OpenShift Container Platform 4.18

Advanced networking

Installing and configuring OpenShift Container Platform clusters

Last Updated: 2026-04-01

OpenShift Container Platform 4.18 Advanced networking

Installing and configuring OpenShift Container Platform clusters

Legal Notice

Abstract

This document provides information about installing OpenShift Container Platform and details about some configuration processes.

Table of Contents

CHAPTER 1. VERIFYING CONNECTIVITY TO AN ENDPOINT

The Cluster Network Operator (CNO) runs a controller, the connectivity check controller, that performs a connection health check between resources within your cluster. By reviewing the results of the health checks, you can diagnose connection problems or eliminate network connectivity as the cause of an issue that you are investigating.

1.1. CONNECTION HEALTH CHECKS THAT ARE PERFORMED

To verify that cluster resources are reachable, a TCP connection is made to each of the following cluster API services:

- Kubernetes API server service
- Kubernetes API server endpoints
- OpenShift API server service
- OpenShift API server endpoints
- Load balancers

To verify that services and service endpoints are reachable on every node in the cluster, a TCP connection is made to each of the following targets:

- Health check target service
- Health check target endpoints

1.2. IMPLEMENTATION OF CONNECTION HEALTH CHECKS

The connectivity check controller orchestrates connection verification checks in your cluster. The results for the connection tests are stored in **PodNetworkConnectivity** objects in the **openshift-network-diagnostics** namespace. Connection tests are performed every minute in parallel.

The Cluster Network Operator (CNO) deploys several resources to the cluster to send and receive connectivity health checks:

Health check source

This program deploys in a single pod replica set managed by a **Deployment** object. The program consumes **PodNetworkConnectivity** objects and connects to the **spec.targetEndpoint** specified in each object.

Health check target

A pod deployed as part of a daemon set on every node in the cluster. The pod listens for inbound health checks. The presence of this pod on every node allows for the testing of connectivity to each node.

You can configure the nodes which network connectivity sources and targets run on with a node selector. Additionally, you can specify permissible *tolerations* for source and target pods. The configuration is defined in the singleton **cluster** custom resource of the **Network** API in the **config.openshift.io/v1** API group.

Pod scheduling occurs after you have updated the configuration. Therefore, you must apply node labels that you intend to use in your selectors before updating the configuration. Labels applied after updating your network connectivity check pod placement are ignored.

Refer to the default configuration in the following YAML:

Default configuration for connectivity source and target pods

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  # ...
    networkDiagnostics: 1
      mode: "All" 2
      sourcePlacement: 3
        nodeSelector:
          checkNodes: groupA
        tolerations:
        - key: myTaint
          effect: NoSchedule
          operator: Exists
      targetPlacement: 4
        nodeSelector:
          checkNodes: groupB
        tolerations:
        - key: myOtherTaint
          effect: NoExecute
          operator: Exists
```

1 Specifies the network diagnostics configuration. If a value is not specified or an empty object is specified, and **spec.disableNetworkDiagnostics=true** is set in the **network.operator.openshift.io** custom resource named **cluster**, network diagnostics are disabled. If set, this value overrides **spec.disableNetworkDiagnostics=true**.

2 Specifies the diagnostics mode. The value can be the empty string, **All**, or **Disabled**. The empty string is equivalent to specifying **All**.

3 Optional: Specifies a selector for connectivity check source pods. You can use the **nodeSelector** and **tolerations** fields to further specify the **sourceNode** pods. These are optional for both source and target pods. You can omit them, use both, or use only one of them.

4 Optional: Specifies a selector for connectivity check target pods. You can use the **nodeSelector** and **tolerations** fields to further specify the **targetNode** pods. These are optional for both source and target pods. You can omit them, use both, or use only one of them.

1.3. CONFIGURING POD CONNECTIVITY CHECK PLACEMENT

As a cluster administrator, you can configure which nodes the connectivity check pods run by modifying the **network.config.openshift.io** object named **cluster**.

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

1. Edit the connectivity check configuration by entering the following command:

```
$ oc edit network.config.openshift.io cluster
```

2. In the text editor, update the **networkDiagnostics** stanza to specify the node selectors that you want for the source and target pods.

3. Save your changes and exit the text editor.

Verification

- Verify that the source and target pods are running on the intended nodes by entering the following command:

```
$ oc get pods -n openshift-network-diagnostics -o wide
```

Example output

```
NAME                                  READY   STATUS    RESTARTS   AGE     IP            NODE
NOMINATED NODE   READINESS GATES
network-check-source-84c69dbd6b-p8f7n   1/1     Running   0          9h      10.131.0.8    ip-10-0-40-
197.us-east-2.compute.internal   <none>           <none>
network-check-target-46pct              1/1     Running   0          9h      10.131.0.6    ip-10-0-40-197.us-
east-2.compute.internal   <none>           <none>
network-check-target-8kwgf              1/1     Running   0          9h      10.128.2.4    ip-10-0-95-74.us-east-
2.compute.internal    <none>           <none>
network-check-target-jc6n7              1/1     Running   0          9h      10.129.2.4    ip-10-0-21-151.us-
east-2.compute.internal   <none>           <none>
network-check-target-lvwnn              1/1     Running   0          9h      10.128.0.7    ip-10-0-17-129.us-
east-2.compute.internal   <none>           <none>
network-check-target-nslvj              1/1     Running   0          9h      10.130.0.7    ip-10-0-89-148.us-east-
2.compute.internal   <none>           <none>
network-check-target-z2sfx              1/1     Running   0          9h      10.129.0.4    ip-10-0-60-253.us-
east-2.compute.internal   <none>           <none>
```

1.4. PODNETWORKCONNECTIVITYCHECK OBJECT FIELDS

The **PodNetworkConnectivityCheck** object fields are described in the following tables.

Table 1.1. PodNetworkConnectivityCheck object fields

Field	Type	Description

Field	Type	Description
metadata.name	**string**	The name of the object in the following format: **<source>-to-<target>**. The destination described by **<target>** includes one of following strings: • **load-balancer-api-external** • **load-balancer-api-internal** • **kubernetes-apiserver-endpoint** • **kubernetes-apiserver-service-cluster** • **network-check-target** • **openshift-apiserver-endpoint** • **openshift-apiserver-service-cluster**
metadata.namespace	**string**	The namespace that the object is associated with. This value is always **openshift-network-diagnostics**.
spec.sourcePod	**string**	The name of the pod where the connection check originates, such as **network-check-source-596b4c6566-rgh92**.
spec.targetEndpoint	**string**	The target of the connection check, such as **api.devcluster.example.com:6443**.
spec.tlsClientCert	**object**	Configuration for the TLS certificate to use.
spec.tlsClientCert.name	**string**	The name of the TLS certificate used, if any. The default value is an empty string.
status	**object**	An object representing the condition of the connection test and logs of recent connection successes and failures.
status.conditions	**array**	The latest status of the connection check and any previous statuses.
status.failures	**array**	Connection test logs from unsuccessful attempts.
status.outages	**array**	Connect test logs covering the time periods of any outages.
status.successes	**array**	Connection test logs from successful attempts.

The following table describes the fields for objects in the **status.conditions** array:

Table 1.2. status.conditions

Field	Type	Description
lastTransitionTime	**string**	The time that the condition of the connection transitioned from one status to another.
message	**string**	The details about last transition in a human readable format.
reason	**string**	The last status of the transition in a machine readable format.
status	**string**	The status of the condition.
type	**string**	The type of the condition.

The following table describes the fields for objects in the **status.conditions** array:

Table 1.3. status.outages

Field	Type	Description
end	**string**	The timestamp from when the connection failure is resolved.
endLogs	**array**	Connection log entries, including the log entry related to the successful end of the outage.
message	**string**	A summary of outage details in a human readable format.
start	**string**	The timestamp from when the connection failure is first detected.
startLogs	**array**	Connection log entries, including the original failure.

1.4.1. Connection log fields

The fields for a connection log entry are described in the following table. The object is used in the following fields:

- **status.failures[]**
- **status.successes[]**
- **status.outages[].startLogs[]**
- **status.outages[].endLogs[]**

Table 1.4. Connection log object

Field	Type	Description
latency	**string**	Records the duration of the action.
message	**string**	Provides the status in a human readable format.
reason	**string**	Provides the reason for status in a machine readable format. The value is one of **TCPConnect**, **TCPConnectError**, **DNSResolve**, **DNSError**.
success	**boolean**	Indicates if the log entry is a success or failure.
time	**string**	The start time of connection check.

1.5. VERIFYING NETWORK CONNECTIVITY FOR AN ENDPOINT

As a cluster administrator, you can verify the connectivity of an endpoint, such as an API server, load balancer, service, or pod, and verify that network diagnostics is enabled.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Confirm that network diagnostics are enable by entering the following command:

```
$ oc get network.config.openshift.io cluster -o yaml
```

Example output

```
# ...
status:
# ...
  conditions:
  - lastTransitionTime: "2024-05-27T08:28:39Z"
    message: ""
    reason: AsExpected
    status: "True"
    type: NetworkDiagnosticsAvailable
```

2. List the current **PodNetworkConnectivityCheck** objects by entering the following command:

```
$ oc get podnetworkconnectivitycheck -n openshift-network-diagnostics
```

Example output

```
NAME                                                                                                                                AGE
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1   73m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2   75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-apiserver-service-cluster                               75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-default-service-cluster                                 75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-external                                         75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-load-balancer-api-internal                                         75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-master-0            75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-master-1            75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-master-2            75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh      74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-worker-c-n8mbf      74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-ci-ln-x5sv9rb-f76d1-4rzrp-worker-d-4hnrz      74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-network-check-target-service-cluster                               75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0    75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-1    75m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-2    74m
network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-openshift-apiserver-service-cluster                                75m
```

3. View the connection test logs:

   a. From the output of the previous command, identify the endpoint that you want to review the connectivity logs for.

   b. View the object by entering the following command:

      ```
      $ oc get podnetworkconnectivitycheck <name> \
        -n openshift-network-diagnostics -o yaml
      ```

      where **<name>** specifies the name of the **PodNetworkConnectivityCheck** object.

      Example output

      ```
      apiVersion: controlplane.operator.openshift.io/v1alpha1
      kind: PodNetworkConnectivityCheck
      metadata:
      ```

```
  name: network-check-source-ci-ln-x5sv9rb-f76d1-4rzrp-worker-b-6xdmh-to-kubernetes-
apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0
  namespace: openshift-network-diagnostics
  ...
spec:
  sourcePod: network-check-source-7c88f6d9f-hmg2f
  targetEndpoint: 10.0.0.4:6443
  tlsClientCert:
    name: ""
status:
  conditions:
  - lastTransitionTime: "2021-01-13T20:11:34Z"
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
      connection to 10.0.0.4:6443 succeeded'
    reason: TCPConnectSuccess
    status: "True"
    type: Reachable
  failures:
  - latency: 2.241775ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
      to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
      connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:10:34Z"
  - latency: 2.582129ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
      to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
      connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:09:34Z"
  - latency: 3.483578ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: failed
      to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443: connect:
      connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:08:34Z"
  outages:
  - end: "2021-01-13T20:11:34Z"
    endLogs:
    - latency: 2.032018ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        tcp connection to 10.0.0.4:6443 succeeded'
      reason: TCPConnect
      success: true
      time: "2021-01-13T20:11:34Z"
    - latency: 2.241775ms
      message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
        failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
        connect: connection refused'
      reason: TCPConnectError
      success: false
      time: "2021-01-13T20:10:34Z"
    - latency: 2.582129ms
```

