ADL Enterprise Learner Record Repository Systems Integration Plan

11 May 2021

This work was supported by the U.S. Advanced Distributed Learning (ADL) Initiative (24361820D0001). 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)*
11-05-2021	Document	

4. TITLE AND SUBTITLE

ADL Enterprise Learner Record Repository Systems Integration Plan

5a. CONTRACT NUMBER

FA701420STS10

5b. GRANT NUMBER

N/A

5c. PROGRAM ELEMENT NUMBER

0603769D8Z

6. AUTHOR(S)

Flamer, Eric
Adusei-Poku,Adjoa

5d. PROJECT NUMBER

N/A

5e. TASK NUMBER

N/A

5f. WORK UNIT NUMBER

N/A

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Deloitte Consulting LLP
1919 North Lynn Street
Arlington, VA 22209

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Activity for Non-Traditional Education Support
Advanced Distributed Learning Initiative
13501 Ingenuity Drive, Suite 248
Orlando, Florida 32826

10. SPONSOR/MONITOR'S ACRONYM(S)

DANTES/ADLI

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

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution A

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The scope of the ELRR Prototype Systems Integration Plan (SIP) is to provide technical guidance for integrating ELRR Prototype with ADL TLA sandbox components, such as the Authoritative Learner Record Store (LRS), Competency and Skills System (CaSS), and Enterprise Course Catalog (ECC), along with a recommended resource mixture to conduct the integration activities. The intended audiences for this document are developers or engineers who are familiar with TLA standards and core cloud computing principles.

15. SUBJECT TERMS

Learner Records, Systems Integration, Enterprise

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	35	19b. TELEPHONE NUMBER *(Include area code)* 5714804640

Reset

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Document Control Information

Document Name	Systems Integration Plan
Client	Advanced Distributed Learning (ADL) Initiative
Task Order Name	Enterprise Leaner Record Repository
Document Owner	Eric Flamer, Adjoa Adusei-Poku (Deloitte Consulting LLP)
Document Version	1.2
Document Status	Update to Baseline

Document Edit History

Version	Date	Additions/Modifications	Prepared/Revised by
1.0	3/15/2021	Baseline submitted to ADL	Deloitte
1.1	3/29/2021	ADL comments submitted and reviewed with Deloitte team.	ADL
1.2	4/12/2021	Addressed all of ADL's comments related to document structuring, level of detail updates, and syntax. Deloitte added a new section (3.0) to specifically outline interfaces between ADL TLA sandbox and ELRR Prototype components.	Deloitte
1.3	5/6/2021	Accepted ADL QA comments submitted to the Deloitte team.	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
Ashley Howell	Government Technical Point of Contact
Brent Smith	Government Technical Lead
Florian Tolk	Government Solution Engineer

Table of Contents

1.0 Introduction **1**
1.1 Deliverable Purpose 1
1.2 Deliverable Scope 1
2.0 ELRR Prototype Integration Plan and Strategy **2**
2.1 ELRR Overview 2
2.2 ELRR Prototype Deployment Strategy 5
2.3 System Integration Roles and Responsibilities 6
2.4 Assumptions 7
3.0 ELRR Integration Approach with ADL TLA **10**
3.1 Authoritative Learner Record Store (LRS) 10
3.2 Competency and Skills System (CaSS) 11
3.3 Enterprise Course Catalog (ECC) 11
3.4 DATASIM 12
4.0 Appendix **13**
Appendix A – References 13
Appendix B - Hardware / Software 14
Appendix C - Deployment Scripts 16
Appendix C-1 Security, Networking, and Identity and Access Management 16
Appendix C-2 Terraform 17
Appendix C-3 ELRR Storage / Databases 17
Appendix C-4 ELRR Application 24
Appendix C-5 ELRR Presentation 28
Appendix D: Key Terms 31

1.0 Introduction

1.1 Deliverable Purpose

The Enterprise Learner Record Repository (ELRR) is one of three Enterprise Digital Learning Modernization (EDLM) lines of effort supported by the ADL Initiative. Today, learner records for Department of Defense (DoD) personnel are stored in disparate locations, along with inconsistent data formats, which complicates the transport, management, and governance of the learner records across and within DoD organizations.

The goal of the ELRR is to aggregate learner records 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 ELRR Prototype's initial implementation; guidance on the required interfaces between the ELRR Prototype and the ADL Total Learning Architecture (TLA) reference implementation in an Amazon Web Services (AWS) sandbox environment; a corresponding integration approach; and a stakeholder engagement strategy to support a reference implementation of the ELRR Prototype.

The Deloitte Team, comprised of Deloitte practitioners and our Lingatech sub-contractors, will perform system integration activities to support an ELRR Prototype Test and Evaluation (T&E) period during the months of May and June 2021. ELRR Prototype implementation efforts will begin within the ADL's TLA environment starting in May and conclude in August 2021 after the completed government acceptance testing activities. Beyond September 2021, ADL may continue to mature, test and harden the implemented ELRR Prototype in pursuit of the appropriate accreditations required to support additional testing within stakeholder environments using live learner data, which may include integration with other DoD systems.

1.2 Deliverable Scope

The scope of the ELRR Prototype Systems Integration Plan (SIP) is to provide technical guidance for integrating ELRR Prototype with ADL TLA sandbox components, such as the Authoritative Learner Record Store (LRS), Competency and Skills System (CaSS), and Enterprise Course Catalog (ECC), along with a recommended resource mixture to conduct the integration activities. 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 xAPI basic concepts is advantageous for reading this document. Readers should also have a fundamental understanding of how data are 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. An understanding of infrastructure-as-code (IaC) concepts, such as Terraform, is recommended.

2.0 ELRR Prototype Integration Plan and Strategy

The following section outlines the core components of the ELRR Prototype, the ELRR Prototype deployment approach, and an integration strategy with target ADL TLA systems. The ELRR Prototype consists of three core components: Presentation, Application / Data Mesh, and Data Storage. 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 often modeled as repositories as well.) The development team will continue to review the existing TLA core services prior to T&E activities starting in June 2021 in order to assess if additional TLA core services connections are required for the ELRR data model, and configuration details (e.g., API keys, API endpoints, schemas) are accessible. A detailed description of the ELRR Prototype components is defined within **Section 2.1**.

The ELRR Prototype 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 GitHub. Our deployment approach is orchestrated by a continuous integration / continuous delivery (CI/CD) pipeline, which uses industry leading software—such as Terraform, Jenkins, and SonarQube—to operate. 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 ELRR Prototype deployment approach.

