

System Design and Architecture

For

**Mobile eBenefits Contract
(CLIN 0002AA, Contract: VA118-13-C-0006)**



**Submitted To:
Department of Veterans Affairs**



**February 1, 2014
Submitted by:**

CHANTILLY, VA 20151

This data shall not be disclosed outside the Government and shall not be duplicated, used, or disclosed—in whole or in part—for any purpose other than to evaluate this proposal; provided that if a contract is awarded to this BAA responder as a result of—or in connection with—the submission of this data, the Government shall have the right to duplicate, use, or disclose the data to the extent provided in the contract. This restriction does not limit the Government's right to use information contained in this proposal if it is obtained from another source without restriction. The data subject to this restriction are contained in sheets so marked.

Table of Contents

1.	Background	1
2.	Architecture Design Overview	2
2.1.	System Overview/Application Context	2
2.2.	Mobile eBenefits High-Level Architecture Overview	3
2.3.	Apache Web Server	6
2.4.	HTTP Server	6
2.5.	Reverse Proxy	6
2.6.	Load Balancer	6
2.7.	Launchpad Server	7
2.8.	Application Server	8
2.9.	Applications	8
2.10.	Authorization Server	10
3.	Authentication and System Integrity Controls	10
3.1.	User Authentication	10
3.2.	Data Access Integrity Controls	13
3.3.	REST Data Services Authorization Cache	15
3.4.	Resource Last Accessed Time and Timeout	15
3.5.	Resource Authorization	15
4.	Services Detailed Implementation	16
4.1.	Claims Explorer	16
4.2.	Loan Explorer	18
5.1.	Education Explorer	20
5.2.	FAQ Explorer	21
5.3.	Jobs Explorer	21
6.	Technical Specification	22
6.1.	Source Code Projects	22
6.2.	Runtime Environments	23
6.3.	Build and Deployment Environments	24
6.4.	Development Tools	26
7.	Deployment and Configuration	26
7.1.	Launchpad	27
7.2.	Claims Explorer	31
7.3.	Loan Explorer	34
7.4.	Education Explorer	36
7.5.	FAQ Explorer	39
7.6.	Jobs Explorer	42
8.	Implementation Plan	44
8.1.	Amazon EC2	44

8.2.	Mobile Application Environment (MAE).....	44
8.3.	FISMA HIGH MAE 009 Deployment.....	46

1. Background

The Mobile eBenefits Application will include the following components:

- **Mobile eBenefits Launchpad App (LP):** This provides the authorized user with a single view and access point to MeB functionality without having to re-enter identifying information. It is integrated with the Identity Access Management (IAM) service.
- **Mobile Claims Explorer:** This allows Veterans or Service members to review the status of compensation and pension (C&P) claims or appeals, the processing stage that the claim is in, and the expected completion date. It is integrated with Benefits Enterprise Platform (BEP) as well as VIERS (Veteran Information Eligibility Services) for appeals. The Claims Explorer External System Access Points section at the end of this document lists the VA systems and access points used for retrieving data for the Claims Explorer application.
- **Mobile Education Explorer:** This application provides Veterans with the ability to view existing information regarding their status and potential entitlement for Chapter 33 (Post-9/11 GI Bill) benefits. Options can be provided to use this information to apply for benefits, to receive a Certificate of Eligibility (COE) or to request a record review by a Veterans Claims Examiner (VCE). The app will also allow Veterans to view in real-time Chapter 33 claim status and Certified Enrollments for each enrollment. It is integrated with Chapter 33 LTS. The Education Explorer External System Access Points section at the end of this document lists the VA systems and access points used for retrieving data for the Education Explorer.
- **Mobile Jobs Explorer:** This provides Veterans with current information on jobs that they may be pursuing or for which they are eligible. The explorer is built so that it can be integrated with National Resource Directory (NRD), VA for Vets, or VetSuccess through the application's configuration data. The Jobs Explorer System Access Points section at the end of this document lists the access points for retrieving data for this explorer.
- **Mobile FAQ Explorer:** This app provides the Veteran with useful tools to locate VA/DoD facilities based on their current or supplied location. It also provides the ability to view, search, or navigate through available FAQs. These FAQ articles and topics are provided via REST services provided by the eGain® KnowledgeAgent. The FAQ Explorer System Access Points section at the end of this document lists the access points for retrieving data for this explorer.
- **Mobile VA Loan Guaranty Benefits Explorer:** This app provides the Veteran the capability to access information and resources to assist them in the home loan process. It is supportive of the VA's Loan Guaranty Program (LGY) and can provide information on the veterans eligibility for the benefits offered through the LGY program, trigger an application and provide a starting point for other self-service apps. The Loan Explorer External System Access Points section at the end of this document lists the access points for retrieving and updating data through this explorer.

These components will be developed and deployed independently of each other into the various runtime environments. The integration between the component explorer applications will be managed and made visible through the Mobile eBenefits Launchpad. Additionally the

Launchpad provides a single sign-on capability for accessing any of the applications made available through it.

This document will discuss the technical architecture of the Mobile eBenefits applications, the component pieces of that architecture and the requirements for hosting and managing the applications.

2. Architecture Overview

This section describes the architecture of the Mobile eBenefits applications focusing on the major components that comprise the solution and the supporting infrastructure within the VA Core Cloud.

2.1. System Overview/Application Context

The figure below provides a conceptual view of the Mobile eBenefits application and the interfacing systems with particular emphasis on the legacy VA systems that will provide data to the various explorer applications. At the core of the system the eBenefits applications are responsible for:

- Providing individual HTML5 application user interfaces which are made available to the user through the Launchpad application
- Receiving requests from the user through their mobile device's browser
- Formatting these requests into a suitable format for transmission to the services layer. For Mobile eBenefits this interface is a REST-based interface using HTTP methods (GET, PUT, POST, DELETE)
- Coordinating the response to the request by accessing data in various legacy VA systems within the REST services layer
- Converting the data into a suitable representation (JSON) for presentation to the user interface layer
- Returning the JSON data in the REST methods response and presenting it to the user in the browser.

Access to the legacy VA systems is accomplished through one or more data service adapters contained within the server components of the application. These will be discussed in more detail below.

Logon capability for the user is managed through DSLogon and the associated VA IAM SSO authentication services that are separately deployed and managed within the runtime environment. See the Authentication section below for more detail on the implementation.

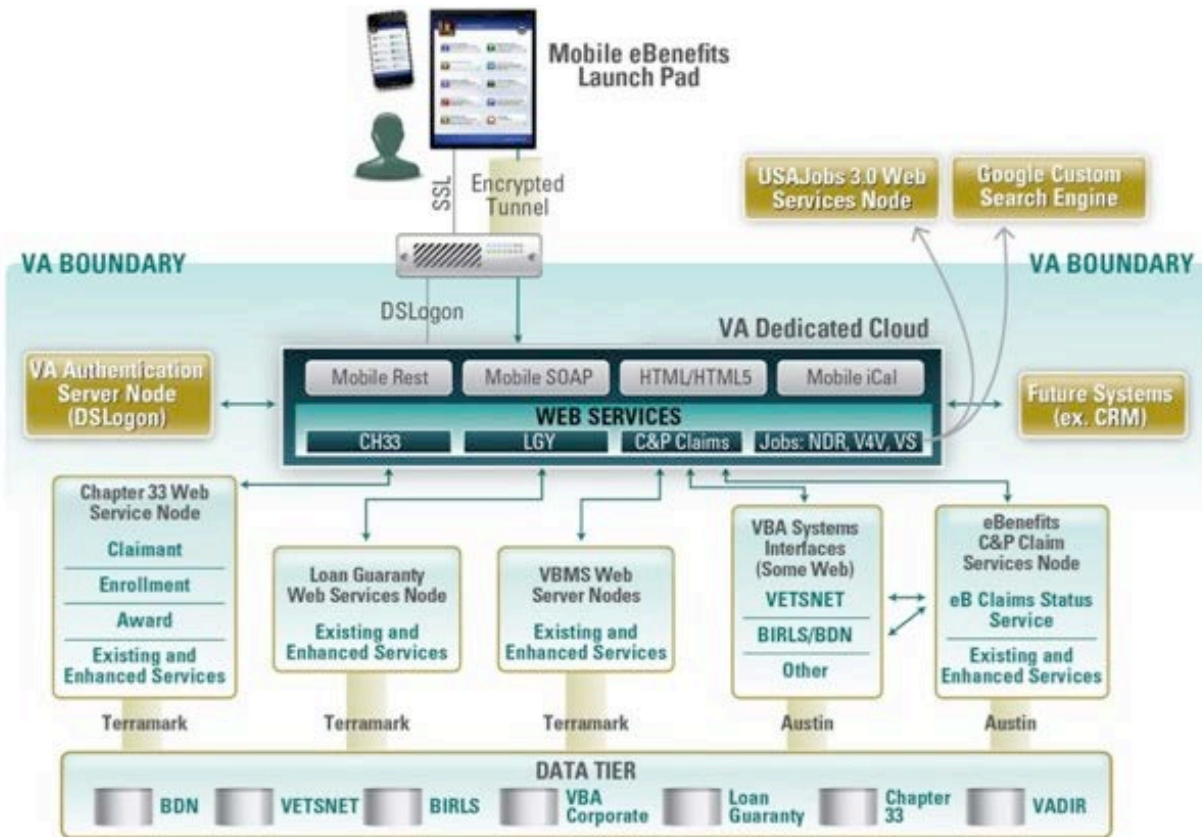


Figure 1 Application Context

The next section will provide a more detailed overview of the architecture of the Mobile eBenefits application.

