# Contents

1.0 **Introduction** ........................................................................................................................................... 7  
  1.1 Intended Audiences ................................................................................................................................. 7  
  1.2 Notes and Symbol Convention .................................................................................................................. 7  
  1.3 Terminology ................................................................................................................................................ 7  
  1.4 References .................................................................................................................................................. 8  

2.0 **Chapter Heading** ................................................................................................................................. 10  
  2.1 Core Ubuntu* Installation ........................................................................................................................... 10  
  2.2 Configure Server Internet Access ............................................................................................................. 10  
  2.3 OpenJDK 1.8 Java* Runtime Environment Installation ............................................................................. 10  
    2.3.1 Adding the PPA Repository for OpenJDK (Ubuntu 14.04.4 Only) ......................................................... 10  
    2.3.2 Installing the OpenJDK Java Runtime Environment ............................................................................ 10  
    2.3.3 Verifying the OpenJDK Installation ....................................................................................................... 10  
  2.4 PostgreSQL 9.5 Installation ....................................................................................................................... 11  
    2.4.1 Adding the Repository for PostgreSQL .................................................................................................. 11  
    2.4.2 Installing PostgreSQL 9.5 ................................................................................................................... 11  
    2.4.3 Verifying the PostgreSQL 9.5 Installation ............................................................................................ 11  
  2.5 Additional Package Installation ................................................................................................................... 11  
  2.6 Custom iPXE ............................................................................................................................................... 12  
    2.6.1 Custom iPXE Compilation .................................................................................................................... 12  
    2.6.2 Custom iPXE Installation ....................................................................................................................... 12  
  2.7 Setting Up Grand Unified Bootloader ......................................................................................................... 12  
  2.8 Network Time Protocol (NTP) Configuration ........................................................................................... 13  
  2.9 Package Signature ...................................................................................................................................... 13  
    2.9.1 Signing a Package ................................................................................................................................. 14  
    2.9.2 Checking Package Signature ................................................................................................................ 14  
    2.9.3 Automated Installation with a Check of the Package Signature .......................................................... 14  
  2.10 Installation of PODM by Using .deb Packages ....................................................................................... 14  

3.0 **Configuration** ......................................................................................................................................... 15  
  3.1 Discovery .................................................................................................................................................... 15  
    3.1.1 Requirements ....................................................................................................................................... 15  
    3.1.2 Customization ...................................................................................................................................... 15  
  3.2 Service Connection .................................................................................................................................... 16  
    3.2.1 Rest Client Configuration ...................................................................................................................... 16  
  3.3 Event Handling .......................................................................................................................................... 17  
    3.3.1 Northbound .......................................................................................................................................... 17  
    3.3.2 Southbound ......................................................................................................................................... 17  
    3.3.3 Customization ...................................................................................................................................... 17  
    3.3.4 Impact on Composed Node Status ....................................................................................................... 18  
  3.4 Service Root Universally Unique ID (UUID) Configuration ......................................................................... 18  
  3.5 Storage Service Configuration ................................................................................................................... 19  
    3.5.1 Storage Service Working on Drawer's Computer System ......................................................................... 19  
    3.5.2 Storage Service Working on External Host Attached to Rack's Storage Network .................................... 19  
  3.6 Network Management ............................................................................................................................... 19  
  3.7 Retention Policy ....................................................................................................................................... 20  
  3.8 Task Synchronization ................................................................................................................................. 20  

4.0 **Security** .................................................................................................................................................. 21  
  4.1 Securing the PODM Northbound Application Programming Interface .................................................... 21
4.1.1 Basic Authentication Management

4.2 Securing the PODM Southbound Application Programming Interface
   4.2.1 PODM as a Server
   4.2.2 PODM as a Client

4.3 PODM Client Certificate

4.4 PODM Server Certificate Verification

4.5 Detailed Configuration

4.6 Certificate Management
   4.6.1 Creating Certificates
   4.6.2 Certificate Creation Examples
   4.6.3 Importing Certificates
   4.6.4 Distributing CA Certificates
   4.6.5 Supplying RMM and PSME with Certificates
   4.6.6 Supplying Storage Service with Certificates

4.7 Using Custom VAULT Configuration

5.0 Deployment
   5.1 Starting the PostgreSQL Service
   5.2 Starting the PODM Service
   5.3 Accessing the PODM REST Application Programming Interface

6.0 Features
   6.1 Composed Node Lifecycle Management
   6.1.1 Composed Node by Using the JSON Template
   6.1.2 Specifying Requirements for a Composed Node
   6.1.3 General Assumptions for Allocation
   6.1.4 Allocation Algorithm
   6.2 NVMe over Fabrics (NVMe-oF)
      6.2.1 Recommended Quality of Service (QoS) Settings for NVMe over Fabrics
   6.3 Using PODM with RMM Service
      6.3.1 Rack Lifecycle Policy (Chassis of Type Rack)
   6.4 Deep Discovery
      6.4.1 General Requirements, Assumptions, and Limitations
      6.4.2 Building the Linux* Utility Image
      6.4.3 Configuring Deep Discovery
   6.5 InBand Service Support
      6.5.1 How PODM Detects InBand Service
      6.5.2 Enabling Aggregation of InBand Service Resources
      6.5.3 How Resources Are Aggregated
      6.5.4 Sub-resource Collections Aggregation
   6.6 Link Aggregation Group
      6.6.1 Creating the Link Aggregation Group
      6.6.2 Modifying the Link Aggregation Group
      6.6.3 Removing the Link Aggregation Group
      6.6.4 Limitations
   6.7 PODM Northbound Interface Eventing
      6.7.1 Supported Events
      6.7.2 Event Subscription
      6.7.3 Registering New Subscription
   6.8 Booting with Internet Small Computer Systems Interface Out-of-band Feature
      6.8.1 Booting from iSCSI
      6.8.2 Setting Chap Data
      6.8.3 Setting Boot Data
6.8.4 Limitations...................................................................................................................................... 44

7.0 Appendix .......................................................................................................................................... 46

7.1 Database Status across PODM Restarts .......................................................................................... 46

7.2 Source Code Compilation (Ubuntu*) ................................................................................................ 46

7.2.1 Key Components .......................................................................................................................... 46

7.2.2 OpenJDK 1.8 Java Development Kit Installation .......................................................................... 47

7.2.3 Verifying OpenJDK Installation .................................................................................................... 47

7.2.4 PostgreSQL 9.5 Installation ........................................................................................................ 47

7.2.5 Compilation of the PODM Application ......................................................................................... 48

7.2.6 Configuration of ISC DHCP Server ............................................................................................... 48

Tables

Table 1 Terminology ................................................................................................................................... 7
Table 2 Reference Documents and Resources .......................................................................................... 8
Table 3 Packages ...................................................................................................................................... 14
Table 4 Sub-resource Collections Aggregation ....................................................................................... 39
Table 5 Key Values to Create Link Aggregation Group .......................................................................... 40
Table 6 Key Values to Modify Link Aggregation Group .......................................................................... 41
Table 7 Key Components for Source Code Ubuntu Compilation .............................................................. 47
## Revision History

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>001US</td>
<td>Initial release of Intel® RSD PODM User Guide software v2.3</td>
<td>May 2018</td>
</tr>
</tbody>
</table>
1.0 Introduction

This document contains information about the installation and configuration of Software Release version 2.3 of Intel® Rack Scale Design (Intel® RSD) PODM and is referred to as Pod Manager (PODM) throughout this document.

1.1 Intended Audiences

The intended audiences for this document include:

- Independent Software Vendors (ISVs) of pod management software, who make use of PODM to discover, compose, and manage drawers, regardless of the hardware vendor, and/or manage drawers in a multivendor environment
- Original Equipment Manufacturers (OEMs) of PSME firmware who would like to provide the Intel® RSD PODM REST API Specification Software v2.3 on top of their hardware platform. Refer to Table 2.

1.2 Notes and Symbol Convention

Symbol and note conventions are similar to typographical conventions used in the Cloud Infrastructure Management Interface 6 (CIMI) Model and RESTful HTTP-based Protocol 7 An Interface for Managing Cloud Infrastructure specification, refer to Table 2. Notation used in JSON* serialization description:

- Values in italics indicate data types instead of literal values.
- Characters are appended to items to indicate cardinality:
  - ? (0 or 1)
  - * (0 or more)
  - + (1 or more)
- Vertical bars, |, denote choice. For example, a|b means a choice between a and b.
- Parentheses, ( ), indicate the scope of the operators ? , * , + , and |.
- Ellipses, ..., indicate points of extensibility. The lack of an ellipsis does not mean no extensibility point exists; rather, it is just not explicitly called out.

1.3 Terminology

Table 1 provides a list of terminology used throughout this document and their definitions.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>BMC</td>
<td>Integrated Baseboard Management Controller</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate Authority</td>
</tr>
<tr>
<td>CM</td>
<td>Control Module</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IBL</td>
<td>Intel Business Link</td>
</tr>
<tr>
<td>iSCSI</td>
<td>Internet Small Computer System Interface</td>
</tr>
<tr>
<td>IQN</td>
<td>iSCSI Qualified Name</td>
</tr>
</tbody>
</table>

Table 1 Terminology
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISVs</td>
<td>Independent Software Vendors</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>LAG</td>
<td>Link Aggregation Group</td>
</tr>
<tr>
<td>LUI</td>
<td>Linux* Utility Image</td>
</tr>
<tr>
<td>MMP</td>
<td>Management Midplane</td>
</tr>
<tr>
<td>NIC</td>
<td>Network Interface Card</td>
</tr>
<tr>
<td>NVMe-ofF*</td>
<td>NVM Express over Fabrics*, for more information refer to: <a href="http://nvmexpress.org/resources/specifications">http://nvmexpress.org/resources/specifications</a></td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>OOB</td>
<td>Out-of Band</td>
</tr>
<tr>
<td>PKCS #12</td>
<td>Personal Information Exchange Syntax Standard</td>
</tr>
<tr>
<td>POD</td>
<td>A physical collection of multiple racks</td>
</tr>
<tr>
<td>PODM</td>
<td>Pod Manager</td>
</tr>
<tr>
<td>PPA</td>
<td>Personal Package Archives</td>
</tr>
<tr>
<td>PSME</td>
<td>Pooled System Management Engine</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RDMA</td>
<td>Remote Direct Memory Access</td>
</tr>
<tr>
<td>Redfish*</td>
<td>DMTF standard, for more information refer to: <a href="https://www.dmtf.org/standards/redfish">https://www.dmtf.org/standards/redfish</a></td>
</tr>
<tr>
<td>REST</td>
<td>Representational state transfer</td>
</tr>
<tr>
<td>RMM</td>
<td>Rack Management Module</td>
</tr>
<tr>
<td>RSA</td>
<td>Public key cryptosystem</td>
</tr>
<tr>
<td>RSS</td>
<td>RSD Storage Service</td>
</tr>
<tr>
<td>SB</td>
<td>Southbound API</td>
</tr>
<tr>
<td>SSDP</td>
<td>Simple Service Discovery Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>TFTP</td>
<td>Trivial File Transfer Protocol</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>ToR</td>
<td>Top of Rack</td>
</tr>
<tr>
<td>UEFI</td>
<td>Unified Extensible Firmware Interface</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally Unique Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
</tbody>
</table>

### 1.4 References

Table 2 provides a list of documents and resources referenced in this document.