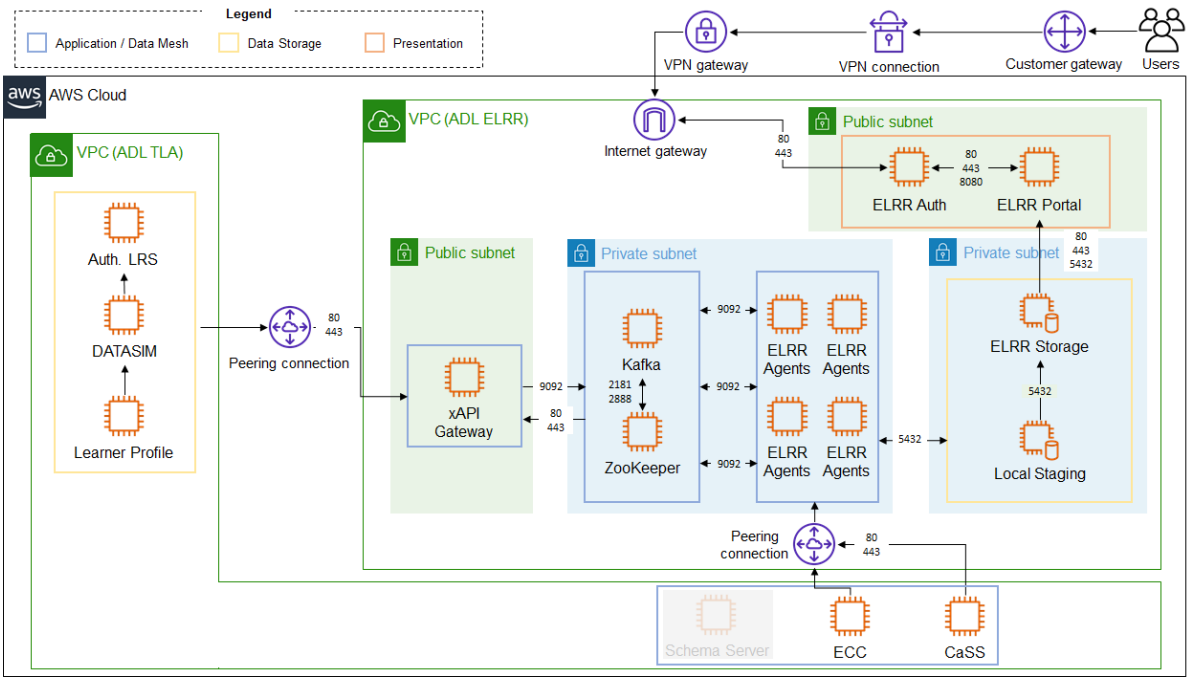
Once the ELRR infrastructure is deployed, the next step is to integrate with source systems within the ADL TLA sandbox. The three ADL TLA source systems used by the ELRR Prototype to develop complete learner profiles include the Authoritative LRS, CASS, and ECC. Additionally, the ELRR Prototype connects with the Data and Training Analytics Simulated Input Modeler (DATASIM) to produce xAPI records to conduct unit testing for the ELRR. Refer to **Section 3.0** for detailed information regarding the ELRR Prototype integration strategy with ADL TLA sandbox.

2.1 ELRR Overview

The ELRR is designed to offer flexibility in automating machine-to-machine connectivity between source LRS systems, while offering learning managers and learners the ability to look at learning progress across the enterprise and view learner details through an ELRR Portal front-end. The data underpinning the ELRR is sourced from multiple source systems, such as an LRS, by custom connectors, known as ELRR Agents, which transform and standardize input learner and competency data. The ELRR Agents are designed to load data from a specific data source and run 'on-demand' when data is queued via Apache Kafka. Once invoked, the ELRR Agents load the data into an initial “Local Staging” database to aggregate the raw data from the various ADL TLA data sources. Once the data have been stored within the Local Staging database, ELRR Agents conduct additional data transformations and validations before loading the data into the ELRR Storage database. The ELRR Storage acts as the authoritative data source for ELRR, and data is presented to DoD learning managers and learners on the ELRR Portal front-end.

Please refer to **Figure 1** below for a visual representation of the components and connections comprising the ELRR. While the figure depicts an AWS-hosted environment, the ELRR can likely run on other major cloud providers (e.g., Azure, Google Cloud Platform) or on-premise environments with minor modifications. A general assessment of what modifications would be needed to prepare ELRR for another cloud provider is included within "*The development team will have access to an AWS cloud environment*" rationale in **Section 2.4 Assumptions**.

***Figure 1** – ELRR Infrastructure*



Zone 1 - ELRR Presentation

The **ELRR Presentation** layer represents the user-facing components of the ELRR to enable users to interact with the aggregated learner data. The front-end offers views for learners, system administrators, and training and career managers with tailored data. The ELRR Presentation layer consists of the following proposed components:

- **ELRR Portal** – The user interface facilitating discovery of aggregated enterprise learner records by learners, training and career managers, and system administrators. This is the web application, which contains two main portals, ELRR Learner and ELRR Admin portals. The ELRR Learner portal enables record consumers (e.g., learners, training and career managers) to view aggregated learner records and to discover enterprise learner record data. To support maintenance and governance of learner records, the ELRR Admin portal provides system administrators a view of imports, endpoints and errors.
- **ELRR Auth -** The enterprise authentication client configured with the ADL TLA reference architecture identity and access management solution, KeyCloak. When a user signs in through KeyCloak their role (e.g., learner, training or career manager, system admin) will dictate what data the user will see on the ELRR Portal. The roles within

KeyCloak ensure all users within the ELRR have the least amount of access to ELRR to conduct their job. For more information regarding KeyCloak, please refer to Appendix A: References "KeyCloak (ADL TLA)".

Zone 2 - ELRR Application / Data Mesh

The **ELRR Application / Data Mesh** contains the business or functional process logic for the ELRR Prototype. This process logic is critical for receiving, transforming, validating, and storing learner record data from ADL TLA systems. Below is a description of the expected common services and how they will be used for the ELRR Prototype:

- **API Gateway** – enables data to be transferred from the Authoritative LRS to the ELRR. The learner records are transferred individually from the Authoritative LRS as JavaScript Object Notation (JSON) to the API Gateway, which routes the learner record to the correct Kafka topic. The API Gateway can consist of many different paths and components that are defined by a configuration file. The xAPI specification allows for validation of the Authoritative LRS learner records, which can improve ELRR data quality by rejecting records that do not match the expected structure. The API Gateway delivers scalability and reliability to the ELRR by adding additional API paths for future ELRR data model enhancements, in addition to being able to handle large amount of simultaneous learner records requests.
- **Kafka / Zookeeper** – provides the streaming service to manage the various data streams from the Authoritative LRS. Kafka stores these data streams on partitions, known as 'Topics', which are aligned to the data source to enable auditability and governance of the source data. The streaming service operates within a publish-subscribe construct and provides a ledger of all communications between ELRR and source systems. The Kafka ledger ensures that any data streams successfully loaded into Kafka will be processed, thus improving ELRR reliability. For more information on Apache Kafka, please refer to **Appendix A: References "Apache Kafka"**.
- **ELRR Agents -** are fine-grained services that execute the ELRR business logic by interacting with the Learner Profile, CaSS and ECC systems. Some of the ELRR Agent(s) are associated with a Kafka 'Topic' and execute their business logic to transform, validate, and load data into ELRR databases, Local Staging and ELRR Storage. Due to the highly specified function, ELRR Agents can rapidly scale to meet high demand for data ingest into the ELRR.