2.2. Mobile eBenefits High-Level Architecture Overview

The Mobile eBenefits deployment environment has the following major components:

- A web server to serve as the access point to the application from browsers on a variety of platforms, including mobile devices
- A Launchpad server to service requests for displaying the Launchpad page to a user
- One or more application servers to support each of the applications that are accessed through the Launchpad
- An authentication server to verify user credentials and to support single sign-on
- External VA legacy systems that provide the data for the various applications

A key aspect of the architecture is the use of resource-oriented services to expose the functionality of individual applications that in turn comprise the Launchpad. These services

provide security/content-related services (authentication, Single Sign-on [SSO], context) and have the following general characteristics:

- Accept a JSON formatted set of input data
- Return a JSON formatted set of output data
- Adhere to the REST interface for request/response actions

The figure below shows the major components of the solution broken down into Internet-based components, VA Dedicated Core components and internal VA legacy systems that provide data for each application as well as the IAM Single Sign-On capability.

Each application that comprises Mobile eBenefits will be deployed as two separate deployable artifacts. The web-based artifact including HTML, Javascript and CSS will be deployed within a ZIP file to an Apache Web Server where it will be unpacked/unzipped. The server artifact including REST resources and VA data source adapters will be deployed to the Application Server in a single JAR file that contains both the server infrastructure needed to host the application as well as the application-specific components.

The sections of the document below give further details about the purpose, design and interactions of each of these components.

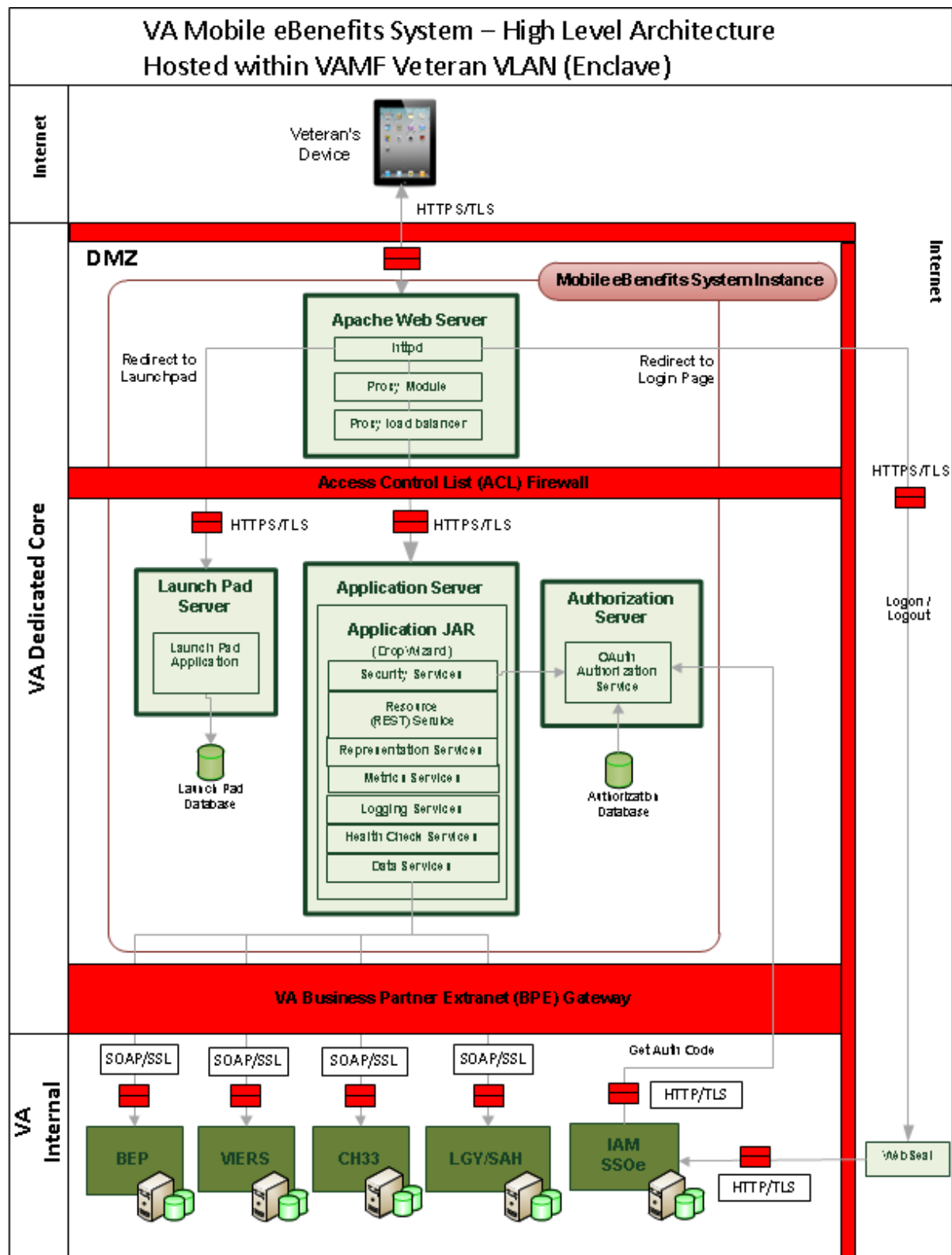


Figure 2 Mobile eBenefits Architecture

2.3. Apache Web Server

The Apache web server receives HTTP requests either directly from the browser as a result of a user-initiated action or indirectly through REST method calls from the Javascript that comprises the Mobile eBenefits application and executes within the browser on the user's behalf.

When client traffic increases and node performance begins to deteriorate, this server is capable of scaling horizontally by adding more servers to address performance if additional hardware resources are not desired or appropriate. The server maintains data privacy and integrity between client devices and the application servers via the TLS protocol.

2.4. HTTP Server

The HTTP Server uses the standard Apache httpd module to provide services for HTTP requests. Common implementation and configuration practices, such as using a "jail" to isolate the web server, will be implemented to ensure that the server, which executes in the DMZ, is properly protected. Additionally it is expected that the same Multi-Processing Module (MPM) performance configuration as is used on other mobile applications will be used for Mobile eBenefits.

Each web application that is deployed to the Apache web server is deployed from a ZIP/JAR file in an "exploded" directory structure with the following major component types:

Javascript – the underlying executable content of the application on the browser that is responsible for coordinating internal model changes with the view and accessing data from the application server via REST calls.

CSS – Cascading Style Sheets

HTML – The Hypertext Markup Language pages. Generally there will be a single HTML page that is dynamically changed and updated through the Javascript based on user actions.

2.5. Reverse Proxy

This server also functions as a reverse proxy to route specific HTTP requests to the application servers. Generally these are REST resource requests that originate from the application's Javascript contained within a user's browser. The reverse proxy within the Apache server then decides where to send those requests, and returns the content as if it was itself the origin.

The reverse proxy is also used to convert local Mobile eBenefits Authorization Service URLs into the actual URLs for the service. The proxied values are necessary since the single origin policy requires that all requests are made to the originating web server. The Authorization Service is installed on a separate host that is accessible and known to Mobile eBenefits only through the proxy entries within Apache.

2.6. Load Balancer

The load balancer will work in concert with the reverse proxy module to route requests to two or more logically parallel application servers. A round-robin algorithm that sequentially cycles through a list of application server instances is the recommended approach. This is a simple and

predictable algorithm for balancing the request load.

2.7. Launchpad Server

This server is responsible for hosting the Launchpad application as well as the database that supports it. The Launchpad is similar to a landing page in a web application, it is the “home base” for a user, and will display all apps in the VA mobile family that the user may access. The Launchpad will also prompt a user for their credentials, and support single sign-on for all of the VA apps.

The VA Launchpad app provides an elegant approach to two pressing issues:

- How will a Veteran find all of the applications within the Mobile eBenefits suite?
- How can context information (used for Single Sign-On (SSO) and able to be used to prevent redundant data entry) be shared between apps without taking the security risk of persisting PHI or PII data on the device?

Whenever the Launchpad is opened, it will provide the user with a listing of all relevant applications that are authorized for use. These applications and the corresponding URL for the icon and application are configured through the Launchpad database. The Launchpad also provides for the collection of context data to support continuity between apps, but does not persist them on the device. Through the Launchpad database the Launchpad can easily be configured to present a Veteran with the most current selection of eBenefits apps, as is illustrated in Figure 4.



Figure 3 Mobile eBenefits Launchpad (Conceptual)

2.8. Application Server

The application server will host one or more of the Mobile eBenefits applications: Claims Explorer, Loan Explorer, Job Explorer, Education Explorer and the FAQ Explorer. These applications provide REST-based web services and the necessary adapters to retrieve application data from a VA system. All of the functional components of each application, except for the web-based components (HTML, Javascript, CSS), which are deployed to the Apache Web Server, will be deployed together to the same application server instance.

If necessary, based on performance throughput considerations, an application may be deployed to two or more servers and load-balanced through the web server.

2.9. Applications

Each application will be deployed in a single JAR (Java Archive) file. This file contains both the runtime HTTP server (Jetty) for the web services offered by the application (Jetty Web Server) and the Java classes that are created to support the specific application. This enables a very simple deployment and runtime environment as all required dependencies, apart from the application's configuration data, is contained within a single deployable file. There is no need to separately install and configure a runtime container or manage Java JVM problems related to memory usage and garbage collection for multiple applications.

At the core of each application is the DropWizard suite of tools. DropWizard straddles the line between being a library and a framework. Its goal is to provide high performance, reliable implementations of everything a production-ready web service needs. Packaged within the DropWizard framework are:

- Jetty to provide web server and servlet container services
- Jersey to provide REST services
- Jackson to provide JSON parsing and formatting services
- Metrics for reporting on request processing times and performance
- Guava
- Logback
- Hibernate Validator
- JDBI
- Joda Time

These tools are combined with the application-specific classes into a single deployable JAR file providing a simple deployment into the runtime environment.

Each application contains the following general services:

Security Services

Spring Security is used to safeguard all exposed REST methods so that only an authorized request, one with a valid authentication token, can access the method and associated data. This service surrounds the REST method so that the method execution is separated from its authorization. This provides a clean separation between functional aspects of the application and security.

