Intel® Rack Scale Design POD Manager (PDOM)

User Guide

Software v2.2

December 19, 2017

Revision 001

Contents

Appendices

Tables

Revision History

Revision	Description	Date
001	Initial release.	December 19, 2017

§

1 Introduction

This document contains information about the installation and configuration of the Intel® Rack Scale Design (Intel® RSD) Pod Manager (PDOM) v2.2. This document will be referred to at the Pod Manager (PDOM) throughout this document.

1.1 Intended audience

The intended audiences for this document include:

- Software Vendors (xSVs) 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.
- Hardware Vendors (OxMs) of PSME firmware who would like to provide Intel® RSD PODM API on top of their hardware platform.

1.2 Conventions

The key words/phrases "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119, refer to Table 2.

1.3 Terminology

Table 1. Terminology

Term	Definition
ACL	Access Control List
CA	Certificate Authority
CM	Control Module
HTTP	Hypertext Transfer Protocol
IBL	Intel Business Link
JSON	JavaScript Object Notation
LAG	Link Aggregation Group
LUI	Linux* Utility Imag4e
MMP	Management Midplane
PKCS #12	Personal Information Exchange Syntax Standard
Pod	A physical collection of multiple racks
PODM	Pod Manager
PSME	Pooled System Management Engine
Redfish*	DMTF standard
REST	Representational state transfer
RMM	Rack Management Module
RSA	Public key cryptosystem
RSS	RSD Storage Service
SB	Southbound API
SSL	Secure Socket Layer
TLS	Transport Layer Security
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
URL	Uniform Resource Locator

1.4 Related Documents

Table 2. Related Documents

Doc ID	Title	Location
336811	Intel® Rack Scale Design (RSD) Conformance and Software Reference Kit Getting Started Guide v2.2, Revision 001	*http://www.intel.com/intelRSD*
336814	Intel® Rack Scale Design Pod Manager (PDOM) Release Notes, Software v2.2, Revision 001	
336816	Intel® Rack Scale Design PSME Release Notes, Software v2.2, Revision 001	
336810	Intel® Rack Scale Design PSME User Guide, Software v2.2, Revision 001	
336855	Intel® Rack Scale Design PSME REST API Specification, Software v2.2, Revision 001	
336856	Intel® Rack Scale Design Storage Services API Specification, Software v2.2, Revision 001	
336857	Intel® Rack Scale Design Pod Manager REST API Specification, Software v2.2, Revision 001	
336858	Intel® Rack Scale Design Rack Management Module (RMM) API Specification, Software v2.2, Revision 001	
336859	Intel® Rack Scale Design Generic Assets Management Interface API Specification, Software v2.2, Revision 001	
336860	Intel® Rack Scale Design Firmware Extension Specification, Software v2.2, Revision 001	
336861	Intel® Rack Scale Design Architecture Specification, Software v2.2, Revision 001	
336862	Intel® RSD v2.2 Solid State Drive (SSD) Technical Advisory	
RFC2119	Key words for use in RFCs to Indicate Requirement Levels, March 1997	*https://www.ietf.org/rfc/rfc2119.txt*
SDP0266	Scalable Platforms Management API Specification v1.1.0	*https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.1.0.pdf*
DSP8010	Redfish Schema v2016.3	*https://www.dmtf.org/sites/default/files/standards/documents/DSP8010_2016.3.zip*

1.5 Typographical conventions

Notation used in JSON serialization description:

- Values in italics indicate data types instead of literal values.
- Characters are appended to items to indicate cardinality:
  - "*" (wildcard)
  - "?" (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, "(" and ")", are used to indicate the scope of the operators "?", "*", "+" and "|".
- Ellipses (i.e., "…") indicate points of extensibility.

Note: The lack of an ellipses does not mean no extensibility point exists, rather it is just not explicitly called out.

§

2 Installation

Pod Manager Software can be installed on both Ubuntu* Server 14.04.4 and Ubuntu Server 16.04.1.

2.1 Core Ubuntu* installation

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

Pod Manager Software installation requires access to public software repositories on the Internet. Confirm the server network, firewall, and proxy configurations are configured properly to allow internet access.

2.3 OpenJDK 1.8 Java Runtime environment installation

OpenJDK 1.8 Java Runtime Environment is required by Pod Manager. If OpenJDK 8 Java Runtime Environment is already installed on the system, then refer to Section 2.3.3, Verifying OpenJDK installation to verify installation.

If OpenJDK 8 is not installed, then follow the steps to set up the Java environment correctly. Refer to 2.3.2 Installing OpenJDK Java Runtime environment.

2.3.1 Adding PPA repository for OpenJDK (Ubuntu v14.04.4 only)

Add the OpenJDK 1.8 repository to the `/etc/apt/sources.list` file by adding the following line:

```
deb http://ppa.launchpad.net/openjdk-r/ppa/ubuntu trusty main
```

Download key:

```
sudo su -
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 86F44E2A
```

Update the apt-get repositories:

```
sudo apt-get update
```

2.3.2 Installing OpenJDK Java Runtime environment

Install the OpenJDK 1.8 Java Runtime Environment package:

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

2.3.3 Verifying OpenJDK installation

Check the default java version by typing:

```
java -version
```

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 Pod Manager. If PostgreSQL 9.5 is already installed on the system, then refer Verifying PostgreSQL 9.5 installation section to verify installation.

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

2.4.1 Adding repository for PostgreSQL

Create the file `/etc/apt/sources.list.d/pgdg.list` and add the following 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 -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo
apt-key add -
```

Update the apt-get repositories:

```
sudo apt-get update
```

2.4.2 Installing PostgreSQL 9.5

Install PostgreSQL 9.5 packages:

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

2.4.3 Verifying PostgreSQL 9.5 installation

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

```
pg_lsclusters
```

If it is installed, the above 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

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 tftpd-hpa
sudo apt-get install ntp
sudo apt-get install vlan
sudo apt-get install acl
```

Update the apt-get repositories:

```
sudo apt-get update
```

2.6 Custom iPXE

Custom iPXE is required for full Pod Manager Functionality. It is not provided as part of the Pod Manager `.deb` packages created after compiling Pod Manager Source.

2.6.1 Custom iPXE compilation

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

```
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.

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

2. Copy the following file from the Pod Manager Source package to the `ipxe/src/ directory`.

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

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

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

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

Note: In case of iPXE dependency or compilation issues, refer to:

http://ipxe.org/docs

2.6.2 Custom iPXE installation

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

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

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

Make `symlink` to `podmipxe.0`

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

2.7 Setting up GRUB

Configure the system to use `ethX` naming convention for ethernet interfaces.

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

```
#GRUB_HIDDEN_TIMEOUT
#GRUB_HIDDEN_TIMEOUT_QUIET
```

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

```
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:

```
sudo update-grub
```

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

2.8 NTP configuration

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

```
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 NTP service:

```
sudo service ntp restart
```

2.9 Package signature

If user is planning to distribute PODM package, then the PODM package must be signed first. Once signed, end user has an option to verify (e.g. before installation) that package can be trusted and was not subjected to any modifications.

To sign `.deb` package user should follow steps described in Section 2.9.1 Signing a package. If PODM package has been signed, before installing PODM package user can verify package signature using steps described in Section 2.9.2. Checking package signature to sign `*.deb` packages, use linux* GPG and debsigs:

1. To install debsigs, use:

```
sudo apt-get install debsigs
```

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

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

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

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

4. When prompted, specify the key type, desired key size, length of time the key should be valid, key owner and email address of the owner. For more information, refer to: *https://help.ubuntu.com/community/GnuPrivacyGuardHowto*

Note: Step 4 requires a good source of entropy. Failure to provide sufficient entropy can result in a failure with error "**not enough entropy**".

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

```
gpg --armour --output /tmp/podm.key --export <owner name>
```

2.9.1 Signing a package

The following procedure provides instructions on signing a package.

1. To sign a .deb package use the command below:

```
debsigs --sign=origin -k <key_id> <deb package>
```

2. Once the packages are signed, use the *GNU Privacy Handbook Getting Started Guide* for instructions on how to exchange your GPG key with the recipient, refer to Table 2.

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 and create a keyring. Follow the steps below to create a keyring:

1. Import the key used during package signing:

```
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 check of package signature

To automate installation process of signed package (with its signature verification) there is possibility to use convenient installation script. To use it, run command in `SW` directory created with `srcTar task`. The prerequisite to run the script is `python3*`. Also `dpkg-sig` package must be installed. The supported version of `gpg` is `v1.4.20`.

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

2.10 Installation of Pod Manager using DEB packages

Pod Manager `.deb` packages can be generated manually as described in the Source Code Compilation (Ubuntu) section.

Pod Manager `.deb` packages should be installed in the following order show in Table 3.

Table 3. Order of .deb package installation

Package	Required	Description
`pod-manager`	Yes	Creates user for services, also contains the web server, database, update scripts and configuration for Pod Manager.
`pod-manager-networking`	Yes	Contains configuration for network, DHCP, NTP, TFTP ***Caution:*** This package overrides the settings for network interfaces in the user system. If the user has custom network settings on the machine on which the Pod Manager is installed, the custom settings will be wiped out by the Pod Manager during package installation.

§

3 Configuration

3.1 Discovery

There are two available mechanisms to discover new services, based on DHCP protocol or based on SSDP protocol. By default, both of them are enabled and same service can be detected by both mechanisms. It is highly recommended the user use either one of the two mechanisms to discover 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 using DHCP

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

Hostname of PSME services must start with 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 RMM service must start with string `"rmm"`.

Example: `psme1, storage1, lui, rmm2`

3.1.1.2 Service name based discovery using SSDP

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

The property "Name" available on service entry point (`/redfish/v1`) must be set to the following values:

- For PSME service: `"PSME Service Root"`
- For RSS service: `"RSS Service Root"`
- For LUI service: `"LUI Service Root"`
- For RMM service: `"Root Service"`

Property values can be changed in the configuration file under `“ServiceMapping”` as described in Section 3.1.2, Customization.

Caution: If 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 PSME.

3.1.2 Customization

Configuration for discovery mechanisms is located at `/etc/pod-manager/service-detection.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".`
- `NumberOfRetriesForFailedServiceCheck` – Defines how many times PODM will retrieve service root for newly discovered services.
- `FailedEndpointRecheckIntervalInSeconds` – Defines the interval after which PODM will retry to retrieve service root from the service that failed to be available during initial discovery.
- `ServiceTypeMapping` - contains mapping between service types and property "Name" available on service root of 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 messaged. `"0.0.0.0/0"` means that all networks are accepted.
  - DHCP:
    - `FilesCheckIntervalInSeconds` - how often PODM will refresh data from DHCP.

3.2 Event handling

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

3.2.1 Northbound

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

3.2.1.1 Southbound

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

3.2.2 Customization

Configuration for events is located at `/etc/pod-manager/events.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",
        "PodManagerIpAddress": "127.0.0.1"
      },
      {
        "ServiceType": "inband",
        "NetworkInterfaceName": "lo",
        "PodManagerIpAddress": "127.0.0.1"
      }
    ]
  }
}
```

`Northbound` - this section groups configuration parameters related with Pod Manager Northbound Interface Eventing feature:

`* DeliveryRetryAttempts` - How many times Pod Manager will try to deliver particular event before subscription is removed

`* DeliveryRetryIntervalSeconds` - Interval between delivery attempts in seconds

`Southbound`: This section groups configuration parameters related with Pod Manager Southbound Interface Event subscription feature

`* EventSubscriptionIntervalSeconds` - how often Pod Manager checks if it is subscribed to a service. In case of subscription absence, Pod Manager subscribes to this service.

`* BufferedEventProcessing` - when 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, inband. `* NetworkInterfaceName` - name of network interface used to communicate with specified service. It is used to determine Pod Manager IP address for event subscription in `PSME/Storage Service/RMM services`. When `PodManagerIpAddress` is present, this parameter is ignored.
- `* PodManagerIpAddress` - Pod Manager IP address used for event subscription in `PSME/Storage Service/RMM` service. This parameter is optional.

3.2.3 Impact on Composed Node status

As a result of event handling, Composed Node State may be set to Offline and Health to Critical when:

- Associated Computer System's or RemoteTarget's State is different than Enabled
- Associated Computer System is removed or Remote Target is deleted

3.3 Service root UUID configuration

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

Sample UUID format:

```
{
  "UUID": "00000000-0000-0000-0000-000000000000"
}
```

3.4 Storage service configuration

Storage service can be configured in two ways:

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

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

3.4.1 Storage service working on drawer's computer system

To have storage service working on drawer's computer system, the user must deploy the storage service image onto the computer system's storage device. 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 via PODM REST API, then it may be targeted for Deep Discovery (the computer system will be rebooted and left in a powered off state).

Note: The hostname for storage service must start with string `"storage"`.

3.4.2 Storage service working on external host attached to rack's storage network

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

Note: Additional settings on ToR may be required for this type of deployment. Also, the hostname for storage service must start with string `"storage"`.

3.5 Network management

The Pod Manager has the following reserved VLANs preconfigured:

- 4091 - Production Network
- 4094 - Service Management Network
- 4092 - Rack Backplane Management
- 4088 - In-band 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 the `/etc/network/interfaces` file to make this change permanent between the Pod Manager updates.

More information about these files are found on the Interfaces Manpage on the Ubuntu website: *http://manpages.ubuntu.com/manpages/wily/man5/interfaces.5.html*.

Remember 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 the subsequent record would also report failure.

To disable the Pod Manager 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.

3.6 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 Pod Manager 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 given external service (and its assets) at the time of its absence notice. The `"0"` value might be helpful especially when Pod Manager communicates with an older version of the RSD software modules.

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

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

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

§

4 Security

4.1 Securing Pod Manager northbound API

Custom certificates can be configured to secure Pod Manager 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, its password, localization, and stored keys. Keystore path, password and alias are provided by an entry in `"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 VAULT tool. Detailed information is provided here:

```
https://developer.jboss.org/wiki/JBossAS7SecuringPasswords
```

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

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.

Custom certificate import example:

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

For more information, use the following command:

```
man keytool
```

4.1.1 Basic authentication management

Default login credentials for Pod Manager are:

