloud computing principles. Readers are not required to know all the details about each standard. However, some knowledge of basic containerization concepts is advantageous for reading this document. Readers should also have a fundamental understanding of how data is exchanged across the internet including how to represent and exchange data using JavaScript Object Notation (JSON), and how to access Representational State Transfer (REST) Application Programming Interfaces (API) on the web.

2.0 ECC IOC Integration Plan and Strategy

The following section outlines the core components of the ECC IOC, the ECC IOC deployment approach, and integration strategy with target ADL TLA systems within the Platform One Cloud. The ECC IOC consists of eight core components: Experience Discovery Service (Front and Back End), Experience Management Service (Front and Back End), Experience Search Engine, Experience Index Agent(s), Experience Index Service, Experience Schema Service. Components are divided into three tiers: ECC Client Tier, ECC Application Tier, and ECC Data Tier. Refer to **Figure 1** below for visual representation. Components are defined as independently deployable software services which, combined together, comprise the system. Components are made up of one or more software packages, libraries, or modules, depending on the technology used to construct the solution; component source code is typically modeled as a repository, although packages, libraries, and modules are also often modeled as repositories as well. A detailed description of the ECC IOC components is defined within **Section 2.1**.

The ECC IOC application deployment approach relies on repeatable, well-defined steps that involve compiling code, building the application, conducting automated unit tests, provisioning servers, and uploading code to Platform One GitLab. Our deployment approach is orchestrated by a continuous integration / continuous delivery (CI/CD) pipeline. Leveraging the CI/CD pipeline enables quicker releases and improves code quality resulting in less bugs following a release. Refer to **Section 2.2** for more detailed information on the ECC IOC deployment approach.

2.1 ECC Overview

The ECC is designed to offer flexibility in connecting to multiple source catalogs. Once invoked, the ECC Index Agents load the data into the Experience Index Service for a global search capability of ingested courses. Once the data has been stored within the XIS, data is pushed to Elasticsearch where high performance search engine is used to retrieve course data. Please refer to **Figure 2** below for a visual representation of the components and connections comprising the ECC. While the figure depicts an AWS-hosted environment, the ECC can likely run on other major cloud providers (e.g., Azure, Google Cloud Platform) or on-premise environments with minor modifications.

Figure 1 *– ECC System Tiers*

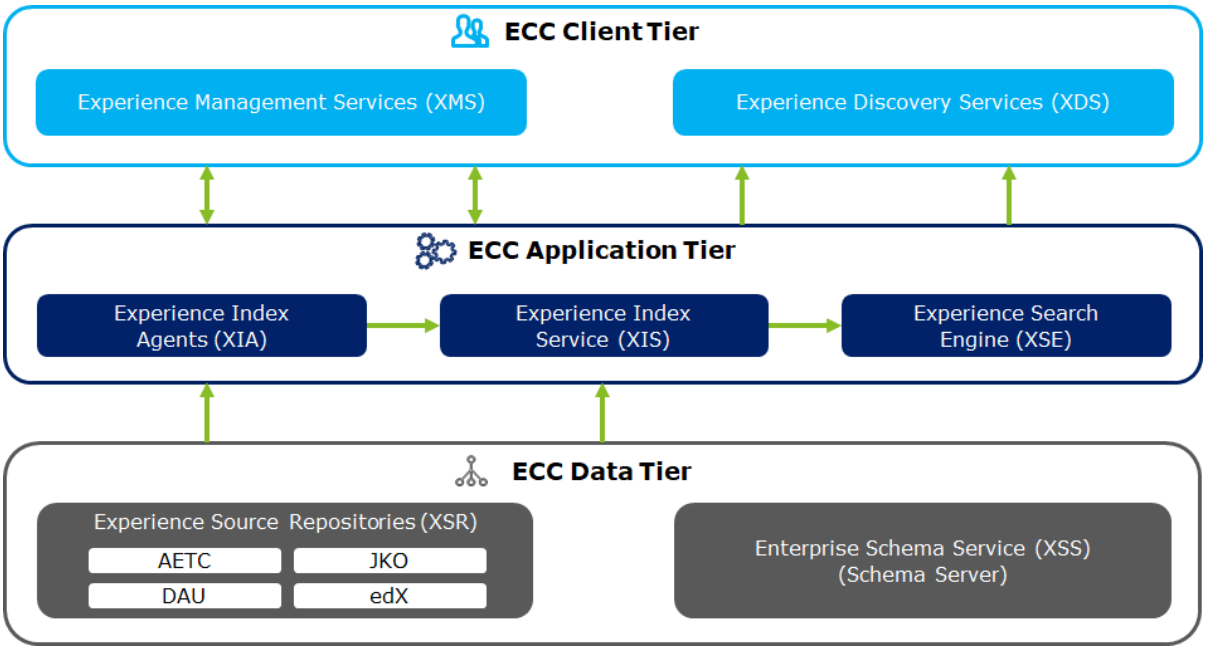
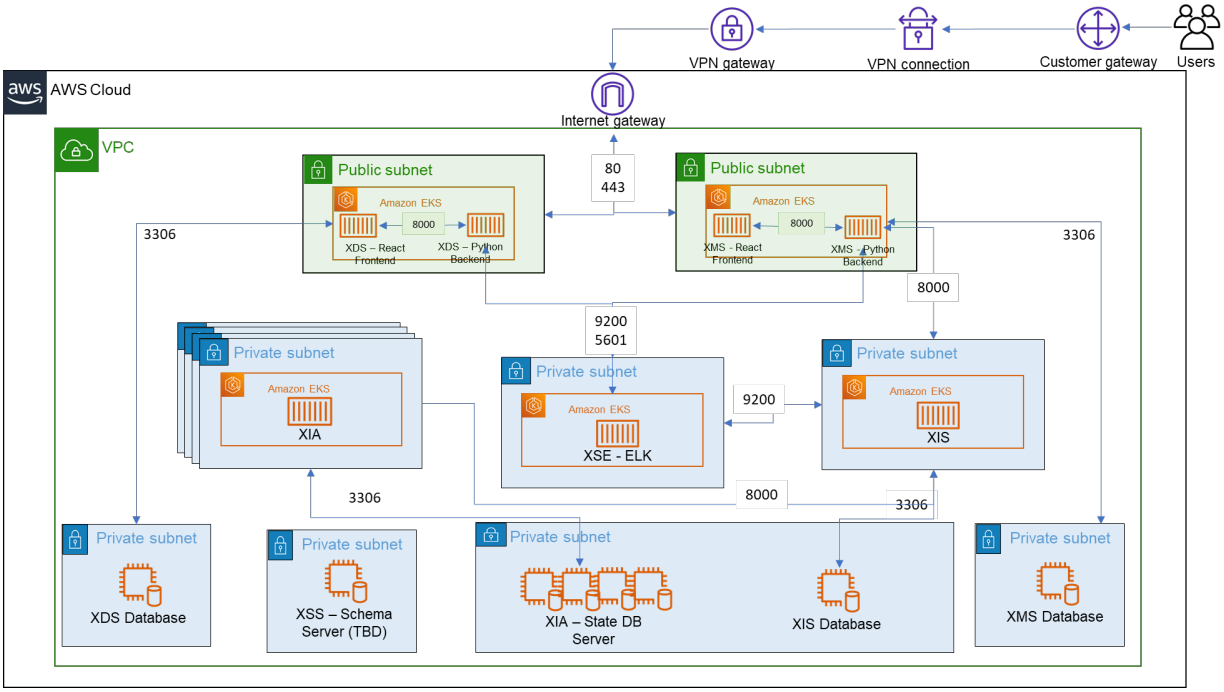