Resource Services

Resource services are the REST (Representational State Transfer) methods that receive requests and return responses using an appropriate representation of an underlying resource. Generally the representation will be formatted as JSON. See the Representation Services section below for more detail.

Jersey, the Java API for RESTful Web Services (JAX-RS) reference implementation, is used to provide the underlying services to support the REST methods.

Domain Model

The “resources” referenced by the RESTful interface are contained within an internal domain model. This domain model represents the logical relationships between data relevant to the application’s context as well as providing entities that contain logically related data elements. The domain model is formulated using Plain Old Java Objects (POJO) and is independent of any representation that may be used for the data. i.e. JSON, Plain Text, etc.

Representation Services

These services are responsible for transforming REST resources, described by the domain model, into an appropriate external representation. As previously mentioned JSON will generally be used as the external representation.

Conversion between the resources and their representations is accomplished by using Jackson. Jackson is a parser and generator for JSON, written in Java, that is extremely fast and configurable.

Metrics

DropWizard provides the ability to annotate any of the REST methods with `@Timed`, `@Metered` or `@ExceptionMetered` annotations to record critical statistics related to the execution performance of the method. These annotations allow these critical statistics to be gathered and analyzed, using a variety of output formats and tools, on an as-needed basis. Although specific recommendations and standards for the Mobile eBenefits metrics gathering have not yet been developed, the expectations is that most or all methods will provide this information to aid in performance tuning.

Logging and Auditing

Logging services for the application are built using Logback and SLF4J (Simple Logging Framework for Java). Configuration of the logging follows the standard Logback approach. Auditing of requests and responses is provided for both REST requests as well as back-end data requests.

Health Check

Health check services, built into the application, will support ad hoc requests to check on the availability and proper functioning of critical resources used by the application. These will include components such as VA systems, databases and messaging infrastructure. The health check is available through a separate administration console port provided by the application.

Data Services

Data services are used to access, and potentially update, data within the VA legacy systems. These services are exposed to the application via an interface to isolate the actual implementation of the service. This will allow the incorporation of mock data services as-needed as well as the replacement of particular implementations without affecting the remainder of the application services.

2.10. Authorization Server

The Authorization Server hosts authentication and authorization for all mobile applications that are accessed through the Launchpad. These include both Mobile eBenefits applications as well as those of the Health Adapter. This server contains the components that make up the Authorization Service and that were previously described.

3. Authentication and System Integrity Controls

This section of the document describes the approach for validating a user's credentials, giving that user access to the application and ensuring that only data for that user can be viewed.

3.1. User Authentication

The ability to access information from multiple VA systems on behalf of the veteran and to work within an enterprise setting requires the ability to properly authenticate, and pass appropriate credentials/tokens to multiple legacy applications.

The VA has collaborated with the DoD to provide a user authentication process (DS Logon) in conjunction with Identity Access Management SSOE capability for both Service members and Veterans. This SSOE will establish an individual's identity and allow that user to complete transactions without having to re-enter information as the user chooses different Launchpad applications. To start the process a user selects the Login button within the Launchpad and will

be presented with a login screen. After logging in with their DS Logon credentials, users are presented with a screen of options. No subsequent capabilities developed under this contract require further login screens to access any of the functionality discussed in the sections below.

The figure below depicts the flow of requests and responses as part of the authentication process. Although it references the Health Adapter as the supported application the flow is the same for Mobile eBenefits as a shared server will be used for both applications.

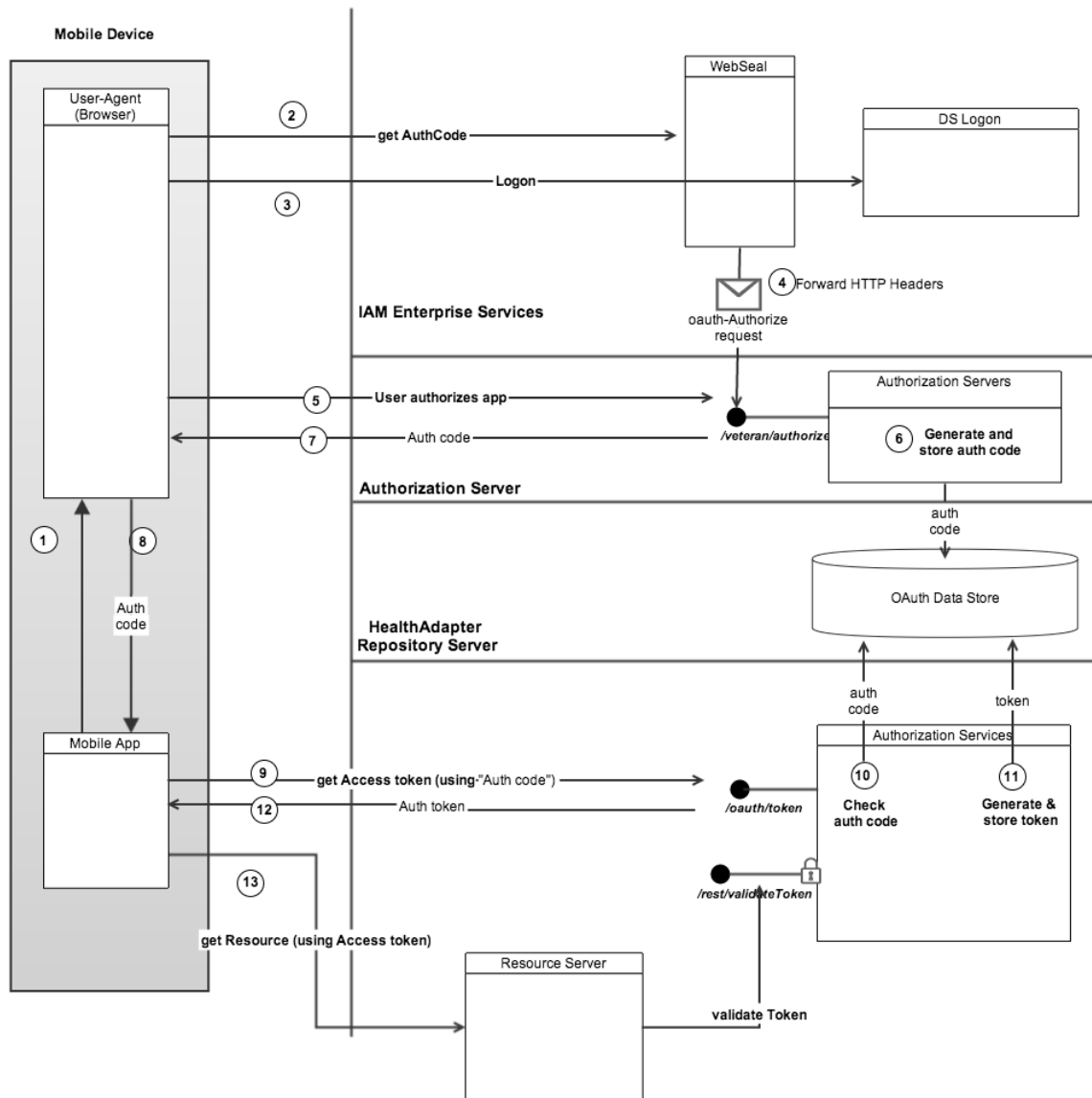


Figure 4 User Authentication Process

Once the application has been launched, the user then clicks "login". This causes the application to perform an "openURL" on the "oauth-authorize" resource, which launches the Mobile

eBenefits browser, (note that this is **not** an embedded browser), acting in the OAuth role as the user-agent [**#1 in diagram**].

The browser attempts to perform an HTTP GET [**#2 in diagram**] to the Authorization Server via WebSEAL.

Because the browser is not authenticated, WebSEAL returns a HTTP 302, redirecting the browser to the DS Logon Authentication Page [**#3 in diagram**].

The user enters credentials into the DS Logon Authentication Page and submits. Once successful, WebSEAL forwards or redirects [**#4 in diagram**] the oauth-authorize request to the Authorization Server (this includes information as to the identity of the user such as the user's EDIPI). The Authorization Server's oauth-Authorization Endpoint trusts the call from WebSEAL (because of the use of mutual TLS authentication).

The Authorization Server asks the user to approve the app to access their clinical resources [**#5 in diagram**]. Note that although explicit user approval is specified by the OAuth 2.0 specification it is not required as part of this authentication process. Instead, the Authorization Service uses a configurable list to indicate those applications that are implicitly accepted by the user who is logging in. A login request for access to any application in the list will result in an automatic approval of the application. See the Client Registration section below for more details.

Once this approval has occurred the Auth server generates and stores auth-code [**#6 in diagram**] and redirects the browser to the Mobile eBenefits Launchpad application [**#7 in diagram**].

This returns a redirect (launchpad://?authcode=XXX) and auth code [**#8 in diagram**] which causes the device to launch the mobile app which then performs an HTTP POST against the token resource [**#9 in diagram**], passing in the authcode.

After the Authorization Service validates the auth code against the OAuth Data store [**#10 in diagram**] it then generates and returns a token to the app [**#11,12 in diagram**]. The Authorization Service also stores the information related to the token (user, expiration) in memory for later retrieval. See the REST service token validation section for more details on how the token is validated.

The access token is passed as an HTTP header (HTTP Authorization Header) [**#13**] to any of the Mobile eBenefits REST services. Those resource servers invoke the Authorization Service to validate the token. Since the token must be validated for all calls, it is likely that token validation would be cached to prevent network service invocations. The REST services must be able to translate the access token into the corresponding user information in order to access PII/PHI data that is not stored on the web browser or mobile device.

Note:

- the user-agent is the mobile browser, not an embedded browser within the app

-
- in #2 the browser attempts to access a resource via WebSEAL, and because the browser is not authenticated, the browser is redirected to DS Logon Authentication page
 - During logout the mobile app invokes the OAuth provider, which invalidates the token.