```
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:09:34Z"
  - latency: 3.483578ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:08:34Z"
  message: Connectivity restored after 2m59.999789186s
  start: "2021-01-13T20:08:34Z"
  startLogs:
  - latency: 3.483578ms
    message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0:
      failed to establish a TCP connection to 10.0.0.4:6443: dial tcp 10.0.0.4:6443:
      connect: connection refused'
    reason: TCPConnectError
    success: false
    time: "2021-01-13T20:08:34Z"
successes:
- latency: 2.845865ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:14:34Z"
- latency: 2.926345ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:13:34Z"
- latency: 2.895796ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:12:34Z"
- latency: 2.696844ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:11:34Z"
- latency: 1.502064ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:10:34Z"
- latency: 1.388857ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
```

```
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:09:34Z"
- latency: 1.906383ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:08:34Z"
- latency: 2.089073ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:07:34Z"
- latency: 2.156994ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:06:34Z"
- latency: 1.777043ms
  message: 'kubernetes-apiserver-endpoint-ci-ln-x5sv9rb-f76d1-4rzrp-master-0: tcp
    connection to 10.0.0.4:6443 succeeded'
  reason: TCPConnect
  success: true
  time: "2021-01-13T21:05:34Z"
```

CHAPTER 2. CHANGING THE MTU FOR THE CLUSTER NETWORK

As a cluster administrator, you can change the maximum transmission unit (MTU) for the cluster network after cluster installation. This change is disruptive as cluster nodes must be rebooted to finalize the MTU change.

2.1. ABOUT THE CLUSTER MTU

During installation, the cluster network maximum transmission unit (MTU) is set automatically based on the primary network interface MTU of cluster nodes. Typically, you do not need to override the detected MTU, but in some instances you must override it.

You might want to change the MTU of the cluster network for one of the following reasons:

- The MTU detected during cluster installation is not correct for your infrastructure.
- Your cluster infrastructure now requires a different MTU, such as from the addition of nodes that need a different MTU for optimal performance.

Only the OVN-Kubernetes network plugin supports changing the MTU value.

2.1.1. Service interruption considerations

When you initiate a maximum transmission unit (MTU) change on your cluster the following effects might impact service availability:

- At least two rolling reboots are required to complete the migration to a new MTU. During this time, some nodes are not available as they restart.
- Specific applications deployed to the cluster with shorter timeout intervals than the absolute TCP timeout interval might experience disruption during the MTU change.

2.1.2. MTU value selection

When planning your maximum transmission unit (MTU) migration there are two related but distinct MTU values to consider.

- **Hardware MTU**: This MTU value is set based on the specifics of your network infrastructure.
- **Cluster network MTU**: This MTU value is always less than your hardware MTU to account for the cluster network overlay overhead. The specific overhead is determined by your network plugin. For OVN-Kubernetes, the overhead is **100** bytes.

If your cluster requires different MTU values for different nodes, you must subtract the overhead value for your network plugin from the lowest MTU value that is used by any node in your cluster. For example, if some nodes in your cluster have an MTU of **9001**, and some have an MTU of **1500**, you must set this value to **1400**.

IMPORTANT

To avoid selecting an MTU value that is not acceptable by a node, verify the maximum MTU value (**maxmtu**) that is accepted by the network interface by using the **ip -d link** command.

2.1.3. How the migration process works

The following table summarizes the migration process by segmenting between the user-initiated steps in the process and the actions that the migration performs in response.

Table 2.1. Live migration of the cluster MTU

User-initiated steps	OpenShift Container Platform activity
Set the following values in the Cluster Network Operator configuration: • **spec.migration.mtu.machine.to** • **spec.migration.mtu.network.from** • **spec.migration.mtu.network.to**	**Cluster Network Operator (CNO)**: Confirms that each field is set to a valid value. • The **mtu.machine.to** must be set to either the new hardware MTU or to the current hardware MTU if the MTU for the hardware is not changing. This value is transient and is used as part of the migration process. If you set a hardware MTU different from the current value, you must manually configure it to persist. Use methods such as a machine config, DHCP setting, or kernel command line. • The **mtu.network.from** field must equal the **network.status.clusterNetworkMTU** field, which is the current MTU of the cluster network. • The **mtu.network.to** field must be set to the target cluster network MTU. It must be lower than the hardware MTU to allow for the overlay overhead of the network plugin. For OVN-Kubernetes, the overhead is **100** bytes. If the values provided are valid, the CNO writes out a new temporary configuration with the MTU for the cluster network set to the value of the **mtu.network.to** field. **Machine Config Operator (MCO)**: Performs a rolling reboot of each node in the cluster.
Reconfigure the MTU of the primary network interface for the nodes on the cluster. You can use one of the following methods to accomplish this: • Deploying a new NetworkManager connection profile with the MTU change • Changing the MTU through a DHCP server setting • Changing the MTU through boot parameters	N/A

User-initiated steps	OpenShift Container Platform activity
Set the **mtu** value in the CNO configuration for the network plugin and set **spec.migration** to **null**.	**Machine Config Operator (MCO)**: Performs a rolling reboot of each node in the cluster with the new MTU configuration.

2.2. CHANGING THE CLUSTER NETWORK MTU

As a cluster administrator, you can increase or decrease the maximum transmission unit (MTU) for your cluster.

IMPORTANT

You cannot roll back an MTU value for nodes during the MTU migration process, but you can roll back the value after the MTU migration process completes.

The migration is disruptive and nodes in your cluster might be temporarily unavailable as the MTU update takes effect.

The following procedures describe how to change the cluster network MTU by using machine configs, Dynamic Host Configuration Protocol (DHCP), or an ISO image. If you use either the DHCP or ISO approaches, you must refer to configuration artifacts that you kept after installing your cluster to complete the procedure.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster using an account with **cluster-admin** permissions.
- You have identified the target MTU for your cluster. The MTU for the OVN-Kubernetes network plugin must be set to **100** less than the lowest hardware MTU value in your cluster.
- If your nodes are physical machines, ensure that the cluster network and the connected network switches support jumbo frames.
- If your nodes are virtual machines (VMs), ensure that the hypervisor and the connected network switches support jumbo frames.

2.2.1. Checking the current cluster MTU value

To ensure network stability and performance in a hybrid environment where part of your cluster is in the cloud and part is an on-premise environment, you can obtain the current maximum transmission unit (MTU) for the cluster network.

Procedure

- To obtain the current MTU for the cluster network, enter the following command:

```
$ oc describe network.config cluster
```

Example output

```
...
Status:
  Cluster Network:
    Cidr:               10.217.0.0/22
    Host Prefix:        23
  Cluster Network MTU:  1400
  Network Type:         OVNKubernetes
  Service Network:
    10.217.4.0/23
...
```

2.2.2. Preparing your hardware MTU configuration

Many ways exist to configure the hardware maximum transmission unit (MTU) for your cluster nodes. The following examples show only the most common methods. Verify the correctness of your infrastructure MTU. Select your preferred method for configuring your hardware MTU in the cluster nodes.

Procedure

1. Prepare your configuration for the hardware MTU:

   - If your hardware MTU is specified with DHCP, update your DHCP configuration such as with the following dnsmasq configuration:

     ```
     dhcp-option-force=26,<mtu>
     ```

     where:

     <mtu>

     Specifies the hardware MTU for the DHCP server to advertise.

   - If your hardware MTU is specified with a kernel command line with PXE, update that configuration accordingly.

   - If your hardware MTU is specified in a NetworkManager connection configuration, complete the following steps. This approach is the default for OpenShift Container Platform if you do not explicitly specify your network configuration with DHCP, a kernel command line, or some other method. Your cluster nodes must all use the same underlying network configuration for the following procedure to work unmodified.

     a. Find the primary network interface by entering the following command:

        ```
        $ oc debug node/<node_name> -- chroot /host nmcli -g connection.interface-name c
        show ovs-if-phys0
        ```

        where:

        <node_name>

        Specifies the name of a node in your cluster.

     b. Create the following **NetworkManager** configuration in the **<interface>-mtu.conf** file:

```
[connection-<interface>-mtu]
match-device=interface-name:<interface>
ethernet.mtu=<mtu>
```

where:

<interface>

Specifies the primary network interface name.

<mtu>

Specifies the new hardware MTU value.

2.2.3. Creating MachineConfig objects

Use the following procedure to create the **MachineConfig** objects.

Procedure

1. Create two **MachineConfig** objects, one for the control plane nodes and another for the worker nodes in your cluster:

   a. Create the following Butane config in the **control-plane-interface.bu** file:

NOTE

The Butane version you specify in the config file should match the OpenShift Container Platform version and always ends in **0**. For example, **4.18.0**. See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.18.0
metadata:
  name: 01-control-plane-interface
  labels:
    machineconfiguration.openshift.io/role: master
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf (1)
      contents:
        local: <interface>-mtu.conf (2)
      mode: 0600
```

(1) Specifies the **NetworkManager** connection name for the primary network interface.

(2) Specifies the local filename for the updated **NetworkManager** configuration file from an earlier step.

   b. Create the following Butane config in the **worker-interface.bu** file:

NOTE

The Butane version you specify in the config file should match the OpenShift Container Platform version and always ends in **0**. For example, **4.18.0**. See "Creating machine configs with Butane" for information about Butane.

```
variant: openshift
version: 4.18.0
metadata:
  name: 01-worker-interface
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
    - path: /etc/NetworkManager/conf.d/99-<interface>-mtu.conf 1
      contents:
        local: <interface>-mtu.conf 2
      mode: 0600
```

1. Specifies the **NetworkManager** connection name for the primary network interface.
2. Specifies the local filename for the updated **NetworkManager** configuration file from an earlier step.

2. Create **MachineConfig** objects from the Butane configs by running the following command:

```
$ for manifest in control-plane-interface worker-interface; do
    butane --files-dir . $manifest.bu > $manifest.yaml
  done
```

WARNING

Do not apply these machine configs until explicitly instructed later in this procedure. Applying these machine configs now causes a loss of stability for the cluster.

2.2.4. Beginning the MTU migration

Start the maximum transmission unit (MTU) migration by specifying the migration configuration for the cluster network and machine interfaces. The Machine Config Operator performs a rolling reboot of the nodes to prepare the cluster for the MTU change.

Procedure

1. To begin the MTU migration, specify the migration configuration by entering the following command. The Machine Config Operator performs a rolling reboot of the nodes in the cluster in preparation for the MTU change.

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": <overlay_from>, "to": <overlay_to> } ,
"machine": { "to" : <machine_to> } } } } }'
```

where:

<overlay_from>

Specifies the current cluster network MTU value.

<overlay_to>

Specifies the target MTU for the cluster network. This value is set relative to the value of **<machine_to>**. For OVN-Kubernetes, this value must be **100** less than the value of **<machine_to>**.

<machine_to>

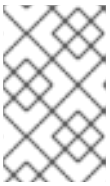
Specifies the MTU for the primary network interface on the underlying host network.

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": { "mtu": { "network": { "from": 1400, "to": 9000 } , "machine": { "to" :
9100} } } } }'
```

2. As the Machine Config Operator updates machines in each machine config pool, the Operator reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get machineconfigpools
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.



