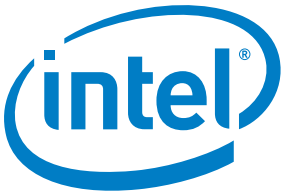


Intel® Rack Scale Design (Intel® RSD) POD Manager (PODM)

User Guide
Software v2.4

April 2019

Revision 001

Document Number: 608489-001

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and noninfringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications.

This document contains information on products, services, and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications, and roadmaps.

Copies of documents that have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2019 Intel Corporation. All rights reserved.

Contents

Figures

Tables

Revision History

Revision	Description	Date
001	Initial release	April 2019

§

1.0 Introduction

This document contains information about the installation and configuration of Software Release v2.4 of Intel® Rack Scale Design (Intel® RSD) POD Manager (PODM) and is referred to as PODM throughout this document.

1.1 Scope

This document contains information about the installation and configuration of Software Release version 2.4.0.498.0 of Intel® Rack Scale Design (Intel® RSD) Pod Manager called Pod Manager throughout this document.

1.2 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 S*pecification Software v2.4* on top of their hardware platform (refer to Table 2).

1.3 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). The 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.

1.4 Terminology

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

Table 1. Terminology

Term	Definition
ACL	Access Control List
BMC	Integrated Baseboard Management Controller
CA	Certificate Authority
CM	Control Module
cURL	Client URL
DHCP	Dynamic Host Configuration Protocol
DMTF	Distributed Management Task Force
GPG	GNU Privacy Guard
HTTP	Hypertext Transfer Protocol
IBL	Intel Business Link
iPXE	Preboot eXecution Environment
iSCSI	Internet Small Computer System Interface
IQN	iSCSI Qualified Name
ISVs	Independent Software Vendors
JSON	JavaScript Object Notation
LAG	Link Aggregation Group
LUI	Linux* Utility Image
MMP	Management Midplane
mTLS	mutual Transport Layer Security
NIC	Network Interface Card
NVMe-oF*	NVM Express over Fabrics*, for more information refer to *http://nvmexpress.org/resources/specifications*
OEM	Original Equipment Manufacturer
OOB	Out-of Band
PKCS #12	Personal Information Exchange Syntax Standard
POD	A physical collection of multiple racks
PODM	POD Manager
PPA	Personal Package Archives
PSME	Pooled System Management Engine
QoS	Quality of Service
RDMA	Remote Direct Memory Access
Redfish*	DMTF standard, for more information, refer to *https://www.dmtf.org/standards/redfish*
REST	Representational state transfer
RMM	Rack Management Module
RSA	Public key cryptosystem
RSS	RSD Storage Service
SB	Southbound API
SSDP	Simple Service Discovery Protocol
SSL	Secure Socket Layer
TFTP	Trivial File Transfer Protocol
TLS	Transport Layer Security
ToR	Top of Rack
UEFI	Unified Extensible Firmware Interface
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier
URL	Uniform Resource Locator

1.5 Reference Documents and Resources

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

Table 2. Reference Documents and Resources

Doc ID	Title	Location
608486	*Intel® Rack Scale Design (Intel® RSD) Pooled System Management Engine (PSME) User Guide Software v2.4*	***Note:*** *https://www.intel.com/content/www/us/en/architecture-and-technology/rack-scale-design/rack-scale-design-resources.html*
608487	*Intel® Rack Scale Design (Intel® RSD) Conformance and Software Reference Kit Getting Started Guide v2.4*	
608488	*Intel® Rack Scale Design (Intel® RSD) POD Manager (PODM) Release Notes Software v2.4*	
608489	*Intel® Rack Scale Design (Intel® RSD) POD Manager (PODM) User Guide Software v2.4*	
608490	*Intel® Rack Scale Design (Intel® RSD) Pooled System Management (PSME) Release Notes Software v2.4*	
608491	*Intel® Rack Scale Design Storage Services API Specification Software v2.4*	
608492	*Intel® Rack Scale Design (Intel® RSD) Architecture Specification Software v2.4*	
608493	*Intel® Rack Scale Design (Intel® RSD) Pod Manager (PODM) Representational State Transfer (REST) API Specification Software v2.4*	
608494	*Intel® Rack Scale Design (Intel® RSD) Rack Management Module (RMM) Representatinal State Transfer (REST) API Specification Software v2.4*	
608495	*Intel® Rack Scale Design (Intel® RSD) Generic Assets Management Interface (GAMI) API Specification v2.4*	
608496	*Intel® Rack Scale Design (Intel® RSD) Pooled System Management Engine (PSME) REST API Specification Software v2.4*	
608497	*Intel® Rack Scale Design (Intel® RSD) Conformance Test Suite (CTS) Release Notes*	
608298	*Field Programmable Gate Array (FPGA) over Fabric Protocol Architecture Specification*	*https://cdrdv2.intel.com/v1/dl/getContent/608298*
596167	*Intel® Rack Scale Design (Intel® RSD) for Cascade Lake Platform Firmware Extension Specification*	*https://cdrdv2.intel.com/v1/dl/getContent/596167*
N/A	*Key Words for Use in RFCs to Indicate Requirement Levels, March 1997*	*https://ietf.org/rfc/rfc2119.txt*
DSP0266	*Scalable Platforms Management API Specification v1.5.0*	*https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.5.0.pdf*
N/A	*NVM Express over Fabrics*	*http://nvmexpress.org/wp-content/uploads/NVMe_over_Fabrics_1_0_Gold_20160605-1.pdf*
N/A	*Get Docker CE for Ubuntu*	*https://docs.docker.com/install/linux/docker-ce/ubuntu/*
N/A	*How to download and install prebuilt OpenJDK packages*	*http://openjdk.java.net/install/*
N/A	*Official PostgreSQL charts*	*https://github.com/helm/charts/tree/master/stable/postgresql*
N/A	*Istio Connect, secure, control, and observe services*	*https://istio.io/*
N/A	*ceph-storage*	*https://github.com/rook/rook/blob/v0.9.3/Documentation/ceph-storage.md*
N/A	*Ceph Storage Quickstart*	*https://github.com/rook/rook/blob/v0.9.3/Documentation/ceph-quickstart.md*

Doc ID	Title	Location
N/A	*Block Storage*	*https://github.com/rook/rook/blob/v0.9.3/Documentation/ceph-block.md*
N/A	*Cleaning up a Cluster*	*https://github.com/rook/rook/blob/v0.9.3/Documentation/ceph-teardown.md*

NOTE: Copies of documents having an order number, referenced in this document, which cannot be accessed may be obtained by calling 1-800-548-4725 or by visiting *www.intel.com/design/literature.htm* and download a copy.

§

2.0 Pod Manager (PODM) Build and Deployment

Steps necessary to build PODM from source code and deploy it on Kubernetes cluster.

2.1 Prerequisites

Components and tools are necessary for PODM deployment.

2.1.1 Operating System

The natural development environment for the PODM is Ubuntu* v16.04 (server distro).

Note: Any snippets available in this user guide works with Ubuntu OS, but there is no guarantee these snippets will work on other operating systems.

2.1.2 Java

Make sure that Java compiler is available:

Important: The PODM requires OpenJdk v1.8.x.

```
javac --version
```

sample output would be:

```
javac 1.8.0_161
```

If the compiler is not installed, refer to Table 2, *How to download and install prebuilt OpenJDK packages*.

2.1.3 Docker*

Make sure that Docker* is installed (`>= 18.02.0-ce`). Refer to Table 2 to *Install Docker CE*.

```
docker version
```

Sample output:

```
Client:
 Version:   18.02.0-ce
 API version:   1.36
 Go version:    go1.9.3
 Git commit:    fc4de44
 Built: Wed Feb  7 21:16:33 2018
 OS/Arch:   linux/amd64
 Experimental:  false
 Orchestrator:  swarm

Server:
 Engine:
  Version:  18.02.0-ce
  API version:  1.36 (minimum version 1.12)
  Go version:   go1.9.3
  Git commit:   fc4de44
  Built:    Wed Feb  7 21:15:05 2018
  OS/Arch:  linux/amd64
  Experimental: false
```

2.1.4 Kubernetes*

The PODM application is designed to be installed on the Kubernetes* cluster. If the instance of the Kubernetes* cluster is not running, refer to Appendix A, Kubernetes* (One Node Cluster) Installation.

2.1.5 Private Docker* Registry

The Kubernetes* cluster should have access to the Docker* repository where all required PODM binary artifacts are exposed. To use the PODM, provide the private Docker* registry. To run private registry (in simplest non production mode) follow these steps:

1. Login to the Kubernetes target node:
```
$ ssh user@targetnode
```
2. Run the registry:
```
$ docker run -d -p 5000:5000 --restart=always --name registry registry:2
```
The private registry should now be running and exposing the API under `localhost:5000`.
3. Create an `SSH` tunnel between the machine where the PODM Sources and `targetNode` are kept:
```
$ ssh -fN -L 5000:localhost:5000 vagrant@targetnode
```
4. Verify the connection between the host and `targetNode`:
```
$ curl localhost:5000/v2/_catalog
```
5. Sample result:
```
{
  "repositories": []
}
```

2.1.6 Database

The PODM application is designed to use the `PostgreSQL` database.

Note: The `PostgreSQL` is not included with the PODM deployment. `PostgreSQL` must be installed and configured on the Kubernetes* cluster by the user. It is recommended to use official `PostgreSQL` charts, refer to Table 2.