When a second application, such as the Loan Explorer, needs to logon, it will attempt to access the "Authorization Endpoint" (via WebSEAL). Because there is an existing session between the mobile device and WebSEAL, the browser is able to access the authorization endpoint without prompting the user to authenticate a second time.

At the conclusion of the authentication process, users will be known by the identifier associated with their DS Logon account, which is their DoD identifier (EDIPI).

Mobile eBenefits also stores the information related to the token (user, expiration) in memory. This is necessary since Mobile eBenefits acts a Policy Enforcement Point, ensuring that the resource being accessed is allowed by this user as well as the verification that the user's authentication token is still valid. This requires the ability to translate the token to the corresponding user. Steps 7 through 13 in the figure above illustrate this process.

3.2. Data Access Integrity Controls

Resource requests made through the Mobile eBenefits REST services will be validated by ensuring that a valid access token is contained within the Authorization header of the HTTP request. This validation will guarantee that all such requests originate from an authenticated user. To implement this validation all REST requests will be intercepted by a Servlet Filter that is registered with the REST service container as part of the startup processing for each application. The registration process is performed through a common Application base class to ensure that all applications enforce security in a consistent manner.

The filter is configured through the application's JSON configuration using a common SecurityConfiguration section. A snippet of that configuration section is shown below.

```
"securityConfiguration": {  
  "urlPattern": "/*",  
  "trustedPorts": [50001, 50011],  
  "endPoint": "",  
  "cacheSize": 2000,  
  "cacheDuration": 30  
},
```

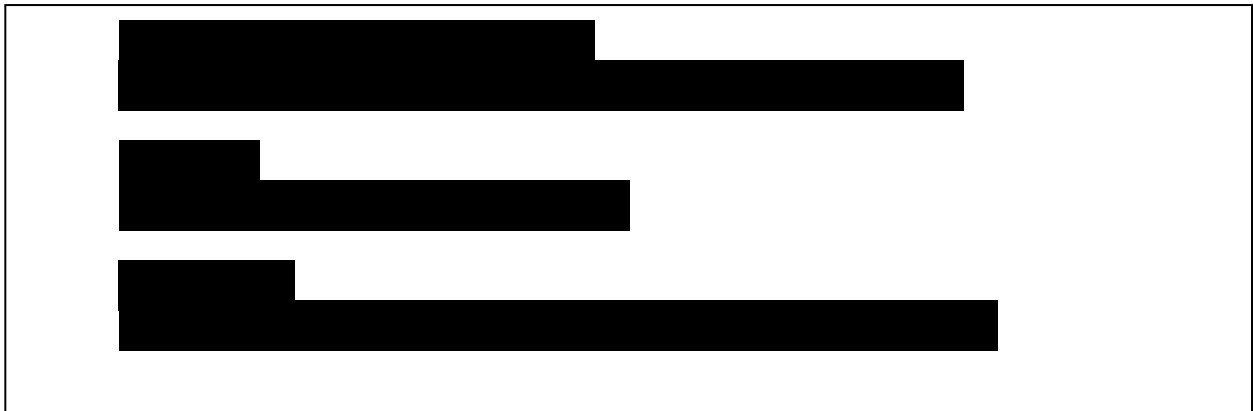
The urlPattern property determines which REST requests will be intercepted by the filter. The trusted ports property determines which server ports will always be trusted without requiring an access token. In this example the only trusted ports are for the application's metrics and health check pages. See the port assignment page in this WIKI for details on the port assignments for each application. The endPoint property contains the full resource request string for the Authentication Service to validate the access token. Finally, the validateToken boolean property is used to dynamically turn on/off the validation of the access token by the security filter.

The security filter will extract the access token from the HTTP Authorization header and strip out the "Bearer" prefix. An example of the format of that header with the HTTP message is shown below.

Headers:

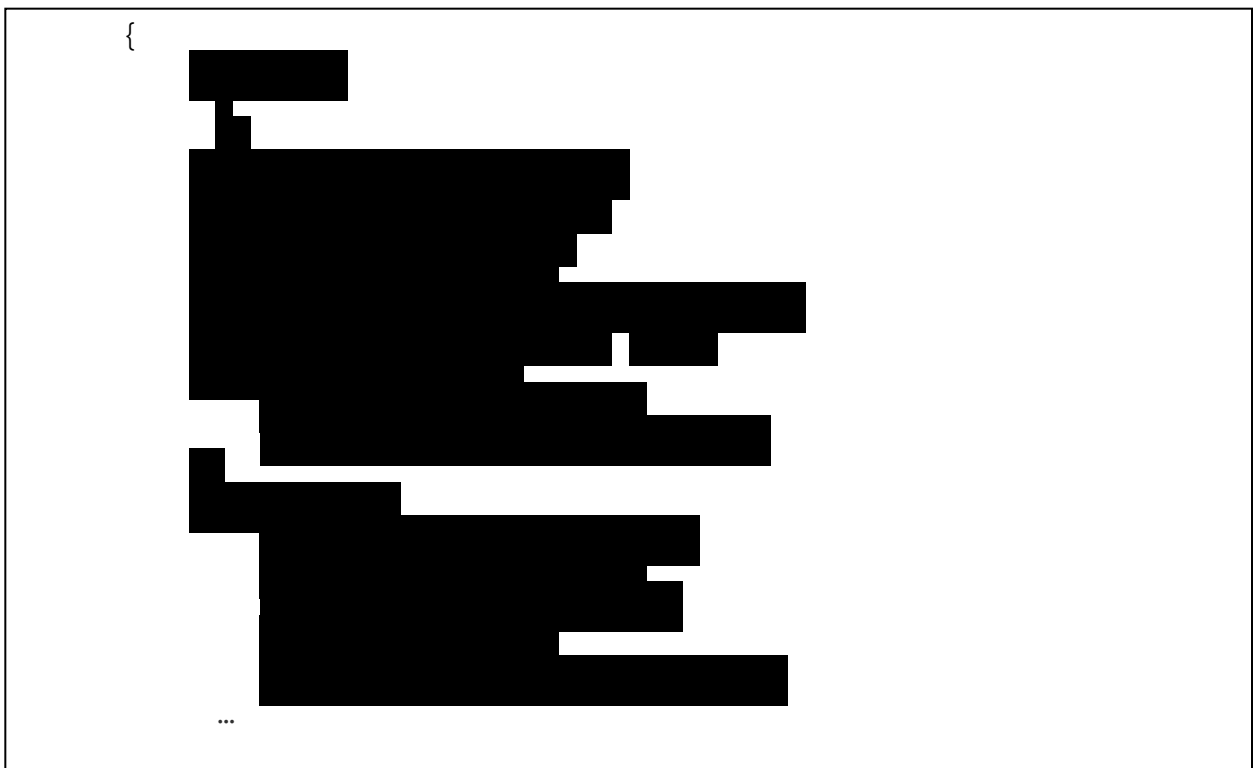
Authorization Bearer e89e3ddd-7c9d-48aa-9ed3-27420b964894

After extracting the access token the filter will send a POST request to the Authorization Server containing the token to be validated. If the token is valid, the service will return the client and user information associated with that access token. The following code blocks depict a



validation request sent to the Authorization Service and a sample response for an authenticated user.

Sample validateToken Response



3.3. REST Data Services Authorization Cache

The REST data access validation process uses a cache of recently validated users, keyed by the access token, to eliminate many concurrent requests to the Authorization Service for the same user. For example, when building a given page within the user interface it is likely that several REST calls will be needed to obtain the necessary data. To avoid the unnecessary revalidation of the current user a short-lived cache is used. Each cache entry expires after a configurable number of seconds. It is important to understand that the entry expires based on the time when the cache entry was written rather than the last access time. This ensures that the user's access token will always be validated at least every X seconds. The current setting for the expiration is 30 seconds. I.e. each entry expires 30 seconds after it was last validated and written to the cache. Only validated users will be written to the cache so the cache will not contain an access token that is not currently active and valid. One exception to this rule is logout. Since the REST services are not aware of the logout action the cache entry can persist for a short period of time after the user has logged out. Based on current settings this is between 0 and 30 seconds after the last access token validation has occurred.

3.4. Resource Last Accessed Time and Timeout

Each call to the Authorization Service to validate the access token will also result in an update of the "last access time" for the user. If a token has expired because the user has not been validated within a specified period of time then an HTTP 419 (Authentication timeout) will be returned from the Authorization Service. This in turn will result in a redirect of the user to the application login page – DS Logon.

3.5. Resource Authorization

In addition to validating the user's right to access the application, via the access token validation, each REST service performs checks to ensure that the data being requested through the RESTful URL can be viewed by the user making the request. Generally the URL will contain one or more identifiers for the specific data that is being requested. To ensure that the URL has not been modified maliciously to reference different identifiers each REST service uses one or more data validators to check the authorized user against the data being requested. For example, within the Claims Explorer the "claimant ID" is a key identifier on most of the URLs. When the request is processed a "claimant validator" will be executed to ensure that the claimant ID on the URL is one of the known claimant IDs for the current user. If the claimant ID on the URL is not recognized then an HTTP 403 status code (Forbidden) will be returned to the client indicating that access to the data is not allowed.

4. Services Detailed Implementation

This section of the document describes the specific services that are used by each explorer to retrieve data and, in the case of the Loan Explorer, to update data within the backend VA systems.

None of the explorers store data in local databases or other persistent storage mechanisms. All data persistence occurs in backend VA systems.

4.1. Claims Explorer

The Claims Explorer services use the Benefits Enterprise Platform (BEP) web services and VIERS (Veteran Information Eligibility Services) web services to retrieve data. No data is written as part of this service; it is read-only. This section describes each of the VA systems that are accessed by the Claims Explorer application, the specific functions and calls that are used, and the type of data that is accessed.