Zone 3 - ELRR Storage / Database

The **ELRR Storage / Database** contains the persistent storage in a relational database management system, comprised of the following planned components:

- **ELRR Local Staging -** stores the consolidated xAPI statements from ADL's Authoritative LRS, according to the TLA MOM standard. Learning event and learner state records are in the JSON format defined for the xAPI (IEEE P9274), in addition to profile, competency data from CaSS, and course metadata from ECC source systems. In

the future, this component will accommodate other DoD organization authoritative learner record stores (e.g., Navy's MyNavy Learning (MNL) or Army's Army Training Information System (ATIS) LRS).

- **ELRR Storage -** is the ELRR application database, which stores the aggregated learner records. The data model is similar to the ELRR Local Staging database; however, it also includes metadata to enhance learner records. The ELRR Portal leverages this database to populate the front-end screens used by learners, training and career managers, and system administrators.

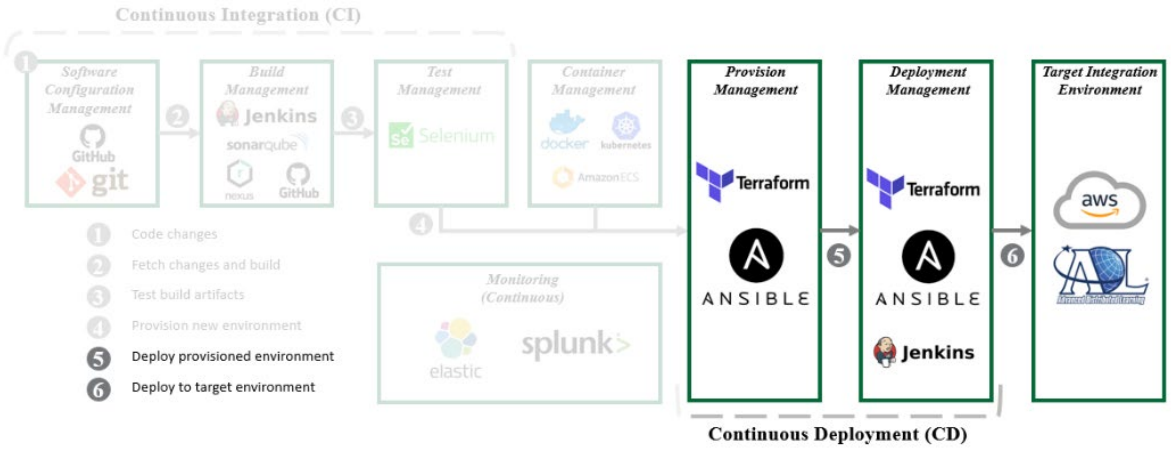
2.2 ELRR Prototype Deployment Strategy

The internal ELRR Prototype application development leverages a robust CI/CD pipeline that enables quality and secure application builds of the ELRR. The CI/CD pipelines offer more consistency and trust that the latest versions of the ELRR will operate as intended within the ADL TLA sandbox. Underpinning the CI/CD pipeline is Jenkins, an industry-leading open source CI/CD pipeline automation tool, to help orchestrate the build, testing, and deployment of applications. Code quality is achieved through SonarQube, performing static code analysis for the ELRR to catch errors within code prior to a deployment.

In alignment with the ELRR development approach, the goal for the CI/CD is to offer a consistent and repeatable experience to install and configure ELRR within an existing ADL TLA sandbox. The software utilized for the ELRR platform has been carefully selected to maintain portability to any cloud environment across the various major cloud providers. At the time of this document release, the ELRR Prototype has only been tested/verified on the AWS platform; though with slight modifications, developers may leverage other cloud providers.

In the future, ELRR will be deployed using IaC scripts with Terraform and configuration management with Ansible, as specified in Steps 5 and 6 in **Figure 2** below. The IaC scripts using Terraform define all facets of the ELRR application from networking, servers, and storage to deliver a repeatable, consistent deployment experience. Once the infrastructure has been built, Ansible uses defined 'playbooks' to specify the installation and configuration of applications. For example, a Terraform IaC script can be used to build the Kafka / Zookeeper server specifications, storage, and networking to run a cluster of servers at the run of a single command. Once Terraform completes the build of the infrastructure, an Ansible playbook starts the Kafka / Zookeeper application via Docker-Compose and conducts additional application configuration, along with unit tests to ensure the application has been successfully deployed. Using Terraform and Ansible together mitigates the risk of user error when deploying the ELRR Prototype. IaC secures the deployment of the infrastructure by easily scripting the specifications of each resource and it can be duplicated across multiple environments.

***Figure 2** – ELRR Prototype CI/CD Pipeline Supporting the Planned System Integration*



2.3 System Integration Roles and Responsibilities

To support the deployment of the ELRR Prototype into the ADL TLA sandbox environment, the following staffing plan, roles and quantity, are accounted for in **Table 1** below.

***Table 1** – ELRR Integration Roles and Responsibilities*

ELRR Prototype Role	ELRR Prototype Responsibilities
Development Engineers (x2)	▪ Verifies the network and firewall rules (e.g., ports) have been opened to ensure communication between ELRR and ADL TLA sandbox components. Refer to Appendix C-1 "Security, Networking, and Identity and Access Management" for a complete list of firewall rules for ELRR. ▪ Maintains and executes deployment scripts for the ELRR components, such as ELRR Portal, ELRR Auth, xAPI Gateway, Kafka / Zookeeper, ELRR Staging, Local Storage, and ELRR Storage. Validates administrator access to each ELRR component. ▪ Maintains and executes installation and configuration of the software for ELRR components. The process is executed by automated Ansible scripts, which install the application with Docker Compose, configure the application and conduct automated unit tests to validate the ELRR components are running as expected. ▪ Troubleshoots issues with related to ELRR deployment, installation, and configuration scripts.
Data Engineers (x2)	▪ Establishes, executes and monitors API connections between Authoritative LRS and the xAPI Gateway. Verifies if the Authoritative LRS requests to xAPI Gateway match the expected data structure.

ELRR Prototype Role	ELRR Prototype Responsibilities
	▪ Designs, configures, and monitors the Kafka / Zookeeper pipelines for each ADL TLA sandbox component connection. Validates that xAPI Gateway and ELRR Agents can send and receive data to Kafka / ZooKeeper. ▪ Establishes, executes and monitors connections between the ELRR Agents and target systems (e.g., Kafka / ZooKeeper, ELRR Local Staging, ELRR Storage, CaSS, ECC). Verifies the ability to query data from CaSS and ECC, post a message to Kafka / ZooKeeper, and load data into ELRR Local Staging and ELRR Storage. ▪ Troubleshoots issues with related to connections between ADL TLA sandbox components and ELRR xAPI Gateway and ELRR Agents, in addition to internal ELRR connections to ELRR Local Staging and ELRR Storage.
Tester (x2)	▪ Conducts unit testing of ELRR components to validate if individual components are functioning as expected. Baseline unit tests are developed to test ELRR component functionality based on PWS requirements. ▪ Executes systems integration test cases that verify the connections between internal ELRR components, in addition to external ADL TLA sandbox components. Verifies end-to-end processes within ELRR from collecting, processing, and storing data from ADL TLA sandbox components and visualizing with in the ELRR Portal. ▪ Captures test results within an Application Lifecycle Management tool for defect tracking and mitigation. Coordinates with the ELRR development team to verify the mitigation of defects.
Project Manager (x1)	▪ Manages scope and schedule as defined in the ELRR technical management work plan. ▪ Supports overall project functional activities, including stakeholder management, risk and issue resolution, and stakeholder communications.

