Data Management Environment Architecture

Design Document

Version 1.0

2/9/2024

# Version History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version Number | Implemented By | Revision Date | Approved By | Approval Date | Description of Change |
| 1.0 | Yuri Dinh | 2/12/2024 |  |  | Initial Version |

Table of Contents

[Version History 2](#_Toc158385279)

[1 Introduction 4](#_Toc158385280)

[1.1 Purpose of this document 4](#_Toc158385281)

[1.2 Background 4](#_Toc158385282)

[2 Architecture Design 8](#_Toc158385283)

[2.1 Technology Stack 8](#_Toc158385284)

[3 Software Interfaces 9](#_Toc158385285)

[4 Software Design 10](#_Toc158385286)

[4.1 Introduction 10](#_Toc158385287)

[4.2 API Server Design 10](#_Toc158385288)

[4.2.1 Common Frameworks 12](#_Toc158385289)

[4.2.2 Scheduled Task Design 14](#_Toc158385290)

[4.3 Command Line Interface Design 16](#_Toc158385291)

[4.4 User Interface Design (Web application) 16](#_Toc158385292)

[4.5 System Integration 17](#_Toc158385293)

[4.5.1 Integrated Systems 17](#_Toc158385294)

[4.5.2 iRODS Integration 17](#_Toc158385295)

[4.5.3 S3 Object Store 17](#_Toc158385296)

[4.5.4 LDAP Integration 17](#_Toc158385297)

[4.5.5 Globus Integration 17](#_Toc158385298)

[5 Database Design 19](#_Toc158385299)

[5.1 iRODS Schema 19](#_Toc158385300)

[5.2 DME Table Design 19](#_Toc158385301)

[5.3 Materialized View Design 21](#_Toc158385302)

# Introduction

## Purpose of this document

The NCI Data Management Environment (DME) offers open-ended storage and management of scientific research datasets to the NCI community and allows users to manage the transfer, access, and data sharing across disparate systems securely and efficiently. The purpose of this document is to record the current architecture and design of the Data Management Environment.

## Background

Users can access the NCI Data Management Environment (DME) programmatically by using the DME REST API. The REST API is an HTTP interface to DME. The REST API is the core of DME where the users and other systems can integrate with DME using the standard HTTP requests. The DME Web Application is a web-based user interface for registering, managing, and transferring the data stored in DME. The DME Command Line Interface provides commands for a set of DME services to issue commands or build scripts to perform DME tasks. Both the DME Web Application and the DME Command Line interface utilize the DME REST API.

The DME API server architecture is a multilayered architecture to support clear separation of concerns and boundaries among the various components of the server. The architecture includes the following six horizontal layers.

1. REST Web service layer
2. Batch layer
3. Business service layer
4. Application service layer
5. Integration layer
6. DAO layer

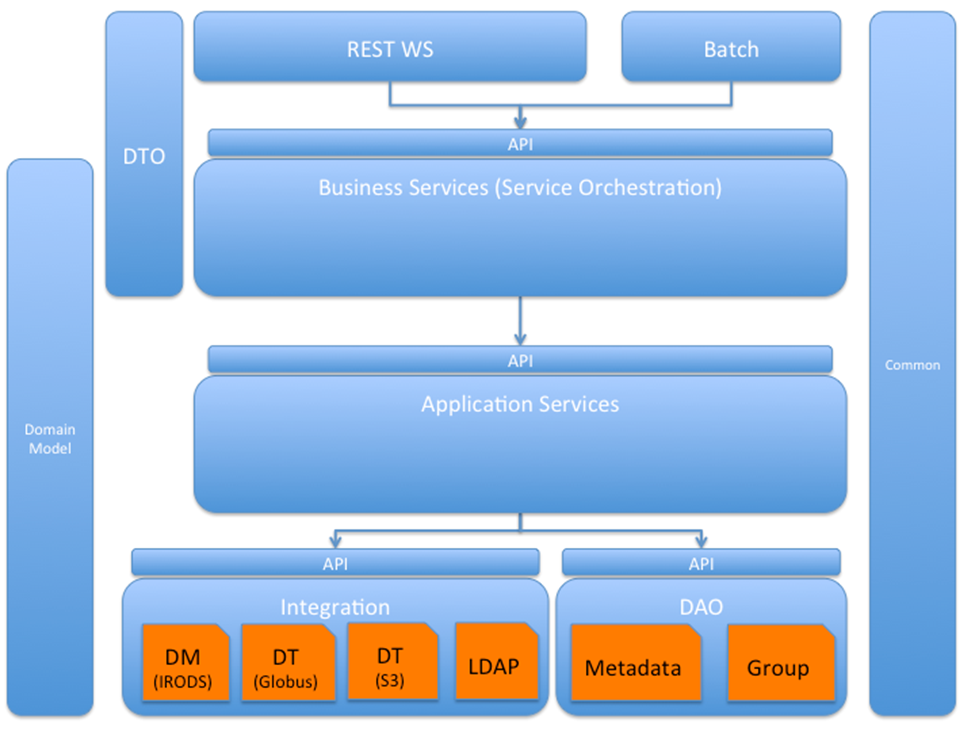
Each horizontal layer is comprised of 2 OSGi bundles (dynamically loadable JAR), one contains the layer’s API and the other contains the API’s implementation. A layer’s API bundle is configured (via a manifest file) to export (or make available) the API interface packages. A layer’s implementation bundle is configured to hide all its code. This approach has the following advantages:

1. It ensures layers are communicating with each other via APIs since the implementation is physically not visible outside the bundle.
2. It ensures circular dependencies avoided among the bundles and thus promotes the architecture reusability.
3. It enables a hot deployment of implementation code, i.e. swapping a layer’s implementation bundle in runtime without stopping the server.

The architecture also includes the following 3 vertical layers, which include POJO code that is used by multiple horizontal layers to communicate.

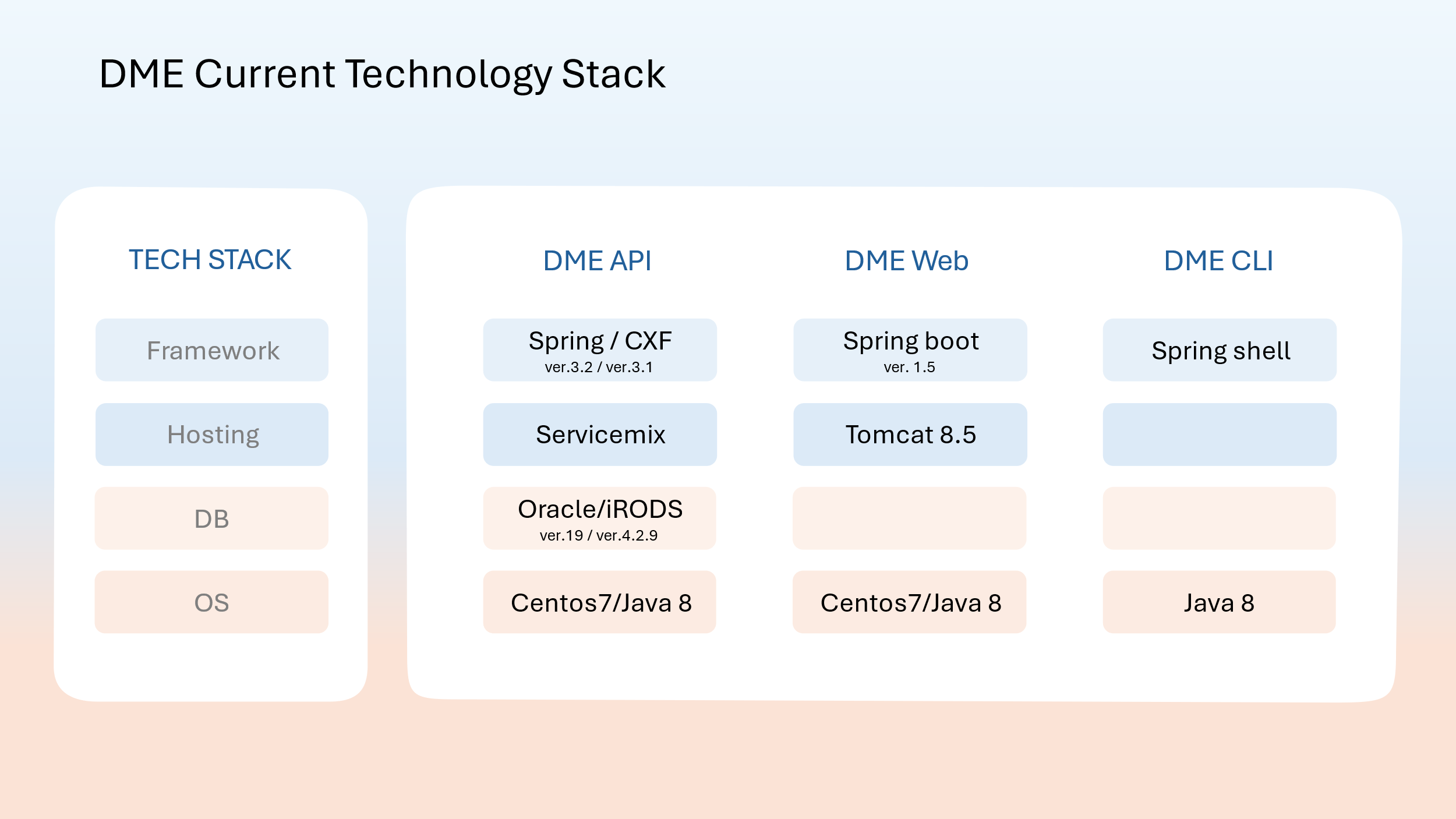
1. Domain layer
2. DTO Layer
3. Common layer