```
User: admin
Password: admin
```

Basic authentication mechanism can be managed by the `"add-user"` utility.

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

4.2 Securing Pod Manager southbound API

The Pod Manager on southbound API can act as both client and server. South bound API communication is established using TLSv1.2 protocol.

While establishing a secure connection with external services (southbound communication), Pod Manager uses `TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256` cipher suite. This means that cipher suite implements `Elliptic - Curve Diffie - Hellman` Ephemeral key exchange. `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 here:

https://tools.ietf.org/html/rfc5289

Important points to note:

- For communication to be properly established by TLS with client authentication at 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 NTP service on Pod Manager to be used for time synchronization between Pod Manager and other services (RMM, PSME, LUI, Storage Service). Each service using Pod Manager as 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 client signed certificate is a well formatted container that includes a full certification chain.

4.2.1 Pod Manager as server

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

4.2.2 Pod Manager as client

The Pod Manager acts as a client when obtaining information from external services and is configured to send its own certificate to be authenticated by server. LUI does not authenticate the Pod Manager on the SB API. The Pod Manager does not perform server authentication.

The CA signed certificate is used by the Pod Manager. The CA certificate should be added to external services trust store.

The default client certificate and CA certificate are provided with Pod Manager `.deb` packages. The client certificate has been signed by CA certificate which is also provided.

The Pod Manager (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.

TLS can be turned **ON/OFF** by modifying following configuration file (it's turned **ON** by default):

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

The TLS can be configured separately for each external service type, by setting the `"true"` or `"false"` value accordingly.

Example to set TLS **OFF** for PSME:

```
{
  "SslEnabledForRmm" : true,
  "SslPortForRmm" : 8091,
  "DefaultPortForRmm" : 8090,
  "SslEnabledForPsme" : false,
  "SslPortForPsme" : 8443,
  "DefaultPortForPsme" : 8888,
  "SslEnabledForRss" : true,
  "SslPortForRss" : 8443,
  "DefaultPortForRss" : 8888,
  "SslEnabledForLui" : true,
  "SslPortForLui" : 8443,
  "DefaultPortForLui" : 8888
}
```

4.2.3 Detailed configuration

Pod Manager 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
```

Keystore `"client.jks"` is protected by a password. To hash the password, the WildFly hashing tool called VAULT is used. Detailed information can be obtained from:

https://developer.jboss.org/wiki/JBossAS7SecuringPasswords

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 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 Pod Manager must have the CA certificate provided.

Caution: The certificate distributed with the Pod Manager 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 certificate distributed with Pod Manager is located at:

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

To check the certificate validation period from the `client.jks` file, use the keytool:

```
keytool -v -list -keystore client.jks
```

4.3 Certificate management

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

4.3.1 Creating client certificate

The script to generate the client certificate is located at:

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

The script must be executed with root privileges by using:

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

Print script using --help option:

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

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

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

The above 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 --ca-certificate
/path/to/ca/file/ca.pfx --client-certificate
"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, 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 - using interactive mode.

During this process, the user will be asked to pass the client certificate attributes manually and follow the instructions displayed on screen to complete the process:

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

Upon completion, the above command creates a new client certificate under `/var/lib/pod-manage`r directory and extracted CA certificate and private key are copied to the `/var/lib/pod-manager/ca` directory.

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

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

Take note of the escape sequence used to escape `"white space"` and existing CA certificate path.

Note: When using TLS configuration on PSMEs, authentication from the Pod Manager to PSMEs is certificate-based and PSMEs rest servers will not communicate with clients that do not perform certificate-based authentication.

Note: To access the PSME 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 Pod Manager for use with other HTTP clients.

The following steps describe the process to access the PSME APIs using the `cURL` client only:

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

```
sudo keytool -importkeystore -srckeystore /var/lib/pod-manager/client.jks
-srcstoretype JKS -destkeystore client.p12 -deststoretype pkcs12
```

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

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

The `.pem` certificate file can be used by the `cURL` or imported into a web browser such as Firefox* or Chrome*.

Example use with `cURL` client:

```
curl -k -i --cert client.pem --cacert /var/lib/pod-manager/root.crt -X
GET https://<psme_ip>:8443/redfish/v1/
```

4.3.2 Importing client certificate

Script to import a signed client certificate to the container is located at:

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

The Script must be executed with root privileges as shown:

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

Print script using:

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

Note: It is assumed the signed certificate container used is of type `PKCS #12` and contains a full certificate chain.

Example:

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

The above command creates a container of type JKS with imported client certificate signed by CA certificate, located at:

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

4.3.3 Distributing CA certificate

The CA certificate must be propagated to all external services (RMM, PSME, Storage Services) that authenticate the Pod Manager.

Which CA certificate to use:

- The default CA certificate provided with Pod Manager 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
  ```

4.3.3.1 Provide RMM and PSME with CA certificate

The following procedures provides the RMM and PSME with CA certicate using the following procedures.

RMM:

1. The selected CA certificate must be copied to RMM. The RMM requires the CA certification file to be named `"podm.cert"`.
2. Rename the CA certificate file `“podm.cert”` and copy the CA certificate to following location:
   ```
   /etc/rmm/podm.cert
   ```
3. Once the is copied, the RMM will 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 the following location:

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

PSME:

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

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

4.3.3.2 Provide storage service with CA certificate

Note: To copy the CA certificate the RMM requires the CA certificate file to be named `"podm.cert"`.

The user must rename the CA certificate file to `"podm.cert"` and copy it to following location:

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

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

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

4.3.4 Using custom VAULT configuration

The VAULT is used to read the keystore password:

- By Pod Manager while reading the certificate from keystore files
- By script to generate/import certificates to set up keystore password

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

https://developer.jboss.org/wiki/JBossAS7SecuringPasswords

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 following two files:

WildFly configuration file “standalone.xml” located at:

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

Scripts configuration file `"vault.json"` used to generate/import certificates located at:

```
/var/lib/pod-manager/vault/vault.json
```

§

5 Deployment

Pod Manager is Java EE application that uses Wildfly Application Server and PostgreSQL as persistent storage engine.

5.1 Starting PostgreSQL service

```
sudo service postgresql start
```

To confirm that the DB is running, use this command:

