Installation Guide

OpenNMS Meridian 2019.1.27, Last updated 2021-12-13 14:18:20 UTC

Table of Contents

Chapter 1. Compatibility

OpenNMS Meridian 2019.1.27 requires the following component versions:

Component	Version Compatibility
OpenNMS Helm	3+
OpenNMS Integration API	0.2.x
Cassandra	3.11.+
Elasticsearch	7.x
Java Development Kit	OpenJDK 8, OpenJDK 11
Kafka	1.x - 2.x
PostgreSQL	10.x - 12.x
RRDTool	1.7.x

Chapter 2. Setting up a basic OpenNMS Meridian

The *OpenNMS Meridian* platform can be installed on multiple OS families. This guide provides instructions for installing the platform on *Red Hat Enterprise Linux (RHEL)*-based operating systems.

2.1. Objectives

- Installing *OpenNMS Meridian* components on a single node using the built-in *JRobin* as time series storage
- Setup *OpenNMS Meridian* on recommended operating systems
- Login the Web User Interface and change the default admin password

2.2. Before you begin

The following abbreviations will be used to refer to their respective entry through this documentation.

Table 1. Operating Systems

RHEL	Red Hat Enterprise Linux 7 or higher, CentOS 7 or higher
OpenJDK 11 Development Kit	Installed OpenJDK 11 Development Kit

It is recommended to meet the following requirements:

Table 2. Installation Requirements

Minimal Hardware	2 CPU, 2 GB RAM, 20 GB disk
Operating System	The latest version of *RHEL* is recommended. Please be aware *OpenNMS Meridian* is developed and mostly operated on Linux systems.
DNS Setup	Please make sure your DNS settings for the OpenNMS server are correct and the localhost name can be resolved. If there is an incorrect or missing *A Resource Record* for the server hostname, OpenNMS might not start correctly. The Java security manager might not initialize and an *RMI class loader disabled* exception will be shown.

Depending on the installed operating system, the path for *OpenNMS Meridian* is different. If the instruction refers to `${OPENNMS_HOME}`, the path is resolved to the following directories:

Table 3. Directory Structure

RHEL	`/opt/opennms`

2.3. Installing on RHEL

The following steps will be described:

1. Installation of the opennms meta package which handles all dependencies

2. Initialize *PostgreSQL* database and configure access

3. Initialize *OpenNMS Meridian* database and start

4. Log in to the Web User Interface and change default admin password

All commands on the command line interface need to be executed with *root* permissions.

Step 1: Install OpenNMS Meridian

Add yum repository and import GPG key

```
dnf -y install https://yum.opennms.org/repofiles/opennms-repo-stable-rhel8.noarch.rpm
rpm --import https://yum.opennms.org/OPENNMS-GPG-KEY
```

Installation of with all built-in dependencies

```
dnf -y install opennms
```

The following packages will be automatically installed:

- *jicmp6* and *jicmp*: *Java* bridge to allow sending *ICMP messages* from *OpenNMS Meridian* repository.

- *opennms-core*: *OpenNMS Meridian* core services, e.g. *Provisiond*, *Pollerd* and *Collectd* from *OpenNMS Meridian* repository.

- *opennms-webapp-jetty*: *OpenNMS Meridian* web application from *OpenNMS Meridian* repository

- *postgresql*: *PostgreSQL* database server from distribution repository

- *postgresql-libs*: *PostgreSQL* database from distribution repository

With the successful installed packages the *OpenNMS Meridian* is installed in the following directory structure:

```
[root@localhost /opt/opennms]# tree -L 1
.
└── opennms
    ├── bin
    ├── contrib
    ├── data
    ├── deploy
    ├── etc
    ├── jetty-webapps
    ├── lib
    ├── logs -> /var/log/opennms
    ├── share -> /var/opennms
    └── system
```

> 💡 We recommend disabling the OpenNMS Meridian repository after installation to prevent unwanted upgrades while it is running. *OpenNMS Meridian* requires some manual steps upon upgrade configuration files or migrate database schemas to a new version. For this reason, it is recommended to exclude the OpenNMS Meridian packages from update except when you are planning on performing an upgrade.

```
dnf config-manager --disable opennms-repo-stable-*
```

Step 2: Initialize and setup PostgreSQL

Initialization of the PostgreSQL database

```
postgresql-setup --initdb --unit postgresql
```

System startup configuration for PostgreSQL

```
systemctl enable postgresql
```

Startup PostgreSQL database

```
systemctl start postgresql
```

Create an opennms database user with a password and create an opennms database which is owned by the user opennms

```
su - postgres
createuser -P opennms
createdb -O opennms opennms
```

Set a password for Postgres super user

```
psql -c "ALTER USER postgres WITH PASSWORD 'YOUR-POSTGRES-PASSWORD';"
exit
```

> ℹ The super user is required to be able to initialize and change the database schema for installation and updates.

Change the access policy for PostgreSQL

```
vi /var/lib/pgsql/data/pg_hba.conf
```

Allow OpenNMS Meridian accessing the database over the local network with a MD5 hashed password

```
host    all             all             127.0.0.1/32            md5①
host    all             all             ::1/128                 md5①
```

① Change method from `ident` to `md5` for *IPv4* and *IPv6* on localhost.