The following diagram depict the API server architecture:



These OSGi bundles are hosted on Apache Service Mix.

The diagram below shows the existing technology stack, and the version of libraries that are being used for Data Management Environment System.



Many libraries are dropping OSGi support. For example, the latest CXF has removed OSGi support. There is also an additional development overhead to import and export packages from the OSGi bundles. In addition, there is an overhead in managing the OSGi dependencies which makes third party library upgrades difficult and security updates more time consuming. In addition, Apache Service Mix is no longer actively being updated. The latest Service Mix cannot work with the recent Spring libraries. Therefore, we have evaluated alternative hosting servers listed under the Technology Catalog hosting the DME REST Services in the future.

The table below lists the Pros and Cons of the evaluated Technology.

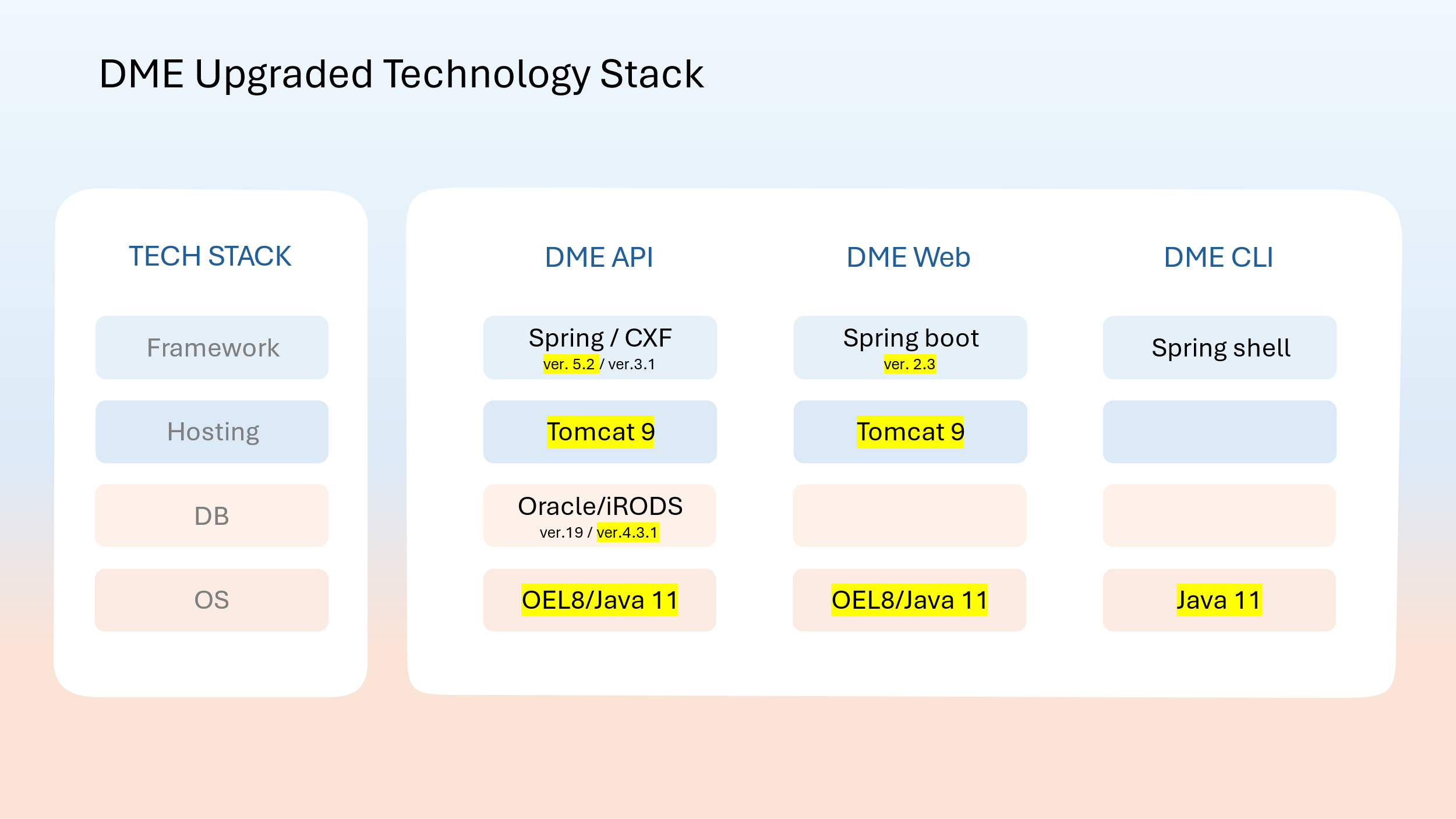
|  |  |  |
| --- | --- | --- |
| Server | Pros | Cons |
| TomEE | Full Java EE server  Open-source application server  Stable.  Lightweight. | Heavyweight than Tomcat and more difficult to operate |
| WebLogic | Full Java EE server  Offers a wide range of management and monitoring tools, as well as integration with other Oracle products. | Commercial application server  Complex to setup  Slow to support newer Java versions |
| WildFly (JBoss) | Full Java EE server  Open-source application server  Red Hat is also one of the most respected software vendors.  Uses Undertow application server.  Has more features than TomEE | Heavyweight than Tomcat and more difficult to operate.  Ret Hat does not offer support contract for WildFly |
| Tomcat | Supports the latest implementation of specifications such as the Servlet and JSP API quicker.  JAX-RS to Spring REST.  Well documented and has a large user base and community support.  Easy to upgrade.  Support tomcat clusters.  Default embedded server when you create a Spring Boot microservice.  Embedded servlet containers in Spring boot for future migration of | Java servlet container and NOT a full Java EE server.  Console manager app is underwhelming.  Tomcat clusters can be difficult to configure. |
| Jetty | Faster throughput and lower latency, suitable choice for high-performance applications.  Embedded servlet containers in Spring boot for future migration of JAX-RS to Spring REST. | Java servlet container and NOT a full Java EE server.  Does not support the latest implementation of specifications such as the Servlet and JSP API quickly. |
| Undertow | More lightweight than Jetty and Tomcat.  Embedded servlet containers in Spring boot for future migration of JAX-RS to Spring REST. | Java servlet container and NOT a full Java EE server.  Lower throughput and latency than Jetty. |