Figure 2 *– ECC Infrastructure*



Tier 1 - ECC Presentation/Client

The **ECC Presentation/Client** layer represents the user-facing components of the ECC to enable users to interact with the aggregated course catalog. The front-end offers views for Experience Participant (Facilitator and Consumer), system operators, experience owners, and experience managers with tailored data and role-based access control (RBAC). The ECC Presentation/client layer consists of the following proposed components:

- **Experience Discovery Service** – The Experience Discovery Service (XDS) is the human-facing application enabling an Experience Consumer or Experience Facilitator to easily locate a pertinent learning experience metadata record that has been indexed by the Experience Index Service (XIS). Because the XDS is a separate application, it can be deployed in a separate environment from the XIS, and can even be configured to point to a different XIS as needed. In addition, multiple XDS applications can be deployed and point to the same XIS, allowing for a great amount of installation and configuration flexibility.

  The Experience Discovery Service (XDS) focuses on enabling two main use cases:

  - Direct Search: An Experience Consumer or Experience Facilitator interacts directly with the XDS in order to locate one or more pertinent learning experiences using search/browse features presented by the component.
  - Indirect Search: An Experience Consumer or Experience Facilitator receives communications asynchronously from the XDS in order to become aware of one or more pertinent learning experiences.
- **Experience Management Service -** The Experience Management Service is the human-facing application enabling an Experience Owner or Experience Manager to modify or augment a learning metadata record ingested by the Experience Index Service (XIS). Because the XMS is a separate application, it can be deployed in a separate environment from the XIS, and can even be configured to point to a different XIS as needed. In addition, multiple XMS applications can be deployed and point to the same XIS, allowing for a great amount of installation and configuration flexibility.

  In addition to managing the XIS, the XMS also enables an Experience Owner to monitor Experience Index Agent (XIA) operational health in order to identify problems with the aggregation workflow.

Tier 2 - ECC Application Tier

The **ECC Application Tier** contains the functional process logic for the ECC IOC. This process logic is critical in extracting, validating, transforming, validating, and loading the course metadata from each source. Below is a description of the expected common services and how they will be used for the ECC IOC:

- **Experience Index Agent** – Experience Index Agents (XIA) are the lifeblood of the ECC system. XIAs are purpose-built pieces of technology – network-based, independently-

deployed software services – configured to work with a specific Experience Source Repository (XSR).

Experience Index Agents are deployed within a containerized operating environment such as Docker. XIA containers can be deployed in proximity to their designated Experience Source Repository (XSR), for security and/or management reasons, or they can be deployed within the ECC environment – or another environment entirely – and connect to the configured target XSR over a network protocol. XIAs operate on a defined/configured interval utilizing an internal scheduler. When the XIA is able to connect to its corresponding XSR, it then executes the following workflow operations:

- **Extract**: XIAs pull pertinent learning experience metadata records from their corresponding XSR, based on configuration.
- **Validate**: XIAs compare extracted learning experience metadata against the configured standard XSR reference schema stored in the Experience Schema Service (XSS)
- **Transform**: XIAs transform extracted learning experience metadata to the configured target schema using the XSR-to-Target transformation stored in the Experience Schema Service (XSS)
- **Validate**: XIAs compare transformed learning experience metadata against the configured target XSR reference schema stored in the Experience Schema Service (XSS)
- **Load**: XIAs push validated and transformed learning experience metadata to the destination Experience Index Service (XIS) for downstream processing
- **Log**: XIAs log error, warning, informational, and debug events to a data store which can be monitored.

- **Experience Index Service** – The Experience Index Service (XIS) is the core of the Enterprise Course Catalog solution. It is the central sink for learning experience metadata collected by the Experience Index Agent (XIA) components. In addition, the XIS can receive supplemental learning experience metadata – field name/value overrides and augmentations – from the Experience Management Service (XMS).

  Learning experience metadata is received from XIAs is stored in the Metadata Loading Area and processed asynchronously in order to enhance overall system performance and scalability. Processed metadata is combined with supplemental metadata provided by an Experience Owner or Experience Manager and the "composite record" is stored in the Metadata Repository. Metadata Repository record addition/modification events are logged to a job queue and the metadata is then sent to the Experience Search Engine (XSE) for indexing and high-performance location/retrieval.

  It is possible for an XIS to syndicate its composite records to another XIS. The record set can be filtered by one or more facets/dimensions in order to transmit a subset of the overall composite record repository. In addition, the transmitted field set can be configured to contain redacted values for specified fields when information is considered too sensitive for syndication.