Auditing

For all data requests and updates that are transmitted to VA systems an auditing record will be recorded in the application log file. This audit record records the specific request being made, the user name of the user making the request and the date and time of the request. At the conclusion of the request the success or failure of the request will also be recorded for auditing purposes.

Benefits Enterprise Platform (BEP) Web Services

BEP contains a number of individual web services (web service ports) that are accessed to provide claims and payment data to the Mobile eBenefits applications. This section enumerates those services and the individual methods that are accessed within each service.

Security header values are set in the SOAP header before sending a request to any of the BEP web services. These identify the source system making the request and include a user name and password required by BEP to authorize the execution of the web service request. The specific values for these header elements are managed through the application configuration files.

Web Service Name and Method	Usage
BEP EBenefitsBnftClaimStatusWebService findBenefitClaimsStatusBySSN	Return all current and historical claims for the given claimant with a given Social Security Number.
BEP EBenefitsBnftClaimStatusWebService	Retrieve claim details for a specific claim.

findBenefitClaimDetailsByBnftClaimId	
BEP ClaimantService findGeneralInformationByFileNumber	Retrieve information about the claimant by using the claimant's VA file number. The Claims Explorer usage of this service always uses the claimant's Social Security Number as the file number argument.
BEP PaymentInformation retrievePaymentSummaryWithBDN	Retrieve all payment information for a claimant. The arguments to this call are the veteran's participant ID, Social Security Number and a payee code set to "00".
BEP PaymentInformation retrievePaymentDetail	Retrieve payment detail for a specific payment using its unique ID.
BEP TrackedItemServiceRemote findTrackedItems	Retrieve "items never received" for a specific claim.
VIERS AppealPort findAppeals	Retrieves either OPEN or HISTORICAL appeals for a given Social Security Number.
VIERS AppealPort getAppeal	Uses an appeal identifier returned by the findAppeals call to retrieve the appeal details.

4.2. Loan Explorer

This section describes each of the VA systems that are accessed by the Loan Explorer application, the specific functions and calls that are used, and the type of data that is accessed.

The Loan Explorer services reuse web services from the Loan Guaranty/Special Adaptive Housing (LGY/SAH) portlet service. This portlet was originally created to support the eBenefits web application. Mobile eBenefits is reusing the web services exposed by this portlet. Unlike other explorers within the Mobile eBenefits application this explorer both reads and writes data.

Auditing

For all data requests and updates that are transmitted to VA systems an auditing record will be recorded in the application log file. This audit record records the specific request being made, the user name of the user making the request and the date and time of the request. At the conclusion of the request the success or failure of the request will also be recorded for auditing purposes.

Loan Guaranty/Special Adaptive Housing Web Services

The LGY/SAH web services are accessed to provide data for the Loan Explorer for both VA Loan applications as well as Special Adaptive Housing (SAH) applications.

Web Service Name	Web Service Method	Usage
SAHPortletService	determineCoeEligibilityEdiPi	Retrieves a loan applicant's status (active, pending, denied, unsuccessful) and the status of a Certificate of Entitlement (COE) application, if any.
SAHPortletService	getDmdcPersonForProfile	Retrieve basic applicant information including Social Security Number, name and date of birth.
SAHPortletService	insertEligibilityCorrespondence	Insert one or more documents to support a COE application.
SAHPortletService	insertCoeApplication	Insert a COE application.
SAHPortletService	getCoeFile	Retrieve a previously submitted COE application for display to the user.
SAHPortletService	isVeteranEligible	Returns a boolean value

		indicating if the veteran is eligible for the SAH Grant application process.
SAHPortletService	activeGrant	Get information about active SAH grants for a veteran.
SAHPortletService	summary	Get summary information about SAH grants for a veteran.
SAHPortletService	processExternalApplication	Insert an SAH application.

5.

5.1. Education Explorer

This section describes each of the VA systems that are accessed by the Education Explorer application, the specific functions and calls that are used, and the type of data that is accessed.

Auditing

For all data requests and updates that are transmitted to VA systems an auditing record will be recorded in the application log file. This audit record records the specific request being made, the user name of the user making the request and the date and time of the request. At the conclusion of the request the success or failure of the request will also be recorded for auditing purposes.

Education Explorer Web Services

The CH33 web services are accessed to provide data for the Education Explorer for entitlement and enrollment information. For CH33 claimant and entitlement data and enrollment data the Education Explorer directly accesses the CH33 ClaimantService and EnrollmentService web services.

Benefits Enterprise Platform (BEP) Web Services

CH33 payment information the BEP PaymentInformationService is also used. This is the same web service that is used by the Claims Explorer for payment information.

Web Service Name	Web Service Method	Usage
ClaimantService	getClaimant	Get CH33 claimant data
EnrollmentService	getEnrollment	Get Ch33 enrollment data
PaymentInformationService	retrievePaymentSummaryWithBDN	Retrieve all payment information for a claimant. The arguments to this call are the veteran's participant ID, Social Security Number and a payee code set to "00".
PaymentInformationService	retrievePaymentDetail	Retrieve payment detail for a specific payment using its unique ID.

ClaimantService (BEP)	findGeneralInformationByFileNumber	Retrieve information about the claimant by using the claimant's VA file number. The Claims Explorer usage of this service always uses the claimant's Social Security Number as the file number argument.
-----------------------	------------------------------------	--

5.2. FAQ Explorer

This page describes the technical details of the FAQ Explorer application that are specific to that app.

The FAQ Explorer has 3 major service components:

1. REST interfaces to eGain KnowledgeAgent repository. This interface is used to support searching for and retrieving frequently asked question (FAQ) articles and topics of interest to the user.
2. SOAP-based interface to a VA locator service. This interface provides VA facility information in response to a request for a given state code.
3. HTTP/XML interface to a Dept. of Defense installation locator service. This interface provides searching either by ZIP code or by state code.

Service	Service Method	Usage
Dept. of Defense Installation Locator	HTTP GET request that returns XML and escaped XML.	Retrieves a list of installations that match the request search arguments – either search by ZIP code or search by state.
VA Facility Locator	SOAP request	Retrieves a list of facilities that match the request search argument – state code.
eGain v11 REST services	REST calls	Search for and retrieve topics and articles from the eGain FAQ repository.

5.3. Jobs Explorer

This section describes each of the VA systems that are accessed by the Jobs Explorer, the specific functions and calls that are used, and the type of data that is accessed.

Auditing

For all data requests that are transmitted to external systems an auditing record will be recorded in the application log file. This audit record records the specific request being made, the user name of the user making the request and the date and time of the request. At the conclusion of the request the success or failure of the request will also be recorded for auditing purposes.

External Services

The Jobs Explorer currently accesses either VetSuccess or Vets.Jobs based on configuration parameters. The interface returns JSON that is identical for both endpoints so the internal codebase is not affected by a change from one to the other. Requests are sent via an HTTP GET to the service endpoint. The response JSON is then parsed into internal domain model objects and returned to the user interface.

The Jobs Explorer also submits reverse geocoding requests to the Google Geocoding API to convert latitude/longitude coordinates into a city, state/province and country for use in searching for Jobs.

Service	Service Method	Usage
VetSuccess or Vets.Jobs	HTTP GET request that returns JSON	Retrieves a list of jobs that matches the GET request search arguments.
Google Geocoding API	HTTP GET request that returns JSON	Retrieve the geocoding data for a given latitude and longitude.

6. Technical Specification

6.1. Source Code Projects

This section provides the name and a short description of the major source code repository projects that make up the Mobile eBenefits suite of applications.

Mobile-ebenefits root directory:

- **product**
 - **integration-tests** – tests to verify the REST service interface
 - **mock-services** – to simulate the legacy VA system interfaces; primarily SOAP-based web services
 - **production** – the root folder for artifacts that will be deployed to production
 - **backend/services** – root folder for service layer artifacts

-
- **benefit-gateway-services** – WSDL and XSD files for the BEP/BGS SOAP-based web services
 - **claims** – DropWizard service project for the Claims Explorer
 - **common** – common code used by two or more explorer services
 - **loan-guaranty-services** – WSDL and XSD for the LGY/SAH SOAP-based web services.
 - **loans**
 - **frontend** – root folder for web application/UI artifacts
 - **claims** – HTML, Javascript and CSS for the Claims Explorer web application
 - **common** – common Javascript for two or more applications
 - **loans** – HTML, Javascript and CSS for the Loan Explorer web application
 - **infrastructure** – root folder for tools that are used to build, test and deploy the application
 - **chef** – automated deployment scripts
 - **gradle** – Gradle templates and build scripts
 - **jenkins** – scripts for configuring the Jenkins build server(s)
 - **vagrant** – scripts to start local virtual machines in the developer’s workspace

6.2. Runtime Environments

The figure below illustrates the runtime environments currently used and envisioned to deploy and execute the Mobile eBenefits applications.

Development of the applications currently occurs within an Amazon EC2 environment. Mock services are used to simulate access to the various legacy VA systems. In addition a common Authorization Service and Launchpad are accessed by each developer’s local deployment of the applications. That local deployment uses Vagrant to create two separate local virtual machines: an Apache Server for the web application component (exploded ZIP file) and the DropWizard virtual machine that loads and executes the application JAR file.

Within the Mobile Application Framework (MAE) no development resources are currently used for runtime testing of the applications.

The MAE integration test environment will provide the first access to the actual VA legacy systems, although they will be using obfuscated test data instead of live data containing PII/PHI information. Each legacy application that provides data to the MeB applications must have an exposed interface in this environment.

The MAE integration test environment will use a common Authorization Service that is shared between Mobile eBenefits and other VAMF applications such as the HealthAdapter. A Launchpad instance that is dedicated to the Mobile eBenefits application will be installed in this environment.

The MAE production environment will follow the same deployment strategy as the integration test environment.