Based on the evaluation above, we have concluded to move to Tomcat 9 and remove the current OSGi infrastructure.

# Architecture Design

## Technology Stack

The diagram below shows the new technology stack after the upgrade for the Data Management Environment System.



# Software Interfaces

NCI Data Management Environment (DME) offers three interfaces to the users.

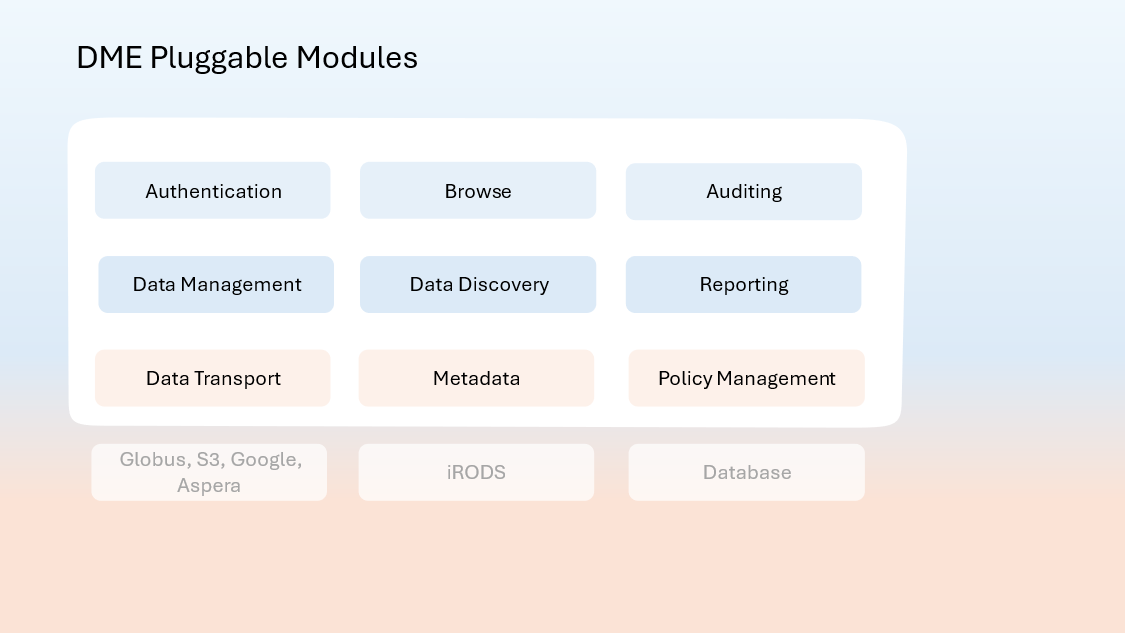
1. Users can access DME programmatically by using the DME REST API. The REST API is an HTTP interface to DME. The REST API is the core of DME where the users and other systems can integrate with DME using the standard HTTP requests.
2. The DME Web Application is a web-based user interface for registering, managing, and transferring the data stored in DME. The DME Web Application internally calls the DME REST API services.
3. The DME Command Line Interface provides commands for a set of DME services to issue commands or build scripts to perform DME tasks. The DME Command Line Interface internally calls the DME REST API services.