- **Experience Search Engine** – The Experience Search Engine (XSE) system component is responsible for enabling fast identification and retrieval of learning experience metadata records aggregated by the Experience Index Service (XIS). After being loaded into one or more XSE indices, learning experience metadata records can be identified specifically by identifier, or more broadly by a keyword and/or facet match.

  Because search technology is generally a solved problem, the ECC project team will not be implementing a custom service. Instead, the team will be utilizing Elasticsearch, which is well-known for performance and scalability. Elasticsearch supports a variety of features which should prove to be useful in the ECC solution, including search, faceting, and percolation.

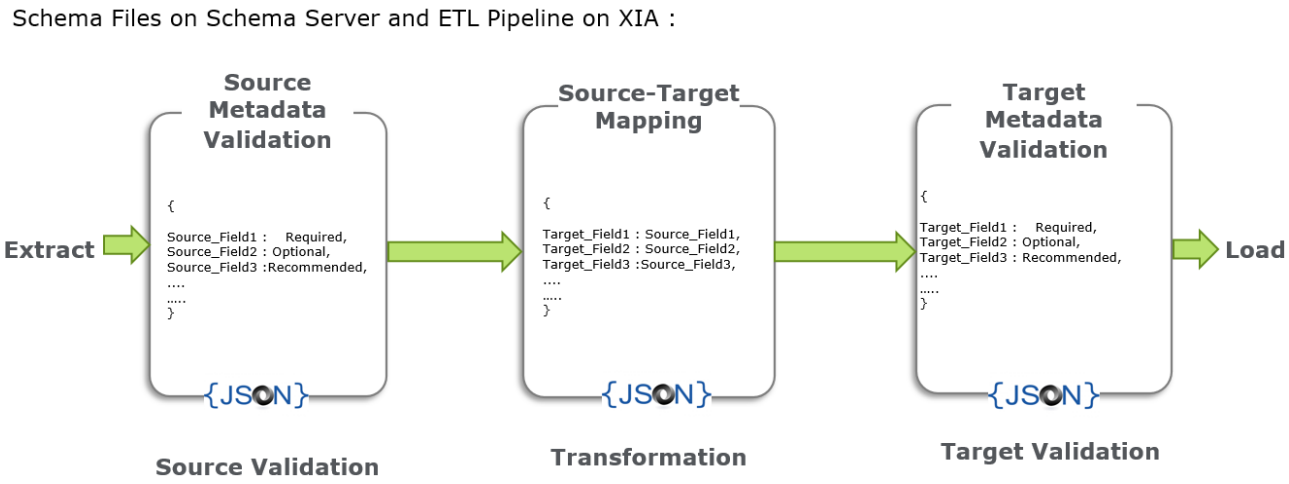
Tier 3 - ECC Data Tier

ECC data tier leverages Amazon Simple Storage Service (S3) and Amazon Relational Database Service (RDS) in Platform One. S3 is a highly reliable, scalable, fault-tolerant, and secure object storage system. Fault tolerance and availability of S3 buckets are managed by AWS. ECC will leverage S3 buckets to store schema files for ECC components to extract, validate and transform source metadata. ECC will leverage Amazon RDS MySQL for persistent data store and index service data. RDS MySQL offers a highly scalable, reliable, fault-tolerant, and secure database. RDS will be designed to fail over between availability zones that eliminates interruptions to the database availability.

The **ECC Data Tier** contains the following components:

- **Experience Source Repositories (XSR) -** Experience Source Repositories are the primary data resources supporting the entire Enterprise Course Catalog solution. Without the learning experience metadata extracted from the XSRs, the entire system would depend on Experience Owners and Experience Managers to enter all of the metadata manually. This would introduce an additional workload on top of the work they are already performing to populate their own experience delivery systems and related catalog services. ECC can extract data via scheduled file drops, direct connection to the source database, upload data and connection to API endpoints. Source data can be in XML, JSON, and CSV file formats when being extracted into ECC.
- **Experience Schema Service (XSS) -** The Experience Schema Service maintains referential representations of domain entities, as well as transformational mappings that describe how to convert an entity from one particular schema representation to another. The XSS is also responsible for storing schema mappings used by the XIA for source metadata transformation and validation. Both metadata and supplemental metadata ledgers get extracted from respective XSR repositories and transformed using the P2881 standards defined in the XSS. **Figure 3** below represents ETL pipeline performed by XIA using the XSS to validate the schema standards.

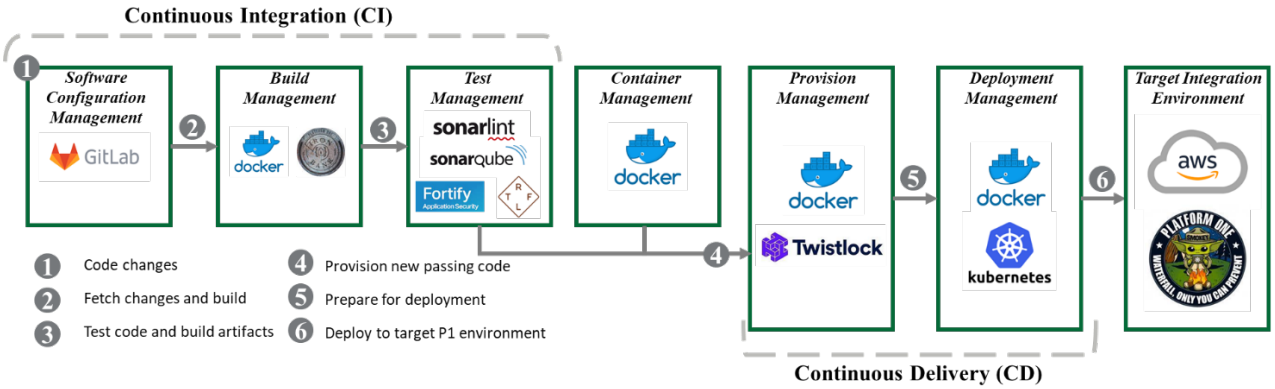
***Figure 3** – ECC ETL Pipeline*



2.2 ECC IOC Deployment Strategy

The internal ECC IOC application leverages a robust CI/CD pipeline that enables quality and secure application builds of the ECC. The CI/CD pipeline offers more consistency that the latest versions of the ECC will operate as intended and to meet Platform guidelines within the US Air Force's Platform One Cloud environment. ECC IOC will be integrated with the Platform One Gitlab CI/CD pipeline to help orchestrate the build, testing, and deploying applications to Kubernetes. Code quality is achieved through SonarQube, performing static code analysis for the ECC to catch errors within code prior to a deployment. Platform One CI/CD pipeline scans each repository for a Dockerfile that meets compliance, unit tests using lint, static code analysis using Fortify, Sonarqube and Trufflehog. Figure 4 provides a summary representation of each stage in Platform One's CI/CD process based on the standardized Platform One pipeline.