All virtual machines in the MAE will use Linux 6.3 or similar binary-compatible operating system. An Oracle database instance is required to provide the database used by the Launchpad application for storing URL and icon image information for each application. The footprint of this database is consequently quite small as the number of applications are small.

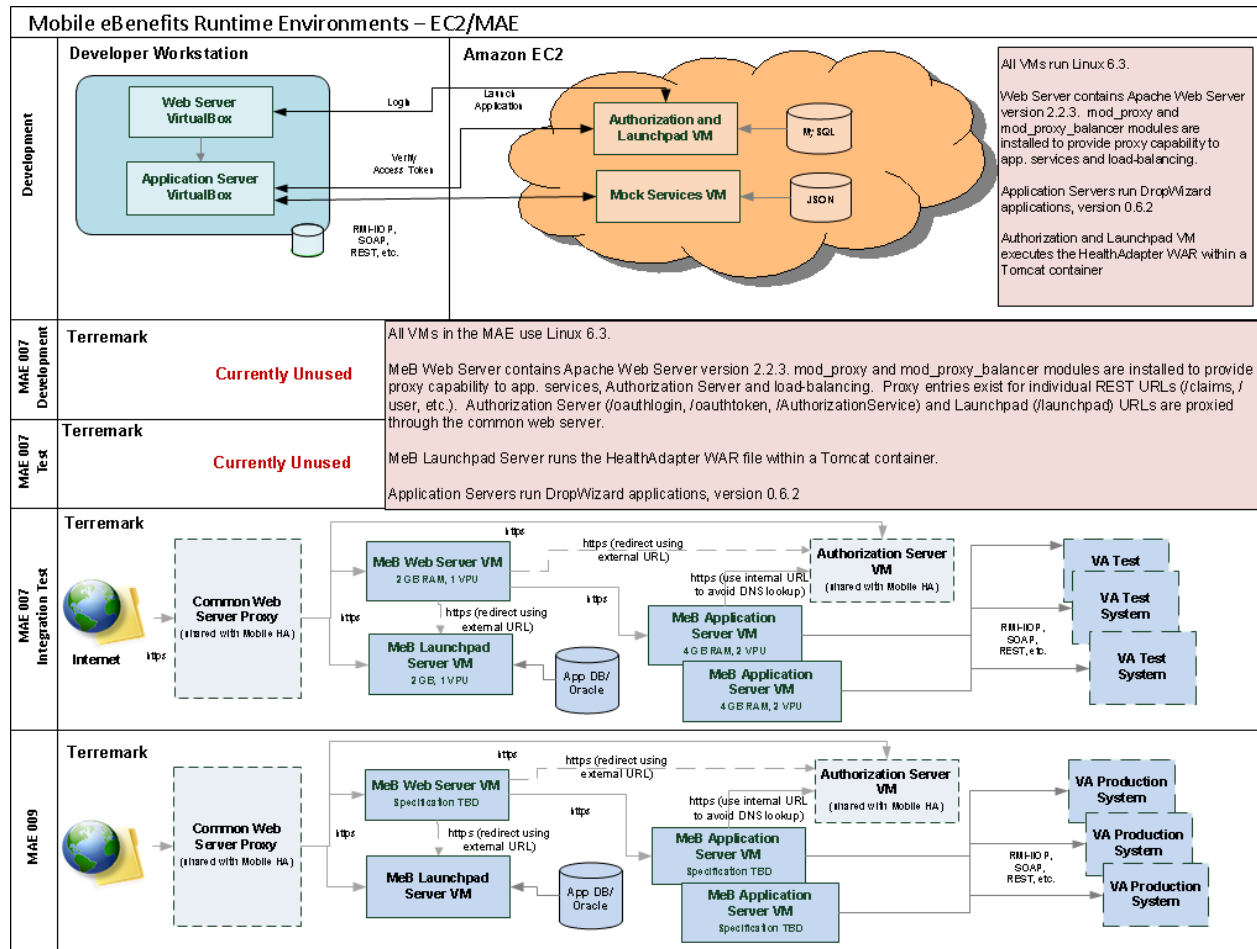


Figure 5 Runtime Environments

6.3. Build and Deployment Environments

The figure below illustrates the development environments currently used to build and test the Mobile eBenefits applications.

As described above, development of the applications currently occurs within an Amazon EC2 environment. Automated builds using Chef, Ruby, Jenkins and EC2 virtual machines are used to build the application and execute automated tests for the application.

Within the Mobile Application Framework (MAE) no development resources are currently used other than a Git repository that is loaded with the latest version of the Mobile eBenefits Git repository contents. This repository is used for Fortify security scans.

The MAE integration test environment will use automated deployment tools to build and install the web application and DropWizard artifacts as well as the application configuration files.

Finally, the MAE production environment will also use automated deployment tools to build and install the applications in a similar fashion to the integration test environment.

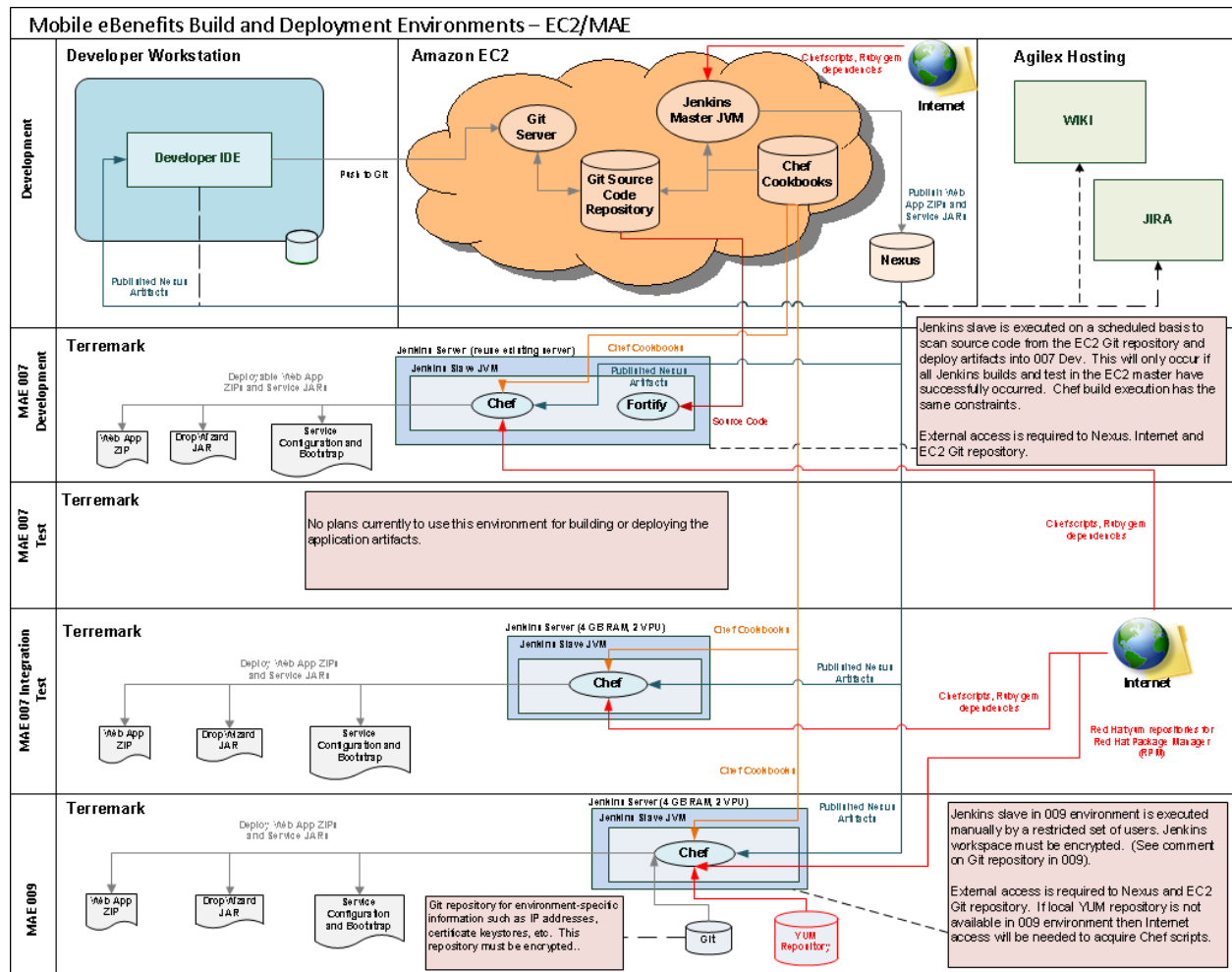


Figure 6 Development and Testing Environments

6.4. Development Tools

The primary tools used in developing and testing the Mobile eBenefits applications are listed below.

Infrastructure Tools

- Chef for automated deployments of the application
- Gradle to build and test the application artifacts
- Jenkins to manage the automated build and test of the application
- Ruby to execute Chef scripts and execute acceptance and integration tests
- Vagrant to create local virtual machines in the developer's workspace

UI

- Node.js and NPM for dependency management
- Grunt for building, deploying, running tests
- Yeoman to create application scaffold
- BackboneJS generator for Yeoman
- Jasmine generator for Yeoman
- Text editors: Sublime Text 3
- IDE: Developer choice (Eclipse, IntelliJ, WebStorm) We strongly recommend WebStorm. Since the GIT integration is exceptionally good.

Services

- DropWizard and associated libraries. See the DropWizard description above in this document for more details.
- Gradle for performing builds of the service artifacts.
- Httparty for integration tests of REST services
- Ruby/Rspec for integration tests

7. Deployment and Configuration

This section provides further detailed information on the configuration and installation of each component of the Mobile eBenefits application. This section would be useful to anyone who must configure or deploy the application.