# Software Design

## Introduction

The NCI Data Management Environment system is designed based on modular application development to provide flexible, pluggable, and easily configurable modules that can be replaced or extended without impacting the application.

The diagram below shows the DME pluggable modules. For example, data transport modules can be extended to support data transfers through various transfer mechanisms into various storage technologies. The metadata management module is currently using iRODS metadata management system. DME provides a flexible way to configure the metadata attributes using the policy management module to enforce metadata rules.

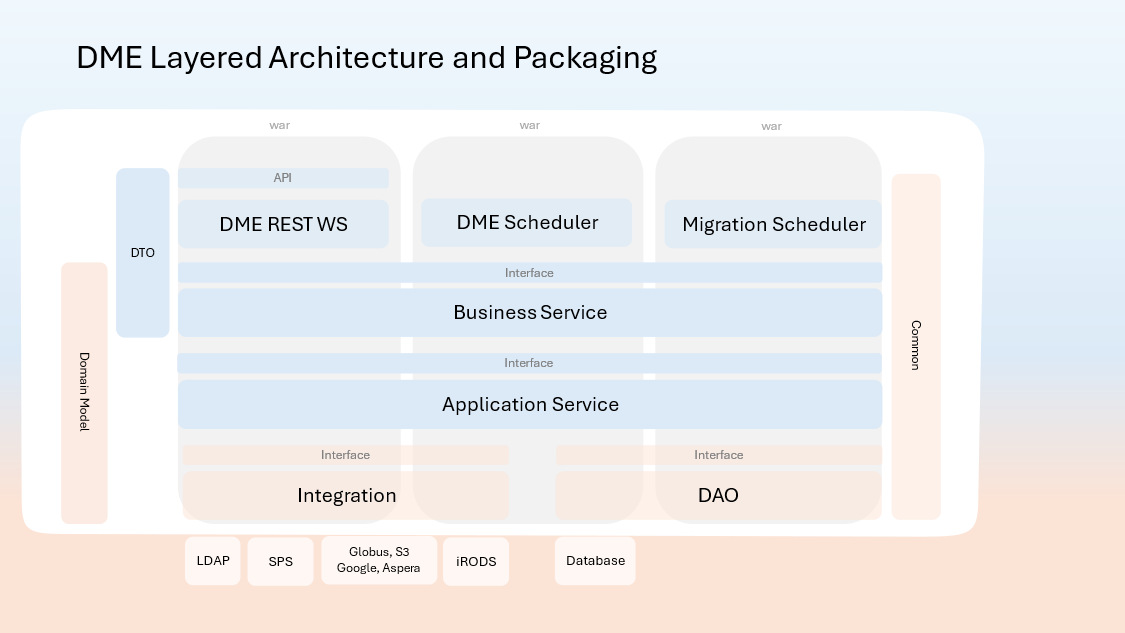


## API Server Design

The NCI Data Management Environment API server hosts a collection of APIs to provide services to various capabilities of the system. In a larger categorization, those components consist of Security (Authentication, User and Group Management), Data Management (Data Transport, Data Downloads, Permission Management, Metadata Management), Data Discovery (Browsing and Searching), Data Migration, Auditing and Reporting.

The DME RESTful Web Services API is packaged into a single WAR file (hpc-server.war) and is deployed into a Servlet Container. The package contains the collection of APIs for the various supported services. The API endpoints are defined and implemented in the REST Web Service Layer. The REST Web services layer invokes the appropriate Business Service via the Business Service interface. The business service layer defines and implements the core functionalities of the system by orchestrating a series of application services. The application service is responsible for implementing business logic and integrating with external systems. Each of these layers and components are independent jar files and packaged in the WAR file via dependency management.

The following diagram depicts the layered architecture of DME.