***Figure 4** – ECC-Adopted Platform One CI/CD Pipeline for Party Bus Deployment*



2.3 System Integration Roles and Responsibilities

To support the deployment of the ECC IOC into the Platform One Cloud environment, the following staffing plan, roles, and quantity are accounted for in **Table 1** below.

Table 1 – *ECC Integration Roles and Responsibilities*

ECC IOC Role	ECC IOC Responsibilities
Deloitte Team Development Engineers (x6)	▪ Develop ECC code ▪ Verify ECC code meets Platform One standards ▪ Verify ECC code passed all analysis and quality checks to ensure pipeline is green ▪ Update README files accordingly for deployment steps
Deloitte Team Testers (x2)	▪ Conducts ECC tests in the QA environment on Deloitte AWS sandbox and in Platform One
Deloitte Team Project Manager (x1)	▪ Manages scope and schedule as defined in the ECC technical management work plan ▪ Supports overall project functional activities, including stakeholder management, risk and issue resolution, and stakeholder communications
ADL Engineers	▪ Verify ADL EDLM code meets Platform One standards ▪ Verify ADL EDLM code passed all analysis and quality checks to ensure pipeline is green ▪ Work with Deloitte Team Development Engineers to verify ECC IOC deployment for ECC IOC Test and Evaluation activities.
Platform One DevOps Engineers	▪ Develop ECC pipelines matching ECC components for deployment to test, pre-production, and production Platform One environments. ▪ Verify ECC code passed all analysis and quality checks to ensure the pipeline is green. ▪ Work with Deloitte Team and ADL Engineers to troubleshoot deployment and integration error within the Platform One environments.

2.4 Assumptions

In relation to the guidance provided within the ECC IOC SIP, the following assumptions have been made within **Table 2** below.

Table 2 – *ECC IOC SIP Assumptions*

Assumption	Rationale
Development will rely on Platform One's DevOps processes to successfully deploy ECC	ECC Development team will work closely with the Platform One DevOps team to integrate ECC with Platform One's Cloud environment. Development team will be responsible of keeping the CI/CD pipeline green through various quality & security checks. To the extent needed, Deloitte will update this SIP deliverable to match the expected integration and deployment practices established by Platform One for the ECC IOC.

Assumption	Rationale
Development Team will have significantly limited access to the Platform One cloud environment after code is pushed to Gitlab	Development team will have limited access/control of the deployment into AWS. All deployment activities will be handled by Platform One's DevOps team. To the extent needed, Deloitte will update this SIP deliverable to match Platform One's deployment processes, workflows, and practices once our ECC pipelines are operational.
Development Team will rely on Platform One to confirm system integration with ADL's newly deployed EDLM systems/enclave	In the previous Operational Prototype development cycle, Deloitte relied on the ADL's TLA Sandbox environment for integration with other TLA components, allowing for a greater degree of access/control via ADL. With Platform One, the assumption above relating to access impacts ADL's own control of its other planned pipeline deployments for services that may impact the ECC IOC (specifically the xAPI Profile Server). Deloitte will coordinate with ADL on the deployment and integration practices for these other systems and update this SIP deliverable to match the expected integration steps with these other EDLM systems.

3.0 ECC Integration Approach with Platform One Party Bus

The ECC IOC relies on connections to source repositories to extract, transform and load course metadata. The four source repositories used during the prototype phase were Defense Acquisition University (DAU), Air and Education Training Command (AETC), Joint Knowledge Online (JKO) and an external source edX. The ECC communicates with the source repositories via API endpoints or from an exported CSV/Excel file from the source provider. The ECC development team will follow Platform One's Dockerfile guidelines to deliver secure and approved containers. Dockerfiles will be limited to non-root users in the scripts. The base image in each Dockerfile will be pointing to a hardened container in Registry One's repository. Even in multi-stage Dockerfiles, the team will ensure hardened containers are referenced.

3.1 Experience Index Agent (XIA)

The XIA is the data processing engine for the ECC. XIA agents are configured to connect to source repositories, extract the data via an API endpoint or CSV. Once extracted, XIA agents will automatically validate the extracted data, transform the data to the mapped data schema, validate the transformed data, then load the data into XIS.

The identified system interfaces between an Authoritative XSR and XIA are depicted within **Table 3** below.

Table 3 *– XIA System Interfaces*

ECC Component	Interfaces	Connection Duration	Connection Frequency
ECC XSR	API endpoint for XIA to connect and run ETL on the data	Ephemeral	On-demand
ECC XIA	Connects to Experience Source Repositories to run ETL and map course data to the schema	Ephemeral	On-demand

3.2 Experience Index Service (XIS)

The XIS is the component that holds the transformed data from the XIAs. When data is loaded from respective XIA agents, XIS validates the data against the XSS schema standards. Once data is stored in the XIS, data is pushed to XSE (Elasticsearch). The XDS component retrieves user queries directly from XSE and XMS component connect directly to XIS to retrieve detailed course data and allow experience managers to make modifications to the metadata

The identified system interfaces between XIA, XIS, XSE and XMS are depicted within **Table 4** below.

