|  |  |
| --- | --- |
| Current version | 1.0 DRAFT |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description of change |
| 1.0 | 6 October 2015 | Sriram Nambakam | Initial draft |
| 1.0 | 6 October 2015 | Kumar Kaushik | Adding use-case for LDAP |
|  |  |  |  |

**Reviewers and Signoff History**

|  |  |  |  |
| --- | --- | --- | --- |
| Reviewer | Role | Comments | Date |
|  |  |  |  |

1. Summary 5

1.1. Overview 5

1.2. Purpose 5

1.2.1. Business Driver(s) 5

1.2.2. Technical Driver(s) 5

1.3. Goal 6

2. Problem Description 7

2.1. Problem 7

2.2. Current Product Limitations 7

2.3. Use Cases 7

2.3.1. LDAP over REST for VMware Directory 7

2.3.1.1. REST client interface. 8

2.3.1.2. Rest Engine. 8

2.3.1.3. LDAP Server hookup 9

2.3.2. REST interface for VMware Authentication Framework 9

2.3.3. REST interface for VMware Certificate Authority 9

3. General Architecture. 10

3.1. Server 10

3.1.1. RestEngine. 10

3.1.1.1. Protocol head HTTP 10

Request Line 10

Status/Response line. 11

General Headers 11

Request Headers. 12

Response Headers. 13

Entity Headers 13

3.1.1.2. Protocol head HTTPS 14

3.1.1.3. API’s Exposed 14

3.1.2. Rest Daemon 14

3.2. Common 14

3.2.1. API’s Exposed 14

3.3. Client 14

3.3.1. API’s Exposed 15

3.4. Transport 15

3.4.1. API’s Exposed 15

3.5. Tools 15

4. Product Requirements 16

4.1. Compatibility 16

4.2. Software Lifecycle 16

4.2.1. Installation 16

4.2.2. Deployment 16

4.2.2.1. Third party component deployment 16

4.2.3. Upgrade 16

4.2.3.1. Legacy 16

4.2.3.2. Future 16

4.2.4. Backup/Restore 16

4.3. Performance and Scalability 16

4.3.1. Large Scale Performance Considerations 16

4.3.2. Performance and Scalability Targets 16

4.4. User interfaces 16

5. Other Considerations 17

5.1. Licensing 17

5.2. Disk footprint 17

5.3. Memory footprint 17

6. References 18

# Summary

## Overview

This document provides the specification for a minimal Representational State Transfer (ReST) engine that can be embedded in services that are implemented in ANSI C.

## Purpose

### Business Driver(s)

User interfaces and other services require a HTTP(S) based REST interface for easy integration with our various existing services that are implemented in ANSI C.

A ReST interface is preferred to easily negotiate proxies, firewalls and also provide easier integration with cloud enabled services.

### Technical Driver(s)

The primary technical drivers are the following.

* Size
* Performance
* Extensibility

## Goal

The primary deliverable is a shared library for the ReST engine. It will be supported on the following platforms.

* Linux
* Windows
* Mac

The ReST Engine will support the following protocols.

* HTTP
* HTTPS

The following payload format will be supported.

* Javascript Object Notation (JSON)

The following components will be provided to support the ReST engine.

* A multi-threaded server that consumes the services of the ReST engine
* A multi-threaded client that communicates with the server
* The server and client will support a ReST API that utilizes a message format implemented using JSON.

# Problem Description

## Problem

Various VMware products requires ReST based interface to exchange information in client server architecture. And most of these products are native which is written in ANSI C.

The deliverable of this project (rest engine library) will be consumed by all such native application to provide HTTP(S) based interaction. This engine will implement transport and application layer of network services using HTTP(S) and TCP protocol headers. The payload format of data exchanged will be JSON which will be used by server module to support various use-cases like interaction with LDAP etc.

## Current Product Limitations

The ReST engine must be implemented using ANSI C.

All encryption must be handled through OpenSSL.