### Common Frameworks

#### Authentication

The NCI Data Management Environment authenticates the user requests with the NCI Active Directory. All user requests are intercepted by the Authentication interceptor to validate the credentials. The users have the option to supply one of the following credentials.

1. Basic Authentication

This is the NCI Active directory (NCI LDAP) username and password supplied in the authorization header.

1. DME Token

DME has an authenticate API for users to obtain a token accessing the authenticate API using method 1 or 3. This JWT token (JSON Web Token) is supplied in the authorization header and can be repeatedly used until it is expired.

1. Site Minder Token

For applications which uses the Site Minder Web agent like the DME Web application, the SMSESSION token can be provided to authenticate against the SiteMinder Secure Proxy Server (SPS).

When a DME user account is created, the user’s iRODS account is also created. With the current state of integration with iRODS, a user would need to be authenticated by iRODS system for secure access. iRODS uses a secure password system for user authentication (Standard Authentication). The user passwords are scrambled and stored in the iCAT database. Additionally, iRODS supports user authentication via PAM (Pluggable Authentication Modules), which can be configured to support many things, including the LDAP authentication system. PAM can be configured to support various authentication systems. Current authentication scheme is set to Standard Auth.

DME provides user authorization based on iRODS roles.

iRODS primarily provides three roles,

* Admin - can manage user accounts and access permissions on different objects and collections.
* User - owner of collections or objects can assign different access types to different users on the owned collections or objects.
* Groupadmin - can create users, groups and add or delete users from groups.

These roles are used to authorize access to each API endpoint using Apache CXF's SimpleAuthorizingInterceptor.

#### Exception

There are mainly two types of exceptions returned from the NCI Data Management Environment, integrated systems exceptions, and user exceptions. The integrated systems exception occurs when there is an error returned from one of the following integrated systems.

* GLOBUS
* IRODS
* LDAP
* ORACLE
* CLOUDIAN
* AWS
* USER\_S\_3\_PROVIDER
* GOOGLE\_STORAGE
* VAST

The error is captured at the business service layer and an email notification is sent to the system administrators notifying of the error and providing the stack trace. The user will also receive an appropriate server error response for their requests. For user exceptions such as authentication, authorization and validation failures, a proper http code is returned along with the error type and message. This mapping is performed at the Web service layer.

#### Notification

Notification from the NCI Data Management Environment to the user is mostly subscription base.

The user may subscribe to DME notifications events such as upload, download, updates etc. to get notified via email. Each of these tasks generates an event in DME which is then processed, and emails generated for the subscribers.

Here are the notifications which are sent by the system that are not subscription base.

* New user registration notification
* System administrator notification

The email templates for the subject and the body for all notifications are defined in **notificationFormats.json** file**.**

#### Logging

NCI Data Management Environment uses the Simple Logging Facade for Java (SLF4J) for logging so that the desired logging framework can be plugged in at deployment. Currently Logback is used, and the logging is configured in **logback.xml file.**

The location of the log files on the server is configured as follows:

/var/log/tomcat/hpc-server

### Scheduled Task Design

#### Background

A key component of DME architecture is a set of tasks that are executed asynchronously to fulfil a user’s request such as a down of file from S3 archive to Globus destination, or fixed scheduled tasks to deliver system functionality such as producing daily reports.

Spring framework capabilities are used to implement this HPC-DME component. Specifically, Spring’s @Scheduled annotation with cron expression is used to declare all HPC-DME’s methods to be scheduled and executed as async tasks. Spring’s TaskScheduler and TaskExecutor configured w/ thread pools are responsible to manage the runtime execution.

There are 2 separate projects containing scheduled tasks.

1. hpc-scheduler – All core data management capabilities – a total of 34 scheduled tasks
2. hpc-scheduler-migration – Scheduled tasks to implement data migration requests from one archive to another – a total of 6 scheduled tasks.

The data migration tasks were separated so that they can be deployed on their own on dedicated server(s) for migration, and scale at times there is a high demand for migration requests.

In the current architecture, each set of scheduled tasks (project) is configured as a karaf feature, so it can be deployed and run separately to Servicemix. The features configuration is in the hpc-features project.

#### Deployment to Tomcat

To deploy the scheduled tasks to Tomcat, we need to package the projects as web-apps (WAR). The web-apps will have no servlet definitions and thus not listening to any incoming HTTP requests. However, as Tomcat deploys the code, the Spring task scheduler/executor will be initiated, and asynchronous execution of tasks will begin.