***Table 4** – XIS System Interfaces*

ECC Component	Interfaces	Connection Duration	Connection Frequency
ECC XIS	API Endpoint	Ephemeral	On-demand
ECC XIA	API Endpoint	Ephemeral	On-demand
ECC XSE	API Endpoint	Ephemeral	On-demand
ECC XMS	API Endpoint	Ephemeral	On-demand

3.3 Experience Search Engine (XSE)

The XSE is the core search engine for ECC. Elasticsearch will handle indices, queries, and highly available search capabilities for the users. Data is pushed from XIS which is then used to retrieve queries from XDS.

The identified system interfaces between XSE, XDS, XIS are depicted within **Table 5** below.

***Table 5** – XSE System Interfaces*

ECC Component	Interfaces	Connection Duration	Connection Frequency
ECC XSE	API Endpoint	Ephemeral	On-demand
ECC XIS	API Endpoint	Ephemeral	On-demand
ECC XDS	API Endpoint	Ephemeral	On-demand

3.4 Experience Discovery Service (XDS)

The XDS is the main interface for the users to interact with. This is where users will get the full experience as experience participants (facilitator and consumer). XDS will interact with the XSE component to retrieve queries, highlight course, save courses, and be taken directly to the course provider.

The identified system interfaces between XDS, Users, XSE are depicted within **Table 6** below.

Table 6 – *XDS System Interfaces*

ECC Component	Interfaces	Connection Duration	Connection Frequency
ECC XDS	API Endpoint	Ephemeral	On-demand
Internet/Users	HTTPS endpoint	Ephemeral	On-demand
ECC XSE	API Endpoint	Ephemeral	On-demand

3.5 Experience Management Service (XMS)

The XMS is the management system where updates to the metadata can be made. Each experience facilitator can be granted roles to the XMS that matches their line of organization to modify the metadata for the organization's preferences. XMS will connect directly with the XIS to make these modifications.

Table 7 – *XMS System Interfaces*

ECC Component	Interfaces	Connection Duration	Connection Frequency
ECC XDS	API Endpoint	Ephemeral	On-demand
Internet/Users	HTTPS endpoint	Ephemeral	On-demand
ECC XSE	API Endpoint	Ephemeral	On-demand

3.6 Experience Schema Service (XSS)

The Experience Schema Service is meant to serve as the TLA component responsible for managing pertinent object/record metadata schemas and the mappings for transforming records from a source metadata schema to a target metadata schema. The ECC Development team has developed a reference implementation of the schema service to support this linked data service for the ECC solution. Global metadata registry along with interfaces for indexing services and agents. This component will also be used to store and link vocabularies from stored schema. The schema service is used to validate metadata against the defined metadata standards. XSS can be used by other services as the global linked vocabulary service.

Table 8 – *XSS System Interfaces*

ECC Component	Interfaces	Connection Duration	Connection Frequency
ECC XIA	API Endpoint	Ephemeral	On-demand
Internet/Users	HTTPS endpoint	Ephemeral	On-demand
ECC XIS	API Endpoint	Ephemeral	On-demand

3.7 ECC Conformance Alerting (Notifications)

The ECC conformance alerting/notification component feature is designed to send email notifications to the system managers, system owners, and participants. Notifications will consist of warnings/error trap reports or record updates automatically sent to subscribers. ECC Conformance alerting is part of the automated scheduler celery tasks. The system operators can add/remove subscribers from the Django admin page. System operators can also customize the content of the email notifications. This component is an independently deployable PyPi package.

ECC conformance alerting consists of the configuration types below, which help System Operators quickly configure and set up conformance alerting.

- **Sender Email Configurations:** ECC OpenLXP-Notification package has the ability to configure sender email. System Operator can choose an email address to send conformance emails from.
- **Receiver Email Configurations:** ECC OpenLXP-Notification package has the ability to configure receiver email addresses. System Operators can add the list of email addresses to send conformance alerts.
- **Notification Email Content:** Conformance alert's email body contents is easily editable. System Operators can choose their own email body content. To learn more information on the editing of the email body, please review the ECC team's GitHub documentation found here.

3.8 ECC Automated Scheduling & Tasks

The ECC ETL (Extract, Transform, Load) process is fully automated using Celery tasks in both XIA and XIS components. In the XIA, Celery is used to extract, validate, transform, validate, load, and validate the metadata before loading transformed data into the XIS. When the ETL process is completed, celery runs a task to load data into the XIS and send notifications to subscribed users. After the data is loaded into the XIS, celery tasks kick off in the XIS to consolidate metadata ledgers and load the metadata into the XSE.

Automating these tasks eliminates manual work performed by the system operator. Hence to gain better performance, ECC tasks run on a customizable schedule to minimize disruption. System Operators can schedule daily, weekly, or monthly tasks in ECC. In the case of emergency planning or meeting specific scheduling needs, system operators can choose to run tasks manually by running an API. The parameters for scheduling the ECC automated tasks using Celery tasks includes the following:

- The System Operator can specify the Name for the task.
- The System Operator can select the applicable tasks from the registered ECC tasks.
- The System Operator can enable or disable the task scheduling by selecting or unselecting the 'Enabled' option for each task.

- The System Operator can specify interval schedule, including the interval period and the number of periods.
- The System Operator can specify the Start Date Time for the scheduled task
- The System Operator can enable or disable the task as a one-time action by selecting or unselecting the ‘One-Off’ option for each task

4.0 Appendix

Appendix A - Hardware / Software

ECC IOC Hardware

Table 9 *– ECC Hardware*

Component	Service	Operating System	Count
XIA	Kubernetes	Docker	1
XIS	Kubernetes	Docker	1
XDS	Kubernetes	Docker	1
XDS-UI	Kubernetes	Docker	1
XMS	Kubernetes	Docker	1
XMS-UI	Kubernetes	Docker	1
XSE	Kubernetes	Docker	1
XSS	Kubernetes	Docker	1

ECC IOC Software

Table 10 *– ECC Software*

Configuration Item (CI)	Name	Version
Security / Authentication	KeyCloak	7.0.x
	Django Administration	3.1.x
Application	Nginx	1.14.0
	Elasticsearch	7.16
	Django	3.1.x
	React	
Data / Storage	RDS MySQL	13.1
Container	Docker	20.10.x
Container Orchestration	Kubernetes	1.19.x
Operating System	Linux/Ubuntu	18.04
Administration	Gitlab CICD	

Appendix B - Deployment Scripts

Appendix B-1 Security, Networking, and Identity and Access Management

The necessary firewall ports for ECC, based on the specified component/infrastructure, is outlined within **Table 11** below.