Important: It is required to install `PostgreSQL` charts on the Kubernetes* cluster using the `podm-db` release name. For example:
```
helm install --name podm-db stable/postgresql
```

2.1.6.1 Database Persistence

For information about configuring optional "Persistent Volume" for `PostgreSQL`, refer to Appendix C, Persistent Volumes (PV). Enable persistence for the `PostgreSQL` by installing charts with the following command:
```
helm install --name podm-db --set persistence.enabled=true stable/postgresql
```

2.2 Building Pod Manager

The assumption is that source code exists in the PODM directory. The first time build, and compilation of the PODM sources takes a bit longer because a set of external dependencies are downloaded.

Make sure build machine has access to the Internet and run:
```
cd PODM
./gradlew build
```

2.3 Building PODM Docker* Images

The PODM is targeted to run on the Kubernetes* cluster. To deploy the PODM on Kubernetes*, pack the PODM application into a set of Docker* images.

```
cd PODM
./buildAllImages.sh
```

After packing has completed, all PODM images should be available in local Docker*:

```
docker images
```

Sample output:

```
REPOSITORY            TAG                 IMAGE ID            CREATED             SIZE
podm-dhcp             1.0-SNAPSHOT        5d71692c8fd8        3 minutes ago       59.4MB
resource-manager      1.0-SNAPSHOT        0ce62b70b037        3 minutes ago       172MB
node-composer         1.0-SNAPSHOT        c6d3024831d0        3 minutes ago       161MB
service-detector      1.0-SNAPSHOT        53912e1e20e5        3 minutes ago       140MB
aaa-service           1.0-SNAPSHOT        e59621fc0e0f        3 minutes ago       151MB
podm-gateway          1.0-SNAPSHOT        4ed6ed172a40        3 minutes ago       128MB
service-registry      1.0-SNAPSHOT        b8f29e2b71e6        3 minutes ago       136MB
event-service         1.0-SNAPSHOT        e4fbb0a241a3        3 minutes ago       127MB
```

2.4 Pushing PODM Images to the Private Docker* Registry

Push images built in the previous step to private Docker Registry.

```
cd PODM
./pushAllDockerImages.sh
```

Verify the PODM images are exposed on the registry:

```
$ curl localhost:5000/v2/_catalog
```

Sample result:

```
{
    "repositories": [
        "aaa-service",
        "event-service",
        "node-composer",
        "podm-dhcp",
        "podm-gateway",
        "resource-manager",
        "service-detector",
        "service-registry"
    ]
}
```

2.5 Building Helm Charts

Build the PODM Helm charts by running following command in PODM source code directory.

```
./createHelmChart.sh
```

`pod-manager-0.99.tgz file` should be created under the PODM directory. Below is the sample output of the above command:

```
Hang tight while we grab the latest from your chart repositories...
Update Complete. ⎈Happy Helming!⎈
Saving 4 charts
Deleting outdated charts
Successfully packaged chart and saved it to PODM/pod-manager-0.99.tgz
```

2.6 Deploying PODM

The PODM application can be deployed by running the following command:

```
helm install --name podm --set global.registry=localhost:5000/ pod-manager-0.99.tgz
```

Verify the status of the PODM deployment:

```
helm status podm
```

Sample output:

```
LAST DEPLOYED: Wed Apr  4 15:10:55 2018
NAMESPACE: default
STATUS: DEPLOYED

RESOURCES:
==> v1/Service
NAME                      TYPE       CLUSTER-IP      EXTERNAL-IP  PORT(S)          AGE
mypodm-podm-gateway       NodePort   172.20.55.111   <none>       8080:31544/TCP   15s
mypodm-postgres           ClusterIP  172.20.110.148  <none>       5432/TCP         15s
mypodm-service-registry   ClusterIP  172.20.96.30    <none>       80/TCP           15s

==> v1beta2/Deployment
NAME                      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
mypodm-podm-gateway       1        1        1           1          15s
mypodm-postgres           1        1        1           1          15s
mypodm-resource-manager   1        1        1           0          15s
mypodm-service-registry   1        1        1           1          15s

==> v1/Pod(related)
NAME                                    READY  STATUS   RESTARTS  AGE
mypodm-podm-gateway-59f4f7974f-wsznb    1/1    Running  0         15s
mypodm-postgres-5fff75c596-l5nm9        1/1    Running  0         15s
mypodm-resource-manager-5965c6b785-jrqmh  0/1  Running  0         15s
mypodm-service-registry-6977bc747-nwh8m   1/1  Running  0         15s


NOTES:
Enjoy!
```

2.7 PODM Redfish API

Run the following command to determine the Kubernetes* cluster IP:

```
kubectl cluster-info
```

Sample output:

```
Kubernetes master is running at https://172.28.128.10:6443
KubeDNS is running at https://172.28.128.10:6443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy
```

Reported IP address: **172.28.128.10** is an address of the external IP of the Kubernetes* cluster and reported port: 31544 is a port where the PODM application is exposed. In this example, the URI of the Redfish API of PODM application will be `targetNode:31544/redfish/v1`. Send requests against this API:

```
curl targetNode:31544/redfish/v1
```

Sample output:


```
{
  "@odata.context": "/redfish/v1/$metadata/#ServiceRoot",
  "@odata.id": "/redfish/v1",
  "@odata.type": "#ServiceRoot.v1_1_1.ServiceRoot",
  "Id": "serviceRoot",
  "Name": "Instance ID: mypodm-resource-manager-5965c6b785-jrqmh",
  "Description": "desc",
  "RedfishVersion": "1.5.0",
  "UUID": "34e60059-0d9a-44ee-9e57-09f9bcccf40f",
  "Chassis": {
    "@odata.id": "/redfish/v1/Chassis"
  },
  "Systems": {
    "@odata.id": "/redfish/v1/Systems"
  },
  "Managers": {
    "@odata.id": "/redfish/v1/Managers"
  },
  "Fabrics": {
    "@odata.id": "/redfish/v1/Fabrics"
  },
  "StorageServices": {
    "@odata.id": "/redfish/v1/StorageServices"
  },
  "TaskService": {
    "@odata.id": "/redfish/v1/TaskService"
  },
  "Links": {
    "Oem": {}
  },
  "Oem": {
    "Intel_RackScale": {
      "@odata.type": "#Intel.Oem.ServiceRoot",
      "ApiVersion": "2.4.0",
      "EthernetSwitches": {
        "@odata.id": "/redfish/v1/EthernetSwitches"
  },
      "TelemetryService": {
        "@odata.id": "/redfish/v1/Oem/Intel_RackScale/TelemetryService"
      }
    }
  }
}
```


§

3.0 Pod Manager Configuration

This chapter provides information on the configuration of the PODM behavior.

3.1 Configuring Properties for Spring Boot-Based Applications

Most of RSD pods contain Spring Boot*-based applications. Properties for these applications (which in non-containerized environments are usually placed in application.properties or `application.yml` files) can be set in `values.yaml` in section `applicationProperties`.

- Example of changing application server port in `values.yaml`:

```
applicationProperties:
  server:
    port: 18999
```

- It can also be done during the installation of the helm chart:

```
helm install --name podm \
--set node-composer.applicationProperties.server.port=18999,\
global.registry=localhost:5000/ pod-manager-0.99.tgz
```

- Configuring properties after deployment:

```
kubectl edit configmap {CONFIG_NAME}
```

  `ConfigMaps` names can be displayed using the command: `kubectl get configmap`. After every change, restart the container to upload new the `ConfigMap`. Every properties field should be set in `data.application.yml`:

  Example field `allocation.reserved-vlan-ids=1,170,4088,4091,4094` should be put in config map this way:

```
data:
  application.yml: |-
    allocation:
      reserved-vlan-ids: 1,170,4088,4091,4094
```

- Another way is to provide a file with overrides during installation of helm chart:

```
new-values.yaml:
node-composer:
  applicationProperties:
    server:
      port: 18999
```

  Deployment command:

```
helm install --name podm global.registry=localhost:5000/ \
-f new-values.yaml pod-manager-0.99.tgz
```

3.2 Discovery Configuration

There are three available mechanisms to discover new services and resources: DHCP, SSDP, and registration of services using endpoints exposed by the REST API. By default, all three mechanisms are enabled, and the same service can be detected by all mechanisms.

Note: It is highly recommended that the user use either one of the mechanisms to discover RSD resources.

Important: Discovery interval is by default set at 60 seconds. It is the time between the last completed discovery and the start of a new one.

Important: If a new resource is created, the resource needs to be discovered by the PODM before it is available for other actions, such as attaching Volume.

During the deployment step, set the discovery interval by adding the variable

"node-composer.applicationProperties.discovery.interval-seconds" into the helm install command.

Installing PODM with different discovery interval:

```
helm install --name podm \
--set node-composer.applicationProperties.discovery.interval-seconds=60,\
global.registry=localhost:5000/ pod-manager-0.99.tgz
```

3.3 Configuring Northbound Communication Security

This section describes the process of configuring TLS including generation of certificates, choosing secure ciphersuites and promotes good practices in key management. In addition, it provides guidelines to user management and authorization using both Basic Access Authentication and Redfish Sessions.

3.3.1 TLS Configuration

This section describes a sample configuration of TLS authentication for the PODM Gateway application. PODM Gateway is a single entry point for any REST requests incoming to the PODM application stack. To configure one way TLS authentication for the PODM Gateway, provide a Java Key Store (JKS) containing required certs. This JKS is stored in K8s secret which is finally consumed by containers running inside the K8s cluster.