2.4 Assumptions

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

***Table 2** – ELRR Prototype SIP Assumptions*

Assumption	Rationale
The ADL TLA sandbox environment will be stood-up and validated	The ELRR Prototype relies on multiple systems within the ADL TLA sandbox, such as Authoritative LRS and CaSS, to load learner records into ELRR. Since these source systems have separate development teams, installation and configuration of an ADL TLA sandbox is outside the scope of the ELRR development team. Many of the ADL TLA sandbox systems remain in active development so bugs and errors when

Assumption	Rationale
	deploying the applications are likely, which results in troubleshooting and takes away time spent on ELRR Prototype development. The ELRR Prototype development team will work with ADL on a timeline for accessing the ADL TLA sandbox and any necessary credentials to access the environment.
Deloitte will rely on an ADL-provided, ELRR-aligned LRS for system implementation and testing	The TLA LRS, corresponding xAPI profiles/services, and Kafka messaging service represent a significant set of application layer services that are vital to the intended system operation of the TLA implementation within the ADL TLA sandbox. Similar to the assumption rationale above, these systems remain in active development and are complex, which increases the likelihood for bugs during deployment. The development team requested its own dedicated version of the ADL-deployed LRS to lower the complexity of the ELRR Prototype integration, increase the likelihood of timely integration testing, and provide Deloitte with the ability to successfully perform ELRR-specific regression testing as it tracks ELRR anomalies throughout the ELRR Prototype test period.
Learner records in the scope of the ELRR Prototype will conform to the xAPI standard	The development team expects that all planned learner records for use in the ELRR Prototype will match the formatting and standards set forth in the xAPI standard. The ELRR Agent scripts are built based on the xAPI standard format, so any modifications to the xAPI format or other data format will result in additional development scope to accommodate additional formats and standards. ADL will notify the development team of any proposed or upcoming changes to the xAPI standard, and the development team will provide an assessment on the level of effort needed to accommodate the change within a current or future release.
The ELRR Prototype will lack some security controls by design	The ELRR Prototype will conform to a certain degree of security requirements, but ultimately will use simulated learner record data because the key outcome of the prototype is the successful demonstration of the evidentiary chain, rather than full or partial attainment of a future security level. The development team will work with ADL and C5ISR project stakeholders to identify security-related requirements as part of the development process, document these requirements, prioritize them as required.
The development team will have access to an AWS cloud environment	All the ELRR code can be deployed on all major cloud environments (e.g., Azure, Google Cloud Platform, AWS) with minor modifications to the underlying IaC scripts. These minor modifications may include service names (e.g., Elastic Compute Cloud (EC2) to Azure Virtual Machine (VM)) and virtual machine types (e.g., t2.medium vs. Standard_B2s). The following guide has been written specifically for an AWS environment deployment: this document will be specific to AWS machine types, as described in **Appendix B - Hardware / Software - Hardware / Software**. Additionally, cloud providers have inconsistent services that are certified at hardened security environments (Impact Level (IL) 2, IL4, IL5, IL6). For example, at the time of this draft, Azure Kubernetes Service (AKS) is certified up to IL5, while Amazon Elastic Kubernetes Service (EKS) is certified up to IL4. The development team will continue to monitor the Scope in Service pages, linked within **Appendix A – References "Services in Scope",** for the major cloud environments, and will work

Assumption	Rationale
	with ADL to support the transition from the AWS environment to EDLM Azure IL 4 environment.
Additional coordination with the ECC Prototype Development team is anticipated	The ECC Prototype development team will also be integrating with the ADL TLA sandbox environment during the same timeline as ELRR (e.g., May – August 2021). While the ECC is a target source system, development timelines may not align for the ECC to be integrated as part of this ELRR release. The ELRR development team, along with ADL, will continue to coordinate with the ECC development team prior to and as part of system integration activities to align on an implementation timeline.
Schema Server components remain undetermined and outside of the current scope	Across both ELRR and ECC Prototype projects, the concept of an enterprise service bus or middleware service for linked data remains a work in progress without a comprehensive timeline for design, development, or implementation into the ADL TLA reference implementation or corresponding ADL TLA sandbox environment. The ECC Prototype development team will implement an initial Experience Schema Service as part of the ECC Prototype, which will not support ELRR schema server use cases.

3.0 ELRR Integration Approach with ADL TLA

The ELRR Prototype relies on establishing connections to existing components within the ADL TLA sandbox to acquire, aggregate, validate, and display consolidated learner records. The three ADL TLA source systems used by the ELRR Prototype to develop complete learner profiles include the Authoritative LRS, CaSS, and ECC. If an organization has multiple components, such as multiple LRSs, the ELRR is able to scale with connections to additional systems. The ELRR communicates with the ADL TLA sandbox systems through the xAPI Gateway, which manages all of the learner record requests into the ELRR. To conform with modern application development standards, the xAPI Gateway utilizes a RESTful API, which is compatible with the existing ADL TLA sandbox components. The xAPI Gateway conducts initial data quality checks and only accepts requests from components deployed to the ADL's TLA sandbox that meet pre-determined schema standards, such as xAPI standards.

While the xAPI Gateway is the external facing interface to the ADL TLA sandbox components, there are internal components within ELRR, Kafka / Zookeeper and ELRR Agents, that queue the requests from the xAPI Gateway and execute business logic to transform, validate, and store the learner records within the Local Storage and ELRR Storage databases. To assist with unit testing, the ELRR connects with DATASIM to produce xAPI records to verify the ELRR application is functioning as expected.

3.1 Authoritative Learner Record Store (LRS)

The Authoritative LRS is a core data source within the ADL TLA sandbox that stores consolidated xAPI statements from multiple edge systems, known as 'noisy' LRSs. The xAPI statements track a learner's progress and actions throughout courses and quizzes towards competency mastery. The data ingested is standardized according to the TLA MOM standard.

The Authoritative LRS provides critical live data regarding DoD learner's progress towards associated competency areas. Assuming organizations may have multiple LRS systems, the ELRR will connect to the source Authoritative LRS systems, aggregate DoD learner's data into a centralized database and display learner metrics on the ELRR Portal user interface. The ELRR receives learner records from the Authoritative LRS via requests to the ELRR xAPI Gateway. Once a request is received, the ELRR xAPI Gateway will queue the response within the respective Topic within the ELRR Kafka instance. Based on the Topic the data is queued within on ELRR Kafka / ZooKeeper instance, the respective ELRR Agent will upload the record to the ELRR Storage database.