***Table 11** – ECC Firewall Ports*

Component	Ports (Ingress)	Ports (Egress)
Experience Index Agent (XIA)	:22 (per IP) :443 Public :8000 (per IP)	All Traffic
Experience Index Service (XIS)	:22 (per IP) :443 Public :8100 (per IP)	All Traffic
Experience Search Engine (XSE)	:22 (per IP) :443 Public :9200 (per IP)	All Traffic
Experience Schema Service (XSS)	:22 (per IP) :443 Public :8080 (per IP)	All Traffic
Experience Discovery Service (XDS)	:22 (per IP) :443 (per IP) :8000 (per IP) :3000 (per IP)	All Traffic
Experience Management Service (XMS)	:22 (per IP) :443 Public :8000 (per IP) :3000 (per IP)	All Traffic

Appendix B-2 Deployment Scripts

ECC is deployed and integrated using microservices approach for each component. Although some components rely on others for the data to be transformed and pushed into the global catalog system, each component can be deployed independently. This allows for flexible configuration modifications based on each organization's requirements. The Experience Index Agent and Notification components are python packages available on PyPi. Eventually, XIA and XIS syndication will be implemented. Organizations will have the options to syndicate data into the deployed ECC application or from ECC into a local index service. ECC allows any component to be deployed individually and connect to/from the ADL deployed ECC.

Integration Use Case 1: Deploying an Experience Index Agent to Contribute to the ADL ECC IOC

Experience Index Agents are implemented using docker-compose. Experience Index Agents is configurable to allow any organizations to connect to their source repository. This is done by configurable API endpoints for the metadata within the XIA Django administration page. Whether the new organization exports CSV files or has API endpoints integrated with their

source repository, new source repositories can be added to OpenLXP. Docker-compose.yaml file presented below is a generic representation of the ECC docker-compose file that can be customized where <XSR> can be updated based on source repository.

```
version: "3"

services:
  db_xia_<XSR>:
    image: mysql:5.7
    ports:
      - '3306:3306'
    environment:
      MYSQL_DATABASE: "${DB_NAME}"
      MYSQL_PASSWORD: "${DB_PASSWORD}"
      MYSQL_ROOT_PASSWORD: "${DB_ROOT_PASSWORD}"
      MYSQL_HOST: ''
    networks:
      - openlxp

  app_xia_<XSR>:
    build:
      context: .
    ports:
      - "8000:8020"
    command: >
      sh -c ". /opt/app/start-app.sh"
    environment:
      DB_NAME: "${DB_NAME}"
      DB_USER: "${DB_USER}"
      DB_PASSWORD: "${DB_PASSWORD}"
      DB_HOST: "${DB_HOST}"
      DJANGO_SUPERUSER_USERNAME: "${DJANGO_SUPERUSER_USERNAME}"
      DJANGO_SUPERUSER_PASSWORD: "${DJANGO_SUPERUSER_PASSWORD}"
      DJANGO_SUPERUSER_EMAIL: "${DJANGO_SUPERUSER_EMAIL}"
      BUCKET_NAME: "${BUCKET_NAME}"
      AWS_ACCESS_KEY_ID: "${AWS_ACCESS_KEY_ID}"
      AWS_SECRET_ACCESS_KEY: "${AWS_SECRET_ACCESS_KEY}"
      AWS_DEFAULT_REGION: "${AWS_DEFAULT_REGION}"
      REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      SECRET_KEY_VAL: "${SECRET_KEY_VAL}"
      LOG_PATH: "${LOG_PATH}"
      CELERY_BROKER_URL: "${CELERY_BROKER_URL}"
```

```
      CELERY_RESULT_BACKEND: "${CELERY_RESULT_BACKEND}"
    volumes:
      - ./app:/opt/app/openlxp-xia-<XSR>
    depends_on:
      - db_xia_<XSR>
    networks:
      - openlxp

  redis:
    image: redis:alpine
    networks:
      - openlxp

  celery:
    build:
      context: .
    command: celery -A openlxp_xia_edx_project worker -l info --pool=solo
    volumes:
      - ./app:/opt/app/openlxp-xia-<XSR>
    environment:
      REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
    env_file:
      - ./.env
    depends_on:
      - db_xia_<XSR>
      - redis
      - app_xia_<XSR>
    networks:
      - openlxp
    restart: on-failure

  celery-beat:
    build:
      context: .
    command: celery -A openlxp_xia_<XSR>_project beat --scheduler
django_celery_beat.schedulers:DatabaseScheduler --loglevel=info --
pidfile=/tmp/celerybeat.pid
    volumes:
      - ./app:/opt/app/openlxp-xia-<XSR>
    environment:
      REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
```

```
      AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
    env_file:
      - ./.env
    depends_on:
      - db_xia_<XSR>
      - redis
      - app_xia_<XSR>
    networks:
      - openlxp
    restart: on-failure

  flower:
    image: mher/flower:0.9.7
    command: [ "flower", "--broker=redis://redis:6379/0", "--port=8888" ]
    ports:
      - 8888:8888
    networks:
      - openlxp

networks:
  openlxp:
    external: true
```

System Operators can use the current implementations of edX, JKO, DAU, and AETC Experience Index Agents by pulling the code from below repositories. Each of these implementations demonstrates a catalog contribution from a distributed system to the ADL's EDLM ECC deployment.

- **edX** (https://github.com/OpenLXP/openlxp-xia-edx)
- **JKO** (https://github.com/OpenLXP/openlxp-xia-jko)
- **DAU** (https://github.com/OpenLXP/openlxp-xia-dau)
- **AETC** (https://github.com/OpenLXP/openlxp-xia-aetc)

Integration Use Case 2: Deploying an Experience Index Service for Local Use of the Full ECC Solution:

The Experience Index Service is implemented using docker-compose. Clone the XIS Github repository and run 'docker-compose up -d'. XIS component is developed to allow syndication with other XIS components. System Operators can run their own XIS if they choose to. Whether it is external to Platform One or internal, any organization can pull or push data into/from other XIS components.