Generating certificate:

Example of creating a simplified development-only chain of certificates to be used by PODM server and its client.

```
# generate keypair for CA
keytool -alias podmca \
        -dname "CN=podmCa, OU=RSD, O=Intel, L=Gdansk, S=Pomerania, C=PL" \
        -keystore podmca.keystore -storetype pkcs12 -storepass podmpodm \
        -genkeypair -keyalg "RSA" -validity 3000 -sigalg SHA384withRSA \
        -keysize 4096 -keypass podmpodm -ext BC:critical="ca:true,pathlen:0"
# export the podm CA cert (self signed)
keytool -exportcert -rfc -keystore podmca.keystore -alias podmca \
        -storepass podmpodm > podmca.pem
# generate keypair for Podm Developer Server
keytool -alias podmserver \
        -dname "CN=Podm Development Server, OU=RSD, O=Intel,\
                L=Gdansk, S=Pomerania, C=PL" \
        -keystore podmserver.keystore -storepass podmpodm -genkeypair \
        -validity 360 -keyalg "RSA" -sigalg SHA384withRSA -keysize 4096 \
        -keypass podmpodm -storetype pkcs12
# sign Podm Developer Server with CA
keytool -alias podmserver \
        -certreq -keystore podmserver.keystore -storepass podmpodm \
        -ext SAN=dns:localhost,dns:dev.podmserver.net | \
        keytool -alias podmca -keystore podmca.keystore -storepass podmpodm \
                -gencert -ext SAN=dns:localhost,dns:dev.podmserver.net \
                -ext ku:c=dig,keyEncipherment -rfc > podmserver.pem
```

Tip: Notice the Subject Alternative Name (SAN) extension provided during subsequent operations. SAN extension plays a crucial role in TLS hostname verification, which is a server identity check.

The check works by verifying that the dnsName in the subjectAltName field of the certificate sent by the server, matches the host portion of the URL used to make the request. Make sure to include the server's hostnames/IPs in that part.

Next, import both the CA certificate and your signed certificate into the keystore.

```
keytool -import -keystore podmserver.keystore -file podmca.pem -alias podmCA \
        -noprompt -trustcacerts -storepass podmpodm
keytool -import -keystore podmserver.keystore -file podmserver.pem \
        -alias podmserver -storepass podmpodm
```

Important: The client that is willing to setup a TLS connection with the PODM server, has to import a certificate of CA that signed the PODM server certificate into its truststore.

The keystore is now prepared to be handed over to the PODM application. Use the K8s secret as the provider.

- K8s secret generation:

```
kubectl create secret generic nb-security-config \
  --from-file=server.ssl.key-store=/absolute/path/to/{jks-name} \
  --from-literal=server.ssl.key-store-password={keypass} \
  --from-literal=server.ssl.key-alias={podm-gateway} \
  --from-literal=server.ssl.key-password={storepass} \
  --from-literal=server.ssl.enabled=true
```

Note: During K8s secret generation, it is recommended to specify the used `ciphersuite` and protocol. This can be done by adding following parameters.

- Specifying the ciphers and protocol:

```
  ...
  --from-literal=server.ssl.ciphers={ciphersuite} \
  --from-literal=server.ssl.protocol={your_preferred_TLS_version}
```

Note: While specifying the `ciphersuite` (specify a comma separated list of `ciphersuites`), follow common security guidelines as specified in JDK documentation (refer to `Table 2`) or fall back to the recommendation in the following table.

Table 3. Recommended ciphersuites

`TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
`TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384`
`TLS_DHE_DSS_WITH_AES_256_CBC_SHA`
`TLS_DHE_RSA_WITH_AES_256_CBC_SHA`
`TLS_RSA_WITH_AES_256_CBC_SHA`
`TLS_DH_DSS_WITH_AES_256_CBC_SHA`
`TLS_DH_RSA_WITH_AES_256_CBC_SHA`

Enhance the regular PODM deployment command with an additional flag:

```
--set podm-gateway.northbound_security.enabled=true
```

After applying the above modification, the deployment command would look like:

```
helm install --name podm \
--set podm-gateway.northbound security.enabled=true,\
global.registry=localhost:5000/ pod-manager-0.99.tgz
```

Once all is in place, the PODM listens on an `SSL` connector.

Consuming service on an SSL connector:

```
curl -v --cacert podmca.pem -u admin:admin \
-X HEAD https://localhost:8888/redfish/v1/SessionService
Warning: Setting custom HTTP method to HEAD with -X/--request may not work the
Warning: way you want. Consider using -I/--head instead.
*   Trying 127.0.0.1...
* TCP NODELAY set
* Connected to localhost (127.0.0.1) port 8888 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* successfully set certificate verify locations:
*   CAfile: podmca.pem
  CApath: /etc/ssl/certs
* (304) (OUT), TLS handshake, Client hello (1):
* (304) (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES256-GCM-SHA384
* ALPN, server did not agree to a protocol
* Server certificate:
*  subject: C=PL; ST=Pomerania; L=Gdansk; O=Intel; OU=RSD; CN=Podm Development Server
*  start date: Mar 14 08:15:27 2019 GMT
*  expire date: Jun 12 08:15:27 2019 GMT
*  subjectAltName: host "localhost" matched cert's "localhost"
*  issuer: C=PL; ST=Pomerania; L=Gdansk; O=Intel; OU=RSD; CN=podmCa
*  SSL certificate verify ok.
* Server auth using Basic with user 'admin'
> HEAD /redfish/v1/SessionService HTTP/1.1
> Host: localhost:8888
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.58.0
> Accept: */*
>
< HTTP/1.1 200
< Date: Thu, 14 Mar 2019 08:42:09 GMT
< Content-Type: application/json;charset=UTF-8
< Content-Length: 0
<
* Connection #0 to host localhost left intact
```

3.3.2 Key and Certificate Management

It is important to follow best security practices when it comes to the Public Key Infrastructure (PKI) because the PODM does not explicitly enforce a way to manage it.

Keys - it demands them to be provided by means of a cloud infrastructure.

Note: It is up to the end user to generate strong keypairs and accurately generate/manage certificates.

It is recommended to set a short validity period for end keys and rotate them once they expire. If you are creating your own CA, it may have a much longer validity time.

Currently, this has to be done manually and requires reinstallation of PODM deployment (maintenance window).

Go to GitHub and download `Key Management Cheat Sheet.md` using the following URL:

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Key_Management_Cheat_Sheet.md

3.3.3 PODM Authentication

The PODM follows the Redfish security guidelines and supports both Basic Access Authentication and Redfish Session tokens to authenticate its clients. Every endpoint beside `/redfish/v1` requires explicit authentication. Access to `/redfish/v1` is possible using both HTTP and HTTPS endpoint.

For reference, refer to Table 2, *Redfish Scalable Platforms Management API Specification*.

Note: Configuring TLS connection alongside any authentication mechanism is crucial. If TLS is configured, then HTTP endpoint provides access only to `/redfish/v1` and redirects all other requests to the HTTPS endpoint.

3.3.3.1 Basic Access Authentication

To authorize using Basic Access Authentication (BA), attach Authorization header to each request. The header takes the following form:

Authorization: Basic <`encoded_credentials`>

Credentials take the form of a `Base64` encoded concatenation of login and password.

Obtaining encoded credentials:

```
$ echo -e "admin:admin" | base64
YWRtaW46YWRtaW4K
```

3.3.3.2 Users Configuration

Manage users employing the RF `AccountService` available at `/redfish/v1/AccountService`.

Warning: The installation contains a predefined admin user (password admin). Modify its password or add a new user and remove the predefined one after installation.

- Creating New User

  To create a new user perform an authorized `POST` operation upon the `/redfish/v1/AccountService` endpoint.

```
$ curl -u admin:admin -v -H 'Content-Type: application/json' \
       -H 'Accept-Type: application/json' -d @create_account.json \
       -X POST http://localhost:8080/redfish/v1/AccountService/Accounts
```

- New user payload:

```
{
   "UserName": "username",
   "Password": "Password!1",
   "RoleId": "Administrator"
 }
```

Note: Provided username cannot be blank and cannot collide with an existing user. Configurable password policies apply to password (size, strength). The `RoleId` has to be an existing role.

- Changing User Password:

  To change/update the password, perform an authenticated `PATCH` request upon `/redfish/v1/AccountService/Accounts/{username}` endpoint.

```
$ curl -u admin:admin -v -H 'Content-Type: application/json' \
       -H 'Accept-Type: application/json' -d '{"Password" : "new password"}' \
       -X PATCH http://localhost:8080/redfish/v1/AccountService/Accounts/username
```

- Removing user:

To remove a user perform an authenticated `DELETE` request upon `/redfish/v1/AccountService/Accounts/{username}` endpoint.

```
$ curl -u admin:admin -v -H 'Content-Type: application/json' \
       -H 'Accept-Type: application/json' \
       -X DELETE http://localhost:8080/redfish/v1/AccountService/Accounts/username
```

- Password Policies:

  Configurable password policies are applied to user passwords. Refer to Section, 3.3.4.2, Finetuning Authentication for configuration parameters which apply to password policy handling.

3.3.4 Authentication with Redfish Sessions

Session authentication allows the user to perform secured operations employing a dedicated authentication token. The token has to be provided in the `X-Auth-Token` header during each request.

To obtain a new token, perform a `POST` operation upon `SessionService's` Sessions collection providing credentials within the operation body.

Note: The following examples assume the Gateway is configured with TLS.

3.3.4.1 Logging in

To authorize using an RF Session, first acquire a session token that will be propagated in all subsequent requests.

- Obtaining RF Session token:

```
curl -v -H 'Content-Type: application/json' -H 'Accept-Type: application/json' \
     -X POST -d @valid_credentials.json \
     http://localhost:8080/redfish/v1/SessionService/Sessions
```

- Credentials payload:

```
{
  "UserName": "admin",
  "Password": "admin"
}
```

  The authentication server validates credentials provided during the call and returns a success response containing the `X-Auth-Token` and `Location` of a freshly created session.