Apply configuration changes for PostgreSQL

```
systemctl reload postgresql
```

Configure database access in OpenNMS Meridian

```
vi ${OPENNMS_HOME}/etc/opennms-datasources.xml
```

Set credentials to access the PostgreSQL database

```
<jdbc-data-source name="opennms"
                  database-name="opennms"①
                  class-name="org.postgresql.Driver"
                  url="jdbc:postgresql://localhost:5432/opennms"
                  user-name="** YOUR-OPENNMS-USERNAME **"②
                  password="** YOUR-OPENNMS-PASSWORD **" />③

<jdbc-data-source name="opennms-admin"
                  database-name="template1"
                  class-name="org.postgresql.Driver"
                  url="jdbc:postgresql://localhost:5432/template1"
                  user-name="postgres"④
                  password="** YOUR-POSTGRES-PASSWORD **" />⑤
```

① Set the database name *OpenNMS Meridian* should use

② Set the user name to access the *opennms* database table

③ Set the password to access the *opennms* database table

④ Set the *postgres* user for administrative access to PostgreSQL

⑤ Set the password for administrative access to PostgreSQL

Step 3: Initialize and start OpenNMS Meridian

Detect of Java environment and persist in /opt/opennms/etc/java.conf

```
${OPENNMS_HOME}/bin/runjava -s
```

Initialize the database and detect system libraries persisted in /opt/opennms/etc/libraries.properties

```
${OPENNMS_HOME}/bin/install -dis
```

Configure systemd to start OpenNMS Meridian on system boot

```
systemctl enable opennms
```

Start OpenNMS Meridian

```
systemctl start opennms
```

Allow connection to the Web UI from your network

```
firewall-cmd --permanent --add-port=8980/tcp
systemctl reload firewalld
```

> 💡 If you want to receive SNMP Traps or Syslog messages you have to allow incoming traffic on your host firewall as well. By default OpenNMS SNMP trap daemon is listening on 162/udp and Syslog daemon is listening on 10514/udp. The SNMP Trap daemon is enabled by default, the OpenNMS Syslog daemon is disabled.

Step 4: First Login and change default password

After starting OpenNMS the web application can be accessed on http://<ip-or-fqdn-of-your-server>:8980/opennms. The default login user is *admin* and the password is initialized to *admin*.

1. Open in your browser http://<ip-or-fqdn-of-your-server>:8980/opennms

2. Login with with admin/admin

3. Click in main navigation menu on "admin → Change Password → Change Password"

4. Set as current password *admin* and set a new password and confirm your newly set password

5. Click "Submit"

6. Logout and login with your new password

Next Steps

Additional information can be found in these follow up documents:

- Getting Started Guide

  Learn the first steps to setup, configure, and maintain an *OpenNMS Meridian.*

- Reference Guide

  Find in-depth information on the detecters, monitors, collectors, and configuration files used by the *OpenNMS Meridian* platform.

Monitor isolated location with Minion

This section describes how to install the *Minion* to monitor devices and services in a location which can't be reached from an *OpenNMS Meridian* instance.

Objectives

- Install a *Minion* to monitor devices and services unreachable from an *OpenNMS Meridian* instance
- Configure an authenticated unencrypted communication between *Minion* and *OpenNMS Meridian* using *ActiveMQ* and *REST*

Before you begin

Setting up a *OpenNMS Meridian* with *Minions* requires:

- Instance of *OpenNMS Meridian* needs to be exact same version as *Minion* packages
- Packages are available as *RPMs* for *RHEL*-based systems alongside OpenNMS in the yum repository
- *OpenNMS Meridian* needs to be installed and communication to the *REST (8980/tcp)* and *ActiveMQ (616161/tcp)* endpoints is possible

Depending on the installed operating system, the path for *Minion* is different. If the instruction refers to `${MINION_HOME}`, the path is resolved to the following directories:

Table 4. Directory Structure

RHEL	`/opt/minion`

Installing on RHEL

1. Setup *OpenNMS Meridian* to allow *Minion* communication
2. Installation of the `opennms-minion` meta package which handles all dependencies
3. Starting *Minion* and access the *Karaf* console over *SSH*
4. Configure *Minion* to communicate with *OpenNMS Meridian*
5. Verify the connectivity between *Minion* and *OpenNMS Meridian*

All commands on the command line interface need to be executed with *root* permissions.

Step 1: Setup OpenNMS Meridian to allow Minion communication

Communication between a *Minion* and *OpenNMS Meridian* uses *REST API* and a messaging system, by default *ActiveMQ*. An authenticated user in *OpenNMS Meridian* is required for these communication channels. The security role *ROLE_MINION* includes the minimal amount of permissions required for a *Minion* to operate.

> ❗ As an example we use in this guide the user name *minion* with password *minion*. Change the credentials accordingly.

Create a user minion in the OpenNMS Meridian web user interface

1. Login the web user interface with a user which has administrative permissions

2. Go in the main navigation to *"Login Name → Configure OpenNMS → Configure Users, Groups and On-Call Roles → Configure Users"*

3. Add a new user with login name *minion* and password *minion* and click *Ok*

4. Assign the security role *ROLE_MINION*, optional fill in a comment for what location and purpose the user is used for and click *Finish*

5. The *minion* user should now be listed in the *User List*

Configure ActiveMQ to allow communication on public network interface

```
vi ${OPENNMS_HOME}/etc/opennms-activemq.xml
```

Remove comments for the transport connector listening on 0.0.0.0 and save

```xml
<transportConnector name="openwire" uri="tcp://0.0.0.0:61616?useJmx=false
&amp;maximumConnections=1000&amp;wireformat.maxFrameSize=104857600"/>
```

Restart OpenNMS Meridian

```
systemctl restart opennms
```

Verify if port 61616/tcp is listening on all interfaces

```
ss -lnpt sport = :61616
State   Recv-Q  Send-Q  Local Address:Port  Peer  Address:Port
LISTEN  0       128     *:61616             *:*   users:(("java",pid=1,fd=706))
```

Step 2: Install the repository and Minion package

Connect with *SSH* to your remote *RHEL* system where you need a *Minion* to be installed.