The following changes are needed to enable running the HPC-DME scheduled tasks projects on Tomcat:

1. **pom.xml**
   1. Change packaging to war
   2. Add the following dependencies:
      1. hpc-bus-service-impl
      2. spring-beans
      3. sprint-context
      4. spring-web
      5. spring-security-web
      6. logback-classic
2. resources folder
   1. **resources/logback.xml** – logging configuration. Each scheduler is configured to log to a separate LOG file under ${catalina.base}/logs/hpc-server
   2. **resources/notificationFormats.json** – this config file contains systems email formats. It is needed via dependency on hpc-app-service jar. A separate change will externalize this file, but as of now – it’s needed under the resources folder.
   3. **resources/WEB-INF/web.xml** – this is the web-app definition. Note that it only includes a section to load spring context file of the scheduler project, but no servlet definitions.
   4. **resources/WEB-INF/spring/hpc-scheduler.properties** – this file includes default properties packaged w/ the code base. Note that an externalized properties file is deployed to each Tomcat which allows for overriding the default values in this file.
   5. **resources/WEB-INF/spring/hpc-scheduler-beans-configuration.xml**
      1. This is the scheduler beans config. The following changes made –
         1. <context:property-placeholder> modified to load the property files. Note that externalized file deployed to Tomcat overrides the default values in the code base.
         2. Add <import> to scan and upload all the dependent HPC-DME spring context files.

## Command Line Interface Design

The DME Command Line Utility consists of two components:

* Bash scripts
* Standalone Java application

The commands are defined and implemented in the bash scripts. Some commands use the curl command to interface with the DME APIs. A handful of commands call the standalone Java application (Command Line Interface Java Application) from the bash script for more complex directory processing, multi-threaded execution, and fault tolerance.

## User Interface Design (Web application)

The Data Management Environment Web application offers a graphical user interface to the users for transferring data, managing user and group permissions, and finding data.

The user authentication is performed with NIH Login which is configured in the Apache Web server. The NIH token is then passed to the DME Web application via cookie and is validated by the DME API server. For environments that do not support NIH Login, users can authenticate using their NIH username and password which will then be authenticated via LDAP by the API server.

DME Web application also integrates with the Google Authentication to allow users to select files/folders to/from their Google Drive for transfer. Google Client account has been setup for the application and configured in the application.

## System Integration

### Integrated Systems

The HPCDME API server provides a set of data management REST services to deposit, annotate, search and retrieve large data objects. The API is implemented by combining capabilities of several external systems. This document provides technical details on how the external systems integration was implemented and lists the dependencies.

### iRODS Integration

iRODS is an open-source data management solution. iRODS provides data registration with metadata catalog. It provides data search, security, audit, rule-engine, and data transfer capabilities. (https://irods.org).

The integration with iRODS was developed using the Jargon API (https://github.com/DICE-UNC/jargon/wiki/Jargon-overview). There are several requirements that could not be satisfied with Jargon, and the server is querying the iRODS DB directly:

1. Complex search: Search by metadata attribute levels and collection types
2. Notifications: Generate events and notify subscribed users
3. Metadata policies: DOC based metadata hierarchies and policies
4. Custom Summarized Reports: Summarized reports by DOC, User, and period
5. Bookmarks: User bookmarks to collections
6. Saved Search: Save search criteria with a name

### S3 Object Store

The integration with various S3 object store was developed using the Amazon AWS SDK for Java (<https://aws.amazon.com/sdk-for-java>).

### LDAP Integration

Microsoft’s directory and authentication system. The integration was done using the Java standard naming API and is trivial so not covered in this document.

### Globus Integration

Globus is an open-source data transfer and sharing platform. Globus provides the ability to transfer large files asynchronously, securely, and reliably. (<https://www.globus.org/>)

The integration with Globus was developed using the Globus transfer API using Globus nexus API to authenticate w/ Globus. Both are Java jars from Globus that are packaged with the API server. The transfer API provides convenient Java API to communicate with the Globus REST services (https://docs.globus.org/api/transfer).

# Database Design

## iRODS Schema

DME adopts iRODS data model to manage metadata and security around that. Each iRODS deployment—or Zone—is composed of an iRODS Metadata Catalog (iCAT), an iCAT-Enabled Server (IES), and optional Resource Servers. The iCAT is a relational database that holds all the information about data, users, and zone that the iRODS servers—IES and any resource servers—need to facilitate the management and sharing of any data registered with HPC DME. The iCAT contains the information about

* the zone for the purposes of sharing across zones,
* data and their metadata,
* the virtual file system,
* resource configuration, and
* user information.

The iRODS data elements are grouped into several tables. The most important tables which map to HPC requirements are the tables representing Collections, Data Objects and the Metadata. These tables are the r\_coll\_main, r\_data\_main and r\_meta\_main. These tables are shown in the image below. The collections and the data tables are connected to the metadata table by the mapping table r\_object\_meta\_map. Metadata may be attached to files, users, groups, collections (iRODS equivalent of sub-directories), and resources.