```
sudo service postgresql status
```

If it is running, the above command should return 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 Pod Manager Service

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

If running, 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 Pod Manager REST API

Pod Manager Service can be accessed at the following address:

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

We recommend using JSON formatting plugins for web browsers e.g `JSONView`, JSON Formatter etc.

Requests should contain HTTP Basic authentication headers.

Table 4. PODM REST API

Resource	URI
Service Root	/redfish/v1
Chassis Collection	/redfish/v1/Chassis
Chassis	/redfish/v1/Chassis/{chassisID}
PowerZone Collection	/redfish/v1/Chassis/{chassisID}/PowerZones
PowerZone	/redfish/v1/Chassis/{chassisID}/PowerZones/{powerzonesID}
ThermalZone Collection	/redfish/v1/Chassis/{chassisID}/ThermalZones
ThermalZone	/redfish/v1/Chassis/{chassisID}/ThermalZones/{thermalzoneID}
Power	/redfish/v1/Chassis/{chassisID}/Power
Thermal	/redfish/v1/Chassis/{chassisID}/Thermal
Drive	/redfish/v1/Chassis/{chassisID}/Drives/{driveID}

Resource	URI
Computer System Collection	/redfish/v1/Systems
Computer System	/redfish/v1/Systems/{systemID}
(Computer System) Ethernet Interface Collection	/redfish/v1/Systems/{systemID}/EthernetInterfaces
(Computer System) Ethernet Interface	/redfish/v1/Systems/{systemID}/EthernetInterfaces/{nicID}
Computer System Metrics	/redfish/v1/Systems/{systemID}/Metrics
(Computer System) VLAN Network Interface Collection	/redfish/v1/Systems/{systemID}/EthernetInterfaces/{nicID}/VLANs
(Computer System) VLAN Network Interface	/redfish/v1/Systems/{systemID}/EthernetInterfaces/{nicID}/VLANs/{vlanID}
(Computer System) Network Interface Collection	/redfish/v1/Systems/{systemID}/NetworkInterfaces
(Computer System) Network Interface	/redfish/v1/Systems/{systemID}/NetworkInterfaces/{networkInterfaceID}
Network Device Function Collection	/redfish/v1/Systems/{systemID}/NetworkInterfaces/{networkInterfaceID}/NetworkDeviceFunctions
Network Device Function	/redfish/v1/Systems/{systemID}/NetworkInterfaces/{networkInterfaceID}/NetworkDeviceFunctions/{networkDeviceFunctionID}
Processor Collection	/redfish/v1/Systems/{systemID}/Processors
Processor	/redfish/v1/Systems/{systemID}/Processors/{processorID}
Memory Collection	/redfish/v1/Systems/{systemID}/Memory
Memory	/redfish/v1/Systems/{systemID}/Memory/{memoryID}
Storage Collection	/redfish/v1/Systems/{systemID}/Storage
Storage	/redfish/v1/Systems/{systemID}/Storage/{StorageID}
Simple Storage Collection	/redfish/v1/Systems/{systemID}/SimpleStorage
SimpleStorage	/redfish/v1/Systems/{systemID}/SimpleStorage/{SimpleStorageID}
Manager Collection	/redfish/v1/Managers
Manager	/redfish/v1/Managers/{managerID}
Network Protocol	/redfish/v1/Managers/{managerID}/NetworkProtocol
(Manager) Network Interface Collection	/redfish/v1/Managers/{managerID}/EthernetInterfaces
(Manager) Network Interface	/redfish/v1/Managers/{managerID}/EthernetInterfaces/{nicID}
(Manager) VLAN Network Interface Collection	/redfish/v1/Managers/{managerID}/EthernetInterfaces/{nicID}/VLANs
(Manager) VLAN Network Interface	/redfish/v1/Managers/{managerID}/EthernetInterfaces/{nicID}/VLANs/ {vlanID}
Storage Service Collection	/redfish/v1/Services
Storage Service	/redfish/v1/Services/{serviceID}
Remote Target Collection	/redfish/v1/Services/{serviceID}/Targets
Remote Target	/redfish/v1/Services/{serviceID}/Targets/{targetID}
Logical Drive Collection	/redfish/v1/Services/{serviceID}/LogicalDrives
Logical Drive	/redfish/v1/Services/{serviceID}/LogicalDrives/{driveID}
Physical Drive Collection	/redfish/v1/Services/{serviceID}/Drives
Physical Drive	/redfish/v1/Services/{serviceID}/Drives/{driveID}
Ethernet Switch Collection	/redfish/v1/EthernetSwitches
Ethernet Switch	/redfish/v1/EthernetSwitches/{switchID}
Ethernet Switch Port Collection	/redfish/v1/EthernetSwitches/{switchID}/Ports
Ethernet Switch Port	/redfish/v1/EthernetSwitches/{switchID}/Ports/{portID}

Resource	URI
(Switch Port) VLAN Network Interface Collection	/redfish/v1/EthernetSwitches/{switchID}/Ports/{portID}/VLANs
(Switch Port) VLAN Network Interface	/redfish/v1/EthernetSwitches/{switchID}/Ports/{portID}/VLANs/{vlanID}
Composed Node Collection	/redfish/v1/Nodes
Composed Node	/redfish/v1/Nodes/{nodeID}
Fabric Collection	/redfish/v1/Fabrics
Fabric	/redfish/v1/Fabrics/{fabricID}
Fabric Switch Collection	/redfish/v1/Fabrics/{fabricID}/Switches
Fabric Switch	/redfish/v1/Fabrics/{fabricID}/Switches/{switchID}
Fabric Switch Port collection	/redfish/v1/Fabrics/{fabricID}/Switches/{switchID}/Ports
Fabric Switch Port	/redfish/v1/Fabrics/{fabricID}/Switches/{switchID}/Ports/{portID}
Fabric Zone Collection	/redfish/v1/Fabrics/{fabricID}/Zones
Fabric Zone	/redfish/v1/Fabrics/{fabricID}/Zones/{zoneID}
EndPoint Collection	/redfish/v1/Fabrics/{fabricID}/Endpoints
Endpoint	/redfish/v1/Fabrics/{fabricID}/Endpoints/{endpointID}
PCIDevice	/redfish/v1/Chassis/{chassisID}/PCIeDevices/{deviceID}
PCIe Device Function	/redfish/v1/Chassis/{chassisID}/PCIeDevices/{deviceID}/Functions/{functionID}
Event Service	/redfish/v1/EventService
Event Service Subscription Collection	/redfish/v1/EventService/Subscriptions
Event Service Subscription	/redfish/v1/EventService/Subscriptions/{subscriptionID}
Telemetry Service Collection	/redfish/v1/TelemetryService
Metric Definition Collection	/redfish/v1/TelemetryService/MetricDefinitions
Metric Definition	/redfish/v1/TelemetryService/MetricDefinitions/{metricDefinitionID}
Metric Report Definition Collection	/redfish/v1/TelemetryService/MetricReportDefinitions
Metric Report Definition	/redfish/v1/TelemetryService/MetricReportDefinitions/{metricReportDefinitionID}

Available resources:

Refer to Intel® Rack Scale Design Pod Manager API Specification in Table 2, for a detailed description of above resources.

6 Features

6.1 Composed node lifecycle management

6.1.1 Composed node using JSON template

To create a Composed Node using Pod Manager REST API, create a JSON template describing requested resources. It must be supplied to Pod Manager by performing an HTTP POST request on the Composed Node Collection Action URI located at `/redfish/v1/Nodes/Actions/Allocate` on the Pod Manager service.

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

6.1.2 Specifying requirements for a composed node

JSON template contains requirements for a single Composed Node. Basic customization covers setting a `"Name"` and `"Description"` 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 Pod Manager will use the default name.

The example below will allocate a single Composed Node with requested name and description:

```
{
    "Name": "Customized Composed Node name",
    "Description": "Description of a customized Composed Node"
}
```

JSON template may contain requirements for: Processors, Memory, Remote Drives, Local Drives and 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 may 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, Allowed values contains additional restrictions to JSON type or hints (e.g. 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 may contain Resource and Chassis objects. Resource must contain the Pod Manager URI (presented as `"@odata.id"`) of the discovered resource (Processor's URI in Processor section, URI to Memory resource in Memory section and so on). Chassis must contain the Pod Manager URI of the discovered Chassis in which applicable resources will be looked for.

6.1.4 Specifying processor requirements

The JSON template may contain requirements for multiple Processors. The example below specifies requirements for a single Processor to be used in Composed Node.