Install the Yum repository

```
dnf -y install https://yum.opennms.org/repofiles/opennms-repo-stable-rhel8.noarch.rpm
rpm --import https://yum.opennms.org/OPENNMS-GPG-KEY
```

Install the Minion package

```
dnf -y install opennms-minion
```

With the successful installed packages the *Minion* is installed in the following directory structure:

```
[root@localhost /opt/minion]# $ tree -L 1
.
├── bin
├── deploy
├── etc
├── lib
├── repositories
└── system
```

The Minion's startup configuration can be changed by editing the `/etc/sysconfig/minion` file. It allows to override the defaults used at startup including:

- Location of the JDK

- Memory usage

- User to run as

Step 3: Starting the Minion and test access to Karaf Shell

Configure systemd to start Minion on system boot

```
systemctl enable minion
```

Startup Minion

```
systemctl start minion
```

Test access to Karaf shell with user admin and password admin and exit with <ctrl-d>

```
ssh -p 8201 admin@localhost
```

Step 4: Configure Minion to communicate with OpenNMS Meridian

Login to the Karaf Shell on the system where your Minion is installed with SSH

```
ssh -p 8201 admin@localhost
```

Configure the Minion's location and endpoint URLs for communication with OpenNMS Meridian

```
[root@localhost /root]# $ ssh -p 8201 admin@localhost
...
admin@minion()> config:edit org.opennms.minion.controller
admin@minion()> config:property-set location Office-Pittsboro
admin@minion()> config:property-set http-url http://opennms-fqdn:8980/opennms
admin@minion()> config:property-set broker-url failover:tcp://opennms-fqdn:61616
admin@minion()> config:update
```

💡 Include the `failover:` portion of the broker URL to allow the *Minion* to re-establish connectivity on failure. For a reference on the different URL formats, see ActiveMQ URI Protocols.

Configure the credentials to use when communicating with OpenNMS Meridian

```
admin@minion()> scv:set opennms.http minion minion
admin@minion()> scv:set opennms.broker minion minion
```

💡 Another way to configure credentials is to use the `scvcli` utility in your *Minion* `bin` directory.

Example of configuring credentials with the command line utility `scvcli`

```
[root@localhost /root]# $ cd /opt/minion
[root@localhost /opt/minion]# $ ./bin/scvcli set opennms.http minion minion
[root@localhost /opt/minion]# $ ./bin/scvcli set opennms.broker minion minion
```

Restart the Minion after updating the credentials

```
[root@localhost /root]# $ systemctl restart minion
```

ℹ️ The credentials are configured separately since they are encrypted on disk.

Step 5: Verifying Connectivity

Connect to Karaf Shell of the Minion

```
ssh -p 8201 admin@localhost
```

Verify connectivity with the OpenNMS Meridian

```
admin@minion()> minion:ping
Connecting to ReST...
OK
Connecting to Broker...
OK
admin@minion()>
```

Sentinel

This section describes how to install the *Sentinel* to scale individual components of OpenNMS Meridian.

> ℹ️ At the moment only flows can be distributed using *Sentinel*. In the future more components will follow.

Before you begin

Setting up a *OpenNMS Meridian* with *Sentinel* requires:

- Instance of *OpenNMS Meridian* needs to be exact same version as *Sentinel* packages
- Packages are available as *RPMs* for *RHEL*-based systems alongside OpenNMS in the yum repository
- *OpenNMS Meridian* needs to be installed and communication to the *REST (8980/tcp)* and *ActiveMQ (616161/tcp)* endpoints is possible
- At least one *Minion* needs to be installed and successful communicate with the *OpenNMS Meridian*

Depending on the installed operating system, the path for *Sentinel* is different. If the instruction refers to `${SENTINEL_HOME}`, the path is resolved to the following directories:

Table 5. Directory Structure

RHEL	`/opt/sentinel`
Debian	`/usr/share/sentinel`

Installing on RHEL

1. Setup *OpenNMS Meridian* to allow *Sentinel* communication
2. Installation of the `meridian-sentinel` meta package which handles all dependencies
3. Starting *Sentinel* and access the *Karaf* console over *SSH*
4. Configure *Sentinel* to communicate with *OpenNMS Meridian*
5. Verify the connectivity between *Sentinel* and *OpenNMS Meridian*

All commands on the command line interface need to be executed with *root* permissions.

Step 1: Setup OpenNMS Meridian to allow Sentinel communication

This step is exactly the same as for *Minion*. Even the role name `ROLE_MINION` can be used, as there does not exist a dedicated role `ROLE_SENTINEL` yet.

Therefore, please refer to section Setup OpenNMS Meridian to allow Minion communication.

> **ℹ** Even if we have to configure the communication to the *OpenNMS Meridian* exactly the same as for *Minion* no ReST requests are made and may be removed at a later state.

Step 2: Install the repository and Sentinel package

Connect with *SSH* to your remote *RHEL* system where the *Sentinel* should be installed.