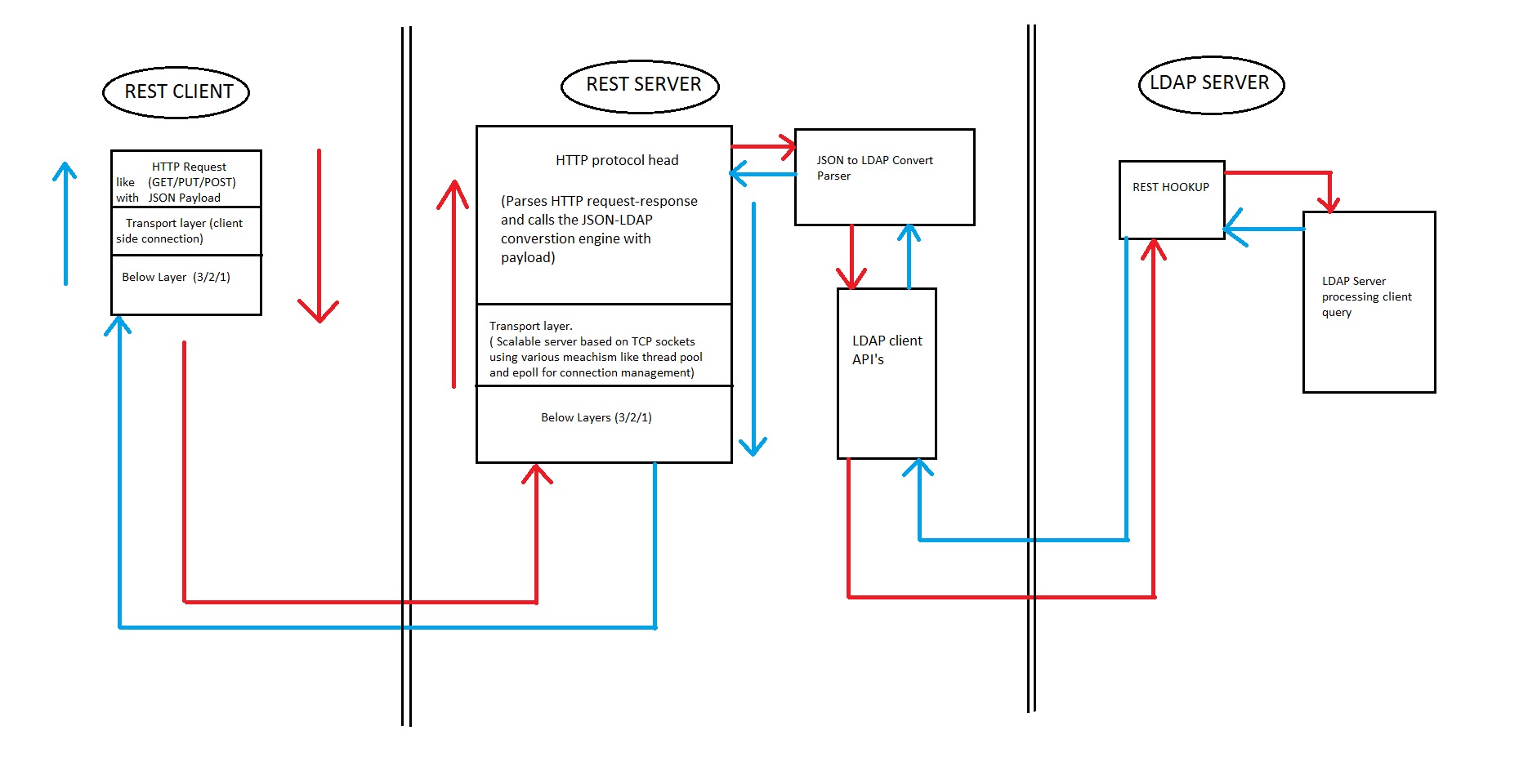
This engine should be highly scalable to handle 10,000’s of client connections.