```
{
    "Processors": [{
        "Model": "Multi-Core Intel(R) Xeon(R) processor 7xxx Series",
         "TotalCores": 2,
        "AchievableSpeedMHz": 3700,
        "InstructionSet": "x86-64",
        "Oem:Intel_RackScale": {
            "Capabilities": [
                "sse",
                "sse2"
            ],
            "Brand": "X7"
        },
        "Resource": {
            "@odata.id": "/redfish/v1/Systems/1/Processors/1"
        },
        "Chassis": {
            "@odata.id": "/redfish/v1/Chassis/1"
        }
    }]
}
```

Table 5. Attribute Definition (for Processor)

Key	JSON type	Allowed values	Nullable	Notes, limitations
Model	String		Yes	String representing Processor model
TotalCores	Number		Yes	Positive integer value expected
AchievableSpeedMHz	Number		Yes	Positive integer value expected
InstructionSet	String	"x86", "x86-64", "IA-64", "ARM-A32", "ARM-A64", "MIPS32", "MIPS64", "OEM"	Yes	One of allowed, enumerated values
Oem	Object		Yes	
Oem -> Brand	String	"E3", "E5", "E7", "X3", "X5", "X7", "I3", "I5", "I7", "Unknown"	Yes	One of allowed, enumerated values
Resource	Object	Exact location of a single Processor.	Yes	See Location requirements section
Chassis	Object	Exact location of a single Chassis.	Yes	See Location requirements section

Allocation assumptions:

- Which Processors will meet supplied requirements?
  - located on the same computer system as other resources
  - with exact match on Model
  - with exact match on Brand
  - with at least TotalCores
  - with at least AchievableSpeedMHz
  - with exact match on InstructionSet

6.1.5 Specifying memory requirements

The JSON template may contain requirements for multiple Memory Modules. The example below specifies the requirements for a single Memory Module to be used in Composed Node.

```
{
    "Memory": [{
         "CapacityMiB": 16000,
        "MemoryDeviceType": "DDR3",
        "SpeedMHz": 1600,
        "Manufacturer": "Intel",
        "DataWidthBits": 64,
        "Resource": {
            "@odata.id": "/redfish/v1/Systems/1/Memory/1"
        },
        "Chassis": {
            "@odata.id": "/redfish/v1/Chassis/1"
        }
    }]
}
```

Table 6. Attribute Definition (for Memory)

Key	JSON type	Allowed values	Nullable	Notes, limitations
CapacityMiB	Number		Yes	Positive value expected
MemoryDeviceType	String	`"DDR", "DDR2", "DDR3", "DDR4", "DDR4_SDRAM", "DDR4E_SDRAM", "LPDDR4_SDRAM", "DDR3_SDRAM", "LPDDR3_SDRAM", "DDR2_SDRAM", "DDR2_SDRAM_FB_DIMM", "DDR2_SDRAM_FB_DIMM_PROBE", "DDR_SGRAM", "DDR_SDRAM", "ROM", "SDRAM", "EDO", "FastPageMode", "PipelinedNibble"`	Yes	One of allowed, enumerated values
SpeedMHz	Number		Yes	Positive integer value expected
Manufacturer	String		Yes	String representing Memory Module manufacturer name
DataWidthBits	Number		Yes	Positive integer value expected
Resource	Object	Exact location of a single Memory Module.	Yes	See Location requirements section
Chassis	Object	Exact location of a single Chassis.	Yes	See Location requirements section

Allocation assumptions:

- Which Memory Modules (represented by Memory resource) will meet supplied requirements?
  - located on the same computer system as other resources
  - with exact match on MemoryDeviceType
  - with at least SpeedMHz
  - with exact match on Manufacturer
  - with at least DataWidthBits
- If a computer system contains Memory Modules of a total size at least CapacityMiB, it will meet the requirements.

6.1.6 Specifying remote drive requirements

The JSON template may contain requirements for multiple Remote Drives, but currently only one set of requirements is supported. The example below specifies requirements for a single Remote Drive to be used in Composed Node.

```
{
    "RemoteDrives": [{
        "CapacityGiB": 80,
        "iSCSIAddress": "iqn.oem.com:fedora21",
        "Master": {
            "Type": "Snapshot",
            "Resource": {
                "@odata.id": "/redfish/v1/Services/1/LogicalDrives/1"
            }
        }
    }]
}
```

Table 7. Attribute Definition (for Remote Drive)

Key	JSON type	Allowed values	Nullable	Notes, limitations
CapacityGiB	Number		Yes	Positive value expected, required if Master Drive supplied. Should be at least the size of Logical Drive used as Master Drive.
iSCSIAddress	String		No	Required. Defines TargetIQN of RemoteTarget. When no Master Drive supplied - it defines IQN of an existing target. Otherwise defines IQN to be set for new Remote Target (should be unique in Pod Manager).
Master	Object		Yes	
Master -> Type	String	"Snapshot", "Clone"	No	One of allowed, enumerated values. Required if Master Drive supplied.
Master -> Address	Object		No	Pod Manager URI of discovered Logical Volume. Required if Master Drive supplied.

6.1.6.1 Using existing remote drive

To use an existing Drive it is necessary to:

- set iSCSIAddress to TargetIQN of existing target,
- do not provide Master, or set it to null

```
{
    "RemoteDrives": [{
        "iSCSIAddress": "iqn.oem.com:fedora21"
    }]
}
```

6.1.6.2 Using a Master Drive for fresh remote drive creation

To use a fresh Drive created from a Master Drive, it is necessary to:

- set CapacityGiB to define the capacity of the new Remote Drive that is at least Master Drive size
- set Address to IQN that is unique in Pod Manager
- set Master -> Type to "Snapshot" or "Clone"
- set Master -> Resource to valid Pod Manager URI of Logical Drive to be used as source Drive

```
{
    "RemoteDrives": [{
        "CapacityGiB": 80,
        "iSCSIAddress": "iqn.oem.com:fedora21",
        "Master": {
            "Type": "Snapshot",
            "Resource": {
                "@odata.id": "/redfish/v1/Services/1/LogicalDrives/1"
            }
        }
    }]
}
```

6.1.7 Specifying local drive requirements

The JSON template may contain requirements for multiple Local Drives (represented by Device resource under System Adapters or by Simple Storage Device under System Simple Storage). The example below specifies requirements for a single Local Drive to be used in Composed Node.

```
{
    "LocalDrives": [{
        "CapacityGiB": 100,
        "Type": "HDD",
        "MinRPM": 5400,
        "SerialNumber": "12345678",
        "Interface": "SATA",
        "Resource": {
            "@odata.id": "/redfish/v1/Systems/1/Adapters/1/Devices/1"
        },
        "Chassis": {
            "@odata.id": "/redfish/v1/Chassis/1"
        }
    }]
}
```

Table 8. Attribute Definition (for Local Drive)

Key	JSON type	Allowed values	Nullable	Notes, limitations
CapacityGiB	Number		Yes	Positive value expected
Type	String	`"HDD", "SSD", "NVMe"`	Yes	One of allowed, enumerated values
MinRPM	Number		Yes	Positive integer value expected
SerialNumber	Number		Yes	
Interface	String	`"PCIe", "SAS", "SATA"`	Yes	One of allowed, enumerated values
Resource	Object	Exact location of a single Device.	Yes	See Location requirements section
Chassis	Object	Exact location of a single Chassis.	Yes	See Location requirements section

Allocation assumptions:

- Which Local Drives will meet supplied requirements?
  - located on the same computer system as other resources
  - with at least CapacityGiB
  - with exact match on Type
  - with at least MinRPM
  - with exact SerialNumber
  - with exact Interface

6.1.8 Specifying Ethernet interface requirements

The JSON template may contain requirements for multiple Ethernet Interfaces. The example below specifies requirements for a single Ethernet Interface to be used in Composed Node.