- Successfully acquiring new token:

```
< HTTP/1.1 200
< X-Auth-Token: b981c650-b553-4857-8c98-f05754ef7cd9
< Location: /redfish/v1/SessionService/Sessions/402100c3-3dd2-48d4-92ba-
7db53fc5ce68
```

Secured conversation with tokens

To convey dialogue upon secured resource, it is required to attach the `X-Auth-Token` to each consecutive call.

Passing authentication token to a secured call:

```
$ curl -vv -H 'Content-Type: application/json' \
       -H 'X-Auth-Token: b981c650-b553-4857-8c98-f05754ef7cd9' \
       -X GET https://localhost:8080/redfish/v1/AccountService/Accounts
```

The session will be kept alive during each user action taking place (it will be prolonged by the session-timeout value). This way Username and Password have to be specified during token acquisition.

Logging out

To log out one has to perform a DELETE operation upon his session URI (which was returned within the Location header during session token acquisition).

Automatic session invalidation

Sessions will be automatically destroyed if the user does not perform any operations within a timespan extending the session timeout.

3.3.4.2 Finetuning Authentication

Currently, the authentication module supports the following parameters:

- `aaa-config.password-policy.minLength` - minimal password length [default `4`]
- `aaa-config.password-policy.maxLength` - maximal password length [default `30`]
- `aaa-config.session-timeout` - session idle time in seconds [default `600`]

The parameters are optional and can be specified to override the defaults.

Overriding parameters during installation of PODM:

```
helm install --name podm \
--set aaa-service.accessVerifier.minPasswordLength=4,\
global.registry=localhost:5000/ pod-manager-0.99.tgz
```

3.4 Configuring Southbound Communication Security

Two way TLS (MTLS) should be configured for PODM southbound communication.

To provide configuration for secure communication, you have to create a Kubernetes secret containing both keystore and truststore that will be used for setting up an MTLS connection.

```
kubectl create secret generic sb-security-config \
--from-file=TRUSTSTORE_PATH=myTrustStore  \
--from-literal=TRUSTSTORE_PASSWORD=myTrustStorePassword \
--from-file=KEYSTORE_PATH=myKeyStore \
--from-literal=KEYSTORE_PASSWORD=myKeystorePassword \
--from-literal=KEYSTORE_ALIAS=keyAliasToUse \
--from-literal=SOUTHBOUNDCONFIG_BASICAUTHTOKEN=basicAuthTokenToUse
```

To generate keys and certificates that have to be imported into the keystore/truststore perform a procedure similar to the one described in the Configuring northbound security section. Generate a dedicated CA for southbound communication or share the one used for the northbound connector. The only difference is that the root certificate of trusted southbound devices (such as in self signed CA) has to be imported into the truststore for MTLS to work properly.

Important: Provided `myTrustStore` and `myKeyStore` files must be in JKS repositories.

Important: Name of the secret: `sb-security-config` cannot be changed because other definitions of PODM application stack deployment relies on it.

Important: Specified `KEYSTORE_ALIAS` has to be contained in the provided JKS repository (`myKeyStore`).

Note: It is recommended to specify the used `ciphersuite` and protocol. As an option, the aforementioned secret generation can be extended with additional parameters.

- Specifying the ciphers and protocol:

```
  ...
--from-literal=server.ssl.ciphersuite={ciphersuite} \
--from-literal=server.ssl.protocol={your_preferred_TLS_version}
```

While specifying the `ciphersuite`, by providing a comma separated list of `ciphersuites`, follow common security guidelines, Refer to Table 2, *How to download and install prebuilt OpenJDK packages*.

Add an additional flag to the regular PODM deployment to enable Two way TLS:

```
--set global.southbound_security.enabled=true
```

- After the above modification deployment command would look like that:

```
helm install \
  --name podm \
  --set global.southbound_security.enabled=true,global.registry=localhost:5000/ \
  pod-manager-0.99.tgz
```

3.4.1 Configuring Southbound Authentication

Redfish supports authentication through Basic Authentication and/or Redfish Sessions. Currently, PODM supports authenticating to its southbound clients by means of Basic Authentication. Redfish Sessions are only supported for northbound clients. While MTLS could be used both for encryption and authentication, Redfish still demands the authentication through additional challenges such as in Basic Authentication.

The credentials that will be used by the PODM for southbound connections need to be provided within the `'sb-security-config'` Kubernetes secret.

- Specifying southbound credentials during `'sb-security-config'` secret creation:

```
--from-literal=SOUTHBOUNDCONFIG_BASICAUTHTOKEN=basicAuthTokenToUse
```

Note: Credentials need to be provided in a standard Basic Authentication format but without the 'Basic' prefix.

- Obtaining encoded credentials:

```
$ echo -e "admin:admin" | base64
 YWRtaW46YWRtaW4K
```

§

4.0 Configuration and Monitoring

The PODM application stack exposes a different set of capabilities related to configuration and monitoring. Selected components of PODM expose `REST` endpoints that provide several options to adjust settings and monitor the state of your application in runtime.

Since major parts of the PODM application stack have been implemented based on Spring Boot framework, configuration and monitoring capabilities come from the Spring Actuator extension, Refer to *Spring Boot Actuator: Production-ready features* in Table 2.

The Rest endpoint that exposes configuration and monitoring capabilities is the same for each PODM component and looks like:

```
service-uri/actuator
```

4.1 Exposed Endpoints

This section describes configuration and monitoring endpoints provided by PODM (based on Spring Boot framework).

4.1.1 @GET /actuator/health

Shows application health information.

4.1.2 @GET /actuator/configprops

Displays a collated list of all properties

4.1.3 @GET @POST @DELETE /actuator/env

Exposes/adds/deletes environment properties

4.1.4 @GET /actuator/env/{toMatch}

Exposes particular property where {toMatch} is property index.

4.1.5 @GET /actuator/loggers

Shows the configuration of loggers in the application.

4.1.6 @GET @POST /actuator/loggers/{name}

Shows and modifies the configuration of the particular logger.

4.1.7 @GET /actuator/threaddump

Performs a thread dump

4.1.8 @GET /actuator/prometheus

Exposes metrics in a format that can be scraped by a Prometheus server.

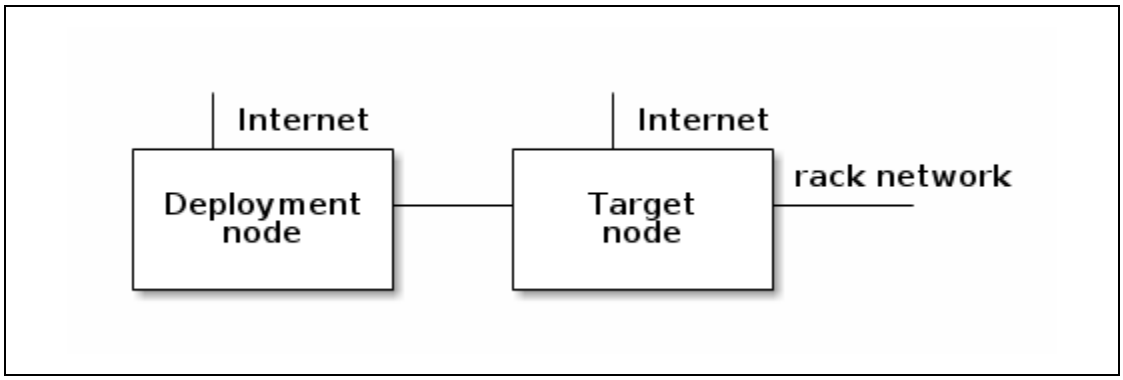
4.1.9 @GET /actuator/httptrace

Displays HTTP trace information (by default, the last 100 HTTP request-response exchanges).

§

Appendix A Kubernetes* (One Node Cluster) Installation

Figure 1. Deployment and Target Nodes



Note: This user guide assumes that the deployment node and target node have Internet connectivity without proxy. Target node should have at least 6 GB of RAM.

A.1 Target Node Preconfiguration

Note: This user guide assumes that:

- the user used for deployment is podm - this user should exist on the target node and have sudo access
- the external IP address of the target node is IP:**192.168.1.1**
- the internal IP address of the target node is IP:**10.3.0.1**

A.1.1 Key Management

Note: This guide assumes that public key is located under `/home/some_user/keys/podm.key.pub` If the key does not exist, generate a pair of public and private keys using:

```
ssh-keygen -t rsa -b 4096 -f podm.key -P ""
```

Copy the public key from the deployment node to target the node:

```
ssh-copy-id -i /home/some_user/keys/podm.key.pub podm@192.168.1.1
```

A.1.2 Configure passwordless sudo for *podm* user

Connect to the target node, using SSH:

```
echo "podm ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/podm
sudo chmod 0440 /etc/sudoers.d/podm
```

A.1.3 Disable Swap on Target Node

```
sudo swapoff -a
sudo sed -i '/ swap / s/^/#/' /etc/fstab
```

A.2 Deployment Node Configuration

A.2.1 Download and untar Kismatic Distribution

```
wget https://github.com/apprenda/kismatic/releases/download/v1.11.0/\
kismatic-v1.11.0-linux-amd64.tar.gz

mkdir kismatic
tar zxf kismatic-v1.11.0-linux-amd64.tar.gz --directory kismatic
cd kismatic
```

A.2.2 Create Cluster Installation Plan with Following Options

```
./kismatic install plan --plan-file single-node-plan.yml
```

```
Plan your Kubernetes cluster:
=> Number of etcd nodes [3]: 1
=> Number of master nodes [2]: 1
=> Number of worker nodes [3]: 1
=> Number of ingress nodes (optional, set to 0 if not required) [2]: 0
=> Number of storage nodes (optional, set to 0 if not required) [0]: 0
=> Number of existing NFS volumes to be attached [0]: 0

Generating installation plan file template with:
- 1 etcd nodes
- 1 master nodes
- 1 worker nodes
- 0 ingress nodes
- 1 storage nodes
- 0 nfs volumes

Wrote plan file template to "single-node-plan.yml"
Edit the plan file to further describe your cluster. Once ready, execute the "install
validate" command to proceed.
```

A.2.3 Edit Generated Plan using Following Configurations

This section describes possible modifications to the plan generated earlier by Kismatic.