Location of XIS GitHub Repository: https://github.com/OpenLXP/openlxp-xis

```
version: "3"

services:
  db_xis:
    image: mysql:5.7
    ports:
      - '3310:3306'
    environment:
      MYSQL_DATABASE: "${DB_NAME}"
#      MYSQL_USER: 'root'
      MYSQL_PASSWORD: "${DB_PASSWORD}"
      MYSQL_ROOT_PASSWORD: "${DB_ROOT_PASSWORD}"
      MYSQL_HOST: ''
    networks:
      - openlxp
  app_xis:
    container_name: openlxp-xis
    build:
      context: .
    ports:
      - "8080:8020"
    command: >
      sh -c ". /opt/app/start-app.sh"
    environment:
      DB_NAME: "${DB_NAME}"
      DB_USER: "${DB_USER}"
      DB_PASSWORD: "${DB_PASSWORD}"
      DB_HOST: "${DB_HOST}"
      DJANGO_SUPERUSER_USERNAME: "${DJANGO_SUPERUSER_USERNAME}"
      DJANGO_SUPERUSER_PASSWORD: "${DJANGO_SUPERUSER_PASSWORD}"
      DJANGO_SUPERUSER_EMAIL: "${DJANGO_SUPERUSER_EMAIL}"
      BUCKET_NAME: "${BUCKET_NAME}"
      AWS_ACCESS_KEY_ID: "${AWS_ACCESS_KEY_ID}"
      AWS_SECRET_ACCESS_KEY: "${AWS_SECRET_ACCESS_KEY}"
      AWS_DEFAULT_REGION: "${AWS_DEFAULT_REGION}"
      REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      SECRET_KEY_VAL: "${SECRET_KEY_VAL}"
      LOG_PATH: "${LOG_PATH}"
      CELERY_BROKER_URL: "${CELERY_BROKER_URL}"
      CELERY_RESULT_BACKEND: "${CELERY_RESULT_BACKEND}"
    volumes:
```

```
    - ./app:/opt/app/openlxp-xis
  depends_on:
    - db_xis
  networks:
    - openlxp

es01:
  image: docker.elastic.co/elasticsearch/elasticsearch:7.11.1
  container_name: es01
  environment:
    - discovery.type=single-node
  ulimits:
    memlock:
      soft: -1
      hard: -1
  volumes:
    - data01:/usr/share/elasticsearch/data
  ports:
    - 9200:9200
    - 9300:9300
  networks:
    - openlxp

redis:
  image: redis:alpine
  networks:
    - openlxp

celery:
  build:
    context: .
  command: celery -A openlxp_xis_project worker -l info --pool=solo
  volumes:
    - ./app:/opt/app/openlxp-xis
  environment:
    REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
    AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
  env_file:
    - ./.env
  depends_on:
    - db_xis
    - redis
```

```
      - app_xis
    networks:
      - openlxp
    restart: on-failure

  celery-beat:
    build:
      context: .
    command: celery -A openlxp_xis_project beat --scheduler
django_celery_beat.schedulers:DatabaseScheduler --loglevel=info --
pidfile=/tmp/celerybeat.pid
    volumes:
      - ./app:/opt/app/openlxp-xis
    environment:
      REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
    env_file:
      - ./.env
    depends_on:
      - db_xis
      - redis
      - app_xis
    networks:
      - openlxp
    restart: on-failure

  flower:
    image: mher/flower:0.9.7
    command: [ "flower", "--broker=redis://redis:6379/0", "--port=8888" ]
    ports:
      - 5555:5555
    networks:
      - openlxp

volumes:
  data01:
    driver: local
networks:
  openlxp:
    external: true
```

The Experience Search Engine is implemented using docker-compose. Clone the Github repository and run 'docker-compose up -d'. There are no additional configurations needed to set up XSE for OpenLXP. Once the Elasticsearch instances are running, indices are created based on configuration from the XIA component.

Location of XSE GitHub Repository: https://github.com/OpenLXP/openlxp-xse

```
version: '2.2'
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.16.2
    container_name: es01
    environment:
      - node.name=es01
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es02,es03
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - elastic

  es02:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.16.2
    container_name: es02
    environment:
      - node.name=es02
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es03
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
```

```
    volumes:
      - data02:/usr/share/elasticsearch/data
    networks:
      - elastic

  es03:
    image: docker.elastic.co/elasticsearch/elasticsearch:7.16.2
    container_name: es03
    environment:
      - node.name=es03
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es02
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data03:/usr/share/elasticsearch/data
    networks:
      - elastic

  kib01:
    image: docker.elastic.co/kibana/kibana:7.16.2
    container_name: kib01
    ports:
      - 5601:5601
    environment:
      ELASTICSEARCH_URL: http://es01:9200
      ELASTICSEARCH_HOSTS: '["http://es01:9200","http://es02:9200","http://e
s03:9200"]'
    networks:
      - elastic

volumes:
  data01:
    driver: local
  data02:
    driver: local
  data03:
```

```
    driver: local

networks:
  elastic:
    driver: bridge
```

The Experience Discovery Service is implemented using docker-compose. Clone the Github repositories and run 'docker-compose up -d' to launch both XDS and XDS-UI. System Operators will configure the XDS component if there is a need to stand up their own discovery service UI.