NOTE

By default, the Machine Config Operator updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

2.2.5. Verifying the machine configuration

Verify the machine configuration on your hosts to confirm that the maximum transmission unit (MTU) migration applied successfully. Checking the configuration state and system settings help ensures that the nodes use the correct migration script.

Procedure

- Confirm the status of the new machine configuration on the hosts:

  a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

  ```
  $ oc describe node | egrep "hostname|machineconfig"
  ```

  Example output

```
kubernetes.io/hostname=master-0
machineconfiguration.openshift.io/currentConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/desiredConfig: rendered-master-
c53e221d9d24e1c8bb6ee89dd3d8ad7b
machineconfiguration.openshift.io/reason:
machineconfiguration.openshift.io/state: Done
```

b. Verify that the following statements are true:

- The value of **machineconfiguration.openshift.io/state** field is **Done**.
- The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

c. To confirm that the machine config is correct, enter the following command:

```
$ oc get machineconfig <config_name> -o yaml | grep ExecStart
```

where:

<config_name>

Specifies the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

The machine config must include the following update to the systemd configuration:

```
ExecStart=/usr/local/bin/mtu-migration.sh
```

2.2.6. Applying the new hardware MTU value

Use the following procedure to apply the new hardware maximum transmission unit (MTU) value.

Procedure

1. Update the underlying network interface MTU value:

- If you are specifying the new MTU with a NetworkManager connection configuration, enter the following command. The MachineConfig Operator automatically performs a rolling reboot of the nodes in your cluster.

```
$ for manifest in control-plane-interface worker-interface; do
    oc create -f $manifest.yaml
  done
```

- If you are specifying the new MTU with a DHCP server option or a kernel command line and PXE, make the necessary changes for your infrastructure.

2. As the Machine Config Operator updates machines in each machine config pool, the Operator reboots each node one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get machineconfigpools
```

A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

NOTE

By default, the Machine Config Operator updates one machine per pool at a time, causing the total time the migration takes to increase with the size of the cluster.

3. Confirm the status of the new machine configuration on the hosts:

   a. To list the machine configuration state and the name of the applied machine configuration, enter the following command:

   ```
   $ oc describe node | egrep "hostname|machineconfig"
   ```

   Example output

   ```
   kubernetes.io/hostname=master-0
   machineconfiguration.openshift.io/currentConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/desiredConfig: rendered-master-
   c53e221d9d24e1c8bb6ee89dd3d8ad7b
   machineconfiguration.openshift.io/reason:
   machineconfiguration.openshift.io/state: Done
   ```

   Verify that the following statements are true:

   - The value of **machineconfiguration.openshift.io/state** field is **Done**.
   - The value of the **machineconfiguration.openshift.io/currentConfig** field is equal to the value of the **machineconfiguration.openshift.io/desiredConfig** field.

   b. To confirm that the machine config is correct, enter the following command:

   ```
   $ oc get machineconfig <config_name> -o yaml | grep path:
   ```

   where:

   <config_name>

   Specifies the name of the machine config from the **machineconfiguration.openshift.io/currentConfig** field.

   If the machine config is successfully deployed, the previous output contains the **/etc/NetworkManager/conf.d/99-<interface>-mtu.conf** file path and the **ExecStart=/usr/local/bin/mtu-migration.sh** line.

2.2.7. Finalizing the MTU migration

Finalize the MTU migration to apply the new maximum transmission unit (MTU) settings to the OVN-Kubernetes network plugin. This updates the cluster configuration and triggers a rolling reboot of the nodes to complete the process.

Procedure

1. To finalize the MTU migration, enter the following command for the OVN-Kubernetes network plugin:

```
$ oc patch Network.operator.openshift.io cluster --type=merge --patch \
  '{"spec": { "migration": null, "defaultNetwork":{ "ovnKubernetesConfig": { "mtu": <mtu> }}}}'
```

   where:

   <mtu>

   Specifies the new cluster network MTU that you specified with **<overlay_to>**.

2. After finalizing the MTU migration, each machine config pool node is rebooted one by one. You must wait until all the nodes are updated. Check the machine config pool status by entering the following command:

```
$ oc get machineconfigpools
```

   A successfully updated node has the following status: **UPDATED=true**, **UPDATING=false**, **DEGRADED=false**.

Verification

1. To get the current MTU for the cluster network, enter the following command:

```
$ oc describe network.config cluster
```

2. Get the current MTU for the primary network interface of a node:

   a. To list the nodes in your cluster, enter the following command:

```
$ oc get nodes
```

   b. To obtain the current MTU setting for the primary network interface on a node, enter the following command:

```
$ oc adm node-logs <node> -u ovs-configuration | grep configure-ovs.sh | grep mtu |
grep <interface> | head -1
```

   where:

   <node>

   Specifies a node from the output from the previous step.

   <interface>

   Specifies the primary network interface name for the node.

   Example output