A.2.3.1 SSH Access Configuration

Note: This User Guide assumes that the private key is located under `/home/some_user/keys/podm.key`.

```
ssh:
    user: podm

    ssh key: /home/some user/keys/podm.key
    ssh_port: 22
```

A.2.3.2 Etcd Nodes are the Ones that Run the etcd Distributed Key-Value Database

```
etcd:
  expected_count: 1

  nodes:
  - host: "abc"
    ip: "192.168.1.1"
    internalip: "10.3.0.1"
```

A.2.3.3 Master Nodes are the Ones that Run the Kubernetes* Control Plane Components

```
master:
  expected_count: 1
  load balanced fqdn: "192.168.1.1"

  load balanced short name: "192.168.1.1"
  nodes:
  - host: "abc"
    ip: "192.168.1.1"
    internalip: "10.3.0.1"
    labels: {}
```

A.2.3.4 Worker Nodes are the Ones that will Run your Workloads on the Cluster

```
worker:
  expected count: 1
  nodes:
  - host: "abc"
    ip: "192.168.1.1"
    internalip: "10.3.0.1"
    labels: {}
```

A.3 Kubernetes* Installation

This section provides information on how to install Kubernets on a target node and make `kubectl` and helm tools available for further use.

A.3.1 To install Kubernetes* on Target Node Run

```
./kismatic install apply --plan-file single-node-plan.yml
```

Installation process generates kubconfig file: generated\config. Generated configuration will be required for tools like `kubectl` or helm (both of them are part of the kismatic distribution).

Note: In case anything goes wrong with the K8S installation, kismatic comes with an option to reset any changes made to the target hosts by 'kismatic apply':

```
./kismatic reset --force
```

A.3.2 To Make kubectl and helm Tools Available for Further Usages

```
sudo cp ./{helm,kubectl} /usr/local/bin
```

§

Appendix B Security Considerations

This appendix contains security recommendations concerning user configuration, password policy, encryption of configuration files, and securing PODM internal communication.

B.1 Configuring Default User

It is recommended to preconfigure the default user.

```
aaa-config:
  default-user:
    name: admin
    password: admin
    role: Administrator
```

- `aaa-config.default-user.name` - username [default admin]
- `aaa-config.default-user.password` - password [default admin]
- `aaa-config.default-user.role` - rolename [default Administrator]

The parameters should be provided during helm install or through `'podm-aaa-service-config'` Kubernetes* `ConfigMap`.

B.2 Configuring Available Password Policies

It is recommended to preconfigure available password policies that will be enforced upon PODM user passwords.

Currently, the authentication module supports the following parameters:

- `aaa-config.password-policy.minLength` - minimal password length [default `4`]
- `aaa-config.password-policy.maxLength` - maximal password length [default `20`]
- `aaa-config.password-policy.noWhitespacesAllowed` - reject whitespaces [default `false`]
- `aaa-config.password-policy.noRepeatedCharsAllowed` - reject repeated characters [default `false`]
- `aaa-config.password-policy.lowercaseCharactersAmount` - minimal lowercase characters amount [default `1`]
- `aaa-config.password-policy.uppercaseCharactersAmount` - minimal uppercase characters amount [default `0`]
- `aaa-config.password-policy.digitCharactersAmount` - minimal digit characters amount [default `0`]
- `aaa-config.password-policy.checkForUsernameInPassword` - reject username as part of password [default `false`]

Optional paramerters can be specified during the helm install or through the `'podm-aaa-service-config'` Kubernetes `ConfigMap`.

B.3 Encrypting Data at Rest

PODM services rely on configuration stored within the environment.

Warning: Embedded defaults are usually meant for development purposes only. Production environment should rely on cloud specific means to configure deployed services in, eg. Kubernetes `ConfigMaps`.

It is advisable to encrypt the key value store used alongside Kubernetes* to export the configuration to deployed applications. Refer to Table 2, *Encrypting Secret Data at Rest* for instructions.

B.4 Encrypting Communication Between Internal Components

It is recommended to protect the communication between PODM services internally that, by default, uses HTTP communication. One way to achieve this is by *incorporating Istio* service* mesh solution (refer to Table 2, Istio Connect, secure, control, and observe services). That has mutual TLS (mTLS) authentication support as one of its many features.

Note: Integration with Istio may require additional work and code changes. Should that be out of the scope, there is still a fallback solution, such as a secure network overlay of your choice.

§

Appendix C Persistent Volumes (PV)

In the case of multinode deployments, selected PODM features might require the existence of PV. This guide provides examples of PV configuration; all of them have been built on top of *rook-ceph*.

C.1 Rook

Rook is an open source cloud-native storage orchestrator for Kubernetes*, providing the platform, framework, and support for a diverse set of storage solutions to natively integrate with cloud-native environments.

C.1.1 Ceph - Rook's Storage Provider

Ceph is a highly scalable distributed storage solution for block storage, object storage, and shared file systems with years of production deployments. More info about Ceph Storage can be found in Table 2, *ceph-storage*.

C.1.1.1 Ceph's Block Storage

Block storage allows you to mount storage to **a single pod**.

C.2 Ceph Cluster Installation

Tip: All manifests required `for Rook-Ceph installation/configuration` have been attached *here*.

Deploy the Rook Operator:

```
kubectl create -f operator.yaml
```

Verify the `rook-ceph-operator`, `rook-ceph-agent`, and `rook-discover` pods are in the Running state before proceeding.

```
kubectl -n rook-ceph-system get pod
```

Create a Rook Cluster:

```
kubectl create -f cluster.yaml
```

Use `kubectl` to list pods in the `rook-ceph` namespace:

```
kubectl -n rook-ceph get pod
```

You should be able to see the following pods once they are all running (it can take several minutes). The number of pods will depend on the number of nodes in the cluster and the number of devices and directories configured.

```
rook-ceph   rook-ceph-mgr-a-57fc559bbc-hmcqs           1/1     Running     0     1h
rook-ceph   rook-ceph-mon-a-5f5cccf46d-d9n92           1/1     Running     0     1h
rook-ceph   rook-ceph-mon-d-58b85869c9-z2vhw           1/1     Running     0     1h
rook-ceph   rook-ceph-mon-e-b84cbbf87-7wn44            1/1     Running     0     1h
rook-ceph   rook-ceph-osd-0-78f5644464-9ztjx           1/1     Running     0     1h
rook-ceph   rook-ceph-osd-prepare-your-machine-7x6lt   0/2     Completed   0     1h
```

For further information, refer to Table 2, *Ceph Storage Quickstart*.

C.3 Ceph's Block Storage Installation and Configuration

Create StorageClass and its storage pool:

```
kubectl create -f storageclass.yaml
```

Tip: To create a storage pool replicated three times use: kubectl create `-f storageclass_3_replicas.yaml`.

The application needs to specify the name of `StorageClass` in its charts to consume block storage provisioned by Rook.

For further information, refer to Table 2, *Block Storage*.

C.4 Cleaning up a Cluster

For further information, refer to Table 2, *Cleaning up a Cluster.*

C.4.1 Cleaning up the Resources Created on Top of the

First, clean up the resources created on top of the Rook cluster, starting with **the applications which consume block storage** provisioned by Rook.

Delete storage pool and StorageClass using this script:

```
kubectl delete -n rook-ceph cephblockpool replicapool
kubectl delete storageclass rook-ceph-block
```

C.4.2 Removing Rook Cluster

After those block and file resources have been cleaned up, then delete the Rook cluster.

Note: It is essential to delete the rock cluster before removing the Rook operator and agent. Otherwise, resources may not be cleaned up properly.

```
kubectl delete -f cluster.yaml
kubectl -n rook-ceph delete cephcluster rook-ceph
```

Verify the cluster has been deleted before continuing to the next step.

```
kubectl -n rook-ceph get cephcluster
```

C.4.3 Removing Persistent Volumes (PV) and Persistent Volumes Claims (PVC)

Remove Persistent Volumes (PV) and Persistent Volumes Claims (PVC) used by your pods.

List all Persistent Volumes:

```
kubectl get pv
```

Remove all Persistent Volumes with `STORAGECLASS rook-ceph-block` by their name:

```
kubectl delete pv fill-name-of-your-pv
```

List all Persistent Volume Claims:

```
kubectl get pvc
```

Remove all Persistent Volume Claims with `STORAGECLASS rook-ceph-block` by their name:

```
kubectl delete pvc fill-name-of-your-pvc
```

C.4.4 Removing the Operator

Delete the Operator:

```
kubectl delete -f operator.yaml
```

Optionally remove the rook-ceph namespace if not in use by any other resources:

```
kubectl delete namespace rook-ceph
```

C.4.5 Deleting the Data on Hosts

Important: The final cleanup step requires deleting files on each host in the cluster.

All files under the `spec.dataDirHostPath` and `spec.storage.directories.path` properties specified in the cluster `CRD` need to be deleted. Otherwise, an inconsistent state remains when a new cluster is started.