Install the Yum repository

```
dnf install -y https://yum.opennms.org/repofiles/opennms-repo-stable-rhel8.noarch.rpm
rpm --import https://yum.opennms.org/OPENNMS-GPG-KEY
```

Install the Sentinel package

```
dnf -y install meridian-sentinel
```

With the successful installed packages the *Sentinel* is installed in the following directory structure:

```
[root@localhost /opt/sentinel]# $ tree -L 1
.
|-- bin
|-- COPYING
|-- data
|-- deploy
|-- etc
|-- lib
`-- system
```

The Sentinel's startup configuration can be changed by editing the `/etc/sysconfig/sentinel` file. It allows to override the defaults used at startup including:

- Location of the JDK
- Memory usage
- User to run as

Step 3: Starting the Sentinel and test access to Karaf Shell

Configure systemd to start Sentinel on system boot

```
systemctl enable sentinel
```

Startup Sentinel

```
systemctl start sentinel
```

Test access to Karaf shell with user admin and password admin and exit with <ctrl-d>

```
ssh -p 8301 admin@localhost
```

Step 4: Configure Sentinel to communicate with OpenNMS Meridian

Login to the Karaf Shell on the system where your Sentinel is installed with SSH

```
ssh -p 8301 admin@localhost
```

Configure the Sentinel's location and endpoint URLs for communication with OpenNMS Meridian

```
[root@localhost /root]# $ ssh -p 8201 admin@localhost
...
admin@sentinel()> config:edit org.opennms.sentinel.controller
admin@sentinel()> config:property-set location Office-Pittsboro
admin@sentinel()> config:property-set http-url http://opennms-fqdn:8980/opennms
admin@sentinel()> config:property-set broker-url failover:tcp://opennms-fqdn:61616
admin@sentinel()> config:update
```

> 💡 Include the `failover:` portion of the broker URL to allow the *Sentinel* to re-establish connectivity on failure. For a reference on the different URL formats, see ActiveMQ URI Protocols.

> ℹ️ Even if the id, location and http-url must be set the same ways as for *Minion*, this may change in future versions of *Sentinel*.

Configure the credentials to use when communicating with OpenNMS Meridian

```
admin@sentinel()> scv:set opennms.http minion minion
admin@sentinel()> scv:set opennms.broker minion minion
```

Username and password is explicitly set to `minion` as it is assumed that they share the same credentials and roles.

> 💡 Another way to configure credentials is to use the `scvcli` utility in your *Sentinel* `bin` directory.

Example of configuring credentials with the command line utility `scvcli`

```
[root@localhost /root]# $ cd /opt/sentinel
[root@localhost /opt/sentinel]# $ ./bin/scvcli set opennms.http minion minion
[root@localhost /opt/sentinel]# $ ./bin/scvcli set opennms.broker minion minion
```

Restart the Sentinel after updating the credentials

```
[root@localhost /root]# $ systemctl restart sentinel
```

> ℹ️ The credentials are configured separately since they are encrypted on disk.

Step 5: Verifying Connectivity

Connect to Karaf Shell of the Sentinel

```
ssh -p 8301 admin@localhost
```

Verify connectivity with the OpenNMS Meridian

```
admin@sentinel()> feature:install sentinel-core
admin@sentinel> health:check
Verifying the health of the container

Verifying installed bundles       [ Success  ]
Connecting to OpenNMS ReST API    [ Success  ]

=> Everything is awesome
admin@sentinel()>
```

> ℹ️ The `health:check` command is a newer and more flexibel version of the original `minion:ping` command. Therefore on *Sentinel* there is no equivalent such as `sentinel:ping`.

Chapter 3. Minion with custom messaging system

Minions and *OpenNMS Meridian* communicate via a messaging system. By default, an embedded *ActiveMQ* broker is used. *OpenNMS Meridian* is designed to work with different messaging systems and based on the system requirements or workload, an alternative to *ActiveMQ* can be used. In general, the communication between *OpenNMS Meridian* and *Minion* is provided by two patterns:

- *Remote Producer Calls (RPCs)* are used to issue specific tasks (such as a request to poll or perform data collection) from an *OpenNMS Meridian* instance to a *Minion* in a remote location.
  - These calls are normally self-contained and include all of the meta-data and information required for them to be performed.
- The *Sink* pattern is used to send unsolicited messages (i.e. *Syslog*, *SNMP Traps* or *Flows*) received from a *Minion* to an *OpenNMS Meridian* instance

High level components used for communication between OpenNMS Meridian and Minions
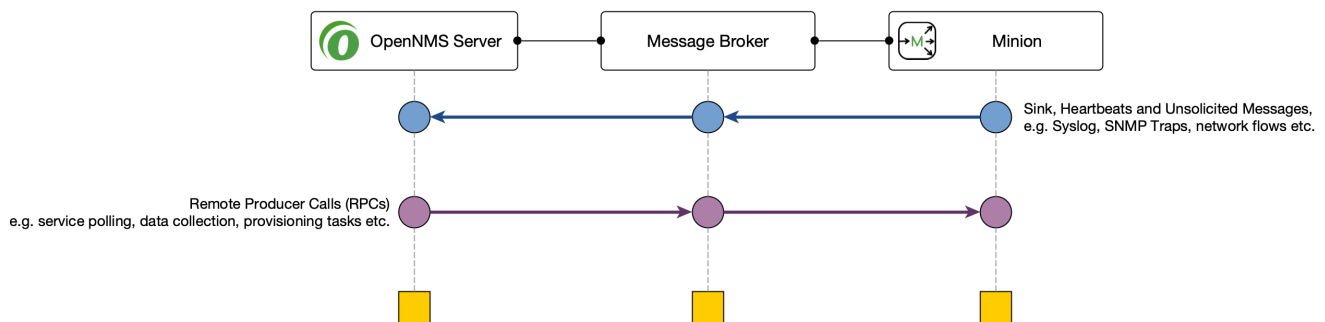
This section describes how you can setup *OpenNMS Meridian* to use other supported messaging systems for the communication with *Minions*.