## DME Table Design

Extending the iRODS schema, DME API requires the following additional tables addressing the requirements with complex search, auditing, maintain data registration tasks, reports, notifications etc.

**User Account:**DME requires users to be registered with its application to access the API, Web UI and CLI. The hpc\_user table holds user registration information. User accounts are never deleted physically from the database. The ACTIVE column stores the status of a user account.

**User Query:**DME users can save search criteria with a name to access it later. The hpc\_user\_query table stores the information about the saved user query.

**User Bookmark:**DME users can bookmark interested collection path so that the bookmarked path can be accessed without typing the entire path at later time. The hpc\_user\_bookmark table stores the information about the bookmarks.

**System Account:**DME’s layered security architecture protects archive data from direct access. Service accounts are used to insulate users accessing the data directly. HPC DME also manages its security with integrated applications through service account. The hpc\_system\_account table holds information about the service accounts. Only System Admin has access to update this information. Currently, DME maintains 4 integration points – 1) S3 Object Store 2) AWS S3 3) Globus 4) iRODS. Whenever there is an update to service account credentials, this table should be updated with the credentials. System Admin can update the information via DME API.

**Notifications:**HPC DME application generates notifications on events such as upload, download, updates etc. Users may subscribe to notifications to get notified via email. The hpc\_notification\_subscription tables manage notifications and subscriptions. Users may opt to unsubscribe at any time.  
The hpc\_notification\_trigger table manages notification subscriptions with its triggering events.

The hpc\_event table manages all current events generated by the application. The hpc\_event\_history table manages the history of all the events. Once an event is processed, it is moved to the event history table. The hpc\_notification\_delivery\_receipt table maintains event notification delivery receipts.

**Data Management configuration:**DME manages different division/office/center information, data archive settings and their metadata policies in hpc\_data\_management\_configuration table. The policy information in JSON format is stored in this table.

**Bulk Data registration:**  
HPC DME API supports registering bulk data or list of data files together. The hpc\_bulk\_data\_object\_registration\_task table keeps track of the requests and the status of individual files. Once the task is completed, the information is moved to hpc\_bulk\_data\_object\_registration\_result table.

**Data Download:**  
DME API supports downloading the data files. hpc\_data\_object\_downlaod\_task tables keep track of the requests and the status of individual files, and results.

DME also supports downloading bulk data (collection) or list of data files together. The hpc\_collection\_download\_task table keeps track of the requests and the status of the collection or list of files, and results.

Both these tasks are moved to the hpc\_download\_task\_result table once the tasks are completed.

**Audit:**DME API enforces auditing on every update and delete requests on any data file or collection. The hpc\_data\_management\_audit table maintains audit information.

## Materialized View Design

**Search:**DME supports searching collections and data objects using the metadata attached to the collection or data object itself, or metadata attached to the parent collections. To improve performance searching through the collection or data object path, the following materialized views are created.

* R\_COLL\_HIERARCHY\_METAMAP
* R\_COLL\_HIERARCHY\_META\_MAIN
* R\_CATALOG\_META\_MAIN
* R\_COLL\_HIERARCHY\_DATA\_OWNER
* R\_DATA\_HIERARCHY\_METAMAP
* R\_DATA\_HIERARCHY\_META\_MAIN
* R\_DATA\_HIERARCHY\_USER\_META\_MAIN

**Reports:**DME supports generating several summarized reports. These reports use multiple tables to generate report data with filters provided by users. To improve performance with reports, the following materialized views are created.

* R\_REPORT\_COLL\_REGISTERED\_BY
* R\_REPORT\_COLL\_REGISTERED\_BY\_BASEPATH
* R\_REPORT\_COLL\_REGISTERED\_BY\_DOC
* R\_REPORT\_COLL\_REGISTERED\_BY\_PATH
* R\_REPORT\_COLLECTION\_PATH
* R\_REPORT\_SOURCE\_FILE\_SIZE
* R\_REPORT\_COLLECTION\_SIZE
* R\_REPORT\_COLLECTION\_TYPE
* R\_REPORT\_DATA\_OBJECTS
* R\_REPORT\_REGISTERED\_BY
* R\_REPORT\_REGISTERED\_BY\_BASEPATH
* R\_REPORT\_REGISTERED\_BY\_DOC
* R\_REPORT\_REGISTERED\_BY\_S3\_ARCHIVE\_CONFIGURATION

These materialized views are refreshed at configured intervals by the background process run by DME API scheduler.