## Use Cases

### LDAP over REST for VMware Directory

The VMware Directory Service will serve LDAP requests over ReST. In order to achieve this, the directory service will embed the ReST engine.

There will be there major modules for rest engine to work with LDAP. These can be called as



#### REST client interface.

This will initiate client connection to REST server which will communicate using HTTP protocol on top of TCP/IP transport. The Payload format for packets

exchanged will be JSON which contains various attributes and values required for rest server to create LDAP query

#### Rest Engine.

This is the main server side module handing HTTP based requests from client and transforming the request to valid LDAP query.

#### LDAP Server hookup

RestEngine will implement callback to be registered with LDAP server for query management.

### REST interface for VMware Authentication Framework

TBD

### REST interface for VMware Certificate Authority

TBD

# General Architecture.

Major modules for REST-based engine can be classified as

## Server

The server module contains one primary deliverable library called rest engine and one process/daemon called vmrestd.

### RestEngine.

RestEngine is the core of this product, which will be eventually consumed by all services intent to use this embedded C based RESTful Engine.

This library will use functionality provided by transport modules to listen for incoming connection and establish TCP connection with them.

Rest engine provides functionality for thread pool (group of worker threads), which can be assigned dynamically to process data from any connection. These threads will use API’s exposed by transport module to perform basic read and write operations.

Once data is ready, it will be passed to protocol head parsing to extract and populate internal data structures.

Following are the major fields of interest for rest engine from HTTP protocol header.

Please refer RFC 2616 for more details.

#### Protocol head HTTP

HTTP request message format.

Request/Response <CR LF>

One or more general headers ( ‘:’ separated attribute-value pair) <CR LF>

One or more request/response headers ( ‘:’ separated attribute-value pair) <CR LF>

One or more entity headers ( ‘:’ separated attribute-value pair) < CR LF>

<CR LF>

Message-body

##### Request Line

<Method> SP <Request-URI> SP <HTTP-Version> CRLF

* Methods

This indicated the operation to be performed on resource identified by Request URI.  
Following request tokens are supported by our rest engine.

GET  
POST  
DELETE

* Request-URI  
  This identifies the resource upon which request has to be applied.
* HTTP-Version  
  Only HTTP/1.1 is supported by rest engine.

##### Status/Response line.

Response Line = <HTTP-Version> SP <Status-Code> SP <Reason-Phrase> CRLF

* HTTP-Version  
  Only HTTP/1.1 is supported by rest engine.
* Status Code   
  - 1xx: Informational - Request received, continuing process.   
  - 2xx: Success - The action was successfully received, understood, and accepted.   
  - 3xx: Redirection - Further action must be taken in order to complete the request.   
  - 4xx: Client Error - The request contains bad syntax or cannot be fulfilled.  
  - 5xx: Server Error - The server failed to fulfill an apparently valid request.

* Reason-Phrase  
  Please refer RFC 2616.

##### General Headers

These header fields are applicable for both request and response messages but does not apply on entity being transferred.

Following general headers are supported by rest engine.

* Cache control  
  TBD
* Connection  
  Connection status after completion of response. This helps in identifying persistent versus non-persistent connection which are necessary for rest engine end use-case.
* Trailer  
  To be used with chunk transfer encoding helping recipient to know what headers field are expected in the trailer.
* Transfer encoding.  
  This header field indicates what transformation has been applied to the message for transfer between sender and receiver.

##### Request Headers.

These are additional header fields in request messages which allows clients to pass additional information about the request and client itself to the server.

Following request headers are supported by restengine.

* Accept.  
  Use to specify media type which are acceptable in response. For rest engine case this should be JSON. The syntax for this header field is little different from attribute-value pair.
* Accept –charset  
  This tell server which character set is supported at client side so that response can be understood.
* Accept-encoding  
  This tell content –coding that are acceptable in response.
* Accept-Language.  
  This restricts the set of natural language that are preferred as a response to request.
* Authorization

TBD

* From  
  Human readable email address of client using the REST services.
* Host  
  Specifies the internet host and port number of resource being requested. This is obtained from URI given by user.
* Referer  
  This tells server what URL lead the request to arrive at server.