3.1. Setup using Apache Kafka

This section describes how to use *Apache Kafka* as a messaging system between *OpenNMS Meridian* and *Minions* in a remote location.

3.1.1. Objectives

- Configure *OpenNMS Meridian* to forward *RPC* to a *Minion*
- Configure *Minion* to forward messages over the *Sink* component to an *OpenNMS Meridian* instance
- Disable the embedded *Active MQ* message broker on the *Minion*.
- Verify the functionality on the *Minion* using the `health:check` command and ensure the *Minion* is registered and monitored in the *OpenNMS Meridian* web interface

3.1.2. Before you begin

The following requirements should be satisfied before you can start with this tutorial:

- At least a minimal Kafka system up and running If you want to start in a lab, the Apache Kafka Quickstart guide is a good starting point

- An instance running with *OpenNMS Meridian* and at least one deployed *Minion*

- Communication between *OpenNMS Meridian*, *Minion* and *Apache Kafka* is possible on TCP port 9092

Network topology used for the following configuration example
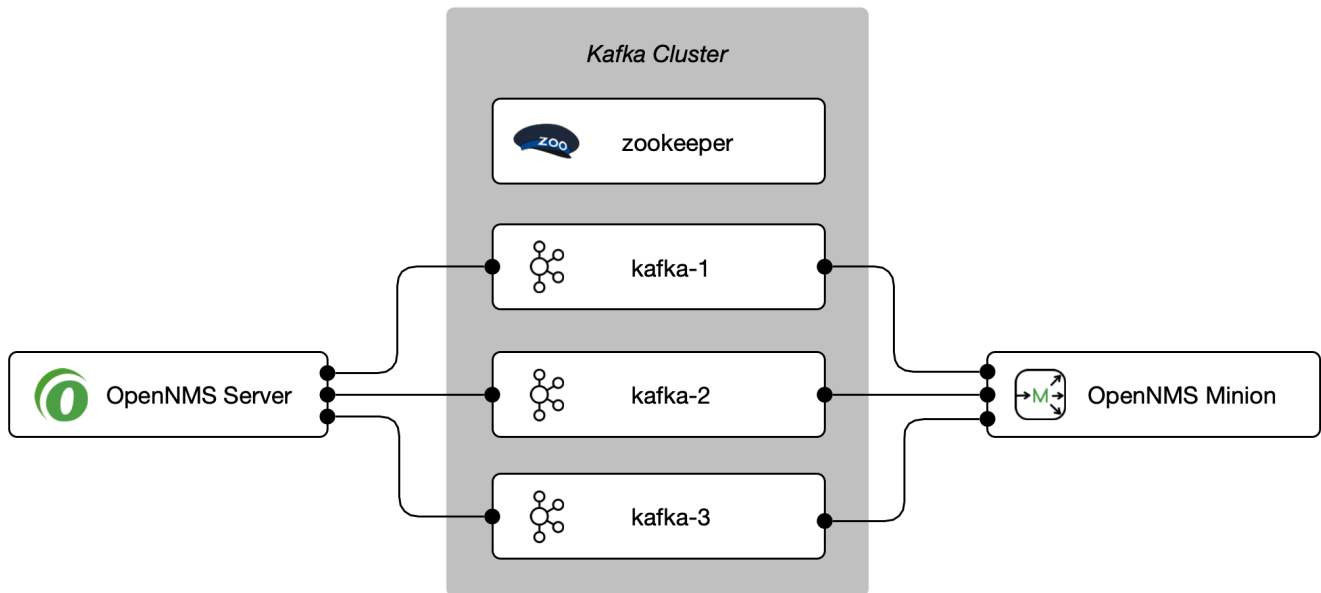
ℹ️	The example is used to describe how the components need to be configured. IP addresses and hostnames need to be adjusted accordingly.

ℹ️	You can add more than one Kafka server to the configuration. The driver will attempt to connect to the first entry. If that is successful the whole broker topology will be discovered and will be known by the client. The other entries are only used if the connection to the first entry fails.

3.1.3. Configure OpenNMS Meridian

Step 1: Set Kafka as RPC strategy and add Kafka server

```
cat <<EOF >${OPENNMS_HOME}/etc/opennms.properties.d/kafka.properties
org.opennms.core.ipc.rpc.strategy=kafka
org.opennms.core.ipc.rpc.kafka.bootstrap.servers=kafka-1:9092,kafka-2:9092,kafka-3:9092
EOF
```

Step 2: Set Kafka as Sink strategy and add Kafka server

```
cat <<EOF >>${OPENNMS_HOME}/etc/opennms.properties.d/kafka.properties
# Ensure that messages are not consumed from Kafka until the system has fully
initialized
org.opennms.core.ipc.sink.initialSleepTime=60000
org.opennms.core.ipc.sink.strategy=kafka
org.opennms.core.ipc.sink.kafka.bootstrap.servers=kafka-1:9092,kafka-2:9092,kafka-
3:9092
EOF
```