```
{
    "EthernetInterfaces": [{
        "SpeedMbps": 1000,
        "PrimaryVLAN": 100,
        "VLANs": [{
            "VLANId": 100,
            "Tagged": false
        }],
        "Resource": {
            "@odata.id": "/redfish/v1/Systems/1/EthernetInterfaces/1"
        },
        "Chassis": {
            "@odata.id": "/redfish/v1/Chassis/1"
        }
    }]
}
```

Table 9. Attribute Requirement (for Ethernet Interfaces)

Key	JSON type	Allowed values	Nullable	Notes, limitations
SpeedMbps	Number		Yes	Positive integer value expected
PrimaryVLAN	Number		Yes	Positive integer value expected
VLANs	Array[Object]		Yes	Null value will be interpreted as absence of this key. Empty array [] will clear all existing vlans, excluding *Reserved VLANs*
VLANs -> VLANId	Number		No	Positive integer value expected
VLANs -> Tagged	Boolean	true, false	No	Boolean value
Resource	Object	Exact location of a single Ethernet Interface.	Yes	See Location requirements section
Chassis	Object	Exact location of a single Chassis.	Yes	See Location requirements section

Allocation assumptions:

- Which Ethernet Interfaces will meet supplied requirements?
  - located on the same Computer System as other resources
  - with at least SpeedMbps
  - ones that are connected with SwitchPorts (when VLANs section is provided)

6.1.8.1 Reserved VLANs

It is possible to restrict the usage of some vlans by changing the configuration file located in /etc/pod-manager/allocation.json.

Example file looks like:

```
{
  "ReservedVlanIds": [170, 4088, 4091, 4094]
}
```

Where 170, 4088, 4091, 4094 are VLANs which are reserved. Reserved VLANs have the following implications:

- Allocation JSON cannot contain such VLANs and such requests result in an error
- Reserved VLANs are not deleted during allocation
- Reserved VLANs are not deleted during disassembly

6.1.9 Allocation algorithm

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

Four major scenarios are currently supported:

- Allocating resources for Composed Node to be booted from Local Drive.
- Allocating resources for Composed Node to be booted from existing Remote Drive.
- Allocating resources for Composed Node to be booted from Remote Drive that need to be created.
- Allocating resources for 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 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.

6.1.9.1 Detailed process of selecting and allocating a Computer System for a Composed Node

1. Find all Computer Systems that are not yet allocated (not used by any other allocated Composed Node) with Status **Enabled** and Health **OK**.
2. Filter Computer Systems by specified Resource and Chassis (if supplied in template).
3. 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,
    - Exactly matching requested instruction set.
4. 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.

5. 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.

6. 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).

7. A first Computer System from resulting filtered collection is then allocated to be used in Composed Node.

6.1.9.2 Connection between Computer System's EthernetInterface and EthernetSwitchPort

In order 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:

- `NeighborMAC` on EthernetSwitchPort resource
- `MacAddress` on EthernetInterface resource

If the Computer System's Ethernet Interface and Ethernet Switch Port properties contain the same values they are treated as connected.

Note: Only Computer Systems with Ethernet Interfaces which are connected to Ethernet Switch Ports can be used in allocation with a specified VLANs requirement.

6.1.9.3 Detailed process of selecting Remote Drives

1. Determine what type of Remote Drive is requested.
2. When requesting existing Remote Drive:
    a. Find all Targets that are not yet allocated (not used by any allocated Composed Node).
    b. Find first Target that exactly matches the requested IQN and allocate it to be used in Composed Node.
3. When requesting a new Remote Drive:
    a. Check if Target does not exist with requested IQN to be set for newly created target.
    b. Check if Logical Drive requested as Master Drive exists on Storage Service handled by Pod Manager, and select this Storage Service to handle new Target creation.
    c. Find all Logical Volume Groups meeting requirements:
        1. Located on selected Storage Service
        2. Having free space of at least requested capacity for a new Remote Drive
    d. A first Logical Volume Group from resulting filtered collection is. selected as a placement for new Logical Volume, which will be exposed as a new Target (Remote Drive)
    e. A new Logical Volume is created on selected Logical Volume Group (as a snapshot or as a clone).
    f. A new Target is created on top of a newly created Logical Volume.
    g. Newly created Target is allocated to be used in Composed Node.

6.1.9.4 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. State of Composed Node is set to `"Allocated"`. An "Allocated" Composed Node is a Pod Manager proposition that can be either accepted or rejected.

- If accepted, the user has to send a HTTP POST request on 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, Composed Node remains `"Assembling"` until Target creation finishes. When Target is successfully assembled to be used with the Composed Node, node's state is set to `"PoweredOff".`
  - Assembly process doesn't end with sending power on request, 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 pick, it is recommended to send HTTP DELETE on all rejected proposals of Composed Nodes to free the resources allocated by them.

6.1.9.5 Disassembly

Upon disassembly of Composed Node, several actions are performed:

- A 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 Using Pod Manager with RMM service

Pod Manager requires that RMM software be installed on external machine and meet the following requirements:

- Host should be part of VLAN 4094
- Hostname must start with a string "rmm"
- Pod Manager certificate is present in `/etc/rmm/podm.cert` - for more information about security refer to Section 4.2, Securing Pod Manager southbound API.

6.2.1 Rack lifecycle policy (Chassis of type Rack)

6.2.1.1 Creating 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 Pod Manager REST API and this Drawer is attached to it.
  - If the Rack with the rack location that this Drawer is reporting (under `'ParentLocationId'`) does exist, the Drawer is attached to this Rack resource.
- When a Drawer has been rediscovered (eg. after slow-poll refresh):
  - If Rack with the rack location that this drawer is reporting does not exist, a new Rack is created in Pod Manager 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 new RMM service is discovered:
  - The New Rack resource is created in the Pod Manager REST API and RMM attributes are connected to it. All racks with the same `'LocationId',` but without RMM (attached) are removed and the Drawers directly linked to those Racks are moved to the newly created rack resource.
- When the RMM has been rediscovered (eg. after slow-poll refresh):
  - If RMM's Rack changes its location id (in RMM: `'RackPuid'` field), Pod Manager's 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 Pod Manager. Drawers can have their `'ParentLocationId'` updated via RMM <-> PSME <-> Pod Manager events notification channel or slow-poll refresh.

6.2.1.2 Rack resource removal

The Rack resource will be deleted under the following conditions:

- When a removed Drawer is the last one that was attached to the Rack and Rack and does not have the RMM service asociated with it.
- When RMM service has disappeared (eg. machine with RMM was turned off) and the Rack does not contain any Drawers.

6.3 Deep discovery

Note: In Intel® RSD 2.2 reference code, all discovery performed by PSME is done out-of-band (OOB). This default OOB discovery enables all RSD POD Manager functionality. The legacy RSD 1.x in-band discovery tool (LUI) is included in the reference code and may be used as an alternative to the OOB discovery. This section describes how to build and configure in-band discovery tool.

6.3.1 Introduction

In general, PSME is responsible for exposing information about Computer Systems. Currently, 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 Pod Manager can read from. Pod Manager reads the basic BMC collected data from PSME, then boots LUI on each computer system. The data visible in the PSME APIs will only be the BMC collected data, whereas the Pod Manager merges both the basic BMC data from the PSME and the additional data discovered from the LUI environment.

6.3.2 General requirements, assumptions and limitations

- Pod Manager and PSME have DiscoveryState property
- 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 -> Pod Manager`: 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 PSME or from deep discovery, but not from both. 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.

6.3.3 Building Linux* Utility Image (LUI)

Please refer to Intel® Rack Scale Design PSME User Guide to get information regarding prerequisites.

6.3.3.1 Preparing rootfs directory

Please refer to Intel® Rack Scale Design PSME User Guide to get information on preparing rootfs directory.

6.3.3.2 Building LUI

Refer to the Intel® Rack Scale Design PSME User Guide in Table 2 to get information on building LUI.

6.3.4 Configuring deep discovery

To enable Deep Discovery, Pod Manager requires LUI. It should be provided at:

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

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

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

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

6.3.4.1 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 lower value to prevent overcurrent and power spikes. Setting this property to higher value will yield an overall shorter time needed to perform 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 threshold value per Drawer for both manual and automatic process triggering.

6.4 InBand Service support

Pod Manager supports InBand / OOB data aggregation mechanism for a subset of Redfish resources that can have a representation originating from either PSME (OOB) or InBand Service or both. Aggregation is performed on `ComputerSystem` and its following subresources: `Processor`, `Memory`, `EthernetInterface`, `SimpleStorage/SimpleStorageDevice`, `Storage/StorageController` and on `Chassis/Drive` resources related to the particular ComputerSystem that gets aggregated.

6.4.1 How Pod Manager detects InBand Service

Note: Refer to `README.md` of `InBand` Service example code bundled with the Pod Manager source code to deploy your own `InBand` Service instance.

It is assumed that a single `InBand` Service instance will handle a set of ComputerSystems that utilize the same agent to provide data via `InBand` channel (one `InBand` Service should be deployed per agent type). Pod Manager will detect `InBand` Service when provided with static endpoint configuration via `/tmp/services.list` file containing a line with InBand Service endpoint, e.g.:

```
https://127.0.0.1:8448/redfish/v1/ inband
```

The Pod Manager will subscribe to events from this service and will discover its resources. The resources will get updated independently from other services.

6.4.2 Enabling aggregation of InBand Service resources

The aggregation feature is disabled by default and can be enabled via configuration file:

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

by setting its content to:

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

6.4.3 How the resources get aggregated

InBand Service resources are discovered independently and are stored separately from their OOB counterparts. The aggregation is performed only when a primary ComputerSystem (discovered via OOB) has a matching representation in InBand resources (based on ComputerSystem's UUID property).

Note: Aggregation is performed on demand, upon GET request on resources from Pod Manager Northbound API.

6.4.3.1 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 value, * if the overlay value is not null then use overlay value (prefer InBand value over OOB) * in other case use existing value

Caution: When using any `InBand` Service implementation make sure that it exposes full and meaningful representation of resources to avoid unintended override of properties obtained via OOB (this includes ComplexTypes and inline expanded collections).

6.4.3.2 OEM data aggregation

The Pod Manager enables merging of Oem namespaces that are specific for a given inband agent (and different than `Intel_RackScale`).

Consider two ComputerSystems are matched by the UUID and have some additional data available in the custom Oem space on `InBand` Service:

Oem from OOB (PSME):

```
{
    "Oem": {
        "Intel_RackScale": {
            "@odata.type": "#Intel.Oem.ComputerSystem",
            "ProcessorSockets": 2
        }
    }
}
Oem from InBand Service:
{
    // other properties removed for brevity
    "Oem": {
        "SomeOem_org.OemSpace": {
            "SomeProp": "SomeVal"
        }
    }
}
```

Merged Oems on Pod Manager:

```
{
    // other properties removed for brevity
    "Oem": {
        "Intel_RackScale": {
            "@odata.type": "#Intel.Oem.ComputerSystem",
            "ProcessorSockets": 2
        },
        "SomeOem_org.OemSpace": {
            "SomeProp": "SomeVal"
        }
    }
}
```

6.4.3.3 Subresource collections aggregation

Apart from properties and Oem aggregation, the linked subresources collections get aggregated using their unique identifiers. Various subresources are uniquely identified using different property values:

Table 10. Subresource Identification Table

Resource	Unique Identifier
ComputerSystem	UUID
Processor	Socket
Memory	DeviceLocator
EthernetInterface	MACAddress
SimpleStorage	UEFIDevicePath
SimpleStorageDevice	Name
Storage	Assumed always one per ComputerSystem
StorageController	DurableName (when in UUID format), source URI otherwise
Chassis	Using matching ComputerSystem's ContainedBy property to gain access to its Drives
Drive	Last path segment of source URI

Based on these values Pod Manager is able to determine the contents of subresource collections (e.g. 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 MAC Address property)

6.5 Link Aggregation Group (LAG)

Link aggregation group 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, thus reducing cost.

6.5.1 Creating LAG

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 Pod Manager REST API, it is necessary to create and supply a proper JSON template to Pod Manager 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"
            }
        ]
    }
}
```

Table 11. Attribute Requirement (for LAG)

Key	JSON type	Allowed values	Nullable	Notes, limitations
`PortId`	String		Yes	Switch port unique identifier. ***Caution:*** The maximum value must not exceed 16 characters and must not contain white spaces.
`PortMode`	String	`"LinkAggregationStatic"`, `"LinkAggregationDynamic"`	Yes	Port working mode. Currently only `LinkAggregationStatic` mode is supported.
`PortMembers`	Array[Link]		Yes	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: • `"PortClass": "Physical"` • `"PortType": "Upstream"` • The same speed. None of these ports can be a member of another LAG. Empty arrays will be interpreted as absence of this requirement key.

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
```

6.5.2 Modifying LAG

To modify LAG using Pod Manager REST API, it is necessary to create and supply the proper JSON template to Pod Manager 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"
            }
        ]
    }
}
```

Table 12. Attribute Requirement (for VLAN)

Key	JSON type	Allowed values	Nullable	Notes, limitations
`AdministrativeState`	String	"Up", "Down"	No	Port link state forced by user.
`LinkSpeedMbps`	Number	Nonnegative number	No	Port speed.
`FrameSize`	Number	Nonnegative number	No	MAC frame size in bytes.
`Autosense`	Boolean	true, false	No	Indicates if the speed and duplex is automatically configured by the NIC
`PrimaryVLAN`	Link		No	Link to VLAN available on this port (only these VLANs are acceptable).
`PortMembers`	Array[Link]		No	Array of ports being member of LAG. Must be placed in Links object. All ports contained in this array must have: • - `"PortClass": "Physical"` • - `"PortType": "Upstream"` • - the same speed None of these ports can be a member of another LAG.’ If PortMembers array is not present in PATCH request, list of port members will not be changed. Otherwise, there must be at least one port.

All of the above properties are optional. If not provided, they won't be changed.