Connect to each machine and delete directories specified by `spec.dataDirHostPath` and `spec.storage.directories.path`:

```
sudo rm -rf /var/lib/rook/
```

§

Appendix D Service Detector

Primary responsibilities of Service Detector are:

- providing information about services being under the management
- exposing actions for manual registration and unregistration of external services.

Different requirements related to service detection comes with different solutions. Therefore, multiple implementations of service detection mechanisms are provided.

D.1 Redfish Registration API

Service detector exposes the following operations:

`GET /redfish/v1/Managers` - gets a collection of all available managers

Response:

```
{
  "@odata.context": "/redfish/v1/$metadata#ManagerCollection.ManagerCollection",
  "@odata.id": "/redfish/v1/Managers",
  "@odata.type": "#ManagerCollection.ManagerCollection",
  "Name": "ManagerCollection",
  "Members@odata.count": 2,
  "Members": [
    {
      "@odata.id": "/redfish/v1/Managers/5490ab10-0515-11e9-b46d-bf8eed3ca1c9"
    },
    {
      "@odata.id": "/redfish/v1/Managers/bd047980-09d2-11e9-9318-7725cee455aa"
    }
  ]
}
```

`POST /redfish/v1/Managers` - creates new manager

Sample body:

```
{
  "RemoteRedfishServiceUri": "http://localhost:9999/redfish/v1"
}
```

`GET /redfish/v1/Managers?$expand=.($levels=1)` - gets expanded collection of all available managers

Response:

```
{
  "@odata.context": "/redfish/v1/$metadata#ManagerCollection.ManagerCollection",
  "@odata.id": "/redfish/v1/Managers",
  "@odata.type": "#ManagerCollection.ManagerCollection",
  "Name": "ManagerCollection",
  "Members@odata.count": 2,
  "Members": [
    {
      "@odata.id": "/redfish/v1/Managers/5490ab10-0515-11e9-b46d-bf8eed3ca1c9",
      "@odata.type": "#Manager.v1_5_0.Manager",
      "Id": "5490ab10-0515-11e9-b46d-bf8eed3ca1c9",
      "Name": null,
      "Status": {
        "State": "Enabled"
      }
      "ServiceEntryPointUUID": "5490ab10-0515-11e9-b46d-bf8eed3ca1c9",
      "RemoteRedfishServiceUri": "http://localhost:10443/redfish/v1",
```

```
      "Oem": {
        "Intel_RackScale": {
          "Trusted": true
        }
      }
    }
  ]
}
```

`GET /redfish/v1/Managers/{id}` - gets information about particular manager

Response:

```
{
  "@odata.id": "/redfish/v1/Managers/5490ab10-0515-11e9-b46d-bf8eed3ca1c9",
  "@odata.type": "#Manager.v1_5_0.Manager",
  "Id": "5490ab10-0515-11e9-b46d-bf8eed3ca1c9",
  "Name": null,
  "Status": {
    "State": "Enabled"
  }
  "ServiceEntryPointUUID": "5490ab10-0515-11e9-b46d-bf8eed3ca1c9",
  "RemoteRedfishServiceUri": "http://localhost:10443/redfish/v1",
  "Oem": {
    "Intel_RackScale": {
      "Trusted": true
    }
  }
```

`DELETE /redfish/v1/Managers/{id}` - deletes existing manager.

D.1.1 Available Configuration Options

Redfish Registration API based detector is always active, and it cannot be disabled. It comes with few configuration options which let users adjust detection functionality to their needs. Configuration options have been implemented on Spring's application profiles. Available profiles:

- **any-service-registrar** - allows registering both HTTP and HTTPS services.
- **https-only-service-registrar** - recommended option (it allows to register only HTTPS services, registration of any HTTP service will be rejected).
- **no-verification** - registered services will be exposed as trusted without any verification.

D.1.2 Trusted/Untrusted Services

Service Detector performs periodical checkup of registered HTTPS services. For all available services (Manager's Status.State = `Enabled`), it tries to validate their certificate. The Sevice Detector also determines whether the service is still trusted which is reflected in the `Oem.Intel_RackScale.Trusted` Manager property.

D.2 SSDP Detector

SSDP detector is disabled by default. To enable, run the `ServiceDetector` application with the appropriate profile. The application profile can be set by property using:

For configuration defined in the `application.properties` file:

```
service-detector.ssdp.enabled=true
```

The same property could be passed to the Helm installation command:

```
--set service-detector.ssdp.enabled=true
```

Additional configuration of SSDP detector is defined in Kubernetes's `configmap`, called the `podm-ssdp-config` which is consumed by the Helm installation command.

D.3 DHCP Detector

DHCP detector is disabled by default. To enable it `ServiceDetector` application has to be run with the appropriate profile. Application profile can be set by property:

For configuration defined in the `application.properties` file:

```
service-detector.dhcpd.enabled=true
```

The same property could be pass to Helm installation command:

```
--set service-detector.dhcpd.enabled=true
```

Additional configuration of DHCP detector is defined in Kubernetes's `configmap` called `podm-dhcp-config` which is consumed by Helm installation command.

§

Appendix E Resource Manager Configuration

Note: Default Resource Manager configuration is located in:

`resource-manager/runner/src/resources/application.yml.`

Tip: Config can be overridden for Kubernetes* deployment by setting `applicationProperties` in Helm Chart (such as via `values.yaml`)

E.1 Spring Base Config

```
spring:
  application:
    name: RESOURCE-MANAGER:PSME
```

E.2 Southbound API

```
southbound-config:
  acceptedHeaders:
    - Location
```

E.3 Spring Cloud Sleuth

```
spring:
  sleuth:
    sampler:
      probability: 1
```

E.4 Spring Cloud Netflix Eureka

```
eureka:
  instance:
    metadata-map:
      requiredType: ${requiredType}
      providedType: ${providedType}
```

E.5 Spring Cloud Netflix Hystrix

Reference: *fallback.isolation.semaphore.maxConcurrentRequests*

```
fallback.isolation.semaphore.maxConcurrentRequests: 200
```

E.6 Events

Table 4. Configurations

Property	Description
`events.submitter`	Configuration for producing events
`events.receiver`	Configuration for consuming events

Table 5. Producing Events - events.submitter

Property	Description
`submitter.endpoint`	Path at Event Service that Resource Manager will produce Events for further processing

Table 6. Consuming Events - events.receiver

Property	Description
`receiver.type`	Specifies the method of determining Resource Manager URI to be used during subscription for events from external sources. *Allowed values:* Fixed, Dynamic *Default value:* Fixed
`receiver.endpoint`	Specifies the REST API endpoint that will be used to receive events from external sources
`receiver.fixed`	Contains static configuration of the event receiving URI at Resource Manager
`receiver.fixed.target-uri`	When receiver.type is set to **Fixed,** this URI will be used for event receiving
`receiver.dynamic`	Used when receiver.type is set to Dynamic. This configuration reflects *Kubernetes Node Port* behavior
`receiver.dynamic.target-port`	The port configured as Node Port for nodes in Kubernetes* cluster
`receiver.dynamic.target-protocol`	The protocol used to build Resource Manager URI
`receiver.dynamic.mapping`	Defines a set of target IP addresses of nodes in Kubernetes* cluster that will be used to build Resource Manager URI. Target IP addresses will be used as a destination during subscription for events from external sources for specific subnets
`receiver.dynamic.mapping.source-subnet`	Defines subnet of external event sources for which this configuration applies. *Allowed format:* CIDR
`receiver.dynamic.mapping.target-ip-addresses`	Defines IP addresses fo nodes in Kubernetes* cluster that are able to receive events from subnet defined by `receiver.dynamic.mapping.source-subnet`.

NOTE: During event subscription attempt when using Dynamic configuration type, first accessible address from target-ip-addresses will be used to build Resource Manager URI that will be used to receive Events from external sources.

Events configuration:

```
events:
  submitter:
    endpoint: /redfish/v1/EventService/Events
  receiver:
    type: Fixed
    endpoint: /events
    fixed:
      target-uri: http://localhost:8600
    dynamic:
      target-port: 30000
      target-protocol: https
      mapping:
        - source-subnet: 10.3.0.0/24
          target-ip-addresses:
            - 10.3.0.1
            - 10.3.0.2
            - 10.3.0.3
        - source-subnet: 10.2.0.0/24
          target-ip-addresses:
            - 10.2.0.1
            - 10.2.0.2
            - 10.2.0.3
```

E.7 Layer: Tagger

```
tagger-config:
  tagDefinitions:
    - resource: /redfish/v1/**
      property: /Oem/Intel RackScale/TaggedValues
      type: OBJECT
    - resource: /redfish/v1/Chassis/pod
      property: /AssetTag
      type: STRING
```

E.8 Layer: Cacher

```
cacher:
  entries-time-to-live: 1d
  max-heap-size-mb: 30
```

E.9 Layer: Unifier

```
unification-task:
  poolSize: 20
```

E.10 Spring Boot Actuator

```
management:
  endpoint:
    health:
      show-details: always
  endpoints:
    web:
      exposure:
        include:
          - health
          - configprops
          - env
          - loggers
          - logfile
          - httptrace
          - threaddump
          - prometheus
```

E.11 Logging

```
logging:
  level:
    root: INFO
    logstash: INFO
    com.intel.rsd.resourcemanager.runner.requiredlayer.RequiredLayer: DEBUG
```

§

Appendix F cluster.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: rook-ceph
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rook-ceph-osd
  namespace: rook-ceph
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rook-ceph-mgr
  namespace: rook-ceph
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-osd
  namespace: rook-ceph
rules:
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: [ "get", "list", "watch", "create", "update", "delete" ]
---
# Aspects of ceph-mgr that require access to the system namespace
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-mgr-system
  namespace: rook-ceph
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - get
  - list
  - watch
---
# Aspects of ceph-mgr that operate within the cluster's namespace
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-mgr
  namespace: rook-ceph
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - services
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - batch
  resources:
  - jobs
  verbs:
```