Step 3: Restart OpenNMS Meridian

```
systemctl restart opennms
```

3.1.4. Configure Minion

Step 1: Disable ActiveMQ for RPC and Sink

Disable ActiveMQ on Minion startup

```
cat <<EOF >${MINION_HOME}/etc/featuresBoot.d/disable-activemq.boot
!minion-jms
!opennms-core-ipc-rpc-jms
!opennms-core-ipc-sink-camel
EOF
```

Step 2: Enable Kafka for RPC and Sink

```
cat <<EOF >${MINION_HOME}/etc/featuresBoot.d/kafka.boot
opennms-core-ipc-rpc-kafka
opennms-core-ipc-sink-kafka
EOF
```

Step 3: Configure Kafka server

Add Kafka server for RPC communication

```
cat <<EOF >${MINION_HOME}/etc/org.opennms.core.ipc.rpc.kafka.cfg
bootstrap.servers=kafka-1:9092,kafka-2:9092,kafka-3:9092
acks=1
EOF
```

Add Kafka server for Sink communication

```
cat <<EOF >${MINION_HOME}/etc/org.opennms.core.ipc.sink.kafka.cfg
bootstrap.servers=kafka-1:9092,kafka-2:9092,kafka-3:9092
acks=1
EOF
```

Step 4: Restart Minion to apply changes

```
systemctl restart minion
```

Step 5: Verify Kafka configuration and connectivity

Login to Karaf Shell

```
ssh admin@localhost -p 8201
```

Test if Kafka RPC and Sink feature is started

```
feature:list | grep opennms-core-ipc-rpc-kafka
opennms-core-ipc-rpc-kafka  | 25.0.0          | x        | Started

feature:list | grep opennms-core-ipc-sink-kafka
opennms-core-ipc-sink-kafka | 25.0.0          | x        | Started
```

Test connectivity to Kafka

```
health:check
Verifying the health of the container

Connecting to OpenNMS ReST API   [ Success  ]
Verifying installed bundles      [ Success  ]
Connecting to Kafka from RPC     [ Success  ]
Connecting to Kafka from Sink    [ Success  ]

=> Everything is awesome
```

Step 6. Verify Minion functionality

Ensure the Minion is registered in the OpenNMS Meridian web interface

1. Login as Administrator

2. Configure OpenNMS

3. Manage Minions

4. Minion should be registered and should be shown as "Up"

5. Click on the name of the Minion and go to the node detail page

6. Verify if the services on the loopback interface *JMX-Minion*, *Minion-Heartbeat*, *Minion-RPC* are monitored and "Up"

3.1.5. Tuning Apache Kafka

The configuration is shipped with sane defaults, but depending on the size and network topology it can be required to tune the *Apache Kafka* environment to meet certain needs. *Apache Kafka* options can be set directly in the `org.opennms.core.ipc.rpc.kafka.cfg` and `org.opennms.core.ipc.sink.kafka.cfg` file.

Alternatively: *Kafka* producer/consumer options can be set by defining additional system properties prefixed with `org.opennms.core.ipc.rpc.kafka` and `org.opennms.core.ipc.sink.kafka`.

You can find available configuration parameters for *Kafka* here:

- Producer Configs for RPC communication
- New Consumer Configs for Sink communication

Multiple OpenNMS Meridian instances

Topics will be automatically created and are prefixed by default with `OpenNMS`. If you want to use an *Apache Kafka* cluster with multiple *OpenNMS Meridian* instances, the topic prefix can be customized by setting `org.opennms.core.ipc.rpc.kafka.group.id` and `org.opennms.core.ipc.sink.kafka.group.id` to a string value which identifies your instance.

Tips for Kafka

💡	For Kafka RPC, the number of partitions should always be greater than the number of minions at a location. When there are multiple locations, partitions >= max number of minions at a location.

💡	By default, Kafka RPC supports buffers greater than >1MB by splitting large buffer into chunks of 900KB(912600). Max buffer size (900KB, by default) can be configured by setting org.opennms.core.ipc.rpc.kafka.max.buffer.size (in bytes).

💡	Default time to live (time at which request will expire) is 20000 msec (20sec). It can be changed by configuring system property `org.opennms.core.ipc.rpc.kafka.ttl` in msec.

Chapter 4. Install other versions than stable

Installation packages are available for different releases of *OpenNMS Meridian* or *Minion*. You will need to choose which release you would like to run and then configure your package repository to point to that release. Configuring a package repository will enable you to install and update the software by using standard Linux software update tools like *yum* and *apt*.

The following package repositories are available:

Table 6. OpenNMS package repositories

Release	Description
`stable`	Latest stable release. This version is recommended for all users.
`testing`	Release candidate for the next stable release.
`snapshot`	Latest successful development build, the "nightly" build.

To install a different release the repository files have to be installed and manually modified.

On *RHEL* systems you can install a snapshot repository with:

```
yum -y install https://yum.opennms.org/repofiles/opennms-repo-snapshot-
rhel7.noarch.rpm
```

i	For branches use `repofiles/opennms-repo-branches-${branch-name}-rhel7.noarch.rpm`.

The installation procedure is the same as with the stable version.

Chapter 5. Setup Minion with a config file

Beside manually configuring a *Minion* instance via the *Karaf CLI* it is possibleto modify and deploy its configuration file through configuration management tools. The configuration file is located in `${MINION_HOME}/etc/org.opennms.minion.controller.cfg`. All configurations set in *Karaf CLI* will be persisted in this configuration file which can also be populated through configuration management tools.