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

Table 3 – *Authoritative LRS / ELRR System Interfaces*

ELRR Component	Interfaces	Connection Duration	Connection Frequency
ELRR xAPI Gateway	API Keys; LRS REST response schema; LRS REST response header; LRS IP address / VPC; ELRR xAPI Gateway endpoint URL	Ephemeral	On-demand

ELRR Component	Interfaces	Connection Duration	Connection Frequency
ELRR Kafka	API Keys; Kafka API Broker; Kafka Topic	Ephemeral	On-demand
ELRR Agents	Kafka Topic	Ephemeral	On-demand

3.2 Competency and Skills System (CaSS)

CaSS enables users and other systems to define, store, manage, and access data objects called "competencies" that are organized into structured collections called "frameworks", which report, store, and retrieve assertions about the competencies held by an individual. CaSS acts as a data source into the ELRR Local Staging Area to build a robust metadata profile for learners. ELRR Agents will transform and standardize competency data from CaSS. The ELRR will connect to CaSS through a VPC Peering Connection. ELRR Agents will connect to CaSS to transform standardized user competency data and store data in the ELRR Local Staging Area.

The identified system interfaces between CaSS and ELRR are depicted within **Table 4** below.

Table 4 – *CaSS / ELRR System Interfaces*

ELRR Component	Interfaces	Connection Duration	Connection Frequency
ELRR Agents	VPC Gateway Endpoint; Kafka Topics	Ephemeral	On-demand
ELRR Local Staging Area	API Endpoint	Ephemeral	On-demand

3.3 Enterprise Course Catalog (ECC)

The ECC is a learning experience discovery service designed to aggregate metadata describing learning experiences from various internal sources as well as external sources. The ELRR will connect to the ECC through a VPC Peering Connection. ELRR Agents will connect to the ECC's API endpoint to retrieve course details based on courses the user has completed from the ECC Experience Index Service.

The identified system interfaces between the ECC and ELRR are depicted within **Table 5** below.

Table 5 – *ECC / ELRR System Interfaces*

ELRR Component	Interfaces	Connection Duration	Connection Frequency
ELRR Agents	VPC Gateway Endpoint; Kafka Topics	Ephemeral	On-demand
ELRR Local Staging Area	API Endpoint	Ephemeral	On-demand

3.4 DATASIM

If a stakeholder organization does not have an operational LRS capturing a high volume of learner records, the DATASIM application can be used to simulate xAPI statements for the LRS. The development team uses DATASIM to facilitate unit testing for the ELRR.

The identified system interfaces between DATASIM and the ELRR are depicted within **Table 6** below.

***Table 6** – DATASIM / ELRR System Interfaces*

ELRR Component	Interfaces	Connection Duration	Connection Frequency
ELRR xAPI Gateway	API Keys; LRS REST response schema; LRS REST response header; LRS IP address / VPC; ELRR xAPI Gateway endpoint URL	Ephemeral	On-demand
ELRR Kafka	API Keys; Kafka API Broker; Kafka Topic	Ephemeral	On-demand
ELRR Agents	Kafka Topic	Ephemeral	On-demand

4.0 Appendix

Appendix A – References

Name	Description	Location
ADL TLA Quick Start	Provides an overview, including technical specifications, for the ADL.	https://adlnet.gov/guides/tla/
Apache Kafka	The Apache Kafka platform provides a distributed publish/subscribe messaging topology built around streams of different data topics.	https://adlnet.gov/guides/tla/service-definitions/TLA-Reference-Implementation.html#kafka-gateway
KeyCloak (ADL TLA)	Details on ADL TLA reference implementation of KeyCloak for identity and access management.	https://adlnet.gov/guides/tla/service-definitions/Authentication-and-Identity-Management.html#keycloak
OpenAPI Specification (OAS)	The OpenAPI Specification (OAS) defines a standard, language-agnostic interface to RESTful APIs, which allows both humans and computers to discover and understand the capabilities of the service without access to source code, documentation, or through network traffic inspection.	https://swagger.io/specification/
Services in Scope	Cloud providers must meet strict security requirements set by US Department of Defense, from IL2 through IL6. While cloud providers have dozens of services, only certain services have passed audits to verify they meet the IL requirements. The linked web pages are updated periodically as services are verified for ILs.	**AWS:** https://aws.amazon.com/compliance/services-in-scope/ **Azure:** https://docs.microsoft.com/en-us/azure/azure-government/compliance/azure-services-in-fedramp-auditscope#azure-government-services-by-audit-scope
xAPI Standard (IEEE P9274)	This Standard describes a JSON data model format and a RESTful APIs for communication between Activities experienced by an individual, group, or other entity and an LRS.	https://standards.ieee.org/project/9274_1_1.html

Appendix B - Hardware / Software

ELRR Prototype Hardware

Table 7 – ELRR Hardware

Component	Service	Type	Operating System	Storage (GB)	Count
API Gateway	EC2	t2.medium	Ubuntu 18.04	20 (gp2-ssd) [weekly snapshots]	1
ELRR Portal	EC2	t2.medium	Ubuntu 18.04	20 (gp2-ssd) [weekly snapshots]	1
ELRRAUTH	EC2	t3.large	Ubuntu 18.04	8 (gp2-ssd)	1
DATASIM	EC2	t2.small	Ubuntu 18.04	20 (gp2-ssd)	1
ELRR Local Staging Area Database	EC2	t3.xlarge	Ubuntu 18.04	100 (gp2-ssd)	1
ELRR Database	EC2	t2.medium	Ubuntu 18.04	100 (gp2-ssd)	1
Jenkins	EC2	t2.medium	Ubuntu 18.04	65 (gp2-ssd)	1
Jenkins Slave	EC2	t2.medium	Ubuntu 18.04	8 (gp2-ssd)	1

ELRR Prototype Software

Table 8 – ELRR Software

Configuration Item (CI)	Name	Version
Security / Authentication	KeyCloak	7.0.x
	Node Version Manager (NVM)	0.35.3
Application	Nginx	1.14.0
	Elasticsearch	7.11
	Kafka	2.6.x
	Django	3.1.x
	Spring Boot	2.4.x

Configuration Item (CI)	Name	Version
Data / Storage	Postgres	13.1
Container	Docker	20.10.x
Container Orchestration	Kubernetes	1.19.x
Operating System	Linux/Ubuntu	18.04
Administration	Terraform	3.32.0
	Ansible	2.10.5
	Jenkins	2.222.1
Languages	Java (JDK)	14
	Python	3.9.x

Appendix C - Deployment Scripts

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

The necessary firewall ports for ELRR, based on the specified component/infrastructure, is outlined within **Table 9** below.