```
  - get
  - list
  - watch
  - create
  - update
  - delete
- apiGroups:
  - ceph.rook.io
  resources:
  - "*"
  verbs:
  - "*"
---
# Allow the operator to create resources in this cluster's namespace
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-cluster-mgmt
  namespace: rook-ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rook-ceph-cluster-mgmt
subjects:
- kind: ServiceAccount
  name: rook-ceph-system
  namespace: rook-ceph-system
---
# Allow the osd pods in this namespace to work with configmaps
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-osd
  namespace: rook-ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rook-ceph-osd
subjects:
- kind: ServiceAccount
  name: rook-ceph-osd
  namespace: rook-ceph
---
# Allow the ceph mgr to access the cluster-specific resources necessary for the mgr
modules
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-mgr
  namespace: rook-ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rook-ceph-mgr
subjects:
- kind: ServiceAccount
  name: rook-ceph-mgr
  namespace: rook-ceph
---
# Allow the ceph mgr to access the rook system resources necessary for the mgr modules
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-mgr-system
  namespace: rook-ceph-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
```

```
  name: rook-ceph-mgr-system
subjects:
- kind: ServiceAccount
  name: rook-ceph-mgr
  namespace: rook-ceph
---
# Allow the ceph mgr to access cluster-wide resources necessary for the mgr modules
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-mgr-cluster
  namespace: rook-ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rook-ceph-mgr-cluster
subjects:
- kind: ServiceAccount
  name: rook-ceph-mgr
  namespace: rook-ceph
---
apiVersion: ceph.rook.io/v1
kind: CephCluster
metadata:
  name: rook-ceph
  namespace: rook-ceph
spec:
  cephVersion:
    # The container image used to launch the Ceph daemon pods (mon, mgr, osd, mds,
rgw).
    # v12 is luminous, v13 is mimic, and v14 is nautilus.
    # RECOMMENDATION: In production, use a specific version tag instead of the general
v13 flag, which pulls the latest release and could result in different
    # versions running within the cluster. See tags available at
https://hub.docker.com/r/ceph/ceph/tags/.
    image: ceph/ceph:v13.2.4-20190109
    # Whether to allow unsupported versions of Ceph. Currently only luminous and mimic
are supported.
    # After nautilus is released, Rook will be updated to support nautilus.
    # Do not set to true in production.
    allowUnsupported: false
  # The path on the host where configuration files will be persisted. If not
specified, a kubernetes emptyDir will be created (not recommended).
  # Important: if you reinstall the cluster, make sure you delete this directory from
each host or else the mons will fail to start on the new cluster.
  # In Minikube, the '/data' directory is configured to persist across reboots. Use
"/data/rook" in Minikube environment.
  dataDirHostPath: /var/lib/rook
  # set the amount of mons to be started
  mon:
    count: 3
    allowMultiplePerNode: true
  # enable the ceph dashboard for viewing cluster status
  dashboard:
    enabled: true
    # serve the dashboard under a subpath (useful when you are accessing the dashboard
via a reverse proxy)
    # urlPrefix: /ceph-dashboard
    # serve the dashboard at the given port.
    # port: 8443
    # serve the dashboard using SSL
    # ssl: true
  network:
    # toggle to use hostNetwork
    hostNetwork: false
  rbdMirroring:
    # The number of daemons that will perform the rbd mirroring.
    # rbd mirroring must be configured with "rbd mirror" from the rook toolbox.
```

```
    workers: 0
  # To control where various services will be scheduled by kubernetes, use the
placement configuration sections below.
  # The example under 'all' would have all services scheduled on kubernetes nodes
labeled with 'role=storage-node' and
  # tolerate taints with a key of 'storage-node'.
  #  placement:
  #    all:
  #      nodeAffinity:
  #        requiredDuringSchedulingIgnoredDuringExecution:
  #          nodeSelectorTerms:
  #          - matchExpressions:
  #            - key: role
  #              operator: In
  #              values:
  #              - storage-node
  #      podAffinity:
  #      podAntiAffinity:
  #      tolerations:
  #      - key: storage-node
  #        operator: Exists
  # The above placement information can also be specified for mon, osd, and mgr
components
  #    mon:
  #    osd:
  #    mgr:
  resources:
  # The requests and limits set here, allow the mgr pod to use half of one CPU core
and 1 gigabyte of memory
  #    mgr:
  #      limits:
  #        cpu: "500m"
  #        memory: "1024Mi"
  #      requests:
  #        cpu: "500m"
  #        memory: "1024Mi"
  # The above example requests/limits can also be added to the mon and osd components
  #    mon:
  #    osd:
  storage: # cluster level storage configuration and selection
    useAllNodes: true
    useAllDevices: false
    deviceFilter:
    location:
    config:
      # The default and recommended storeType is dynamically set to bluestore for
devices and filestore for directories.
      # Set the storeType explicitly only if it is required not to use the default.
      # storeType: bluestore
      databaseSizeMB: "1024" # this value can be removed for environments with normal
sized disks (100 GB or larger)
      journalSizeMB: "1024"  # this value can be removed for environments with normal
sized disks (20 GB or larger)
      osdsPerDevice: "1" # this value can be overridden at the node or device level
# Cluster level list of directories to use for storage. These values will be set for
all nodes that have no `directories` set.
#    directories:
#    - path: /rook/storage-dir
# Individual nodes and their config can be specified as well, but 'useAllNodes' above
must be set to false. Then, only the named
# nodes below will be used as storage resources.  Each node's 'name' field should
match their 'kubernetes.io/hostname' label.
#    nodes:
#    - name: "172.17.4.101"
#      directories: # specific directories to use for storage can be specified for
each node
#      - path: "/rook/storage-dir"
#      resources:
```