Configuration file for Minion

```
id = 00000000-0000-0000-0000-deadbeef0001
location = MINION
broker-url = tcp://myopennms.example.org:61616
http-url = http://myopennms.example.org:8980/opennms
```

The *Minion* needs to be restarted when this configuration file is changed.

💡	In case the credentials needs to be set through the *CLI* with configuration management tools or scripts, the `${MINION_HOME}/bin/client` command can be used which allows to execute *Karaf* commands through the Linux shell.

Chapter 6. Running in non-root environments

This section provides information running *OpenNMS Meridian* and *Minions* processes in non-root environments. Running with a system user have restricted possibilites. This section describes how to configure your *Linux* system related to:

- sending *ICMP* packages as an unprivileged user
- receiving *Syslog* on ports < 1023, e.g. 514/udp
- receiving *SNMP Trap* on ports < 1023,e.g. 162/udp

6.1. Send ICMP as non-root

By default, *Linux* does not allow regular users to perform `ping` operations from arbitrary programs (including *Java*). To enable the *Minion* or *OpenNMS Meridian* to ping properly, you must set a `sysctl` option.

Enable User Ping (Running System)d

```
# run this command as root to allow ping by any user (does not survive reboots)
sysctl net.ipv4.ping_group_range='0 429496729'
```

If you wish to restrict the range further, use the *GID* for the user the *Minion* or *OpenNMS Meridian* will run as, rather than `429496729`.

To enable this permanently, create a file in `/etc/sysctl.d/` to set the range:

/etc/sysctl.d/99-zzz-non-root-icmp.conf

```
# we start this filename with "99-zzz-" to make sure it's last, after anything else
that might have set it
net.ipv4.ping_group_range=0 429496729
```

6.2. Trap reception as non-root

If you wish your *Minion* or *OpenNMS Meridian* to listen to *SNMP Traps*, you will need to configure your firewall to port forward from the privileged trap port (162) to the Minion's default trap listener on port 1162.

Forward 162 to 1162 with Firewalld

```
# enable masquerade to allow port-forwards
firewall-cmd --add-masquerade
# forward port 162 TCP and UDP to port 1162 on localhost
firewall-cmd --add-forward-port=port=162:proto=udp:toport=1162:toaddr=127.0.0.1
firewall-cmd --add-forward-port=port=162:proto=tcp:toport=1162:toaddr=127.0.0.1
```

6.3. Syslog reception as non-root

If you wish your *Minion* or *OpenNMS Meridian* to listen to syslog messages, you will need to configure your firewall to port forward from the privileged *Syslog* port (514) to the Minion's default syslog listener on port 1514.

Forward 514 to 1514 with Firewalld

```
# enable masquerade to allow port-forwards
firewall-cmd --add-masquerade
# forward port 514 TCP and UDP to port 1514 on localhost
firewall-cmd --add-forward-port=port=514:proto=udp:toport=1514:toaddr=127.0.0.1
firewall-cmd --add-forward-port=port=514:proto=tcp:toport=1514:toaddr=127.0.0.1
```

Chapter 7. Use R for statistical computing

R is a free software environment for statistical computing and graphics. *OpenNMS Meridian* can leverage the power of *R* for forecasting and advanced calculations on collected time series data.

OpenNMS Meridian interfaces with *R* via *stdin* and *stdout*, and for this reason, *R* must be installed on the same host as *OpenNMS Meridian.* Note that installing *R* is optional, and not required by any of the core components.

> ❗ The *R* integration is not supported on *Microsoft Windows* systems.

7.1. Install R on RHEL

Ensure the dnf plugin config-manager is installed

```
dnf -y install dnf-plugins-core
```

Enable the PowerTools repository for R dependencies

```
dnf config-manager --set-enabled PowerTools
```

Install the epel-release repository with R packages

```
dnf -y install epel-release
```

Install R-core package

```
dnf -y install R-core
```

7.2. Install R on Debian

Install R

```
apt -y install r-recommended
```

Chapter 8. Using a different Time Series Storage

OpenNMS Meridian stores performance data in a time series storage which is by default [JRobin](#). For different scenarios it is useful to switch to a different time series storage. The following implementations are supported:

Table 7. Supported Time Series Databasees

JRobin	*JRobin* is a clone of *RRDTool* written in *Java*, it does not fully cover the latest feature set of *RRDTool* and is the default when you install *OpenNMS Meridian*. Data is stored on the local file system of the OpenNMS Meridian node. Depending on I/O capabilities it works good for small to medium sized installations.
RRDTool	*RRDTool* is active maintained and the de-facto standard dealing with time series data. Data is stored on the local file system of the OpenNMS Meridian node. Depending on I/O capabilities it works good for small to medium sized installations.
Newts	[Newts](#) is a database schema for [Cassandra](#). The time series is stored on a dedicated *Cassandra* cluster which gives growth flexibility and allows to persist time series data in a large scale.

This section describes how to configure *OpenNMS Meridian* to use *RRDTool* and *Newts*.

> The way how data is stored in the different time series databases makes it extremely hard to migrate from one technology to another. Data loss can't be prevented when you switch from one to another.

8.1. RRDtool

In most *Open Source* applications, [RRDtool](#) is often used and is the de-facto open standard for *Time Series Data*. The basic installation of *OpenNMS Meridian* comes with *JRobin* but it is simple to switch the system to use *RRDtool* to persist *Time Series Data*. This section describes how to install *RRDtool*, the *jrrd2 OpenNMS Java Interface* and how to configure *OpenNMS Meridian* to use it.