<table>
<thead>
<tr>
<th>Doc ID</th>
<th>Title</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>337197</td>
<td>Intel® Rack Scale Design (Intel® RSD) Conformance and Software Reference Kit Getting Started Guide Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337198</td>
<td>Intel® Rack Scale Design (Intel® RSD) POD Manager (PODM) Release Notes Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337200</td>
<td>Intel® Rack Scale Design (Intel® RSD) Pooled System Management Engine (PSME) Release Notes Software v2.3</td>
<td></td>
</tr>
<tr>
<td>Doc ID</td>
<td>Title</td>
<td>Location</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>337201</td>
<td>Intel® Rack Scale Design (Intel® RSD) Firmware Extension Specification Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337202</td>
<td>Intel® Rack Scale Design (Intel® RSD) Storage Services API Specification Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337203</td>
<td>Intel® Rack Scale Design (Intel® RSD) Architecture Specification Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337204</td>
<td>Intel® Rack Scale Design (Intel® RSD) POD Manager (PODM) Representational State Transfer (RESTful) API Specification Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337205</td>
<td>Intel® Rack Scale Design (Intel® RSD) Rack Management Module (RMM) Representational State Transfer (RESTful) API Specification Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337206</td>
<td>Intel® Rack Scale Design (Intel® RSD) Generic Assets Management Interface (GAMI) API Software v2.3</td>
<td></td>
</tr>
<tr>
<td>337207</td>
<td>Intel® Rack Scale Design (Intel® RSD) Pooled System Management Engine (PSME) Representational State Transfer (RESTful) API Specification Software v2.3</td>
<td></td>
</tr>
<tr>
<td>DSP0266</td>
<td>Scalable Platforms Management API Specification v1.4.0</td>
<td><a href="https://www.dmtf.org/sites/default/files/DSP0266_1.4.0.pdf">https://www.dmtf.org/sites/default/files/DSP0266_1.4.0.pdf</a>.</td>
</tr>
<tr>
<td>DSP8010</td>
<td>Redfish Schema v2016.3</td>
<td><a href="https://www.dmtf.org/sites/default/files/standards/documents/DSP8010_2016.3.zip">https://www.dmtf.org/sites/default/files/standards/documents/DSP8010_2016.3.zip</a></td>
</tr>
<tr>
<td>RFC 2119</td>
<td>Key Words for Use in RFCs to Indicate Requirement Levels, March 1997</td>
<td><a href="https://ietf.org/rfc/rfc2119.txt">https://ietf.org/rfc/rfc2119.txt</a></td>
</tr>
<tr>
<td>N/A</td>
<td>NVM Express over Fabrics</td>
<td><a href="http://nvmexpress.org/wp-content/uploads/NVMe_over_Fabrics_1.0_Gold_20160605-1.pdf">http://nvmexpress.org/wp-content/uploads/NVMe_over_Fabrics_1.0_Gold_20160605-1.pdf</a></td>
</tr>
</tbody>
</table>
2.0 Chapter Heading

PODM software can be installed on Ubuntu* Server v14.04.4 and v16.04.1.

2.1 Core Ubuntu* Installation

We recommend using a machine with at least 32 GB of storage space available. Download Ubuntu Server v14.04.4 or v16.04.1. Boot the target machine using one of those images and follow the installation instructions.

2.2 Configure Server Internet Access

PODM software installation requires access to public software repositories on the Internet. Confirm that the server network, firewall, and proxy configurations are configured properly to allow Internet access.

2.3 OpenJDK 1.8 Java* Runtime Environment Installation

OpenJDK v1.8 Java* Runtime Environment is required by the PODM. If the OpenJDK 8 Java* Runtime Environment is already installed, refer Section 2.3.3, Verifying the OpenJDK Installation to verify installation.

If OpenJDK v8 is not installed, then follow the steps below to set up the Java environment correctly.

2.3.1 Adding the PPA Repository for OpenJDK (Ubuntu 14.04.4 Only)

Do the following:

1. Add the OpenJDK 1.8 repository to the /etc/apt/sources.list file by adding this line:
   
   ```
   deb http://ppa.launchpad.net/openjdk-r/ppa/ubuntu trusty main
   ```

2. Download key:
   
   ```
   sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 86F44E2A
   ```

3. Update the apt-get repositories:
   
   ```
   sudo apt-get update
   ```

2.3.2 Installing the OpenJDK Java Runtime Environment

Do the following:

Install the OpenJDK v1.8 Java Runtime Environment package:

```
sudo apt-get install openjdk-8-jre-headless
```}

2.3.3 Verifying the OpenJDK Installation

To verify the installation of the OpenJDK perform the following:

1. Check the default java version by typing:
   
   ```
   java -version
   ```

2. If the above command does not show OpenJDK v1.8, execute the following command to set Java defaults:
   
   ```
   sudo update-alternatives --config java
   ```
2.4 PostgreSQL 9.5 Installation

PostgreSQL 9.5 is required by the PODM. If PostgreSQL 9.5 is already installed on the system, then refer to Section 2.4.3, Verifying PostgreSQL 9.5 installation to verify the installation.

If PostgreSQL 9.5 is not installed, then follow these steps to install it correctly.

2.4.1 Adding the Repository for PostgreSQL

Do the following:

1. Create the file /etc/apt/sources.list.d/pgdg.list and add this repository line:
   - For Ubuntu v14.04:
     
     ```
     deb http://apt.postgresql.org/pub/repos/apt/ trusty-pgdg main
     ```
   - For Ubuntu v16.04.1:
     
     ```
     deb http://apt.postgresql.org/pub/repos/apt/ xenial-pgdg main
     ```
   - Import the repository signing key:
     
     ```
     wget --quiet -0 - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -
     ```

2. Update the apt-get repositories:

   ```
   sudo apt-get update
   ```

2.4.2 Installing PostgreSQL 9.5

Do the following:

Install PostgreSQL 9.5 packages:

   ```
   sudo apt-get install postgresql-9.5 postgresql-contrib-9.5
   ```

2.4.3 Verifying the PostgreSQL 9.5 Installation

Do the following:

1. Check if PostgreSQL 9.5 is installed and runs on port 5432:

   ```
   pg_iscluster
   ```

   If it is installed, this command should return output like:

   ```
   Ver Cluster Port Status Owner Data directory Log file 9.5 main 5432 online postgres
   /var/lib/postgresql/9.5/main /var/log/postgresql/postgresql-9.5-main.log
   ```

2.5 Additional Package Installation

The following is a procedure to install the additional package:

1. Install the following packages:

   ```
   sudo apt-get install isc-dhcp-server
   sudo apt-get install openssh-server
   sudo apt-get install python3
   sudo apt-get install tftp-hpa
   sudo apt-get install ntp
   sudo apt-get install vlan
   sudo apt-get install acl
   ```
2. Update the apt-get repositories:

```bash
sudo apt-get update
```

### 2.6 Custom iPXE

Custom iPXE is required for full PODM functionality.

It is not provided as part of the PODM Debian* (.deb*) packages created after compiling the PODM source.

#### 2.6.1 Custom iPXE Compilation

These packages must be installed on Ubuntu v14.04.4 or v16.04.1.

```bash
sudo apt-get install build-essential
sudo apt-get install genisoimage
sudo apt-get install git
sudo apt-get install liblzma-dev
```

1. **Clone iPXE repository.**

   ```bash
   git clone https://git.ipxe.org/ipxe.git
   ```

2. **Copy the following file from the PODM source package to the `ipxe/src/` directory.**

   ```bash
   SW/external/ipxe-dhcp
   ```

3. **Compile iPXE by executing the following command from the `ipxe/src` directory.**

   ```bash
   make EMBED=ipxe-dhcp
   ``

   Transfer `bin/undionly.kpxe` to the target machine.

   **Note:** In case of iPXE dependency or compilation issues, visit: [http://ipxe.org/docs](http://ipxe.org/docs).

#### 2.6.2 Custom iPXE Installation

Do the following:

1. **Create the `/srv/tftp` directory on the target machine.**

   ```bash
   sudo mkdir -p /srv/tftp
   ```

2. **On the target machine copy** `undionly.kpxe` **to** `/srv/tftp`.

3. **Make symlink to podmipxe.0.**

   ```bash
   cd /srv/tftp
   sudo ln -s undionly.kpxe podmipxe.0
   ```

### 2.7 Setting Up Grand Unified Bootloader

Configure the system to use ethX naming convention for Ethernet interfaces.

Do the following:

1. **Edit the `/etc/default/grub` file and comment these variables:**

   ```bash
   #GRUB_HIDDEN_TIMEOUT
   #GRUB_HIDDEN_TIMEOUT_QUIET
   ```

2. **Edit the `/etc/default/grub` file and modify the following variables:**

   ```bash
   GRUB_CMDLINE_LINUX_DEFAULT="" GRUB_TERMINAL=console GRUB_CMDLINE_LINUX="nomodeset net.ifnames=0 biosdevname=0 acpi-osi=" GRUB_TIMEOUT=2 GRUB_RECORDFAIL_TIMEOUT=2
   ```
3. Save the file and apply changes:

```bash
sudo update-grub
```

4. Restart the system for the changes to take effect.

### 2.8 Network Time Protocol (NTP) Configuration

Do the following:

1. Edit the `/etc/ntp.conf` file and add the following lines:

```plaintext
tos maxdist 16 # In case of lost connection to external NTP server, PODM shall use itself as a NTP server. # This might happen if PODM has no access to worldwide network or there is a temporary connectivity problem. server 127.127.1.0 fudge 127.127.1.0 stratum 10 restrict -4 default kod notrap nomodify nopeer noquery restrict -6 default kod notrap nomodify nopeer noquery restrict 10.3.0.0 255.255.252.0 nomodify notrap restrict 10.2.0.0 255.255.255.0 nomodify notrap restrict 127.0.0.1 restrict ::1
```

2. Restart the Network Time Protocol (NTP) service:

```bash
sudo service ntp restart
```

### 2.9 Package Signature

If user would like to distribute a PODM package, the PODM package should be signed.

If PODM package will be signed end user will have possibility to verify (for example, before installation) that package can be trusted and has not been modified after it was signed.

To sign `.deb` package the user should follow the steps described in Section 2.9.1 Signing a Package. If the PODM package has been signed, before installing the PODM package the user can verify the package signature by using steps described in Section 2.9.2, Checking Package Signature.

To sign `.deb` packages, use Linux* GPG and debsigns:

1. To install debsigns, use:

   ```bash
   sudo apt-get install debsigns
   ```

2. To sign package, gpg uses key-pair. To check existing keys in the system, use following command:

   ```bash
gpg --list-key
   ```

3. To create a new key pair, use the following command:

   ```bash
   gpg --gen-key
   ```

4. When prompted, specify key type, desired key size, length of time the key should be valid, key owner, and email address for the owner. For more information, refer to:

   ```
   ```

**Note:** Above step requires to have a good source of entropy. Failure to provide sufficient entropy can result in failure with error "not enough entropy".

5. To export the created public key to file, use the command:

   ```bash
   gpg --armour --output /tmp/podm.key --export <owner name>
   ```
2.9.1 Signing a Package

To sign a .deb package use the command below:

```
dpkg-sig -s builder <podm package>
```

Once the packages are signed, use the following guide to exchange your GPG key with the recipient:


2.9.2 Checking Package Signature

Before checking a signature of a .deb package, the user must import the GPG public key that was used during package signing.

Do the following:

1. Import the key used during package signing using below command:
   
   ```
gpg --import <gpg public key file>
   ```

2. Verify package signature. To verify a signature of a .deb package, run the following command:

   ```
sudo dpkg-sig -c <podm package>.deb
   ```

2.9.3 Automated Installation with a Check of the Package Signature

To automate the installation process of the signed package (with its signature verification), Run this command:

```
./pod-manager-install.py
```

In the SW directory created with the srcTar task. The prerequisite to run the script is Python3*. Also, the dpkgsig package must be installed. The supported version of the GPG is v1.4.20.

2.10 Installation of PODM by Using .deb Packages

PODM .deb packages can be generated manually as described in Section 7.2, Source Code Compilation (Ubuntu*). Table 3 shows the order for installing the PODM .deb packages.

Table 3. Packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pod-manager</td>
<td>Yes</td>
<td>Creates user for services. Also contains a web server, database, updated scripts, and the PODM configuration.</td>
</tr>
<tr>
<td>pod-manager-networking</td>
<td>Yes</td>
<td>Contains the configuration for the network, Dynamic Host Configuration Protocol (DHCP), NTP, and Trivial File Transfer Protocol (TFTP). <strong>CAUTION</strong> This package overrides the settings for network interfaces in the user system. If the user has custom network settings on the machine on which PODM has been installed, then those custom settings will be wiped out by the PODM package installation.</td>
</tr>
</tbody>
</table>
3.0 **Configuration**

Refer to these sections for information about configuration.

### 3.1 Discovery

There are two available mechanisms to discover new services, based on DHCP protocol or based on Simple Service Discovery Protocol (SSDP) protocol. By default, both of them are enabled and the same service can be detected by both mechanisms. It is highly recommended to use either one of the two mechanisms to discover Intel® RSD resources.

#### 3.1.1 Requirements

This chapter contains all requirements for the service or hostname to be visible for DHCP or SSDP-based discovery.

##### 3.1.1.1 Hostname-based Discovery by Using DHCP

**Note:** Hostname-based Discovery is required only for the DHCP-based discovery mechanism.

The hostname of the PSME services must start with the string "psme", hostname of storage services must start with string "storage", and hostname of deep discovery LUI service must be set to string "lui", hostname of the RMM service must start with string "rmm".

Examples include:

- psme1
- storagel
- lui
- rmm2

##### 3.1.1.2 Service-name-based Discovery by Using Simple Service Discovery Protocol

**Note:** This is required only for the SSDP-based discovery mechanism.

The property names available on the service entry point (/redfish/v1) must be set to these values:

- For PSME service: **PSME Service Root**
- For RSD Storage Service (RSS) service: **RSS Service Root**
- For LUI service: **LUI Service Root**
- For RMM service: **Root Service**

These values can be changed in the configuration file under **ServiceTypeMapping** as described in Section 3.1.2, Customization.

**Caution:** If the PODM is unable to match the value of the "Name" property with any value present in the configuration file, it will assume the service type to be the PSME.

### 3.1.2 Customization

Configuration for discovery mechanisms is at `/etc/pod-manager/service-detection.json`:

```json
{
}
```


```
"EnabledProtocols": [ 
  "DHCP",
  "SSDP"
],
"ServiceTypeMapping": { 
  "PSME": "PSME Service Root",
  "RSS": "RSS Service Root",
  "LUI": "LUI Service Root",
  "RMM": "Root Service",
  "INBAND": "In Band Service"
},
"Protocols": { 
  "SSDP": { 
    "AnnouncementFrequencyInSeconds": 600,
    "MX": 5,
    "Subnets": [
      "0.0.0.0/0"
    ]
  },
  "DHCP": { 
    "FilesCheckIntervalInSeconds": 10,
    "NumberOfRetriesForFailedServiceCheck": 5,
    "FailedEndpointRecheckIntervalInSeconds": 300
  }
}
```

- **EnabledProtocols**: defines which protocols for discovery mechanisms are enabled.
  Possible options are: "DHCP", "SSDP".
- **ServiceTypeMapping**: contains mapping between service types and property "Name" available on service root of the service.
- **Protocols**: contains configuration specific to selected protocols.
  - **SSDP**:
    - **AnnouncementFrequencyInSeconds**: how often the M-SEARCH message will be sent to network
    - **MX**: configuration of the MX parameter
    - **Subnets**: from which subnets PODM will accept "ssdp:alive" or M-SEARCH response message. "0.0.0.0/0" means that all networks are accepted.
  - **DHCP**:
    - **FilesCheckIntervalInSeconds**: defines how often the PODM will refresh data from the DHCP.
    - **NumberOfRetriesForFailedServiceCheck**: defines how many times the PODM will retrieve the service root for newly discovered services.
    - **FailedEndpointRecheckIntervalInSeconds**: defines the interval after which the PODM will retry to retrieve the service root from the service that failed to be available during initial discovery.

## 3.2 Service Connection

Refer to these sections for information regarding service connections.

### 3.2.1 Rest Client Configuration

Configuration for the client connection is located at `/etc/pod-manager/service-connection.json`:
3.3 Event Handling

The PODM is able to subscribe to events and receive them from Redfish* compliant external services when a specific resource is removed, added, or changed.

3.3.1 Northbound

The PODM provides Redfish compliant eventing, and sends events to its subscribers when a specific resource is removed, added, or changed.

3.3.2 Southbound

The PODM is able to subscribe to events and receive them from Redfish compliant external services when a specific resource is removed, added, or changed.

3.3.3 Customization

Configuration for events is located at /etc/pod-manager/events.json:

```json
{
    "Northbound": {
        "DeliveryRetryAttempts": 2,
        "DeliveryRetryIntervalSeconds": 1
    },
    "Southbound": {
        "EventSubscriptionIntervalSeconds": 90,
        "BufferedEventProcessing": {
            "ProcessingWindowSizeInSeconds": 5
        },
        "ServicesConfiguration": [
            {
                "ServiceType": "psme",
                "NetworkInterfaceName": "eth0.4094",
                "PODManagerIpAddress": "127.0.0.1"
            },
            {
                "ServiceType": "rss",
                "NetworkInterfaceName": "eth0.4093",
                "PODManagerIpAddress": "127.0.0.1"
            },
            {
                "ServiceType": "rmm",
                "NetworkInterfaceName": "eth0.4094"
            }
        ]
    }
}
```
Northbound – This section groups configuration parameters related with the PODM Northbound Interface Eventing feature:

- **DeliveryRetryAttempts**: how many times PODM will try to deliver a particular event before the subscription is removed
- **DeliveryRetryIntervalSeconds**: interval between delivery attempts in seconds

Southbound:

- **EventSubscriptionIntervalSeconds**: how often the PODM checks if it is subscribed to a service.

In case of subscription absence, the PODM subscribes to this service.

- **BufferedEventProcessing**: when a section is defined, buffered event processing is enabled
- **ProcessingWindowSizeInSeconds**: incoming events are buffered and processed once per time window (size of which can be adjusted by this attribute)
- **ServicesConfiguration/ServiceType**: name of service type that is configured by this object.

Valid values: “PSME”, “RSS”, “RMM”, and “InBand”.

- **NetworkInterfaceName**: name of network interface used to communicate with a specified service. It is used to determine the PODM IP address for event subscription in PSME/Storage Service/RMM services. When the **PODManagerIpAddress** is present, this parameter is ignored.
- **PODManagerIpAddress**: the PODM IP address used for an event subscription in the PSME/Storage Service/RMM service. This parameter is optional.

### 3.3.4 Impact on Composed Node Status

As a result of event handling, a composed node state might be set to offline and health to critical when:

- Associated computer system’s or remote target’s state is different than Enabled
- Associated computer system is removed or remote target is deleted.

### 3.4 Service Root Universally Unique ID (UUID) Configuration

The Northbound API service root Universally Unique ID (UUID) is stored in the `/var/lib/pod-manager/service-root-uuid.json` configuration file. The PODM will generate a UUID and store it at this location by default. The user can change or set a specific UUID of the PODM service root by editing the above file. Remember, this file must contain a proper UUID of PODM service root in the following format:

```json
{
    "UUID": "00000000-0000-0000-0000-000000000000"
}
```
3.5 Storage Service Configuration

The storage service can be configured in two ways:

1. Storage service working on drawer's computer system.
2. Storage service working on external host attached to rack's storage network.

*Note:* Refer to the Intel® Rack Scale Design PSME User Guide to get details regarding Linux image creation for storage service host. Refer to Table 2.

3.5.1 Storage Service Working on Drawer's Computer System

To have the storage service working on drawer's computer system, the user must deploy the storage service image into the computer system's storage device.

*Note:* Later, the user must allocate (with local boot option enabled) and assemble a new node with this exact computer system.

If such a configured storage system is not assembled by way of the PODM REST API, then it might be targeted for Deep Discovery (the computer system will be rebooted and left in a powered off state).

*Note:* The hostname for the storage service must start with the string “storage”.

3.5.2 Storage Service Working on External Host Attached to Rack's Storage Network

The user can simply attach an external host with the storage service image to the rack's storage network.

*Note:* Additional settings on the Top of Rack (ToR) might be required for this type of deployment. In addition, the hostname for the storage service must start with the string “storage”.

3.6 Network Management

The PODM has these reserved VLANs preconfigured:

- 4091: Production Network
- 4094: Service Management Network
- 4092: Rack Backplane Management
- 4088: InBand Management
- 4093: Storage Management/Access Network
- 4090: External Management Network

Except for VLAN #4090 (which is an external management network), all VLANs are enabled by default with preconfigured IP addresses.

To enable VLAN #4090, the administrator must change the entry (for eth0.4090) in the /etc/network/interfaces.d/pod-manager-network-configuration.conf file or in the /etc/network/interfaces file to make this change permanent between PODM updates.

More information about this file can be found on the Interfaces Manpage* on the Ubuntu website. Remember that the default configuration for this VLAN requires DHCP presence and it must be enabled in this network.

*Note:* The default configuration for VLAN 4090 requires DHCP presence and the virtual VLAN interface must be enabled. Another point to note is that in Ubuntu v16.04, all interfaces present in the above files are read record-by-record. If a record fails to fetch its IP address, then next record will also fail.
To disable the PODM specific network configuration, the administrator must delete 
/etc/network/interfaces.d/pod-manager-network-configuration.conf file and remove this line 
source /etc/network/interfaces.d/pod-manager-network-configuration.conf in the 
/etc/network/interfaces file.

Restart of the networking service is required.

3.7 Retention Policy

The retention policy is the period (given in hours) between the time the external service, like PSME or RMM becomes unavailable and the time the PODM deletes all assets connected to this service from its own database. During this time all assets state will be set to Absent.

The user can provide a special value 0, which will result in the immediate deletion of the given external service (and its assets) at the time of its absence notice. The 0 value might be helpful especially when the PODM communicates with an older version of the Intel® RSD software modules.

To change the retention time change the /etc/pod-manager/external-services.json file:

```json
{
    "RetainUnavailableServicesForHours": 720
}
```

The PODM does not need to be restarted after the change, but loading a new retention time may take time to complete.

3.8 Task Synchronization

In the PODM, various operations have to wait for asynchronous tasks to complete (for example, discovery of an external service). To change the maximum time to wait for asynchronous task completion change /etc/pod-manager/task-synchronization.json file:

```json
{
    "MaxTimeToWaitForAsyncTaskSeconds": 180
}
```

If the asynchronous task (which blocks execution of the requested operation) does not finish within specified time, the requested operation will fail with 503 (Service Temporarily Unavailable) status code.

**Note:** To avoid 503 responses when executing operations on the PODM, it is recommended to set this time to double the maximum time of the longest discovery task.
4.0 Security

Refer to these sections for information about security.

4.1 Securing the PODM Northbound Application Programming Interface

Custom certificates can be configured to secure PODM Northbound API communication. All certificates are stored in a default keystore located at:

/var/lib/pod-manager/keystore.jks

It is possible to change keystore and its password, localization, and stored keys. Keystore path, password, and alias are provided by an entry in the "keystore" xml node contained in file:

/opt/pod-manager/wildfly/standalone/configuration/standalone.xml

Detailed information about WildFly* SSL connections configuration is provided here:

https://docs.jboss.org/author/display/WFLY9/Detailed+Configuration

WildFly supports password hashing for the keystore entry in the standalone.xml file using the VAULT tool. Detailed information appears here:


**Caution:** VAULT uses a container to store the hashed password. Once the user changes the VAULT container, the configuration files MUST be updated as described in Section 4.7, Using Custom VAULT Configuration.

Here is a keystore creation command example:

```
keytool -genkey -alias <alias> -keyalg RSA -keysize 2048 -keystore <keystore_name>
```

Recommended key size is at least 2048 bits. A key with a lower size might lead to compatibility issues.

This is a custom certificate import example:

```
keytool -importcert -file <certificate> -keystore <keystore> -alias <alias>
```

For more information, use this command:

```
man keytool
```

4.1.1 Basic Authentication Management

**Caution:** This section describes configuration of basic access authentication used as a default mechanism to secure the PODM API. In a production environment with stronger security requirements, use a method that is appropriate for its deployment (for example, digest access authentication).

The basic authentication mechanism can be managed by the add-user utility:

```
/opt/pod-manager/wildfly/bin/add-user.sh
```

The default login credentials for the PODM are:

<table>
<thead>
<tr>
<th>User</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>admin</td>
</tr>
</tbody>
</table>
After successful installation default login credentials should be changed. This can be done by this command:

```
/opt/pod-manager/wildfly/bin/add-user.sh -a -u admin -p <new-password>
```

### 4.2 Securing the PODM Southbound Application Programming Interface

The PODM on the Southbound API can act as both a client and server. The Southbound API communication is established by using TLS v1.2 protocol.

While establishing a secure connection with external services (Southbound communication), the PODM employs the `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` cipher suite.

This means the cipher suite implements Elliptic-Curve Diffie-Hellman Ephemeral key exchange. The Elliptic-Curve Digital Signature Algorithm, with AES-128 as the block cipher and SHA-256 HMAC for the authentication hash.

More information about this cipher suite can be found in *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*, Table 2.

These are important points to note:

- For communication to be properly established by the TLS with client authentication at the server side, all communicating components MUST have synchronized time. Time must be set up correctly to fulfill the certificate validation period. "pod-manager-config" package configures the NTP service on the PODM to be used for time synchronization between the PODM and other services (RMM, PSME, LUI, Storage Service). Each service using the PODM as the NTP server must have the correct NTP client configuration.

- The expected format of the client certificate, CA certificate is PKCS #12 (.pfx). It is assumed that the pkcs file with the client signed certificate is a well formatted container that includes a full certification chain.

#### 4.2.1 PODM as a Server

The PODM acts as a server when receiving events (even from clients that are not authenticated).

#### 4.2.2 PODM as a Client

The PODM acts as a client when obtaining information from external services. The PODM is configured to send its own certificate to be authenticated by the server.

*Note:* The LUI does not authenticate the PODM on the Southbound API. The PODM does not perform server authentication.

The PODM uses the CA signed certificate. The CA certificate should be added to external services trust store.

The default client certificate and CA certificate are provided with PODM .deb packages. The client certificate has been signed by the CA certificate, which is also provided.

### 4.3 PODM Client Certificate

PODM (client) certificate is stored in a keystore file:

```
/var/lib/pod-manager/client.jks
```

CA certificates can be created manually and corresponding keys are stored in:

```
/var/lib/pod-manager/ca
```

CA certificate and key is also packed to file in PKCS #12 format so it can be used in other locations.
The TLS can be turned on or off by modifying the following configuration file (turned ON by default):

```
/etc/pod-manager/service-connection.json
```

The TLS can be configured separately by setting the Boolean values accordingly for each external service type.

Example to set TLS to OFF for the PSME:

```
{
    "SslEnabledForRmm" : true,
    "SslPortsForRmm" : [ 8443 ],
    "HttpPortsForRmm" : [ 8888 ],
    "SslEnabledForPsme" : false,
    "SslPortsForPsme" : [ 8443 ],
    "HttpPortsForPsme" : [ 8888 ],
    "SslEnabledForRss" : true,
    "SslPortsForRss" : [ 8443 ],
    "HttpPortsForRss" : [ 8888 ],
    "SslEnabledForLui" : true,
    "SslPortsForLui" : [ 8443 ],
    "HttpPortsForLui" : [ 8888 ],
    "SslPortsForInBand": [ 8448 ],
    "SslPortsForDiscoveryService": [ 8444 ]
}
```

### 4.4 PODM Server Certificate Verification

The PODM verifies server certificate if it has been signed by known authority. Server's root certificate must be imported to the PODM's trust store:

```
/var/lib/pod-manager/server.jks
```

The CA certificates can be created manually and corresponding keys are stored in:

```
/var/lib/pod-manager/ca
```

The CA certificate and key are also packed to file in PKCS #12 format so it can be used in other locations.

Server certificate verification can be turned ON/OFF by modifying following configuration file (turned on by default):

```
/etc/pod-manager/security.json
```

By default server certificate verification is turned on, the property `ServerCertificateVerificationEnabled` is set to True:

```
{
    "ClientKeystorePath": "/var/lib/pod-manager/client.jks",
    "ServerKeystorePath": "/var/lib/pod-manager/server.jks",
    "ServerCertificateVerificationEnabled": true
}
```

To turn OFF server certificate verification:

Set "ServerCertificateVerificationEnabled" property to false:

```
{
    "ClientKeystorePath": "/var/lib/pod-manager/client.jks",
    "ServerKeystorePath": "/var/lib/pod-manager/server.jks",
    "ServerCertificateVerificationEnabled": false
}
```
4.5 Detailed Configuration

The PODM southbound API must be properly configured to establish a secure connection by satisfying the points below:

- Client certificate must be added to keystore located at:
  
  `/var/lib/pod-manager/client.jks`

- Server (PSME/RMM/RSS) root certificate must be added to keystore located at:
  
  `/var/lib/pod-manager/server.jks`

Both keystores `client.jks` and `server.jks` are protected by a password.

To hash the password, the WildFly hashing tool called VAULT is used. Detailed information can be obtained from:


**Caution:** VAULT uses a container to store the hashed password. Once the user changes the VAULT container, the configuration files MUST be updated as described in Using custom VAULT configuration.

- Client and server certificate must be signed by a known CA. A CA certificate may be a self-signed certificate.
- To create a correct key SHA256 with ECDSA, a signature algorithm must be used.
- All servers that will authenticate PODM must have the CA certificate provided.
- All servers root certificates used by PODM to authenticate servers must have the CA certificate provided.

**Caution:** Certificates distributed with PODM has a limited validation period. Check the validity period for certificates before using them. System time CANNOT exceed the certificate expiration date.

The default exported PODM CA certificate distributed with PODM is located at:

`/var/lib/pod-manager/root.crt`

The default server key and certificate are being deployed with PSME/RSS/RMM package. The default keystore that contains default server's CA certificate is at:

`/var/lib/pod-manager/server.jks`

To check the certificate validation period from the `client.jks` or `server.jks` file:

Use the keytool:

```
keytool -v -list -keystore [keystore_file].jks
```

4.6 Certificate Management

Certificates can be created by the user or imported to keystore using tools described in this section.

4.6.1 Creating Certificates

The script to generate client/server certificate is located at:

`/usr/bin/pod-manager-certificate-creation.sh`

The script must be executed with root privileges:

```
sudo /usr/bin/pod-manager-certificate-creation.sh
```

The print script usage using `--help` option:

```
sudo /usr/bin/pod-manager-certificate-creation.sh --help
```
4.6.2 Certificate Creation Examples

User can create client or server certificate using the same script. To create client certificate use this option:

```
certificate-type "client"
```

To create server certificate the user must use this option:

```
certificate-type "server"
```

4.6.2.1 PODM Client Certificate Creation Examples

**Example 1:** Create a new client certificate and sign it with a newly created CA certificate:

```
sudo pod-manager-certificate-creation.sh --certificate-type "client" --ca-certificate-dname "/C=PL/ST=State/L=locality/O=organization/CN=fqdn_or_ip" --certificate-dname "C=PL,ST=State,L=locality,O=organization,CN=fqdn_or_ip"
```

This command creates the client certificate under /var/lib/pod-manager and CA certificate under /var/lib/pod-manager/ca.

**Example 2:** Create a new client certificate and sign it with an existing CA certificate:

```
sudo pod-manager-certificate-creation.sh --certificate-type "client" --ca-certificate-file /path/to/ca/file/ca.pfx --certificate-dname "C=PL,ST=State,L=locality,O=organization,CN=fqdn_or_ip"
```

The above command creates a new client certificate under /var/lib/pod-manager. Whereas, when extracted, the CA certificate and private key will be copied to /var/lib/pod-manager/ca directory.

**Example 3:** Create a new client certificate and sign it with the provided CA certificate by using the interactive mode.

```
sudo /usr/bin/pod-manager-certificate-creation.sh --certificate-type "client" --ca-certificate-file /path/to/ca/file/ca.pfx"
```

During this process, the user will be asked to pass the client certificate attributes manually. Follow the instructions displayed on screen to complete the process. Upon completion, the above command creates a new client certificate under /var/lib/pod-manager directory and extracted CA certificate and private key are copied to /var/lib/pod-manager/ca.

**NOTE:**

1. DO NOT REMOVE keystore file created after certificate generation process for all above examples. This container is used by the PODM application to configure SSL/TLS Connection.

```
/var/lib/pod-manager/client.jks
```

2. Please note the escape sequence used to escape "white space" and existing CA certificate path.

4.6.2.2 PSME/RMM/RSS Server Certificate Creation Examples

**Example 1:** Create a new server certificate and sign it with a newly created CA certificate:

```
sudo pod-manager-certificate-creation.sh --certificate-type "server" --ca-certificate-dname "/C=PL/ST=State/L=locality/O=organization/CN=fqdn_or_ip" --certificate-dname "/C=PL,ST=State,L=locality,O=organization/CN=fqdn_or_ip"
```

This command creates the server certificate, key and server keystore file under /var/lib/pod-manager and CA certificate under /var/lib/pod-manager/ca.

**Example 2:** Create a new server certificate and sign it with an existing CA certificate:

```
sudo pod-manager-certificate-creation.sh --certificate-type "server" --ca-certificate-file /path/to/ca/file/ca.pfx --certificate-dname "/C=PL/ST=State/L=locality/O=organization/CN=fqdn_or_ip"
```

This command creates the server certificate, key and server keystore file under /var/lib/pod-manager and CA certificate under /var/lib/pod-manager/ca.
This command creates a new server certificate, key, and server keystore file under `/var/lib/pod-manager`. Whereas, when extracted, the CA certificate and private key will be copied to `/var/lib/pod-manager/ca` directory.

**Example 3:** Create a new server certificate and sign it with the provided CA certificate - using interactive mode.