Location of XDS GitHub Repository: (https://github.com/OpenLXP/openlxp-xds):

```
version: "3"

services:
  db:
    image: mysql:5.7
    ports:
      - '3306:3306'
    environment:
      MYSQL_DATABASE: "${DB_NAME}"
#      MYSQL_USER: 'root'
      MYSQL_PASSWORD: "${DB_PASSWORD}"
      MYSQL_ROOT_PASSWORD: "${DB_ROOT_PASSWORD}"
      MYSQL_HOST: ''
    networks:
      - openlxp

  app:
    container_name: openlxp-xds
    build:
      context: .
    ports:
      - "8100:8020"
    command: >
      sh -c ". /opt/app/start-app.sh"
    environment:
      DB_NAME: "${DB_NAME}"
      DB_USER: "${DB_USER}"
      DB_PASSWORD: "${DB_PASSWORD}"
      DB_HOST: "${DB_HOST}"
      DJANGO_SUPERUSER_USERNAME: "${DJANGO_SUPERUSER_USERNAME}"
      DJANGO_SUPERUSER_PASSWORD: "${DJANGO_SUPERUSER_PASSWORD}"
      DJANGO_SUPERUSER_EMAIL: "${DJANGO_SUPERUSER_EMAIL}"
```

```
      AWS_ACCESS_KEY_ID: "${AWS_ACCESS_KEY_ID}"
      AWS_SECRET_ACCESS_KEY: "${AWS_SECRET_ACCESS_KEY}"
      AWS_DEFAULT_REGION: "${AWS_DEFAULT_REGION}"
      REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      SECRET_KEY_VAL: "${SECRET_KEY_VAL}"
      ES_HOST: "${ES_HOST}"
      ES_INDEX: "${ES_INDEX}"
      LOG_PATH: "${LOG_PATH}"
      ENTITY_ID: "${ENTITY_ID}"
      SP_PUBLIC_CERT: "${SP_PUBLIC_CERT}"
      SP_PRIVATE_KEY: "${SP_PRIVATE_KEY}"
    volumes:
      - ./app:/opt/app/openlxp-xds
    depends_on:
      - db
    networks:
      - openlxp

networks:
  openlxp:
    external: true
```

Location of XDS-UI GitHub Repository: https://github.com/OpenLXP/openlxp-xds-ui):

```
version: '3'
services:
  # list of containers to run
  nextjs:
    # pointing to the dockerfile locations
    build: ./
  nginx:
    # pointing to the dockerfile location in nginx/
    build: ./nginx
    ports:
      - 80:80
      - 443:443
```

The Experience Management Service is implemented using docker-compose. Use the docker-compose file below to launch both XMS and XMS-UI. System Operators will configure the XMS component if there is a need to stand up their own management service UI.

Location of XMS GitHub Repository: https://github.com/OpenLXP/openlxp-xms

```
version: "3"

services:
  db:
    image: mysql:5.7
    ports:
      - '3306:3306'
    environment:
      MYSQL_DATABASE: "${DB_NAME}"
#      MYSQL_USER: 'root'
      MYSQL_PASSWORD: "${DB_PASSWORD}"
      MYSQL_ROOT_PASSWORD: "${DB_ROOT_PASSWORD}"
      MYSQL_HOST: ''
    networks:
        - openlxp

  app:
    container_name: openlxp-xms
    build:
      context: .
    ports:
      - "8000:8020"
    command: >
      sh -c ". /opt/app/start-app.sh"
    environment:
      DB_NAME: "${DB_NAME}"
      DB_USER: "${DB_USER}"
      DB_PASSWORD: "${DB_PASSWORD}"
      DB_HOST: "${DB_HOST}"
      DJANGO_SUPERUSER_USERNAME: "${DJANGO_SUPERUSER_USERNAME}"
      DJANGO_SUPERUSER_PASSWORD: "${DJANGO_SUPERUSER_PASSWORD}"
      DJANGO_SUPERUSER_EMAIL: "${DJANGO_SUPERUSER_EMAIL}"
      DATA_FILE_NAME: "${DATA_FILE_NAME}"
      SCHEMA_FILE: "${SCHEMA_FILE}"
      BUCKET_NAME: "${BUCKET_NAME}"
      UPLOAD_BUCKET_NAME: "${UPLOAD_BUCKET_NAME}"
      AWS_ACCESS_KEY_ID: "${AWS_ACCESS_KEY_ID}"
      AWS_SECRET_ACCESS_KEY: "${AWS_SECRET_ACCESS_KEY}"
      AWS_DEFAULT_REGION: "${AWS_DEFAULT_REGION}"
      REQUESTS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      AWS_CA_BUNDLE: '/etc/ssl/certs/ca-certificates.pem'
      SECRET_KEY_VAL: "${SECRET_KEY_VAL}"
```

```
      LOG_PATH: "${LOG_PATH}"
    volumes:
      - ./app:/opt/app/openlxp-xms
    depends_on:
      - db
    networks:
      - openlxp

networks:
  openlxp:
    external: true
```

Location of XMS-UI GitHub Repository: https://github.com/OpenLXP/openlxp-xms-ui

```
version: '3'
services:
  # list of containers to run
  nextjs:
    # pointing to the dockerfile locations
    build: ./
  nginx:
    # pointing to the dockerfile location in nginx/
    build: ./nginx
    ports:
      - 80:80
      - 443:443
```

Finally, the notifications implementation of the OpenLXP enables ECC users to receive notifications from the application. This component is also configurable through the Django administration page. Sender/receiver emails, notification body message, and attachment options can be configured. This allows ECC users to be notified of any errors that occurred during the metadata ETL process. OpenLXP notifications is packaged in PyPi to allow installation during the container build of the XIA component. Below is the Github code repository for the notifications.

- Notifications: https://github.com/OpenLXP/openlxp-notifications

Appendix C: Key Terms

Table 12 below summarizes the acronyms referenced in this document.

***Table 7** – Acronyms*

Acronym	Term
ADL	Advanced Distributed Learning
API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration/Continuous Deployment
DoD	Department of Defense
EC2	Elastic Compute Cloud
ECC	Enterprise Course Catalog
EDLM	Enterprise Digital Learning Modernization
EKS	Elastic Kubernetes Service
IAC	Infrastructure as Code
IAM	Identity Access Management
IOC	Initial Operating Capability
JSON	JavaScript Object Notation
OAS	OpenAPI Specification
SIP	Systems Integration Plan
TCP	Transmission Control Protocol
TLA	Total Learner Architecture
UI	User Interface
VM	Virtual Machine
XDS	Experience Discovery Service
XIA	Experience Index Agent
XIS	Experience Index Service
XMS	Experience Management Service
XSE	Experience Search Engine

Acronym	Term
XSS	Experience Schema Service
YAML	YAMLAin't Markup Language