7.1. Launchpad

The Mobile eBenefits (MeB) Launchpad is a web application that is built and installed independently of the individual applications that are made available within it. The component applications are made visible through the Launchpad by adding descriptive information about each application's home page URL and icon to a dedicated Oracle database. The Launchpad Web Application Archive (WAR) file is installed within a Tomcat container running on its own server, separate from the individual MeB applications. Apache web server proxy entries are required to support the single-origin policy for browsers while allowing the applications to be physically deployed in separate servers.

The Launchpad also provides single sign on session management functions for login and logout. These functions access a common Authorization Service and via that service the IAM/DS Logon services. Once authenticated, via this logon process, the user may freely move between the Explorers that comprise the MeB application without having to log into each application individually. The Launchpad also supports a session timeout feature - currently set to 15 minutes - which is implemented via the Authorization Service. Please reference the architecture diagrams above for more details.

The constituent applications - Claims Explorer, Loan Explorer, etc. - have two deployment artifacts for each application. First is the ZIP file containing the web application itself. This includes HTML, Javascript and Cascading Style Sheets (CSS). This file is deployed to the Mobile eBenefits web application server running Apache. The application REST services are contained within a single JAR (Java Archive) file. This file contains the entire server application as well as the supporting infrastructure needed to execute the application - Jersey for REST, Jetty for web services, etc. The REST service is started by creating a Java JVM using the JAR file.

Launchpad Installation and Configuration

Tomcat is used as the Launchpad WAR file servlet and web server container. URLs for Authorization and OAuth services, which are accessed through the Launchpad, are configured through environment variables passed in the CATALINA_OPTS environment variable. See the code block below for an example. These examples are from a "test" installation of the Launchpad. Production installations will use similar values although the MHP_BASEURL variable will change to recognize the specific type of environment. No further configuration of these values should be necessary once the Launchpad is installed.

Tomcat CATALINA_OPTS Configuration



-
- **oauth.authorize_url** is the location of the VAAFI OAuth access grant service.
 - **oauth.ssoe_logout_url** is the location of the VAAFI logout service. This service is responsible for removing the session.
 - **oauth.logout_uri** is the location of the Authorization Services logout service responsible for clearing the currently assigned access token.
 - **oauth.token_url** is the location of the Authorization Service access token assignment and verification service.
 - **authorization_services.url** is the location of the Authorization Services method to generate and save a grant auth code.
 - The value for **org.apache.cocoon.mode** determines whether or not real or mock user login data is used.
 - **hadb_jndiname** is the Tomcat JNDI name for the Launchpad data source. This data source is used to read Launchpad icon and URL information from the Launchpad database. See the next section for details on this database.
 - **authdb_jndiname** is the Tomcat JNDI name for the Authorization data source. This data source is used to read and write to the Authorization token and session management database.

Each individual application is made accessible through the Launchpad by adding an appropriate entry to its Oracle database table. The script below illustrates the addition of the Claims Explorer and the Loan Explorer to the Launchpad. Each entry in the LAUNCHPAD_ITEM database table provides a unique name to the application, a description, an icon/image URL and the URL for the application's home page.

Each new application that is accessed through the Launchpad requires a row in the LAUNCHPAD_ITEM database table.

Launchpad Database Configuration for MeB Application



Web Application Configuration

No additional configuration of the individual web applications is required or supported once they have been installed.

Common REST Service Application Configuration

This section describes the major sections of the REST application service configuration that are common to all applications. As reference, the code block below uses the configuration from the Claims Explorer service however it is consistent from one application to the next.

Note that for all configuration changes the application must be restarted before the changes will be recognized.

Security Configuration

- **urlPattern** represents the REST URLs that will be intercepted by the security filter for validation. Generally all URLs are included in the validation.
- **trustedPorts** is an array of ports that are trusted by the application and do not need to be validated. These should only be ports that are secured behind a firewall so that only administrative level users can access the port.
- **endPoint** is the address of the VA Mobile Framework (VAMF) Authorization Service which is responsible for validating user's access tokens
- **cacheSize** denotes the size of an in-memory cache that is used to limit the number of concurrent access requests to the Authorization Service for a given user.
- **cacheDuration** is the time in seconds that a cache entry will persist from the time that it is written. The time used should be selected to minimize the number of requests that are sent to the Authorization Service for a given user within a relatively short period of time while ensuring that the access token is validated frequently enough. The recommended value for this duration value is 30 seconds.

http Configuration

These configuration entries control how the Jetty web server, embedded within the application's JAR file, is configured to handle HTTP request traffic.

Logging Configuration

These entries control the application's logging levels as well as console-based logging, file-based logging and the file-based archiving strategy for log files.

Common REST Service Configuration Sections

The example below illustrates the common sections of a REST service configuration file including the sections described above. Details on the service-specific configuration can be found further below under the corresponding heading for each explorer.

```
{
  "securityConfiguration": {
    "urlPattern": "/*",
    "trustedPorts": [50001, 50011],
    "endPoint": "<%= node['mhp']['baseurl'] %>/AuthorizationServices/rest/validateToken",
    "cacheSize": 2000,
    "cacheDuration": 30
  },
  "servicesConfiguration": {
    ...
  },
  "http": {
    "port": "50000",
    "adminPort": "50001",
    # The minimum number of threads to keep running to process incoming HTTP requests.
    "minThreads": 8,
    # The maximum number of threads to keep running to process incoming HTTP requests.
    "maxThreads": 1024,
    # The type of connector to use.
    #
    # Possible values are:
    # * blocking: Good for low-latency services with
    #             short request durations. Corresponds
    #             to Jetty's BlockingChannelConnector.
    # * nonblocking: Good for services which use Servlet 3.0
    #               continuations or which maintain a large
    #               number of open connections. Corresponds
    #               to Jetty's SelectChannelConnector.
    # * legacy: Simple, java.io.Socket-based connector.
    #           Corresponds to Jetty's SocketConnector.
    # * legacy+ssl: Corresponds to Jetty's SslSocketConnector.
    # * nonblocking+ssl: Corresponds to Jetty's
    #                   SslSelectChannelConnector.
    "connectorType": blocking,
    # The maximum amount of time a connection is allowed
    # to be idle before being closed.
    "maxIdleTime": 200s,
    # The number of threads dedicated to accepting connections.
    "acceptorThreads": 1,
    "ssl" : {
      "keyStore": "template.keystore",
      "keyStorePassword": "example"
    }
  },
  "logging": {
    "level": "ALL",
    "loggers": {
```

```

        "gov.va.meb.claims.health": "DEBUG",
        "gov.va.meb.common.external.services.adapter.auditing": "INFO",
        "org.eclipse.jetty": "INFO"
    },
    "console": {
        "enabled": "true",
        "threshold": "ALL"
    },
    file: {
        "enabled": "true",
        "threshold": "ALL",
        "currentLogFilename": "./logs/claims.log",
        "archivedLogFilenamePattern": "./logs/claims-%d.log.gz",
        "archivedFileCount": "5",
        "timeZone": "UTC"
    }
}
}

```

7.2. Claims Explorer

The following configuration sections and properties control the Claims Explorer REST service.

Security Configuration

See the Launchpad Common REST Service Application Configuration section for details.

Services Configuration

This section of the configuration data controls access to back-end data services consumed by the REST service. The servicesConfiguration property consists of one or more individual back-end service configuration sections for the individual data service interfaces.

Benefits Gateway Services Configuration

Configures the BGS/BEP service adapter and the SOAP header values that are required for authentication with that service.

Appeals Service Configuration

Contains the SOAP endpoint address for accessing the VIERS Appeals SOAP web service.

Claims Service Configuration

Contains the SOAP endpoint address for accessing the BEP EBenefits Claim Status SOAP web service.

Claimants Service Configuration

Contains the SOAP endpoint address for accessing the BEP Claimant SOAP web service.

Payments Service Configuration

Contains the SOAP endpoint address for accessing the BEP Payment Information SOAP web service.

Claimant Cache Configuration

These configuration parameters control the in-memory cache that is used to temporarily store claimant information in order to reduce the number of round-trip calls to the Claimants Web Service.

http

See the Launchpad Common REST Service Application Configuration section for details.

Logging

See the Launchpad Common REST Service Application Configuration section for details.

The code block below shows an example of the configuration for the Claims Explorer's service component.

```
{
  "url": "https://dev.com:8080/mockservices/PaymentInformationService",
  "method": "POST",
  "headers": {
    "Content-Type": "application/json",
    "Accept": "application/json"
  },
  "body": {
    "cardNumber": "4111 1111 1111 1111",
    "expiryDate": "12/2018-12/2019",
    "cardholderName": "John Doe",
    "cvv": "123"
  }
},
```

```
}  
  {  
    {  
      {  
        {  
          {  
            {  
              {  
                {  
                  {  
                    {  
                      {  
                        {  
                          {  
                        }  
                      }  
                    }  
                  }  
                }  
              }  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

7.3. Loan Explorer

The following configuration sections and properties control the Loan Explorer REST service.

Security Configuration

See the Common Application Configuration section with the Launchpad for details.

Services Configuration

This section of the configuration data controls access to back-end data services consumed by the REST service. The servicesConfiguration property consists of one or more individual back-end service configuration sections for the individual data service interfaces.

LGY/SAH Web Service Configuration

Configures the Loan service adapter with the endpoint address of the SAHPortletService. See the sample Loan Explorer configuration shown below for more information.

http

See the Launchpad Common REST Service Application Configuration section for details.

Logging

See the Launchpad Common REST Service Application Configuration section for details.

The code block below shows an example of the configuration for the Loan Explorer's service component.

```
{  
  {  
    {  
      {  
        {  
          {  
            {  
              {  
                {  
                  {  
                    {  
                      {  
                        {  
                          {  
                        }  
                      }  
                    }  
                  }  
                }  
              }  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

```
}  
}  
}
```

7.4. Education Explorer

The following configuration sections and properties control the Education Explorer REST service.