```
ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 8051
```

2.3. ADDITIONAL RESOURCES

- Using advanced networking options for PXE and ISO installations
- Manually creating NetworkManager profiles in key file format
- Configuring a dynamic Ethernet connection using nmcli

CHAPTER 3. NETWORK BONDING CONSIDERATIONS

You can use network bonding, also known as *link aggregration*, to combine many network interfaces into a single, logical interface. This means that you can use different modes for handling how network traffic distributes across bonded interfaces. Each mode provides fault tolerance and some modes provide load balancing capabilities to your network. Red Hat supports Open vSwitch (OVS) bonding and kernel bonding.

3.1. OPEN VSWITCH (OVS) BONDING

With an OVS bonding configuration, you create a single, logical interface by connecting each physical network interface controller (NIC) as a port to a specific bond. This single bond then handles all network traffic, effectively replacing the function of individual interfaces.

Consider the following architectural layout for OVS bridges that interact with OVS interfaces:

- A network interface uses a bridge Media Access Control (MAC) address for managing protocol-level traffic and other administrative tasks, such as IP address assignment.
- The physical MAC addresses of physical interfaces do not handle traffic.
- OVS handles all MAC address management at the OVS bridge level.

This layout simplifies bond interface management as bonds act as data paths, where centralized MAC address management happens at the OVS bridge level.

For OVS bonding, you can select either **active-backup** mode or **balance-slb** mode. A bonding mode specifies the policy for how bond interfaces get used during network transmission.

3.1.1. Enable active-backup mode for your cluster

The **active-backup** mode provides fault tolerance for network connections by switching to a backup link where the primary link fails.

The mode specifies the following ports for your cluster:

- An active port where one physical interface sends and receives traffic at any given time.
- A standby port where all other ports act as backup links and continously monitor their link status.

During a failover process, if an active port or its link fails, the bonding logic switches all network traffic to a standby port. This standby port becomes the new active port. For failover to work, all ports in a bond must share the same Media Access Control (MAC) address.

3.1.2. Enabling OVS balance-slb mode for your cluster

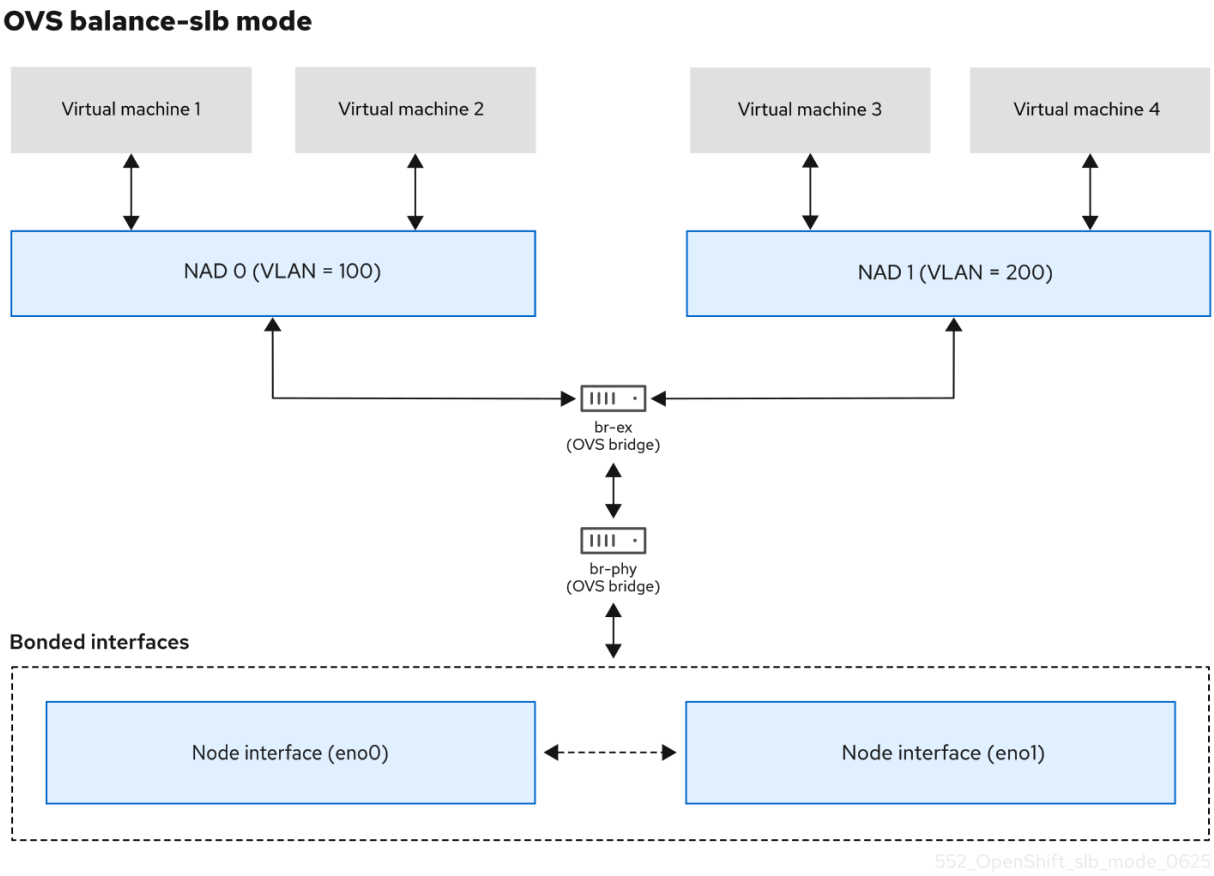
You can enable the Open vSwitch (OVS) **balance-slb** mode so that two or more physical interfaces can share their network traffic. A **balance-slb** mode interface can give source load balancing (SLB) capabilities to a cluster that runs virtualization workloads, without requiring load balancing negotiation with the network switch.

Currently, source load balancing runs on a bond interface, where the interface connects to an auxiliary bridge, such as **br-phy**. Source load balancing balances only across different Media Access Control (MAC) address and virtual local area network (VLAN) combinations. Note that all OVN-Kubernetes pod

traffic uses the same MAC address and VLAN, so this traffic cannot be load balanced across many physical interfaces.

The following diagram shows **balance-slb** mode on a simple cluster infrastructure layout. Virtual machines (VMs) connect to specific localnet **NetworkAttachmentDefinition** (NAD) custom resource definition (CRDs), **NAD 0** or **NAD 1**. Each NAD provides VMs with access to the underlying physical network, supporting VLAN-tagged or untagged traffic. A **br-ex** OVS bridge receives traffic from VMs and passes the traffic to the next OVS bridge, **br-phy**. The **br-phy** bridge functions as the controller for the SLB bond. The SLB bond balances traffic from different VM ports over the physical interface links, such as **eno0** and **eno1**. Additionally, ingress traffic from either physical interface can pass through the set of OVS bridges to reach the VMs.

Figure 3.1. OVS balance-slb mode operating on a localnet with two NAD CRDs



You can integrate the **balance-slb** mode interface into primary or secondary network types by using OVS bonding. Note the following points about OVS bonding:

- Supports the OVN-Kubernetes CNI plugin and easily integrates with the plugin.
- Natively supports **balance-slb** mode.

Prerequisites

- You have more than one physical interface attached to your primary network and you defined the interfaces in a **MachineConfig** file.
- You created a manifest object and defined a customized **br-ex** bridge in the object configuration file.

- You have more than one physical interfaces attached to your primary network and you defined the interfaces in a NAD CRD file.

Procedure

1. For each bare-metal host that exists in a cluster, in the **install-config.yaml** file for your cluster define a **networkConfig** section similar to the following example:

```
apiVersion: v1
kind: InstallConfig
metadata:
  name: <cluster-name>
# ...
networkConfig:
  interfaces:
    - name: enp1s0
      type: ethernet
      state: up
      ipv4:
        dhcp: true
        enabled: true
      ipv6:
        enabled: false
    - name: enp2s0
      type: ethernet
      state: up
      mtu: 1500
      ipv4:
        dhcp: true
        enabled: true
      ipv6:
        dhcp: true
        enabled: true
    - name: enp3s0
      type: ethernet
      state: up
      mtu: 1500
      ipv4:
        enabled: false
      ipv6:
        enabled: false
# ...
```

where:

enp1s0

The interface for the provisioned network interface controller (NIC).

enp2s0

The first bonded interface that pulls in the Ignition config file for the bond interface.

mtu

Manually set the **br-ex** maximum transmission unit (MTU) on the bond ports.

enp3s0

The second bonded interface is part of a minimal configuration that pulls ignition during cluster installation.

2. Define each network interface in an NMState configuration file:

Example NMState configuration file that defines many network interfaces

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
# ...
ovn:
  bridge-mappings:
    - localnet: localnet-network
      bridge: br-ex
      state: present
interfaces:
  - name: br-ex
    type: ovs-bridge
    state: up
    bridge:
      allow-extra-patch-ports: true
      port:
        - name: br-ex
        - name: patch-ex-to-phy
    ovs-db:
      external_ids:
        bridge-uplink: "patch-ex-to-phy"
  - name: br-ex
    type: ovs-interface
    state: up
    mtu: 1500
    ipv4:
      enabled: true
      dhcp: true
      auto-route-metric: 48
    ipv6:
      enabled: false
      dhcp: false
      auto-route-metric: 48
  - name: br-phy
    type: ovs-bridge
    state: up
    bridge:
      allow-extra-patch-ports: true
      port:
        - name: patch-phy-to-ex
        - name: ovs-bond
          link-aggregation:
            mode: balance-slb
            port:
              - name: enp2s0
              - name: enp3s0
  - name: patch-ex-to-phy
    type: ovs-interface
    state: up
```

```
    patch:
      peer: patch-phy-to-ex
  - name: patch-phy-to-ex
    type: ovs-interface
    state: up
    patch:
      peer: patch-ex-to-phy
  - name: enp1s0
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      enabled: true
    ipv6:
      enabled: false
  - name: enp2s0
    type: ethernet
    state: up
    mtu: 1500
    ipv4:
      enabled: false
    ipv6:
      enabled: false
  - name: enp3s0
    type: ethernet
    state: up
    mtu: 1500
    ipv4:
      enabled: false
    ipv6:
      enabled: false
# ...
```

where:

mtu

Manually set the **br-ex** MTU on the bond ports.

3. Use the **base64** command to encode the interface content of the NMState configuration file:

```
$ base64 -w0  <nmstate_configuration>.yml
```

<nmstate_configuration>: Where the **-w0** option prevents line wrapping during the base64 encoding operation.

4. Create **MachineConfig** manifest files for the **master** role and the **worker** role. Ensure that you embed the base64-encoded string from an earlier command into each **MachineConfig** manifest file. The following example manifest file configures the **master** role for all nodes that exist in a cluster. You can also create a manifest file for **master** and **worker** roles specific to a node.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: master
```