***Table 9** – ELRR Firewall Ports*

Component	Ports (Ingress)	Ports (Egress)
xAPI Gateway	:22 (per IP) :80 Public :443 Public :9092 (per IP)	All Traffic
ELRR Portal	:22 (per IP) :80 Public :443 Public :8080 (per IP) :5432 (per IP)	All Traffic
ELRRAUTH	:22 (per IP) :80 Public :443 Public	All Traffic
Kafka / Zookeeper	:22 (per IP) :80 Public :443 Public :9092 (per IP) :2181 (per IP) :2888 (per IP)	All Traffic
ELRR Agents	:22 (per IP) :80 (per IP) :443 (per IP) :9092 (per IP) :5432 (per IP)	All Traffic
DATASIM	:22 (per IP) :80 Public :443 Public :9091 (per IP) :9090 (per IP)	All Traffic
ELRR Local Staging	:22 (per IP) :5432 (per IP)	All Traffic
ELRR Storage	:22 (per IP) :5432 (per IP)	All Traffic
Jenkins / Jenkins Agent	:22 (per IP) :80 Public :443 Public :8080 (per IP) :5986 (per IP)	All Traffic

Appendix C-2 Terraform

The ELRR will be deployed with IaC approach using Terraform. Terraform is a cloud agnostic deployment tool that can deploy any resources described in the Terraform templates to most major cloud providers. Terraform CLI is used to deploy templates into specified cloud accounts. Terraform will create necessary resources to host ELRR components in a highly available and secure way. Additionally, scripting environment definitions and components mitigates user error when creating resources via the console, while reducing the amount of time to construct the environment. Once Terraform templates have been created, the Deployment Engineer will install Terraform and run the following commands to initialize, plan and apply Terraform.

The first command when running Terraform should always be 'terraform init'. Terraform Init initializes the working directory that contain Terraform scripts whether it was created locally or cloned from a Github or another version control repository.

```
# initialize the working directory

terraform init
```

Next, once Terraform configuration scripts are complete and the working directory has been initialized, it is critical to run *'terraform plan'* prior to running *'terraform apply'*. *'Terraform plan'* lists out each resource it is planning on creating based on what is in the Terraform scripts. Terraform analyzes the scripts and displays the output on the command line interface for review. If there are any errors in the scripts, Terraform will list each error with specific file name and line number for reference.

```
# plan out the resources that will be created based on templates

terraform plan
```

Finally, after *'terraform plan'* runs without any errors and the system operator is satisfied with the resources to be created from the scripts, running *'terraform apply'* will verify and start creating resources. Similar to *'terraform plan'*, *'terraform apply'* will do a final check of the scripts and require the system operator to confirm this is the action to be performed in the environment with a simple yes or no answer in the command line. It will tail the log and status directly on the command line interface.

```
# apply Terraform templates and start creating defined resources

terraform apply
```

Appendix C-3 ELRR Storage / Databases

The ELRR system relies on two core databases, ELRR Local Staging Area and ELRR Profile, to support learner progress and other associated metadata from ADL TLA, such as Learner Profiles and Competency records via CaSS. These databases serve as an enterprise aggregation of

transactional data for learners across multiple LRSs. The ELRR platform utilizes DATASIM to generate synthetic xAPI data for unit testing of the API Gateway.

ELRR Local Staging Area Database

The ELRR Local Staging Area database offers centralized storage of all xAPI requests from connected Authoritative LRS systems. The Authoritative LRS systems send requests to the ELRR API Gateway, which routes to the ELRR Local Staging Area database, parses the request, and updates the xAPI request body to conform to the xAPI JSON IEEE P9274 format. The ELRR Local Staging Area in the ELRR reference implementation runs on PostgreSQL v13.1

ELRR Local Staging Area Installation

Once the virtual machine has been started, execute the below bash/shell script to install the necessary components for Docker.

```
#!/bin/bash

sudo apt-get update
sudo apt-get install docker
sudo apt-get install docker-io
sudo groupadd docker
sudo usermod -aG docker $USER
```

ELRR Local Staging Area Configuration

Based on the ELRR reference implementation, Docker Compose will be used to build a PostgreSQL database on the virtual machine. Create a new file called `docker-compose.yml` and copy/paste the following code.

```
## docker-compose.yml

version: '3.0'
services:
    postgres:
        image: postgres:13.1
        restart: always
        environment:
          - POSTGRES_USER=${POSTGRES_USER_STAGING} #update based on secrets defined in .env file
          - POSTGRES_PASSWORD=${POSTGRES_PASS_STAGING} #update based on secrets defined in .env file
        logging:
          options:
            max-size: 10m
```

```
            max-file: "3"
        ports:
          - '5438:5432'
        volumes:
          - ./postgres-data:/var/lib/postgresql/data
```

Within the same directory as the above `docker-compose.yml` run the following command:

```
sudo docker-compose up -d
```

ELRR Local Staging Area Create Tables

Once the container is up and running, setting-up of the database and creating the reference tables can begin, along with their specified schemas, for the ELRR Local Staging Area database. The table schemas will be based on the xAPI JSON IEEE P9274 format. Depending on the number of source Authoritative LRS systems, there could be multiple tables per ELRR Local Staging Area database.

ELRR Profile Database

The ELRR Profile database offers an enterprise view into xAPI requests from all connected Authoritative LRS systems, along with enriched data from CaSS. The ELRR Profile database subscribes to the ELRR Local Staging Area database's topic within Kafka, which triggers a call from the API Gateway to move records from the ELRR Local Staging Area to the ELRR Profile database. Once the record is within the ELRR Profile database, a separate request is sent to CaSS with a unique learner identifier to request additional information, which is then appended to the learner record within the ELRR Profile database. The records within the ELRR Profile database persist for analysis and review by learning managers and individual learners via the ELRR Portal. The ELRR Profile database in the ELRR reference implementation runs on Postgres v13.1.

ELRR Profile Installation

```
#!/bin/bash

sudo apt-get update
sudo apt-get install docker
sudo apt-get install docker-io
sudo groupadd docker
sudo usermod -aG docker $USER
```

ELRR Profile Configuration

Based on the ELRR reference implementation, Docker Compose will be used to build a PostgreSQL database on the virtual machine. Create a new file called `docker-compose.yml` and copy/paste the following code.

```
## docker-compose.yml

version: '3.0'
services:
    postgres:
        image: postgres:13.1
        restart: always
        environment:
          - POSTGRES_USER=${POSTGRES_USER_PROFILE} #update based on secrets defined in .env file
          - POSTGRES_PASSWORD=${POSTGRES_PASS_PROFILE} #update based on secrets defined in .env file
        logging:
          options:
            max-size: 10m
            max-file: "3"
        ports:
          - '5438:5432'
        volumes:
          - ./postgres-data:/var/lib/postgresql/data
```

Within the same directory as the above `docker-compose.yml` run the following command:

```
sudo docker-compose up -d
```

ELRR Profile Create Tables

Once the container is up and running, setup of the and creation of the reference tables can begin, along with their specified schemas, for the ELRR Local Staging Area database. The table schemas will be based on the IEEE ELR 1484.2 format.

DATASIM (Testing / Validation)

To assist with unit testing of the ELRR, the ELRR reference implementation uses DATASIM to simulate xAPI data that can be leveraged within the ELRR pipeline to validate data architecture and performance testing of the application.