8.1.1. Install RRDTool on RHEL

> Following this guide does not cover data migration from *JRobin* to *RRDTool*.

> To install *jrrd2* enable the OpenNMS YUM repository ensure the repositories are enabled. You can enable them with `dnf config-manager --enable opennms-repo-stable-*`.

Step 1: Install RRDTool and the jrrd2 interface

Installation on RHEL

```
dnf -y install rrdtool jrrd2
```

Step 2: Configure OpenNMS Meridian to use RRDTool

```
cat << EOF | sudo tee /opt/opennms/etc/opennms.properties.d/timeseries.properties
org.opennms.rrd.strategyClass=org.opennms.netmgt.rrd.rrdtool.MultithreadedJniRrdStrategy
org.opennms.rrd.interfaceJar=/usr/share/java/jrrd2.jar
opennms.library.jrrd2=/usr/lib64/libjrrd2.so
org.opennms.web.graphs.engine=rrdtool # optional, unset if you want to keep Backshift as default
EOF
```

> 💡 The visualization with the graph engine is optional. You can still use the default graphing engine `backshift` by not setting the `org.opennms.web.graphs.engine` property and use the system default.

Step 3: Restart OpenNMS Meridian and verify setup

```
find /opt/opennms/share/rrd -iname "*.rrd"
```

With the first data collection, *RRDTool* files with extension *.rrd* will be created. The *JRobin* files with extension *.jrb* are not used anymore and are not deleted automatically.

8.1.2. Reference

The following configuration files have references to the *RRDTool* binary and may be changed if you have a customized *RRDTool* setup.

Table 8. References to the RRDtool binary

Configuration file	Property
`opennms.properties`	`rrd.binary=/usr/bin/rrdtool`
`response-adhoc-graph.properties`	`command.prefix=/usr/bin/rrdtool`
`response-graph.properties`	`command.prefix=/usr/bin/rrdtool` `info.command=/usr/bin/rrdtool`
`snmp-adhoc-graph.properties`	`command.prefix=/usr/bin/rrdtool`
`snmp-graph.properties`	`command.prefix=/usr/bin/rrdtool` `command=/usr/bin/rrdtool info`

8.2. Newts for Time Series data

Newts is a time-series data schema for Apache Cassandra. It enables horizontally scale capabilities for your time series storage and is an alternative to JRobin and RRDtool.

The *Cassandra* cluster design, setup, sizing, tuning and operation is out of scope for this section. To install and set up a *Cassandra* cluster please follow the Cassandra installation instructions. For further information see Cassandra Getting Started Guide.

> 💡 To avoid unwanted updates disable the *Cassandra* repository on *DNF/YUM* based distributions or use `apt-mark hold cassandra` on *APT* based distributions.

> ❗ For simplicity we use the `${OPENNMS_HOME}/bin/newts init` command which initializes a *Newts* keyspace for you and the defaults are not optimal tuned for a production-ready environment. If you want to build a production environment please consult Sizing Cassandra for Newts and planning Anti-patterns in Cassandra articles.

8.2.1. Objectives

- Configure *OpenNMS Meridian* to use an existing *Cassandra* cluster
- Initializing the *Newts* keyspace using `newts init` with *STCS* without production-ready tuning
- Verify time series data is stored and can be accessed

8.2.2. Before you begin

- A running instance of *OpenNMS Meridian* running on Linux
- Working data collection and response time metrics from *Collectd* and *Pollerd*
- *Cassandra* cluster with access to the *Cassandra* client port `TCP/9042`

> ⚠️ It is currently not supported to initialize the *Newts* keyspace from *Microsoft Windows Server* operating system. *Microsoft Windows* based *Cassandra* server can be part of the cluster, but keyspace initialization is only possible using a *Linux* operating system.

8.2.3. Configure OpenNMS Meridian to use Newts

Step 1: Configure Cassandra endpoints, keyspace and time series strategy

```
cat << EOF | sudo tee /opt/opennms/etc/opennms.properties.d/timeseries.properties
# Configure storage strategy
org.opennms.rrd.storeByForeignSource=true①
org.opennms.timeseries.strategy=newts②

# Configure Newts time series storage connection
org.opennms.newts.config.hostname={cassandra-ip1,cassandra-ip2}③
org.opennms.newts.config.keyspace=newts④
org.opennms.newts.config.port=9042⑤

# One year in seconds
org.opennms.newts.config.ttl=31540000

# Seven days in seconds
org.opennms.newts.config.resource_shard=604800
EOF
```

① Associate time series data by the foreign ID instead of the database generated Node-ID

② Set time-series strategy to use `newts`

③ Host or IP addresses of the *Cassandra* cluster nodes can be a comma-separated list

④ Name of the keyspace which is initialized and used

⑤ Port to connect to *Cassandra*

Step 2: Initialize the *Newts* **schema in** *Cassandra*

```
${OPENNMS_HOME}/bin/newts init
```

Step 3: Verify if the keyspace was properly initialized

Connect to a *Cassandra* node with a CQL shell

```
cd $CASSANDRA_HOME/bin
./cqlsh

use newts;
describe table terms;
describe table samples;
```

Step 4: Apply changes and verify your configuration

```
systemctl restart opennms
```

Go to the Node detail page from a *SNMP* managed device and verify if you response time graphs for *ICMP* and *Node-level Performance data.*