```
sudo /usr/bin/pod-manager-certificate-creation.sh --certificate-type "server" --ca-certificate-file /path/to/ca/file/ca.pfx"
```

During this process, the user will be asked to pass the client certificate attributes manually. Follow the instructions displayed on screen to complete the process. Upon completion, this command creates a new server certificate, key, and server keystore file under `/var/lib/pod-manager` directory and extracted CA certificate and private key are copied to `/var/lib/pod-manager/ca`.

**NOTE:**

1. DO NOT REMOVE keystore files created after certificate generation process for all above examples. Those containers are used by the PODM application to configure SSL/TLS Connection.

   ```
   /var/lib/pod-manager/client.jks
   /var/lib/pod-manager/server.jks
   ```

2. The escape sequence used to escape "white space" and existing CA certificate path.

3. When using TLS configuration on PSMEs, authentication from the PODM to PSMEs is certificate-based and PSMEs rest servers will not communicate with clients that do not perform certificate-based authentication. To access the PSMEs APIs directly, using a web browser or any command line REST API client (for example cURL), it is necessary to export the client certificate from PODM for use with other HTTP clients.

These steps describe this process for cURL client:

1. Generate a PKCS #12 key based on the root key:

   ```
   ```

2. Create .pem certificate file using the output of the above command:

   ```
   openssl pkcs12 -clcerts -nodes -in client.p12 -out client.pem
   ```

   Now this .pem file can be used by cURL, or imported into a web browser such as Firefox* or Chrome*.

This is the example usage with cURL:

```
```

The following steps describe this process for wget client:

```
wget --no-check-certificate --certificate=/root/client.pem -qO- https://<psme_ip>8443/redfish/v1
```

### 4.6.3 Importing Certificates

The script to import signed client or server certificate to container is at:

```
/usr/bin/pod-manager-certificate-import.sh
```

Ensure the following script is executed with root privileges:

```
sudo /usr/bin/pod-manager-certificate-import.sh
```

Print out the help menu by using the script:

```
sudo /usr/bin/pod-manager-certificate-import.sh --help
```

**Note:** It is assumed that the signed certificate container used is of type PKCS #12 and contains a full certificate chain.
4.6.3.1 Client Certificate Import
This is an example of the script used to import client certificates:

```
sudo pod-manager-certificate-import.sh -t "client" -c /path/to/ca/file/ca.pfx
```

This command creates a container of type .JKS with imported client certificate signed by the CA certificate, located at:

```
/var/lib/pod-manager/client.jks
```

4.6.3.2 Server CA Certificate Import
This is an example of the server CA certification import script:

```
sudo pod-manager-certificate-import.sh -t "server" -c /path/to/ca/file/ca.crt
```

This command creates a container of type .JKS with imported CA certificate used to sign server certificate at:

```
/var/lib/pod-manager/server.jks
```

4.6.3.3 Import Server CA Certificate to the Existing Keystore
This is an example of the script used to import the server CA certification to the existing Keystore:

```
sudo pod-manager-certificate-import.sh -t "server" -c /path/to/ca/file/ca.crt -i -a "alias"
```

This command inserts CA certificate under specified alias to the existing JSK container at:

```
/var/lib/pod-manager/server.jks
```

4.6.4 Distributing CA Certificates
This section provides information pertaining to the distribution of CA certificates.

4.6.4.1 CA Certificate Client
The CA certificate must be propagated to all external services (RMM, PSME, and Storage Service) that will authenticate the PODM.

Which CA certificate to use:
- The default CA certificate provided with PODM packages is located at:
  
  ```
  /var/lib/pod-manager/root.crt
  ```
- If the certificate was created using the "pod-manager-certificate-creation.sh" script, then the certificate is located at:
  
  ```
  /var/lib/pod-manager/ca/root.crt
  ```

4.6.4.2 CA Certificate Server
The CA certificate, server certificate was signed by, must be imported to the PODM's trust store located at:

```
/var/lib/pod-manager/server.jks
```

Which CA certificate to use:
- The default container that contain default CA certificate provided with PODM packages is located at:
  
  ```
  /var/lib/pod-manager/server.jks
  ```
- The default CA certificate provided with PODM packages is located at:
The default CA certificate key provided with PODM packages is located at:

```
/var/lib/pod-manager/server.key
```

- If the certificate was created using the "pod-manager-certificate-creation.sh" script, then the certificate is located at:

```
/var/lib/pod-manager/ca/root.crt
```

### 4.6.5 Supplying RMM and PSME with Certificates

This section refers to supplying the RMM and PSME with Certificates.

#### 4.6.5.1 CA Certificate

The selected CA certificate must be copied to RMM. RMM expects file named "podm.cert". The user must rename the CA certificate file and copy it to this location:

```
/etc/rmm/podm.cert
```

The RMM will then propagate this certificate to all PSME's in the rack it manages.

**Note:** In a case where the RMM is not used, the certificate MUST be copied manually to a specific location on each PSME to this location:

```
/etc/psme/certs/ca.crt
```

The PSME configuration file "/etc/psme/psme-rest-server-configuration.json" MUST be updated NOT to expect a certificate from RMM as shown below:

```
"rmm-present": false,
```

#### 4.6.5.2 Server Certificate and Key

The selected certificate and key file must be copied to the PSME/RMM. The PSME/RMM expects file named "server.key" for key file and "server.crt" for certificate file. The user must rename the CA certificate file and copy it to this location:

```
/etc/psme/certs/server.crt
/etc/psme/certs/server.key
```

### 4.6.6 Supplying Storage Service with Certificates

Refer to these sections for information regarding the supplying of the storage service with certificates.

#### 4.6.6.1 CA Certificate

The selected CA certificate must be copied to Storage Service. Storage Service expects the file to be named "ca.crt". Rename the CA certificate file and copy it to:

```
/etc/psme/certs/ca.crt
```

For Storage Service PSME rest server configuration file "/etc/psme/psme-rest-server-configuration.json" MUST be updated NOT to expect certificate from RMM as shown below:

```
"rmm-present": false,
```
4.6.6.2 Server Certificate and Key

The selected certificate and key file must be copied to RSS. RSS expects file named "server.key" for key file and "server.crt" for certificate file. The user must rename the CA certificate file and copy it to following location:

/etc/psme/certs/server.crt
/etc/psme/certs/server.key

4.7 Using Custom VAULT Configuration

The VAULT is used to read the keystore password:

- by the PODM while reading the certificate from keystore files
- by script to generate/import certificates to set up keystore password

To get default keystore password on PODM with root level privilege:

```
java -jar /usr/lib/pod-manager/vault-decrypter.jar --password-type keystore
```

The VAULT container can be changed. For more information refer:


Once the user changes the VAULT container entry values (KEYSTORE_URL, KEYSTORE_PASSWORD, KEYSTORE_ALIAS, SALT, ITERATION_COUNT, ENC_FILE_DIR) corresponding updates MUST also be made in the these two files:

- The WildFly configuration file "standalone.xml" is at:
  /opt/pod-manager/wildfly/standalone/configuration/standalone.xml
- The scripts configuration file "vault.json" used to generate/import certificates is at:
  /var/lib/pod-manager/vault/vault.json
5.0  Deployment

The PODM is Java EE application that uses the WildFly Application Server and PostgreSQL as persistent storage engine.

5.1  Starting the PostgreSQL Service

To start the PostgreSQL service, use this command:

```bash
sudo service postgresql start
```

To confirm that the DB is running use this command:

```bash
sudo service postgresql status
```

If it is running, this command should return the following message:

```
9.5/main (port <PORT>): online
```

Default login credentials for PostgreSQL are stored securely in WildFly's VAULT and are set to:

- User: administrator
- Password: podm

5.2  Starting the PODM Service

Run these commands to start the PODM service:

```bash
sudo service pod-manager start
sudo service pod-manager status
```

If running, this command should return following message:

```
* Wildfly Application Server is running with PID <PID>
```

If the application is not available, check for deployment error logs in the following locations:

```
/var/log/wildfly/console.log
/opt/pod-manager/wildfly/standalone/log/server.log
```

5.3  Accessing the PODM REST Application Programming Interface

The PODM service can be accessed at this address:

```
https://<target_machine_IP>:8443/redfish/v1
```

**Note:** Intel recommends using JSON formatting plugins for web browsers (for example, JSONView*, JSON Formatter, and so forth).

Requests should contain HTTP Basic authentication headers.

Refer to Table 2, Intel® Rack Scale Design PODM API Specification document for detailed description of resources.
6.1 Composed Node Lifecycle Management

This section provides the information about composed node lifecycle management.

6.1.1 Composed Node by Using the JSON Template

To create a composed node by using the PODM REST API, create a JSON template describing requested resources. It must be supplied to the PODM by performing an HTTP POST request on the composed node Collection Action Uniform Resource Identifier (URI) is located at /redfish/v1/Nodes/Actions/Allocate on the PODM service.

The JSON template may contain various details of resources to be used in the composed node. All JSON template elements are optional, but each requirement should be coherent itself. It is possible to supply the PODM with a JSON template containing no specific requirements (for example, "{}" - a pair of empty curly braces in HTTP request body) thus allowing the PODM to compose a node containing resources chosen arbitrarily by the PODM.

6.1.2 Specifying Requirements for a Composed Node

The JSON template contains requirements for a single composed node.

Basic customization covers setting a "Name" and "Description" string of such System (both being of type String). As the "Name" parameter is required by Redfish for all resources, if it's not provided then the PODM will use the default name.

The example below will allocate a single composed node with requested name and description:

```json
{
   "Name": "Customized composed node name",
   "Description": "Description of a customized composed node"
}
```

JSON template may contain requirements for:

- Processors
- Memory
- Remote Drives
- Local Drives
- Ethernet interfaces

To specify requirements for those resources, a proper section must appear in the JSON template.

6.1.3 General Assumptions for Allocation

Requirements are treated as a minimal required value, so the resulting composed node might have better parameters than requested.

Composed node customization and resource customization sections described below can be used jointly.

Each resource type description has an associated table which contains details about specific requirements:

- Key is the JSON object field.
• JSON type contains data type as defined by json.org (https://www.json.org/),

Allowed values contains additional restrictions to JSON type or hints (for example, for enumerations or Boolean values)
• Nullable indicates if null value can be passed for a specified key.
• Notes, limitations provides additional hints about the specific requirement.

6.1.3.1 Location Requirements

Processor, memory, local drive, and Ethernet interface sections might contain resource and chassis objects.

The resource must contain the PODM URI (presented as "@odata.id") of the discovered resource (Processor's URI in the Processor section, URI to Memory resource in Memory section, and so on).

Chassis must contain the PODM URI of the discovered chassis in which applicable resources will be looked for.

Specifying the requirement of resources (like ex-processor, memory, local drive, Ethernet interface, and so forth) can refer to the PODM API Specification Appendix for details.

6.1.4 Allocation Algorithm

Node composition starts with the HTTP POST request of the JSON template on /redfish/v1/Nodes/Actions/Allocate composed node Collection Action URI on the PODM service. If the JSON template is well-formed and contains a supported set of requirements, the allocation process starts.

There are four major scenarios that are currently supported:
• Allocating resources for the composed node to be booted from the local drive
• Allocating resources for the composed node to be booted from the existing remote drive
• Allocating resources for the composed node to be booted from the remote drive that need to be created
• Allocating resources for the composed node with VLAN requirements specified. This scenario is used with one of the other three.

The allocation process is preceded by a general verification of the JSON template that checks if the requested node can be realized by available resources and consists of:
• Selecting and allocating a computer system that contains resources matching template requirements for processors, memory, local drives, and Ethernet interfaces
• Selecting or creating a remote drive to be used with a previously selected computer system and allocating it

6.1.4.1 Detailed Process of Selecting and Allocating a Computer System for a Composed Node

This is the process:
• Find all computer systems that are not yet allocated (not used by any other allocated composed node) with Status Enabled and Health OK.
• Filter computer systems by specified Resource and Chassis (if supplied in template)
• Filter computer systems by Processors: return all computer systems that contain at least a requested quantity of Processors that meet requirements (if supplied in template):
  – Exactly matching requested model
  – Exactly matching requested brand
  – With at least requested number of cores
  – With at least requested frequency
Features

- Exactly matching requested instruction set
- Filter computer systems by Memory: return all computer systems with at least the total requested size of memory located on the Memory Modules that meet requirements (if supplied in template):
  - Memory of exactly requested DIMM device type
  - With at least requested speed MHz
  - With exact requested manufacturer
  - With at least requested data width bits
- Filter computer systems by Local Drives: return all computer systems that contain for each requested Drive one distinct Device meeting requirements (if supplied in template):
  - With at least requested capacity specified
  - Exactly matching requested Drive type
  - With at least requested min RPM
  - With exact requested serial number
  - With exact Interface
- Filter computer systems by Ethernet Interfaces: return all computer systems that contain for each requested Ethernet Interface one distinct Ethernet Interface meeting requirements (if supplied in template):
  - With at least requested speed
  - If a VLANs section is provided, then computer systems with Ethernet Interfaces which are not connected with EthernetSwitchPorts are filtered out (as described below)
- A first Computer System from resulting filtered collection is then allocated to be used in composed node.

6.1.4.2 Connection between Computer System's Ethernet Interface and Ethernet Switch Port

To enable particular VLAN usage on composed node, there is a need to map the Ethernet switch port and computer system's Ethernet interface. This mapping is done using a MAC address as an identifier. Fields used for this mapping are:

- NeighborMAC on EthernetSwitchPort resource
- MacAddress on EthernetInterface resource

If those two properties contain the same value, the computer system's Ethernet interface and Ethernet switch port are treated as connected.

Only computer systems with Ethernet interfaces, which are connected to Ethernet switch ports, could be used in allocation with a specified VLANs requirement.

6.1.4.3 Detailed Process of Selecting a Remote Drives

This is the process:

- Determine what type of Remote Drive is requested
- When requesting an existing Remote Drive:
  - Find all Targets that are not yet allocated (not used by any allocated composed node),
  - First find the Target that exactly matches the requested iSCSI Qualified Name (IQN) and allocate it to be used in the composed node.
- When requesting a new Remote Drive:
  - Check if the Target does not exist with requested IQN to be set for newly created target,
  - Check if the Logical Drive requested as a Master Drive exists on Storage Service handled by the PODM, and select this Storage Service to handle a new Target creation.
  - Find all Logical Volume Groups meeting requirements:
– Located on selected Storage Service
– Having free space of at least requested capacity for a new Remote Drive
– A first Logical Volume Group from resulting filtered collection is selected as a placement for a new Logical Volume, which will be exposed as a new Target (Remote Drive)
– A new Logical Volume is created on selected Logical Volume Group (as a snapshot or as a clone)
– A new Target is created on top of a newly created Logical Volume.
– Newly created Target is allocated to be used in composed node.

6.1.4.4 Post-allocation Scenarios

A composed node is created as a new REST resource at /redfish/v1/Nodes/{NodeId} when a proper computer system is found and is successfully allocated.

The state of the composed node is set to Allocated.

An allocated composed node is a PODM proposition that can be either accepted or rejected.

- If accepted, the user has to send an HTTP POST request on the ComposedNode.Assemble action of the proposed composed node to assemble it:
  - If no remote drive was requested, a composed node's state is set to PoweredOff.
  - When remote drive is requested, a composed node remains "Assembling" until the target creation finishes. When the target is successfully assembled to be used with the composed node, the node's state is set to PoweredOff.
  - The assembly process doesn't end with sending power on request, so it is necessary to perform a ComposedNode.Reset action to power on a composed node after assembly.
- If rejected, the user can continue sending HTTP POST requests of JSON template on /redfish/v1/Nodes/Actions/Allocate to create more proposals to pick from. When finding the right proposal, it is recommended to send HTTP DELETE on all rejected proposals of the composed nodes to free the resources allocated by them.

6.1.4.5 Disassembly

Upon disassembly of a composed node several actions are performed:

- Graceful shutdown request is sent to computer system.
- All VLANs (except for reserved ones; refer to Reserved VLANs) are removed from associated Ethernet switch ports associated with computer system's Ethernet interfaces.
- The computer system is deallocated.
- The remote target is deallocated (when used in composition).

6.2 NVMe over Fabrics (NVMe-oF)

This feature enables attaching/detaching the NVMe volumes to/from remote hosts over an Ethernet network using Remote Direct Memory Access (RDMA)-capable Network Interface Cards (NICs).

A full NVMe-oF solution consists of:

- One or more hosts providing volumes (the "targets")
- One or more client hosts (the "initiators")
- A single discovery service, which responds to queries from the initiators about volumes which are available to them for attachment.
The PODM provides the API to attach/detach volumes to/from the composed node, which helps to manage targets and discovery service.

### 6.2.1 Recommended Quality of Service (QoS) Settings for NVMe over Fabrics

NVMe requires an appropriate level of network resources to be allocated to minimize latency and maximize bandwidth for NVMe traffic. To ensure that level of resources, Quality of Service (QoS) should be configured on all switch interfaces with NVMe traffic. QoS can be configured through the PODM or PSME APIs. For more details, refer to the API Specifications and Intel® Rack Scale Design (Intel® RSD) Pooled System Management Engine (PSME) User Guide Software v2.3. Refer to Table 2.

### 6.3 Using PODM with RMM Service

The PODM requires that RMM software be installed on the external machine and meet the following requirements:

- Host should be part of VLAN 4094
- Hostname must start with a string "rmm"
- The PODM certificate is present in /etc/rmm/podm.cert; for more information about security, refer to Section, 4.6.5, Supplying RMM and PSME with Certificates.

#### 6.3.1 Rack Lifecycle Policy (Chassis of Type Rack)

This section provides the information about the rack lifecycle policy (chassis of type rack).

##### 6.3.1.1 Creating a New Rack Resource

The following rules are used for creating a new Rack resource:

- When a new Drawer (Chassis of type Drawer) is discovered
  - If discovered Drawer is reporting "null" as "ParentLocationId" Drawer is attached to Pod (Chassis of type Pod)
  - If Rack with the rack location that this Drawer is reporting (under "ParentLocationId") does not exist, a new Rack resource is created in PODM REST API and this Drawer is attached to it
  - If Rack with the rack location that this Drawer is reporting (under "ParentLocationId") does exist, Drawer is attached to this Rack resource
- When an already discovered Drawer has been rediscovered (for example, after slow-poll refresh)
  - If Rack with the rack location that this drawer is reporting does not exist, a new Rack is created in the PODM REST API and this Drawer is attached to it
  - If Rack with the rack location that this Drawer is reporting (under "ParentLocationId") does exist, Drawer is attached to this Rack resource
- When a new RMM service is discovered:
  - New Rack resource is created in PODM REST API and RMM attributes are connected to it. All racks with the same "LocationId" but without the RMM attached, are removed and the Drawers directly linked to those racks are moved to the newly created rack resource.
- When already discovered RMM has been rediscovered (for example, after slow-poll refresh)
  - If the RMM’s Rack changes its location ID (in the RMM: “RackPuid” field), PODMs Rack resource’s parameters "RackPuid" and "LocationId" are being updated to the new values. All drawers contained by this Rack are moved under Rack identified by their respective parent location ID. When there is no single Rack with an old location ID value, the new Rack is being created on the PODM. Drawers can have their “ParentLocationId” updated via RMM <-> PSME <-> PODM events notification channel or slow-poll refresh.
6.3.1.2 Rack Resource Removal

The rack resource will be deleted under these conditions:

- When a removed drawer is the last one that was attached to the rack, and the rack does not have RMM service associated with it.
- When RMM service has disappeared (for example, a machine with RMM was turned off), and rack does not contain any Drawers.

6.4 Deep Discovery

In general, PSME is responsible for exposing information about computer systems. Currently, Integrated Baseboard Management Controller (BMC) does not provide all required information. As a workaround, a tiny Linux* utility image (LUI) can be used to boot the system and perform deep discovery of resources. This image includes a PSME compliant service that PODM can read from. The PODM reads the basic BMC collected data from the PSME, then boots the LUI on each computer system. The data visible in the PSME APIs will only be the BMC collected data, whereas the PODM merges both the basic BMC data from the PSME and the additional data discovered from the LUI environment.

6.4.1 General Requirements, Assumptions, and Limitations

Refer to these points:

- PODM and PSME have DiscoveryState properties
- During the deep discovery process, the computer system cannot be used for any other processes (for example - allocation)
- Networks/VLANs used:
  - iPXE/Deep discovery: Storage Access Network
  - LUI -> PODM: Separate dedicated VLAN
- To prevent power and bandwidth spikes, the deep discovery process is staggered. How staggering is performed is configurable.
- Resource properties should be obtained either from the PSME or deep discovery, but not from both.

Note: It is necessary to avoid overwriting data during the slow poll process. Expected data sources for specific properties are currently hardcoded, but should be configurable in future. If customization is required, source code has to be changed and the application rebuilt.

6.4.2 Building the Linux* Utility Image

Refer to Table 2, Intel® Rack Scale Design PSME User Guide to get information regarding prerequisites.

6.4.2.1 Preparing the rootfs Directory

Refer to Table 2, Intel® Rack Scale Design PSME User Guide to get information on preparing the rootfs directory.

6.4.2.2 Building the Linux Utility Image

Refer to Table 2, Intel® Rack Scale Design PSME User Guide to get information on building LUI.
6.4.3 Configuring Deep Discovery

To enable Deep Discovery, the PODM requires the LUI. It should be provided at:

```
/opt/pod-manager/wildfly/discovery/bzImage
```

Refer to Table 2, Intel® Rack Scale Design PSME User Guide for the LUI building process.

General configuration of discovery is located at `/etc/pod-manager/discovery.json`:

```json
{
    "MaxComputerSystemsCountPerDrawerBeingDeepDiscovered": 1,
    "DeepDiscoveryEnabled": true,
    "DiscoveryIntervalSeconds": 600
}
```

6.4.3.1 Deep Discovery Configuration Notes

- It is possible to enable or disable deep discovery. If disabled, all data would be read from PSME.
- It is possible to configure the number of computer systems per drawer that could be deep discovered at the same time. It could be set to a lower value to prevent overcurrent and power spikes. Setting this property to a higher value will yield an overall shorter time needed to perform the deep discovery on all computer systems.
- It is possible to trigger a deep discovery process manually; manual triggering is queued along with automatic triggering. `MaxComputerSystemsCountPerDrawerBeingDeepDiscovered` configuration property defines the threshold value per Drawer for both manual and automatic process triggering.

6.5 InBand Service Support

The PODM supports InBand /Out-of Band (OOB) data aggregation mechanism for a subset of Redfish resources that can have a representation originating from either the PSME (OOB) or InBand Service or both.

Aggregation occur on the computer system and its following sub-resources:

- Processor
- Memory
- Ethernet interface
- Simple storage/simple storage device
- Storage/storage controller
- Chassis/drive

Resources related to the particular computer system that gets aggregated.

6.5.1 How PODM Detects InBand Service

*Note:* Refer to the README.md file of the InBand service example code bundled with the PODM source code to deploy the user's own InBand service instance.

It is assumed that a single InBand service instance will handle a set of computer systems that use the same agent to provide data via InBand channel (one InBand Service should be deployed per agent type).

The PODM will detect the InBand service when provided with a static endpoint configuration by way of the `/tmp/services.list` file containing a line with InBand service endpoint:

```
https://127.0.0.1:8448/redfish/v1/ InBand
```
The PODM will subscribe to events from this service and will discover its resources. The resources will get updated independently from other services.

### 6.5.2 Enabling Aggregation of InBand Service Resources

The aggregation feature is disabled by default and can be enabled by way of the configuration file

```
/etc/pod-manager/InBand-service-config.json
```

by setting its content to:

```
{
  "InBandServiceSupportEnabled": true
}
```

### 6.5.3 How Resources Are Aggregated

InBand service resources are discovered independently and are stored separately from their OOB counterparts.

The aggregation is performed only when a primary computer system (discovered by way of OOB) has a matching representation in InBand resources (based on the computer system's UUID property).

**Note:** Aggregation is performed on demand, upon GET request on resources from the PODM northbound API.

#### 6.5.3.1 Properties Aggregation

The aggregation is done by getting the values from OOB representation and then overlaying it with property values from InBand representations.

Simple properties are merged using following aggregate condition:

- If the existing value is null or an empty collection, use overlay the value
- If the overlay value is not null, then use the overlay value (InBand value preferred over OOB)
- In other case use existing value

**Note:** When using any InBand service implementation make sure that it exposes full and meaningful representation of resources to avoid unintended override of properties obtained by way of OOB (this includes especially complex types and inline expanded collections).

#### 6.5.3.2 Original Equipment Manufacturer Data Aggregation

The PODM enables merging of the OEM namespaces that are specific for a given InBand agent (and different from Intel BackScale).

Consider two computer systems that got matched by UUID and have some additional data available in the custom OEM space on InBand service:

This is the OEM from OOB (PSME):

```
{
  "Oem": {
    "Intel_RackScale": {
      "ProcessorSockets": 2
    }
  }
}
```

This is the OEM from the InBand service:

```
{
  "Oem": {
    "Intel_RackScale": {
      "ProcessorSockets": 2
    }
  }
}
```
6.5.4 Sub-resource Collections Aggregation

Apart from properties and OEM aggregation, the linked sub-resources collections are aggregated using their unique identifiers. Various sub-resources are uniquely identified using different property values in Table 4:

<table>
<thead>
<tr>
<th>Resource</th>
<th>Unique Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer system</td>
<td>UUID</td>
</tr>
<tr>
<td>Processor</td>
<td>Socket</td>
</tr>
<tr>
<td>Memory</td>
<td>Device locator</td>
</tr>
<tr>
<td>Ethernet interface</td>
<td>MAC address</td>
</tr>
<tr>
<td>Simple storage</td>
<td>UEFI device path</td>
</tr>
<tr>
<td>Simple storage device</td>
<td>Name</td>
</tr>
<tr>
<td>Storage</td>
<td>Assumed always one per computer system</td>
</tr>
<tr>
<td>Storage controller</td>
<td>Durable name (when in UUID format), source URI otherwise</td>
</tr>
<tr>
<td>Chassis</td>
<td>Using matching computer system's contained-by-property to gain access to its drives</td>
</tr>
<tr>
<td>Drive</td>
<td>Last path segment of source URI</td>
</tr>
</tbody>
</table>

Based on these values PODM is able to determine the contents of sub-resource collections (for example, EthernetInterfaces) and include:

- resources present only in OOB (no matching InBand representation using MACAddress property)
- resources present both in OOB and InBand (matched by MACAddress, resource gets merged)
- resources present only in InBand Service (no matching OOB representation using MACAddress property)

6.6 Link Aggregation Group

The Link Aggregation Group (LAG) is a technique used in a high-speed-backbone network to enable the fast and inexpensive transmission of bulk data.

The best feature of link aggregation is its ability to enhance or increase network capacity while maintaining a fast transmission speed and not changing any hardware devices, thereby reducing cost.
Creating the Link Aggregation Group

A LAG can be created by combining at least one physical upstream port resulting in creating an additional virtual port. To create a new LAG using PODM REST API, it is necessary to create and supply a proper JSON template to PODM by performing an HTTP POST request on the Ethernet Switch Port Collection resource located at:

/redfish/v1/EthernetSwitches/{switchID}/Ports/{portID}

Sample JSON template:

```
{
    "PortId" : "LagPort",
    "PortMode" : "LinkAggregationStatic",
    "Links" : {
        "PortMembers" : [
            { "@odata.id" : "/redfish/v1/EthernetSwitches/1/Ports/10" },
            { "@odata.id" : "/redfish/v1/EthernetSwitches/1/Ports/11" }
        ]
    }
}
```

Refer to Table 5 for template key values and notes.

**Table 5. Key Values to Create Link Aggregation Group**

<table>
<thead>
<tr>
<th>Key</th>
<th>JSON Type</th>
<th>Allowed Values</th>
<th>Nullable</th>
<th>Notes, Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>PortId</td>
<td>String</td>
<td></td>
<td>Yes</td>
<td>Switch port unique identifier. CAUTION! The maximum value must not exceed 16 characters and must not contain whitespace.</td>
</tr>
<tr>
<td>PortMode</td>
<td>String</td>
<td>&quot;LinkAggregationStatic&quot;, &quot;LinkAggregationDynamic&quot;</td>
<td>Yes</td>
<td>Port working mode. Currently only LinkAggregationStatic mode is supported.</td>
</tr>
<tr>
<td>PortMembers</td>
<td>Array[Link]</td>
<td></td>
<td>Yes</td>
<td>Array of ports being member of LAG. Must be placed in Links object. There must be at least one port. All ports contained in this array must have:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- &quot;PortClass&quot;: &quot;Physical&quot;</td>
<td></td>
<td>- &quot;PortType&quot;: &quot;Upstream&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- The same speed.</td>
<td></td>
<td>None of these ports can be a member of another LAG.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An empty array &quot;[]&quot; will be interpreted as absence of this requirement key.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Response after the LAG has been successfully created should contain the location to the newly created Ethernet switch port:

HTTP/1.1 201 Created

Location: <PROTOCOL>://<IP>:<PORT>/redfish/v1/EthernetSwitches/1/Ports/99
6.6.2 Modifying the Link Aggregation Group

To modify LAG using the PODM REST API, it is necessary to create and supply the proper JSON template to the PODM by performing an HTTP PATCH request on the existing Ethernet Switch Port resource located at:

```
/redfish/v1/EthernetSwitches/{switchID}/Ports/{portID}
```

If Switch Port modification JSON includes Links to PortMembers, then the modified Ethernet Switch Port needs to be a proper LAG in order for this request to be successful:

- **PortClass** must be set to Logical.
- **PortMode** must be set to LinkAggregationStatic.
- **PortMembers** array cannot be empty.

In the following example, port 12 is added as a member to the LAG along with changing additional properties:

Sample JSON template:

```
{
    "AdministrativeState" : "Up",
    "LinkSpeedMbps" : 40000,
    "FrameSize" : 1500,
    "Autosense" : false,
    "Links" : {
        "PrimaryVLAN" : {
            "@odata.id" : "/redfish/v1/EthernetSwitches/1/Ports/99/VLANs/1"
        },
        "PortMembers" : [{
            "@odata.id" : "/redfish/v1/EthernetSwitches/1/Ports/10"
        }, {
            "@odata.id" : "/redfish/v1/EthernetSwitches/1/Ports/11"
        }, {
            "@odata.id" : "/redfish/v1/EthernetSwitches/1/Ports/12"
        }]
    }
}
```

Refer to **Table 6** for key values used to modify LAG.

### Table 6. Key Values to Modify Link Aggregation Group

<table>
<thead>
<tr>
<th>Key</th>
<th>JSON Type</th>
<th>Allowed Values</th>
<th>Nullable</th>
<th>Notes, Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdministrativeState</td>
<td>String</td>
<td>&quot;Up&quot;, &quot;Down&quot;</td>
<td>No</td>
<td>Port link state forced by user</td>
</tr>
<tr>
<td>LinkSpeedMbps</td>
<td>Number</td>
<td>Nonnegative number</td>
<td>No</td>
<td>Port speed</td>
</tr>
<tr>
<td>FrameSize</td>
<td>Number</td>
<td>Nonnegative number</td>
<td>No</td>
<td>MAC frame size in bytes</td>
</tr>
<tr>
<td>Autosense</td>
<td>Boolean</td>
<td>true, false</td>
<td>No</td>
<td>Indicates if the speed and duplex are automatically configured by the NIC</td>
</tr>
<tr>
<td>PrimaryVLAN</td>
<td>Link</td>
<td></td>
<td>No</td>
<td>Link to VLAN available on this port (only these VLANs are acceptable)</td>
</tr>
<tr>
<td>PortMembers</td>
<td>Array[Link]</td>
<td></td>
<td>No</td>
<td>Array of ports being member of LAG. Must be placed in Links object. All ports contained in this array must have: - &quot;PortClass&quot;: &quot;Physical&quot; - &quot;PortType&quot;: &quot;Upstream&quot; - the same speed None of these ports can be a member of another LAG.</td>
</tr>
</tbody>
</table>
All of the properties in Table 6 are optional. If not provided, they are not changed.

Response after success LAG modification:
HTTP/1.1 204 No Content

6.6.3 Removing the Link Aggregation Group

To remove LAG using PODM REST API, it is necessary to perform an HTTP DELETE request on the existing Ethernet Switch Port resource located at:

/redfish/v1/EthernetSwitches/{switchID}/Ports/{portID}

This is the response after successful LAG port removal:
HTTP/1.1 204 No Content

6.6.4 Limitations

These are the limitations for LAG:

- LAGs can be created only from non-LAG ports
- LAGs can be created only on upstream physical ports that have matching speeds
- Creating a LAG on selected ports removes those ports' VLAN memberships

6.7 PODM Northbound Interface Eventing

Since Intel® RSD v2.1.3, the PODM provides eventing feature enabled on its northbound interface, (implementation has been based on eventing model defined by Redfish).

Events generated by PODM can be utilized by user for getting knowledge about changes of the PODM data model.

6.7.1 Supported Events

Currently PODM is able to generate following kind of events:

- **ResourceAdded**: event generated when new Redfish resource will be detected by the PODM
- **ResourceUpdated**: event generated when PODM will detect that already known Redfish resource has been updated
- **ResourceRemoved**: event generated when PODM will detect that already known Redfish resource has been deleted
- **StatusChange**: event generated when the PODM will detect that state of already known Redfish resource has been changed
- **Alert**: event generated when the PODM will detect that Status of already known Redfish resource enters Critical state
6.7.2 Event Subscription

User can control subscriptions life cycle using following HTTP operations:

- POST on /redfish/v1/EventService/Subscriptions (subscription creation)
- DELETE on /redfish/v1/EventService/Subscriptions/{subscriptionID} (subscription deletion)

Be aware that the user's subscription might be unregistered automatically in any case when the PODM cannot reach subscription endpoint after repeated retries.

Consider this scenario: the user is registering new subscription (provided destination is http://inactive-listener
because EventSubscription is a Redfish resource the PODM will generate ResourceAdded event.

The PODM will try to deliver a generated event to the http://inactive-listener because the http://inactive-listener is not active and the PODM will try to redeliver event. The number of redelivery retries is limited to the value described by DeliveryRetryAttempts.

When the threshold defined by DeliveryRetryAttempts will be reached, the subscription related with http://inactive-listener will be automatically unregistered.

The #repeated retries value has been described as #repeated retries attribute and it is stored in the /etc/podm/event.json file.

For more information about configuring "DeliveryRetryAttempts" and other related attributes, refer to Section 3.1.2, Customization.

6.7.3 Registering New Subscription

Sample subscription creation request can look like:

```json
{
    "Name" : "My Subscription",
    "Destination": "http://eventlistener.com:8000",
    "EventTypes": [
    ],
    "Context": "This is my event subscription",
    "Protocol": "Redfish",
    "OriginResources" : []
}
```

Where Name is name of the user's subscription:

- Destination is reference to the system which is listening for events generated by the PODM.
- EventTypes are events types observed by particular subscriber. Any other events generated on the PODM side will not be delivered to this subscriber.
- Context is a client-supplied string that is stored with the event destination subscription.
- Protocol is the only one value currently supported is Redfish.
- OriginResources is a list of resources for which the service will send events specified in EventTypes array. Empty array or NULL is interpreted as subscription for all resources and assets in subsystem.

6.8 Booting with Internet Small Computer Systems Interface Out-of-band Feature

OOB Internet Small Computer Systems Interface (iSCSI) Boot is a way to boot the node from an iSCSI Target using BMC/BIOS.
### 6.8.1 Booting from iSCSI

To boot from OOB iSCSI the user needs to have the PSME and PODM at least in Intel® RSD v2.1.3. The only requirement is the Network Device Function on the REST API is available on the Computer System and the "BootSourceOverrideMode" property for this System is set to Unified Extensible Firmware Interface (UEFI).

There are two ways to boot from the remote target:

- **Automatically during node composition** (when requesting remote drive in allocation JSON). Additional Network Device Function will be patched with proper iSCSI data by the PODM.
- **Manually** (for already assembled node) when no remote drive was requested in allocation request. User has to patch Computer System and Network Device Function related to Node with iSCSI data. To avoid overwriting iSCSI data by the PODM it has to be done after `#ComposedNode.Assemble` and before `#ComposedNode.Reset` action.

### 6.8.2 Setting Chap Data

To boot using Chap data, it is needed to PATCH the Remote Target with "ChapUser" and "Secret" property. The same data has to be sent to the Network Device Function before the `#ComposedNode.Reset` action through the PATCH request.

**Note:** Secret and `MutualSecret` properties will not be exposed on REST API (will be visible as NULL).

### 6.8.3 Setting Boot Data

Boot data on Computer System will be set by the PODM in the assembly process depending on the allocation request and available data on the PODM REST API.

"BootSourceOverrideTarget" will be different depending on allocation request:

- **When not requesting RemoteTarget**, "BootSourceOverrideTarget" will be set to "Hdd"
- **When requesting RemoteTarget**:
  - When the computer system used to compose the Node has Network Device Function, the property `BootSourceOverrideTarget` will be set to `RemoteDrive` before assembly process
  - When the computer system used to compose the Node does not have network device function, the property `BootSourceOverrideTarget` will be set to `Pxe` before the assembly process

- `BootSourceOverrideEnabled` will be set to Continuous.
- `BootSourceOverrideMode` will not be changed.
- **Be aware that "BootSourceOverrideMode" on Computer System should be set to "UEFI" to boot using this feature. This might require changing this property, by sending PATCH on Computer System, between `#ComposedNode.Assemble` and `#ComposedNode.Reset`. Due to different hardware behavior, it is recommended to check and eventually set this value in every assemble process.**

### 6.8.4 Limitations

These are the limitations:

- **Every manual change on BootSourceOverrideEnabled and BootSourceOverrideTarget on the computer system between `#ComposedNodeCollection.Allocate` and `#ComposedNode.Assemble` will be overwritten by PODM while the `#ComposedNode.Assemble` task works.**
- **The property BootSourceOverrideMode is not changed by the PODM during allocation.**
- The property `BootSourceOverrideMode` will be set to Legacy only while the deep discovery process works.
- The current PODM is able to boot using OOB iSCSI only when the computer system related with the node has a network device function, and this system's `BootSourceOverrideMode` property is set to **UEFI**.
These are relevant appendix materials.

### 7.1 Database Status across PODM Restarts

The PODM retains its asset inventory upon restart. To clear the PODM database upon PODM startup, the user should execute a script that, by default, is installed with the `.deb` package PODM:

```bash
sudo /usr/bin/pod-manager-clean-database-on-next-startup
```

**Note:** The PODM retains asset inventory upon startup. This could lead the PODM to display inaccurate information on its REST API interface.

It is recommended to clear the PODM database to avoid the following:

- **Startup functionality implications:**
  - All assets that were reported in Available state under the REST API before the PODM shut down, are still exposed after a subsequent startup of the PODM.
  - The PODM sets InTest state for all assets available after startup in PODM and originating from RMM/PSME/storage services.
  - The PODM performs rediscovery of RMM/PSME/Storage services that will result in respective state changes of rediscovered assets.

- **Deep Discovery functionality implications:**
  - Assets successfully fetched by way of the LUI service remain unchanged. The user MUST trigger the deep discovery process manually to refresh the Deep Discovery dataset.
  - The PODM assumes that the deep discovery should be triggered on all computer systems in `DeepInProgress` state upon startup.
  - The PODM will not trigger Deep Discovery for computer systems that are in `inTest` state.
  - If Deep Discovery is enabled, then the PODM will trigger deep discovery, forcing asset rediscovery (after respective state change).
  - **Computer systems in** `DiscoveryState:DeepInProgress` **are put into InTest state.**
  - **Computer systems in** `DiscoveryState:DeepFailed` **are left intact.**

- **Composed nodes/allocation functionality implications:**
  - During the allocation process, the PODM does not take into consideration any assets with InTest state.
  - Composed nodes in the Allocating, Allocated, or Assembling state during the PODM startup are presumed Failed.

- **Implication on actions performed on resources:**
  - Actions on assets with InTest state cannot be performed, and yield an HTTP 409 Conflict response from the PODM.

### 7.2 Source Code Compilation (Ubuntu*)

Refer to these sections for information regarding source code compilation for Ubuntu*

#### 7.2.1 Key Components

Refer to [Table 7](#) of key components for source code compilation for Ubuntu.
Table 7. Key Components for Source Code Ubuntu Compilation

<table>
<thead>
<tr>
<th>Component</th>
<th>Name</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>OpenJDK</td>
<td>1.8</td>
</tr>
<tr>
<td>Java EE</td>
<td>Full Profile</td>
<td>7.0</td>
</tr>
<tr>
<td>Libraries</td>
<td>cling</td>
<td>2.0.1</td>
</tr>
<tr>
<td></td>
<td>guava</td>
<td>18.0</td>
</tr>
<tr>
<td></td>
<td>hibernate</td>
<td>5.0.7.Final</td>
</tr>
<tr>
<td></td>
<td>jackson-annotations</td>
<td>2.6.0 (from other jackson libraries), 2.6.3</td>
</tr>
<tr>
<td></td>
<td>jackson-core</td>
<td>2.6.3</td>
</tr>
<tr>
<td></td>
<td>jackson-databind</td>
<td>2.6.3</td>
</tr>
<tr>
<td></td>
<td>jackson-datatype-jsr310</td>
<td>2.6.3</td>
</tr>
<tr>
<td></td>
<td>jackson-jaxrs-base</td>
<td>2.6.3</td>
</tr>
<tr>
<td></td>
<td>jackson-jaxrs-json-provider</td>
<td>2.6.3</td>
</tr>
<tr>
<td></td>
<td>jackson-module-jaxb-annotations</td>
<td>2.6.3</td>
</tr>
<tr>
<td></td>
<td>javassist</td>
<td>3.16.1-GA</td>
</tr>
<tr>
<td></td>
<td>liquibase</td>
<td>3.5.2</td>
</tr>
<tr>
<td></td>
<td>modelmapper</td>
<td>1.1.0</td>
</tr>
<tr>
<td></td>
<td>blueprints-core</td>
<td>2.6.0</td>
</tr>
<tr>
<td></td>
<td>logback-classic</td>
<td>1.1.1</td>
</tr>
<tr>
<td></td>
<td>logback-core</td>
<td>1.1.1</td>
</tr>
<tr>
<td></td>
<td>commons-beanutils</td>
<td>1.8.3</td>
</tr>
<tr>
<td></td>
<td>commons-collections</td>
<td>3.2.2</td>
</tr>
<tr>
<td></td>
<td>commons-digester</td>
<td>1.8.1</td>
</tr>
<tr>
<td></td>
<td>commons-io</td>
<td>2.4</td>
</tr>
<tr>
<td></td>
<td>commons-lang</td>
<td>2.6</td>
</tr>
<tr>
<td></td>
<td>commons-logging</td>
<td>1.2</td>
</tr>
<tr>
<td></td>
<td>commons-validator</td>
<td>1.5.0</td>
</tr>
<tr>
<td></td>
<td>commons-text</td>
<td>1.1.1</td>
</tr>
<tr>
<td>Application server</td>
<td>WildFly</td>
<td>10.0.0.Beta1</td>
</tr>
<tr>
<td>Database</td>
<td>PostgreSQL</td>
<td>9.5</td>
</tr>
</tbody>
</table>

7.2.2 OpenJDK 1.8 Java Development Kit Installation

Before the user begins, ensure that OpenJDK Personal Package Archives (PPA) repository is correctly configured in the user's system. Refer to Section 2.3.1, Adding the PPA Repository for OpenJDK (Ubuntu 14.04.4 Only).

Install OpenJDK v1.8 Java Development Kit package:

```
sudo apt-get install openjdk-8-jdk-headless
```

7.2.3 Verifying OpenJDK Installation

Check default javac version by typing:

```
javac --version
```

If the above command does not show openjdk v1.8, execute this command to set Java defaults:

```
sudo update-alternatives --config javac
```

7.2.4 PostgreSQL 9.5 Installation

Refer to Section 2.4, PostgreSQL 9.5 Installation.
7.2.5 Compilation of the PODM Application

On the development machine, change the directory to the PODM source's root and use gradlew to compile the PODM package:

```
./gradlew buildDeb
```

The above command creates the necessary .deb packages under source root directory which are used to install PODM.

**Note:** If applicable, the user should pass the appropriate proxy information to the gradlew script through the "-DproxyHost= -DproxyPort=" option.

**Caution:** In a rare case, the command can fail with an SSL exception, which is a known issue on Ubuntu with OpenJDK 1.8, arising from the "/etc/ssl/certs/java/cacerts" file NOT being generated properly.

If the scenario described above occurs, execute the following command:

```
sudo /var/lib/dpkg/info/ca-certificates-java.postinst configure
```

Thereafter, re-run the command used for compilation.

To find all packages generated by above command, navigate to the source root directory and execute:

```
find ./ -iname *.deb
```

7.2.6 Configuration of ISC DHCP Server

The recommended ISC DHCP Server configuration is provided by the PODM package, and it is located under this path:

```
/etc/dhcp/dhcpd-pod-manager.inc
```

The default interfaces list on which ISC DHCP Server is listening is available in this file:

```
/etc/default/isc-dhcp-server
```

To change the interfaces list, change this line to match the user configuration:

```
INTERFACES="eth0.4091 eth0.4093 eth0.4094 eth0.4088"
```