```
#        limits:
#          cpu: "500m"
#          memory: "1024Mi"
#        requests:
#          cpu: "500m"
#          memory: "1024Mi"
#    - name: "172.17.4.201"
#      devices: # specific devices to use for storage can be specified for each node
#      - name: "sdb"
#      - name: "nvme01" # multiple osds can be created on high performance devices
#        config:
#          osdsPerDevice: "5"
#      config: # configuration can be specified at the node level which overrides the
cluster level config
#        storeType: filestore
#    - name: "172.17.4.301"
#      deviceFilter: "^sd."
```

§

Appendix G operator.yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: rook-ceph-system
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: cephclusters.ceph.rook.io
spec:
  group: ceph.rook.io
  names:
    kind: CephCluster
    listKind: CephClusterList
    plural: cephclusters
    singular: cephcluster
  scope: Namespaced
  version: v1
  validation:
    openAPIV3Schema:
      properties:
        spec:
          properties:
            cephVersion:
              properties:
                allowUnsupported:
                  type: boolean
                image:
                  type: string
                name:
                  pattern: ^(luminous|mimic|nautilus)$
                  type: string
            dashboard:
              properties:
                enabled:
                  type: boolean
                urlPrefix:
                  type: string
                port:
                  type: integer
            dataDirHostPath:
              pattern: ^/(\S+)
              type: string
            mon:
              properties:
                allowMultiplePerNode:
                  type: boolean
                count:
                  maximum: 9
                  minimum: 1
                  type: integer
              required:
              - count
            network:
              properties:
                hostNetwork:
                  type: boolean
            storage:
              properties:
                nodes:
                  items: {}
                  type: array
                useAllDevices: {}
                useAllNodes:
```

```
                  type: boolean
          required:
          - mon
  additionalPrinterColumns:
  - name: DataDirHostPath
    type: string
    description: Directory used on the K8s nodes
    JSONPath: .spec.dataDirHostPath
  - name: MonCount
    type: string
    description: Number of MONs
    JSONPath: .spec.mon.count
  - name: Age
    type: date
    JSONPath: .metadata.creationTimestamp
  - name: State
    type: string
    description: Current State
    JSONPath: .status.state
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: cephfilesystems.ceph.rook.io
spec:
  group: ceph.rook.io
  names:
    kind: CephFilesystem
    listKind: CephFilesystemList
    plural: cephfilesystems
    singular: cephfilesystem
  scope: Namespaced
  version: v1
  additionalPrinterColumns:
  - name: MdsCount
    type: string
    description: Number of MDSs
    JSONPath: .spec.metadataServer.activeCount
  - name: Age
    type: date
    JSONPath: .metadata.creationTimestamp
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: cephobjectstores.ceph.rook.io
spec:
  group: ceph.rook.io
  names:
    kind: CephObjectStore
    listKind: CephObjectStoreList
    plural: cephobjectstores
    singular: cephobjectstore
  scope: Namespaced
  version: v1
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: cephobjectstoreusers.ceph.rook.io
spec:
  group: ceph.rook.io
  names:
    kind: CephObjectStoreUser
    listKind: CephObjectStoreUserList
    plural: cephobjectstoreusers
    singular: cephobjectstoreuser
  scope: Namespaced
```

```
  version: v1
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: cephblockpools.ceph.rook.io
spec:
  group: ceph.rook.io
  names:
    kind: CephBlockPool
    listKind: CephBlockPoolList
    plural: cephblockpools
    singular: cephblockpool
  scope: Namespaced
  version: v1
---
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: volumes.rook.io
spec:
  group: rook.io
  names:
    kind: Volume
    listKind: VolumeList
    plural: volumes
    singular: volume
    shortNames:
    - rv
  scope: Namespaced
  version: v1alpha2
---
# The cluster role for managing all the cluster-specific resources in a namespace
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: rook-ceph-cluster-mgmt
  labels:
    operator: rook
    storage-backend: ceph
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  - pods
  - pods/log
  - services
  - configmaps
  verbs:
  - get
  - list
  - watch
  - patch
  - create
  - update
  - delete
- apiGroups:
  - extensions
  resources:
  - deployments
  - daemonsets
  - replicasets
  verbs:
  - get
  - list
  - watch
  - create
```

```
  - update
  - delete
---
# The role for the operator to manage resources in the system namespace
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: Role
metadata:
  name: rook-ceph-system
  namespace: rook-ceph-system
  labels:
    operator: rook
    storage-backend: ceph
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - configmaps
  verbs:
  - get
  - list
  - watch
  - patch
  - create
  - update
  - delete
- apiGroups:
  - extensions
  resources:
  - daemonsets
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - delete
---
# The cluster role for managing the Rook CRDs
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRole
metadata:
  name: rook-ceph-global
  labels:
    operator: rook
    storage-backend: ceph
rules:
- apiGroups:
  - ""
  resources:
  # Pod access is needed for fencing
  - pods
  # Node access is needed for determining nodes where mons should run
  - nodes
  - nodes/proxy
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - ""
  resources:
  - events
  # PVs and PVCs are managed by the Rook provisioner
  - persistentvolumes
  - persistentvolumeclaims
  verbs:
  - get
```

```
  - list
  - watch
  - patch
  - create
  - update
  - delete
- apiGroups:
  - storage.k8s.io
  resources:
  - storageclasses
  verbs:
  - get
  - list
  - watch
- apiGroups:
  - batch
  resources:
  - jobs
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - delete
- apiGroups:
  - ceph.rook.io
  resources:
  - "*"
  verbs:
  - "*"
- apiGroups:
  - rook.io
  resources:
  - "*"
  verbs:
  - "*"
---
# Aspects of ceph-mgr that require cluster-wide access
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-mgr-cluster
  labels:
    operator: rook
    storage-backend: ceph
rules:
- apiGroups:
  - ""
  resources:
  - configmaps
  - nodes
  - nodes/proxy
  verbs:
  - get
  - list
  - watch
---
# The rook system service account used by the operator, agent, and discovery pods
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rook-ceph-system
  namespace: rook-ceph-system
  labels:
    operator: rook
    storage-backend: ceph
---
```

```
# Grant the operator, agent, and discovery agents access to resources in the rook-
ceph-system namespace
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-system
  namespace: rook-ceph-system
  labels:
    operator: rook
    storage-backend: ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rook-ceph-system
subjects:
- kind: ServiceAccount
  name: rook-ceph-system
  namespace: rook-ceph-system
---
# Grant the rook system daemons cluster-wide access to manage the Rook CRDs, PVCs, and
storage classes
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: rook-ceph-global
  namespace: rook-ceph-system
  labels:
    operator: rook
    storage-backend: ceph
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: rook-ceph-global
subjects:
- kind: ServiceAccount
  name: rook-ceph-system
  namespace: rook-ceph-system
---
# The deployment for the rook operator
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: rook-ceph-operator
  namespace: rook-ceph-system
  labels:
    operator: rook
    storage-backend: ceph
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: rook-ceph-operator
    spec:
      serviceAccountName: rook-ceph-system
      containers:
      - name: rook-ceph-operator
        image: rook/ceph:v0.9.3
        args: ["ceph", "operator"]
        volumeMounts:
        - mountPath: /var/lib/rook
          name: rook-config
        - mountPath: /etc/ceph
          name: default-config-dir
        env:
        # To disable RBAC, uncomment the following:
        # - name: RBAC ENABLED
        #  value: "false"
```

```
        # Rook Agent toleration. Will tolerate all taints with all keys.
        # Choose between NoSchedule, PreferNoSchedule and NoExecute:
        # - name: AGENT_TOLERATION
        #   value: "NoSchedule"
        # (Optional) Rook Agent toleration key. Set this to the key of the taint you
want to tolerate
        # - name: AGENT TOLERATION KEY
        #   value: "<KeyOfTheTaintToTolerate>"
        # (Optional) Rook Agent mount security mode. Can by `Any` or `Restricted`.
        # `Any` uses Ceph admin credentials by default/fallback.
        # For using `Restricted` you must have a Ceph secret in each namespace storage
should be consumed from and
        # set `mountUser` to the Ceph user, `mountSecret` to the Kubernetes secret
name.
        # to the namespace in which the `mountSecret` Kubernetes secret namespace.
        # - name: AGENT MOUNT SECURITY MODE
        #   value: "Any"
        # Set the path where the Rook agent can find the flex volumes
        # - name: FLEXVOLUME DIR PATH
        #  value: "<PathToFlexVolumes>"
        # Set the path where kernel modules can be found
        # - name: LIB MODULES DIR PATH
        #  value: "<PathToLibModules>"
        # Mount any extra directories into the agent container
        # - name: AGENT MOUNTS
        #  value:
"somemount=/host/path:/container/path,someothermount=/host/path2:/container/path2"
        # Rook Discover toleration. Will tolerate all taints with all keys.
        # Choose between NoSchedule, PreferNoSchedule and NoExecute:
        # - name: DISCOVER TOLERATION
        #   value: "NoSchedule"
        # (Optional) Rook Discover toleration key. Set this to the key of the taint
you want to tolerate
        # - name: DISCOVER TOLERATION KEY
        #  value: "<KeyOfTheTaintToTolerate>"
        # Allow rook to create multiple file systems. Note: This is considered
        # an experimental feature in Ceph as described at
        # http://docs.ceph.com/docs/master/cephfs/experimental-features/#multiple-
filesystems-within-a-ceph-cluster
        # which might cause mons to crash as seen in
https://github.com/rook/rook/issues/1027
        - name: ROOK ALLOW MULTIPLE FILESYSTEMS
          value: "false"
        # The logging level for the operator: INFO | DEBUG
        - name: ROOK_LOG_LEVEL
          value: "INFO"
        # The interval to check if every mon is in the quorum.
        - name: ROOK MON HEALTHCHECK INTERVAL
          value: "45s"
        # The duration to wait before trying to failover or remove/replace the
        # current mon with a new mon (useful for compensating flapping network).
        - name: ROOK MON OUT TIMEOUT
          value: "600s"
        # The duration between discovering devices in the rook-discover daemonset.
        - name: ROOK_DISCOVER_DEVICES_INTERVAL
          value: "60m"
        # Whether to start pods as privileged that mount a host path, which includes
the Ceph mon and osd pods.
        # This is necessary to workaround the anyuid issues when running on OpenShift.
        # For more details see https://github.com/rook/rook/issues/1314#issuecomment-
355799641
        - name: ROOK_HOSTPATH_REQUIRES_PRIVILEGED
          value: "false"
        # In some situations SELinux relabelling breaks (times out) on large
filesystems, and doesn't work with cephfs ReadWriteMany volumes (last relabel wins).
        # Disable it here if you have similar issues.
        # For more details see https://github.com/rook/rook/issues/2417
        - name: ROOK_ENABLE_SELINUX_RELABELING
```

```
          value: "true"
        # In large volumes it will take some time to chown all the files. Disable it
here if you have performance issues.
        # For more details see https://github.com/rook/rook/issues/2254
        - name: ROOK_ENABLE_FSGROUP
          value: "true"
        # The name of the node to pass with the downward API
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        # The pod name to pass with the downward API
        - name: POD_NAME
          valueFrom:
            fieldRef:
              fieldPath: metadata.name
        # The pod namespace to pass with the downward API
        - name: POD_NAMESPACE
          valueFrom:
            fieldRef:
              fieldPath: metadata.namespace
      volumes:
      - name: rook-config
        emptyDir: {}
      - name: default-config-dir
        emptyDir: {}
```

§

Appendix H storageclass.yaml

```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook-ceph
spec:
  replicated:
    size: 1
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
provisioner: ceph.rook.io/block
parameters:
  blockPool: replicapool
  # Specify the namespace of the rook cluster from which to create volumes.
  # If not specified, it will use `rook` as the default namespace of the cluster.
  # This is also the namespace where the cluster will be
  clusterNamespace: rook-ceph
  # Specify the filesystem type of the volume. If not specified, it will use `ext4`.
  fstype: xfs
  # (Optional) Specify an existing Ceph user that will be used for mounting storage
with this StorageClass.
  #mountUser: user1
  # (Optional) Specify an existing Kubernetes secret name containing just one key
holding the Ceph user secret.
  # The secret must exist in each namespace(s) where the storage will be consumed.
  #mountSecret: ceph-user1-secret
reclaimPolicy: Retain
```

§

Appendix I storageclass_3_replicas.yaml

```
apiVersion: ceph.rook.io/v1
kind: CephBlockPool
metadata:
  name: replicapool
  namespace: rook-ceph
spec:
  replicated:
    size: 3
---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: rook-ceph-block
provisioner: ceph.rook.io/block
parameters:
  blockPool: replicapool
  # Specify the namespace of the rook cluster from which to create volumes.
  # If not specified, it will use `rook` as the default namespace of the cluster.
  # This is also the namespace where the cluster will be
  clusterNamespace: rook-ceph
  # Specify the filesystem type of the volume. If not specified, it will use `ext4`.
  fstype: xfs
  # (Optional) Specify an existing Ceph user that will be used for mounting storage
with this StorageClass.
  #mountUser: user1
  # (Optional) Specify an existing Kubernetes secret name containing just one key
holding the Ceph user secret.
  # The secret must exist in each namespace(s) where the storage will be consumed.
  #mountSecret: ceph-user1-secret
reclaimPolicy: Retain
```

§