DATASIM Installation

```
#!/bin/bash
```

```
2
3  if [ ! -d "/home/datasim/code/datasim" ]
4  then
5
6  if [ ! -d "/tmp/ssm" ]
7  then
8  echo "ssm Folder not found, making it" >> /home/ubuntu/startlog.txt
9  sudo mkdir /tmp/ssm
10 fi
11 cd /tmp/ssm
12 wget https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/debian_amd64/amazon-ssm-agent.deb
13 sudo dpkg -i amazon-ssm-agent.deb
14 sudo systemctl enable amazon-ssm-agent
15
16
17 echo "making home folder" >> /home/ubuntu/startlog.txt
18
19 if [ ! -d "/home/datasim" ]
20 then
21 echo "Datasim Folder not found, making it" >> /home/ubuntu/startlog.txt
22 sudo mkdir /home/datasim
23 fi
24 cd /home/datasim
25 echo "runningupdate" >> /home/ubuntu/startlog.txt
26 sudo apt update
27 echo "installing openjdk and curl" >> /home/ubuntu/startlog.txt
28 sudo apt-get install openjdk-8-jre -y >> /home/ubuntu/startlog.txt
29 sudo apt-get install curl -y >> /home/ubuntu/startlog.txt
30 if [ ! -d "/home/datasim/code" ]
31 then
```

```
echo "code Folder not found, making it" >> /home/ubuntu/startlog.txt
sudo mkdir ./code
fi
sudo chmod -R 777 code >> /home/ubuntu/startlog.txt
cd ./code >> /home/ubuntu/startlog.txt
echo "run install script 1" >> /home/ubuntu/startlog.txt
sudo curl -O https://download.clojure.org/install/linux-install-1.10.1.754.sh
sudo chmod +x linux-install-1.10.1.754.sh
sudo ./linux-install-1.10.1.754.sh >> /home/ubuntu/startlog.txt
echo "sleeping..." >> /home/ubuntu/startlog.txt
sleep 20
#git clone https://github.com/nvm-sh/nvm.git ~/.nvm >> /home/ubuntu/startlog.txt
curl -sL https://raw.githubusercontent.com/creationix/nvm/v0.35.3/install.sh -o install_nvm.sh
echo "run install script 2" >> /home/ubuntu/startlog.txt
sudo chmod +x install_nvm.sh >> /home/ubuntu/startlog.txt
echo "changed mod" >> /home/ubuntu/startlog.txt
. ./install_nvm.sh >> /home/ubuntu/startlog.txt
echo "sleeping..." >> /home/ubuntu/startlog.txt
sleep 20
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This loads nvm bash_completion
echo "downloaded nvm binaries" >> /home/ubuntu/startlog.txt
source ~/.bashrc >> /home/ubuntu/startlog.txt
#for pid in `lslocks -rn | grep /var/lib/dpkg/lock|awk '{print $2}'`;
#do
#    sudo kill $pid;
#done
echo "installing nvm" >> /home/ubuntu/startlog.txt
nvm install 14.15.1 >> /home/ubuntu/startlog.txt
```

```
echo "nvm installed" >> /home/ubuntu/startlog.txt
nvm use 14.15.1 >> /home/ubuntu/startlog.txt
echo "installing git make and wrap..." >> /home/ubuntu/startlog.txt
sudo apt-get install git -y >> /home/ubuntu/startlog.txt
sudo apt-get install make -y >> /home/ubuntu/startlog.txt
sudo apt-get install rlwrap -y >> /home/ubuntu/startlog.txt
echo "git make and wrap installed" >> /home/ubuntu/startlog.txt

echo "cloning ELRR Repo..." >> /home/ubuntu/startlog.txt
git clone https://github.com/US-ELRR/Misc.git >> /home/ubuntu/startlog.txt
echo "cloned ELRR Repo; cloning DATASIM repo..." >> /home/ubuntu/startlog.txt
git clone https://github.com/yetanalytics/datasim.git >> /home/ubuntu/startlog.txt
cd datasim/
touch .lein-env && echo "{:credentials generic:generic }" > .lein-env
echo "cloned DATASIM repo. Added credential file" >> /home/ubuntu/startlog.txt
cd ..
echo "cloning DATASIM-UI repo..." >> /home/ubuntu/startlog.txt
git clone https://github.com/yetanalytics/datasim-ui.git >> /home/ubuntu/startlog.txt
echo "cloned DATASIM-UI repo" >> /home/ubuntu/startlog.txt

echo "Updating DATASIM/DATASIM-UI config with Public" >> /home/ubuntu/startlog.txt
export PUBLIC_DNS=`curl http://169.254.169.254/latest/meta-data/public-hostname 2>/dev/null`
sudo sed -i 's/localhost/'$PUBLIC_DNS'/g' /home/datasim/code/datasim-ui/dev.cljs.edn
sudo sed -
i 's/localhost/'$PUBLIC_DNS'/g' /home/datasim/code/datasim/src/server/com/yetanalytics/datasim/server.clj
echo "Creating DATASIM bundle..." >> /home/ubuntu/startlog.txt
cd datasim/
sleep 2
make clean >> /home/ubuntu/startlog.txt
sleep 2
```

```
make bundle >> /home/ubuntu/startlog.txt
sleep 90
echo "Created DATASIM bundle. Launching DATASIM API server..." >> /home/ubuntu/startlog.txt
screen -dm -S "DATASIM" make server
sleep 60
echo "DATASIM API server launched." >> /home/ubuntu/startlog.txt
cd ../datasim-ui/
echo "Launching DATASIM-UI FIG..." >> /home/ubuntu/startlog.txt
screen -dm -S "DATASIM-UI-FIG" make fig
sleep 60
echo "DATASIM-UI FIG launched. Launching DATASIM-UI SASS server..." >> /home/ubuntu/startlog.txt
screen -dm -S "DATASIM-UI-SASS" make build-sass
sleep 60
echo "DATASIM-UI SASS server launched." >> /home/ubuntu/startlog.txt
fi
```

Appendix C-4 ELRR Application

The core of the ELRR is an enterprise API gateway (referred here as the xAPI Gateway), that handles requests and responses from ADL TLA core systems (e.g., CaSS, Authoritative LRS, Learner Profiles, ECC). The incoming requests are funneled through secure data pipelines / topics within Apache Kafka messaging bus. The Kafka cluster has the ability to scale to meet the demand of a production learning system with thousands of concurrent data streams / topic requests. The requests are brokered from Kafka to the API Gateway where the requests are routed to the appropriate source system.

xAPI Gateway

The xAPI Gateway is the API gateway for the ELRR application. The xAPI Gateway contains the API specification that defines the schema and paths associated with the ELRR REST API design.