Response after success LAG modification:

```
HTTP/1.1 204 No Content
```

6.5.3 Removing LAG

To remove LAG using the Pod Manager 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}
Response after successful LAG port removal:
HTTP/1.1 204 No Content
```

6.5.4 Limitations

- 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

6.6 Pod Manager Northbound Interface Eventing

Pod Manager 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 PodM data model.

6.6.1 Supported events

Currently Pod Manager is able to generate following kind of events:

- `ResourceAdded` - event generated when new Redfish resource will be detected by 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 PodM will detect that State of already known Redfish resource has been changed
- `Alert` - event generated when PodM will detect that Status of already known Redfish resource enters Critical state

6.6.2 Event subscription

The user can control subscriptions life cycle using following HTTP operations: * POST on `/redfish/v1/EventService/Subscriptions (subscription creation)` * DELETE on `/redfish/v1/EventService/Subscriptions/{subcriptionID}` (subscription deletion)

Note: Be aware your subscription might be unregistered automatically in any case when Pod Manager (PodM) cannot reach subscription endpoint after some #repeated retries.

Consider following scenario: - the user is registering new subscription (provided destination is *http://inactive-listener* – since the `EventSubscription` is a Redfish resource the PodM will generate `ResourceAdded` event - PodM will try to deliver a generated event to the *http://inactive-listener* – since this link *http://inactive-listener* is not active, the PodM will try to redeliver the event - number of redelivery retries is limited to the value described by `DeliveryRetryAttempts` - when the threshold defined by the `DeliveryRetryAttempts` is reached a subscription related with the link *http://inactive-listener* will be automatically unregistered.

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

For more information about configuring `'DeliveryRetryAttempts'` and other related attributes please refer to Customization

6.6.3 Registering new subscription

Sample subscription creation request can look like:

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

Where: * Name - is name of your subscription

- `Destination` - is reference to the system which is listening for events generated by Pod Manager
- `EventTypes` - events types which will be observed by particular subscriber. Any other events generated on Pod Manager side will not be delivered to this subscriber.
- `Context` - a client-supplied string that is stored with the event destination subscription.
- `Protocol` - the only one value currently supported is 'Redfish'
- `OriginResources` - 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.

6.7 Booting with iSCSI Out of Band feature

OOB iSCSI Boot is a way to boot node from iSCSI Target using BMC/BIOS.

6.7.1 Booting from iSCSI

To boot from Out of Band iSCSI user needs to have PSME and POD Manager at least in 2.1.3 version. The only requirement is that Network Device Function on REST API is available on Computer System and `"BootSourceOverrideMode"` property for this System is set to UEFI.

There are two ways to boot from remote target:

- Automatically during node composition (when requesting remote drive in allocation JSON). Additional Network Device Function will by patched with proper iSCSI data be POD Manager.
- 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 POD Manager it has to be done after `#ComposedNode.Assemble` and before `#ComposedNode.Reset` action

6.7.2 Setting Chap data

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

Note: `"Secret"` and `"MutualSecret"` properties won't be exposed on REST API (will be visible as NULL).

6.7.3 Setting boot data

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

`"BootSourceOverrideTarget"` will be different depending on allocation request:

- When not requesting `RemoteTarget`, `"BootSourceOverrideTarget"` will be set to `"Hdd".`
- When requesting `RemoteTarget`:
  - When Computer System used to compose the Node has Network Device Function, property `"BootSourceOverrideTarget"` will be set to `"RemoteDrive"` before assembly process.
  - When Computer System used to compose the Node does not have Network Device Function, property `"BootSourceOverrideTarget"` will be set to `"Pxe"` before 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 may 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.

6.7.4 Limitations

- Every manual change on `"BootSourceOverrideEnabled"` and `"BootSourceOverrideTarget"` on Computer System between `#ComposedNodeCollection.Allocate` and `#ComposedNode.Assemble` will be overwritten by POD Manager while #ComposedNode.Assemble task.
- Property `"BootSourceOverrideMode"` is not changed by POD Manager during allocation.
- Property `"BootSourceOverrideMode"` will be set to `“Legacy”` only while DeepDiscovery process.
- Currently POD Manager is able to boot using OOB iSCSI only when Computer System related with Node has Network Device Function and this System's `"BootSourceOverrideMode"` property is set to UEFI.

§

A Appendix

A.1 Database status across Pod Manager restarts

Pod Manager retains its asset inventory upon restart. To clear Pod Manager Database upon Pod Manager Startup user should execute a script that, by default, is installed with .deb package pod-manager:

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

Note: The Pod Manager retains asset inventory upon startup. This could lead the Pod Manager to display inaccurate information on its RESTful API interface. Intel recommended to clear the Pod Manager database to avoid the following:

- Startup functionality implications
  - All assets that were reported in the Available state under REST API before Pod Manager shut down are still exposed after a subsequent startup of Pod Manager
  - Pod Manager sets `InTest` state for all assets available after startup in Pod Manager and originating from `RMM/PSME/Storage` services
  - Pod Manager performs rediscovery of `RMM/PSME/Storage` services that will result in respective state changes of rediscovered assets
- Deep Discovery functionality implications
  - Assets successfully fetched via LUI service remain unchanged. The user MUST trigger the deep discovery process manually to refresh the Deep Discovery dataset
  - Pod Manager assumes that Deep Discovery should be triggered on all Computer Systems that are in `DeepInProgress` state upon startup
  - Pod Manager will not trigger Deep Discovery for Computer Systems that are in `"InTest"` state
  - If Deep Discovery is enabled, then Pod Manager 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 Pod Manager does not take into consideration any assets with InTest state
  - Composed Nodes in Allocating, Allocated or Assembling state during Pod Manager 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 Pod Manager

A.2 Source Code Compilation (Ubuntu)

A.2.1 Key components

Table 13. Source Code Compilation key Components

Component	Name	Version
Compiler	OpenJDK	1.8
Java EE	Full Profile	7.0
Libraries	`cling`	2.0.1
	`guava`	18.0
	`hibernate`	5.0.7.Final
	`jackson-annotations`	2.6.0 (from other jackson libraries), 2.6.3
	`jackson-core`	2.6.3
	`jackson-databind`	2.6.3
	`jackson-datatype-jsr310`	2.6.3
	`jackson-jaxrs-base`	2.6.3
	`jackson-jaxrs-json-provider`	2.6.3
	`jackson-module-jaxb-annotations`	2.6.3
	`javassist`	3.16.1-GA
	`liquibase`	3.5.2
	`modelmapper`	0.7.6
	`blueprints-core`	2.6.0
	`logback-classic`	1.1.1
	`logback-core`	1.1.1
	`commons-beanutils`	1.8.3
	`commons-collections`	3.2.2
	`commons-digester`	1.8.1
	`commons-io`	2.4
	`commons-lang`	3.6
	`commons-logging`	1.2
	`commons-validator`	1.5.0
Application server	WildFly	10.0.0.Beta1
Database	PostgreSQL	9.5

A.3 OpenJDK 1.8 Java Development Kit installation

At the start, ensure the OpenJDK PPA repository is correctly configured in your system. For details refer to Section 2.3.1, Adding PPA repository for OpenJDK (Ubuntu v14.04.4 only).

Install the OpenJDK 1.8 Java Development Kit package:

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

A.3.2 Verifying OpenJDK installation

Check default javac version by typing:

```
javac -version
```

If the above command does not show openjdk version 1.8, execute the following command to set Java defaults:

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

A.3.3 PostgreSQL 9.5 installation

Refer to Section 2.4 PostgreSQL 9.5 installation of this document.

A.3.4 Compilation of Pod Manager application

On the development machine, change directory to the Pod Manager source's root and use the gradle build tool* to compile the Pod Manager package:

```
./gradlew buildDeb
```

The above command creates the necessary DEB packages under source root directory which are used to install Pod Manager.

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 source root directory and execute:

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

A.3.5 Configuration of ISC DHCP server

The recommended ISC DHCP Server configuration is provided by Pod Manager Package and it is located under following path:

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

The default interfaces list on which ISC DHCP Server is listening is available in the following file

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

To change the interfaces list, change the following line to match the user configuration

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

§