##### Response Headers.

These are the header, which lets server to pass additional information in the response message, which cannot be provided in the response/status line.

Rest engine supports following response headers.

* Accept range.  
  Allows the server to indicate its acceptance of range request for resource.
* Location

This header field is used for redirecting the recipient to a location other tha n request URI for completion of request. This can be used with 201 (creation) or 3XX response code.

* Proxy-Authenticate.  
  TBD
* Retry-After  
  This can be used with 503 service unable response code to indicate how long service is expected to be unavailable to the requested client.
* Server.  
  The Server response-header field contains information about the software used by the origin server to handle the request. For case of rest engine this can be name of service used to query the request. (LDAP, VMAFD, VMCA etc.)

##### Entity Headers

These are the headers fields associated with the entity body commonly known as payload. This can be used with both request and response messages both.

Rest engine supports following entity headers.

* Allow  
  This lists the set of methods supported by the resource identified. This must be used by 405 status message. Currently our rest engine support GET, POST and DELETE methods only.
* Content-Encoding  
  This specifies the additional content encoding that has been applied to entity body helping in the decoding of media type.
* Content-Language  
  Specifies the natural language of enclosed entity.
* Content-length  
  This indicates the size of entity body in decimal number of Octets.
* Content-Location  
  If location of resource is different from the one which specified in URL.
* Content-MD5  
  TBD
* Content-Range  
  This is used in case of partial entity body to tell where in full entity, this partial entity has to be applied.
* Content-Type  
  Specifies the media type which in case of rest engine would be JSON.

#### Protocol head HTTPS

TBD

#### API’s Exposed

/\*

\* @brief Rest engine initialization

\*

\* @param[in] void

\* @param[out] void

\* @return Returns 0 for success

\*/

uint32\_t

VmRESTEngineInit(

void

);

/\*

\* @brief Rest engine shutdown

\*

\* @param[in] void

\* @param[out] void

\* @return Returns 0 for success

\*/

void

VmRESTEngineShutdown(

void

);

### Rest Daemon

This will use services of rest engine library to host and REST server running in process.

## Common

This modules provides library which can be used by any other module present in the system. This consists of common utility like memory allocation and delocation functions etc.

### API’s Exposed

/\*

\* @brief Allocation of heap memory for rest engine.

\*

\* @param[in] size of memory to be allocated

\* @param[out] pointer to allocated memory

\* @return Returns 0 for success

\*/

uint32\_t

VmRESTAllocateMemory(

size\_t dwSize,

void\*\* ppMemory

);

/\*

\* @brief Free of head memory for rest engine.

\*

\* @param[in] pointer to allocated memory

\* @return Returns 0 for success

\*/

void

VmRESTFreeMemory(

void\* pMemory

);

## Client

This module will provide a library which can be consumed by process/daemon of tools module to initiate a connection to rest server. This will be capable of creating HTTP(S) based packet and will use transport layer exposed API’s for low level communication.

### API’s Exposed

TBD

## Transport

This module is responsible for providing library which can be used by rest engine for handling all TCP/IP layer communication. This will support mac, windows and linux platforms. Entire responsibility of creating and maintaining scalable server sockets lies within scope of this. Server listen for new connection in separate thread and adds any new connection to event queue which will be eventually processed by worker threads of rest engine. This module will expose API ‘s which will perform read and write operations on war sockets.

### API’s Exposed

TBD

## Tools

TBD

# 

# Product Requirements

## Compatibility

## Software Lifecycle

### Installation

### Deployment

#### Third party component deployment

### Upgrade

#### Legacy

#### Future

### Backup/Restore

## Performance and Scalability

### Large Scale Performance Considerations

### Performance and Scalability Targets

## User interfaces

# Other Considerations

## Licensing

The code will be distributed using the Apache 2.0 license

## Disk footprint

TBD

## Memory footprint

TBD

# References