API Gateway Installation

The ELRR leverages a lightweight NGINX API Gateway to handle a high-velocity of REST API requests, while offering ease of maintainability. The ELRR reference implementation leverages Docker to install and run the NGINX as an API gateway.

```
#!/bin/bash

sudo apt-get update
sudo apt-get install docker
sudo apt-get install docker-io
sudo groupadd docker
sudo usermod -aG docker $USER
sudo apt-get install nginx
```

API Gateway Configuration - API Specification

Once the NGINX server has been installed and is running, it is necessary to define the API specification that defines how the API Gateway will handle incoming requests. An industry standard specification is the OpenAPI Specification (OAS), which specifies the API paths, request / response schemas, and status codes. For more information on OAS, refer to **Appendix A – References "OpenAPI Specification (OAS)"**. The API specification file allows for fine-grained control on form validation where requests for meeting specific criteria (e.g., correct timestamp format, integers vs. float) will be granted a response, therefore improving data quality within ELRR. The files are written in *YAML Ain't Markup Language* (YAML) and can be transferred to most major API Gateway tools, if a developer does not wish to use the NGINX API Gateway within the reference implementation. Another benefit of OAS is that documentation of the API is automatically generated, which assists with readability and maintainability of the ELRR solution.

API Gateway Configuration - Security and Authentication

In future development, the ELRR API Gateway will only accept requests from authenticated sources by using protected API keys. Limiting requests to sources with API keys secures the ELRR application from being attacked by malicious actors. To further combat against external malicious actors, the ELRR will employ rate limiters to mitigate the impact of a Distributed Denial of Service (DDoS) attack on the API Gateway. The xAPI Gateway reference NGINX architecture enables the ability to apply rate limits to specific requests, so authenticated requests from an expected API key would have a higher rate limit than a request from a non-authenticated source.

ELRR Kafka / Messaging Bus

The ELRR reference implementation utilizes Kafka as the messaging bus between core ADL TLA systems and the ELRR platform. The implementation is a cluster of brokers and clients that connect to source systems via individual high-performance Transmission Control Protocol (TCP) that handles the requests/responses. Kafka ensures that on a specific broker TCP connection

requests / responses are handled on a first-come-first-serve basis. A Kafka broker can handle multiple TCP connections from various sources simultaneously, within reasonable limits.

ELRR Kafka / Messaging Bus Installation

For development purposes, a clustered Kafka deployment will be used for the ELRR. An additional pre-requisite for Kafka is Apache ZooKeeper to assist with the management of distributed requests / responses across the Kafka brokers. The initial deployment approach involves using Docker Compose to stand-up a virtual cluster on a single VM, an example of the shell script and docker-compose.yml file is below.

```
#!/bin/bash


sudo apt-get update
sudo apt-get install docker
sudo apt-get install docker-io
sudo groupadd docker
sudo usermod -aG docker $USER
sudo curl -
L "https://github.com/docker/compose/releases/download/1.28.5/docker-
compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

Create a new file called `docker-compose.yml` with the following code* **Note:** the ELRR Development Team is exploring leveraging existing ADL TLA Kafka implementation to utilize defined security and control parameters.

```

## docker-compose.yml

version: '3.0'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
  kafka-1:
```

```
    image: wurstmeister/kafka
    ports:
      - "9095:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: kafka1.test.local
      KAFKA_ADVERTISED_PORT: 9095
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LOG_DIRS: /kafka/logs
      KAFKA_BROKER_ID: 500
      KAFKA_offsets_topic_replication_factor: 3
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - ${KAFKA_DATA}/500:/kafka

  kafka-2:
    image: wurstmeister/kafka
    ports:
      - "9096:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: kafka2.test.local
      KAFKA_ADVERTISED_PORT: 9096
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LOG_DIRS: /kafka/logs
      KAFKA_BROKER_ID: 501
      KAFKA_offsets_topic_replication_factor: 3
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - ${KAFKA_DATA}/501:/kafka

  kafka-3:
```

```
  image: wurstmeister/kafka
  ports:
    - "9097:9092"
  environment:
    KAFKA_ADVERTISED_HOST_NAME: kafka1.test.local
    KAFKA_ADVERTISED_PORT: 9097
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_LOG_DIRS: /kafka/logs
    KAFKA_BROKER_ID: 502
    KAFKA_offsets_topic_replication_factor: 3
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - ${KAFKA_DATA}/502:/kafka
```

Within the same directory as the above `docker-compose.yml` run the following command:

```
sudo docker-compose up -d
```

ELRR Kafka / Messaging Bus Topics

Once the Kafka server has been installed and is running, the next step is to build the topics associated for the ELRR. These topics are associated with the portions of the ELRR data model, and based on the request, multiple data sources (e.g., Authoritative LRS, CaSS), can publish to the same topic. The xAPI Gateway 'subscribes' to various topics, which enables a scalable, durable messaging system send / receive the stream of requests / responses from the back-end services. When a new message is published to an ELRR topic the information is stored within Kafka's storage and can be queried for further analysis and troubleshooting. The duration the records are stored within the Kafka brokers can be tuned based on an organizations data retention policy.

Appendix C-5 ELRR Presentation

The ELRR Portal will serve as the presentation layer that enables learners and record managers the ability to view their learning progress. Additionally, learning managers may access an administration application to update specific learner records.

ELRR Portal

The ELRR Portal represents the user-facing interface that facilitates the discovery of aggregated enterprise learner records by learners and observers/reviewers. The ELRR Portal will serve as the web application that enables record consumers (learners, managers, and observers/reviewers) to discover enterprise learner record data.

ELRR Portal Installation (NGINX)

```
#!/bin/bash

sudo apt-get update
sudo apt-get install docker
sudo apt-get install docker-io
sudo groupadd docker
sudo usermod -aG docker $USER
sudo apt-get install nginx
```

ELRR Portal Installation (Spring Boot)

```
package com.example.springboot;

import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!";
    }

}
```

Run the following command to run Spring Boot:

```
./mvnw spring-boot:run
```

ELRR Portal Configuration

After installation of the Nginx and Spring Boot framework is complete, application classes can be configured and added as needed. Unit testing can also be created with the implementation of the Spring Boot framework.

Appendix D: Key Terms

Table 10 below summarizes the acronyms referenced in this document.

***Table 10** – Acronyms*

Acronym	Term
ADL	Advanced Distributed Learning
AKS	Azure Kubernetes Service
API	Application Programming Interface
ATIS	Army Training Information System
AWS	Amazon Web Services
CaSS	Competency and Skills System
CI/CD	Continuous Integration/Continuous Deployment
DATASIM	Data and Training Analytics Simulated Input Modeler
DDoS	Distributed Denial of Service
DoD	Department of Defense
EC2	Elastic Compute Cloud
ECC	Enterprise Course Catalog
EDLM	Enterprise Digital Learning Modernization
EKS	Elastic Kubernetes Service
ELRR	Enterprise Learner Record Repository
IAC	Infrastructure as Code
IAM	Identity Access Management
IL	Impact Level
JSON	JavaScript Object Notation
LRP	Learning Record Provider
LRS	Learner Record Store
MNL	My Navy Learning
MOM	Master Object Model
MVP	Minimum Viable Product

Acronym	Term
OAS	OpenAPI Specification
SIP	Systems Integration Plan
TCP	Transmission Control Protocol
TLA	Total Learner Architecture
VM	Virtual Machine
YAML	YAMLAin't Markup Language