Security Configuration

See the Launchpad Common REST Service Application Configuration section for details.

Services Configuration

This section of the configuration data controls access to back-end data services consumed by the REST service. The servicesConfiguration property consists of one or more individual back-end service configuration sections for the individual data service interfaces.

Benefits Gateway Services Configuration

Configures the BGS/BEP service adapter and the SOAP header values that are required for authentication with that service.

Claimant Web Service Configuration

Contains the SOAP endpoint address for accessing the CH33 Claimant SOAP web service.

Enrollment Web Service Configuration

Contains the SOAP endpoint address for accessing the CH33 Enrollment SOAP web service.

Payments Service Configuration

Contains the SOAP endpoint address for accessing the BEP Payment Information SOAP web service.

Claimant Cache Configuration

These configuration parameters control the in-memory cache that is used to temporarily store claimant information in order to reduce the number of round-trip calls to the Claimant Web Service.

http


See the Launchpad Common REST Service Application Configuration section for details.

Logging

See the Launchpad Common REST Service Application Configuration section for details.

The code block below shows an example of the configuration for the Education Explorer's service component.

T



[REDACTED]

Downloaded from <http://ajphaphysocpharm.sagepub.com/> at 11:06 11 November 2014

Downloaded from <http://ajph.org/> on November 10, 2014

[illegible]

Satisfaction Level	Percentage
Very satisfied	~85%
Not very satisfied	~15%

[illegible]

7.5. FAQ Explorer

The following configuration sections and properties control the FAQ Explorer REST service.

Security Configuration

See the Launchpad Common REST Service Application Configuration section for details.

Services Configuration

This section of the configuration data controls access to back-end data services consumed by the REST service. The `servicesConfiguration` property consists of one or more individual back-end service configuration sections for the data service interfaces.

Knowledge Manager Service Configuration

This section configures the URL and the portal ID used to access the eGain KnowledgeAgent REST services. The URL is a partial URL containing the host name and port plus the web service context. A trailing slash is not required as it will be added internally, if missing, by the adapter before constructing the full URL.

DoD Installation Service Configuration

This section configures the connection to the Department of Defense service that provides names and addresses for selected installations. It also configures the parameters for the HTTP client that is used to send requests to that service. The `"connectionTimeout"` specifies the amount of time in milliseconds that the service will wait while attempting to establish a connection to the DoD service. The `"timeout"` specifies the amount of time in milliseconds that the client will wait for a complete response to be received from that service. Note that this service requires extra time to send and receive a complete response. Consequently a time of 45 seconds is recommended for this value. To improve performance, responses for state-specific searches will be cached internally after the first request for a state. The `"connectionKeepAlive"` specifies time in milliseconds for keeping a connection open and available for reuse before it is closed.

VA Facility Service Configuration

This section configures the VA facility locator service including the URL of the service as well as HTTP/SOAP parameters used in connecting to that service. The `"connectionTimeout"` specifies the amount of time in milliseconds that the service will wait while attempting to establish a connection to the VA service. The `"timeout"` specifies the amount of time in milliseconds that the client will wait for a complete response to be received from that service. Note that this service requires extra time to send and receive a complete response. Consequently a time of 45 seconds is recommended for this value. To improve performance, responses for state-specific searches will be cached internally after the first request for a state. The `"connectionKeepAlive"` specifies time in milliseconds for keeping a connection open and available for reuse before it is closed.

DoD Installations Cache Configuration

This section of the configuration controls the cache for state-specific search results against the Department of Defense installations service. The default values are to store up to 100 results, keyed by the state code, and to retain those results for one hour (3,600 seconds) before the cache

entry is removed. Once removed the next request for DoD installations for a given state will result in a call to the DoD service.

VA Facility Cache Configuration

This section of the configuration controls the cache for state-specific search results against the VA facility locator service. The default values are to store up to 100 results, keyed by the state code, and to retain those results for one hour (3,600 seconds) before the cache entry is removed. Once removed the next request for VA facilities for a given state will result in a call to the service.

http

See the Launchpad Common REST Service Application Configuration section for details.

Logging

See the Launchpad Common REST Service Application Configuration section for details.

[illegible]

},

[REDACTED]

[REDACTED]

|

[REDACTED]

|

[REDACTED]

},

```

    }
  }
}

```

7.6. Jobs Explorer

The following configuration sections and properties control the Jobs Explorer REST service.

Security Configuration

See the Launchpad Common REST Service Application Configuration section for details.

Services Configuration

This section of the configuration data controls access to back-end data services consumed by the REST service. The `servicesConfiguration` property consists of one or more individual back-end service configuration sections for the individual data service interfaces.

Jobs Search Configuration

Configures the connection to the job search service. This includes the URL of that service and connection timeout durations.

Google Geocoding Configuration

Configures the URL and connection timeout values for the connection to the Google Geocoding API service.

http

See the Launchpad Common REST Service Application Configuration section for details.

Logging

See the Launchpad Common REST Service Application Configuration section for details.

The code block below shows an example of the configuration for the Jobs Explorer's service component.

```

{
  "securityConfiguration": {
    "urlPattern": "/*",
    "trustedPorts": [53001, 53011],
    "endPoint": "<%= node['mhp']['baseurl'] %>/AuthorizationServices/rest/validateToken",
    "cacheSize": 2000,
  }
}

```

8. Implementation Plan

8.1. Amazon EC2

Development and demonstration of the application will occur within an Amazon EC2 environment. This approach is necessary as sufficient resources do not exist within the Mobile Application Environment in order to support development. In addition, because this development is outside of the MAE, it will be necessary to develop mock services that simulate the actual VA backend systems.

8.2. Mobile Application Environment (MAE)

Mobile eBenefits applications and servers will be hosted in the VA Dedicated Core Cloud. This leverages prior VA mobile work and allows further leveraging of VA's Mobile MAE environment. The VA Dedicated Core Cloud environment is located at Terremark facilities in Culpeper, VA and meets Federal Cloud Computing Security Requirements Baseline.

The VA's Dedicated Core Cloud environment's C&A is at the FISMA "High Impact" level. The design encompasses: a private networking core, dedicated firewalls and load balancers; dedicated security tools, dedicated encrypted tape backup, dedicated encrypted replicated networks, logical and physical separation between programs, and dedicated network monitoring.

In addition, one of the largest requirements of agencies in moving to the cloud is “trust and verify”. MAE has a portal that will provide the near real time information on the infrastructure to provide situational awareness. This will be especially important in order to provide the most cost efficient match of user demand (increase) to environment scaling. In partnership with the VA, Terremark has worked to give the VA full visibility to all critical components of the Terremark infrastructure. This allows the VA to have constant awareness, of the current status of their systems from a simple and detailed dashboard.

The MAE environment will be used for development, test, pre-production including pilots of the application, and production for the Mobile eBenefits application. The development team will coordinate access to the environment and escalation of software to successive levels of environments (e.g. development to pre-production) with the VA governance board and IV&V team.

The sections below describe the configuration that has been requested for a development and demonstration environment within MAE as well as a separate environment for functional testing of the application.

Integration Testing

The table below contains the list of servers within the MAE 007 environment in order to perform internal verification of the application.

Server Description	Configuration	Number of Servers
Apache web server, proxy and load balancer	2 GB RAM, 1 VPU, 80 GB Disk space	1
Authorization Server	2 GB RAM, 1 VPU, 80 GB Disk space	1
Launchpad Server	2 GB RAM, 1 VPU, 80 GB Disk space	1
Application Servers	4 GB RAM, 2 VPU, 160 GB Disk space, executing two or more DropWizard applications/Java JVMs	2

User Acceptance Testing, Pre-production and Production

The table below contains the list of servers currently planned for the functional test environment for Mobile eBenefits. The type of access to legacy VA systems in this environment is still to be determined. In particular, will this access be to a mock interface or to a live system with obfuscated data.

Server Description	Configuration	Number of Servers
Apache web server, proxy and load balancer	2 GB RAM, 1 VPU, 80 GB Disk space	1
Authorization Server	2 GB RAM, 1 VPU, 80 GB Disk space	1
Launchpad Server	2 GB RAM, 1 VPU, 80 GB Disk space	1
Application Servers	4 GB RAM, 2 VPU, 160 GB Disk space, executing two or more DropWizard applications/Java JVMs	2

8.3. FISMA HIGH MAE 009 Deployment

The FISMA HIGH MAE environments will be used for all pilot and production deployments for the Mobile eBenefits application. As mentioned above, a common Launchpad application will be shared between Mobile eBenefits and the Mobile Health Adapter application. In addition the existing Authorization Server and service will be shared between the applications.

Terremark’s Dedicated Core Cloud has been validated and certified as FISMA High by the VA Office of Cyber Security. Using the core cloud will allow VA to take advantage of the VA-standardized security, monitoring, and other VA hosted processes.

The tactical use of the VA Enterprise Mobile Framework is the reuse of existing infrastructure while fostering innovative app development (by reducing integration overhead) and allowing app developers to focus on the creation of high value, niche workflow apps. This framework exists and can be leveraged for Mobile eBenefits in the VA dedicated FISMA “High” Certified Federal Data Center cloud Environment (a.k.a. VA Core Cloud) currently supported by Terremark.

The deployment of the Mobile eBenefits application into this environment will match that shown in the Runtime Environments section above. Additional servers may be added as needed to support load balancing and distribution. Connections to the VA systems that provide data to the Mobile eBenefits application will occur in this environment.

Within the VA Dedicated Cloud a number of virtual servers make up the current VAMF Production Architecture. Each of these systems serves a specific and distinct purpose. There are VAMF servers that connect to other VA production systems containing PII, systems that connect to VA systems that may or may not contain PII data, and systems with no connectivity to other VA systems. The VA Dedicated Cloud located in Culpeper, Virginia, is replicated at the DR site located in Miami, Florida.