```
  name: 10-br-ex-master
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,
<base64_encoded_nmstate_configuration>
        mode: 0644
        overwrite: true
        path: /etc/nmstate/openshift/cluster.yml
```

where:

name

The name of the policy.

source

Writes the encoded base64 information to the specified path.

path

Specify the path to the **cluster.yml** file. For each node in your cluster, you can specify the short hostname path to your node, such as **<node_short_hostname>**.yml.

5. Save each **MachineConfig** manifest file to the **./<installation_directory>/manifests** directory, where **<installation_directory>** is the directory in which the installation program creates files. The Machine Config Operator (MCO) takes the content from each manifest file and consistently applies the content to all selected nodes during a rolling update.

3.2. KERNEL BONDING

You can use kernel bonding, which is a built-in Linux kernel function where link aggregation can exist among many Ethernet interfaces, to create a single logical physical interface. Kernel bonding allows multiple network interfaces to be combined into a single logical interface, which can enhance network performance by increasing bandwidth and providing redundancy in case of a link failure.

Kernel bonding is the default mode if no bond interfaces depend on OVS bonds. This bonding type does not give the same level of customization as supported OVS bonding.

For **kernel-bonding** mode, the bond interfaces exist outside, which means they are not in the data path, of the bridge interface. Network traffic in this mode is not sent or received on the bond interface port but instead requires additional bridging capabilities for MAC address assignment at the kernel level.

If you enabled **kernel-bonding** mode on network controller interfaces (NICs) for your nodes, you must specify a Media Access Control (MAC) address failover. This configuration prevents node communication issues with the bond interfaces, such as **eno1f0** and **eno2f0**.

Red Hat supports only the following value for the **fail_over_mac** parameter:

- **0**: Specifies the **none** value, which disables MAC address failover so that all interfaces receive the same MAC address as the bond interface. This is the default value.

Red Hat does not support the following values for the **fail_over_mac** parameter:

- **1**: Specifies the **active** value and sets the MAC address of the primary bond interface to always remain the same as active interfaces. If during a failover, the MAC address of an interface changes, the MAC address of the bond interface changes to match the new MAC address of the interface.
- **2**: Specifies the **follow** value so that during a failover, an active interface gets the MAC address of the bond interface and a formerly active interface receives the MAC address of the newly active interface.

CHAPTER 4. USING THE STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

As a cluster administrator, you can use the Stream Control Transmission Protocol (SCTP) on a bare-metal cluster.

4.1. SUPPORT FOR SCTP ON OPENSHIFT CONTAINER PLATFORM

As a cluster administrator, you can enable SCTP on the hosts in the cluster. On Red Hat Enterprise Linux CoreOS (RHCOS), the SCTP module is disabled by default.

SCTP is a reliable message based protocol that runs on top of an IP network.

When enabled, you can use SCTP as a protocol with pods, services, and network policy. A **Service** object must be defined with the **type** parameter set to either the **ClusterIP** or **NodePort** value.

4.1.1. Example configurations using SCTP protocol

You can configure a pod or service to use SCTP by setting the **protocol** parameter to the **SCTP** value in the pod or service object.

In the following example, a pod is configured to use SCTP:

```
apiVersion: v1
kind: Pod
metadata:
  namespace: project1
  name: example-pod
spec:
  containers:
    - name: example-pod
...
      ports:
        - containerPort: 30100
          name: sctpserver
          protocol: SCTP
```

In the following example, a service is configured to use SCTP:

```
apiVersion: v1
kind: Service
metadata:
  namespace: project1
  name: sctpserver
spec:
...
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30100
      targetPort: 30100
  type: ClusterIP
```

In the following example, a **NetworkPolicy** object is configured to apply to SCTP network traffic on port **80** from any pods with a specific label:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: allow-sctp-on-http
spec:
  podSelector:
    matchLabels:
      role: web
  ingress:
  - ports:
    - protocol: SCTP
      port: 80
```

4.2. ENABLING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

As a cluster administrator, you can load and enable the blacklisted SCTP kernel module on worker nodes in your cluster.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a file named **load-sctp-module.yaml** that contains the following YAML definition:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: load-sctp-module
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: /etc/modprobe.d/sctp-blacklist.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,
        - path: /etc/modules-load.d/sctp-load.conf
          mode: 0644
          overwrite: true
          contents:
            source: data:,sctp
```

2. To create the **MachineConfig** object, enter the following command:

```
$ oc create -f load-sctp-module.yaml
```

3. Optional: To watch the status of the nodes while the MachineConfig Operator applies the configuration change, enter the following command. When the status of a node transitions to **Ready**, the configuration update is applied.

```
$ oc get nodes
```

4.3. VERIFYING STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) IS ENABLED

You can verify that SCTP is working on a cluster by creating a pod with an application that listens for SCTP traffic, associating it with a service, and then connecting to the exposed service.

Prerequisites

- Access to the internet from the cluster to install the **nc** package.
- Install the OpenShift CLI (**oc**).
- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a pod starts an SCTP listener:
   a. Create a file named **sctp-server.yaml** that defines a pod with the following YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpserver
  labels:
    app: sctpserver
spec:
  containers:
    - name: sctpserver
      image: registry.access.redhat.com/ubi9/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
      ports:
        - containerPort: 30102
          name: sctpserver
          protocol: SCTP
```

   b. Create the pod by entering the following command:

```
$ oc create -f sctp-server.yaml
```

2. Create a service for the SCTP listener pod.

a. Create a file named **sctp-service.yaml** that defines a service with the following YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: sctpservice
  labels:
    app: sctpserver
spec:
  type: NodePort
  selector:
    app: sctpserver
  ports:
    - name: sctpserver
      protocol: SCTP
      port: 30102
      targetPort: 30102
```

b. To create the service, enter the following command:

```
$ oc create -f sctp-service.yaml
```

3. Create a pod for the SCTP client.

a. Create a file named **sctp-client.yaml** with the following YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: sctpclient
  labels:
    app: sctpclient
spec:
  containers:
    - name: sctpclient
      image: registry.access.redhat.com/ubi9/ubi
      command: ["/bin/sh", "-c"]
      args:
        ["dnf install -y nc && sleep inf"]
```

b. To create the **Pod** object, enter the following command:

```
$ oc apply -f sctp-client.yaml
```

4. Run an SCTP listener on the server.

a. To connect to the server pod, enter the following command:

```
$ oc rsh sctpserver
```

b. To start the SCTP listener, enter the following command:

```
$ nc -l 30102 --sctp
```

5. Connect to the SCTP listener on the server.

   a. Open a new terminal window or tab in your terminal program.

   b. Obtain the IP address of the **sctpservice** service. Enter the following command:

   ```
   $ oc get services sctpservice -o go-template='{{.spec.clusterIP}}{{"\n"}}'
   ```

   c. To connect to the client pod, enter the following command:

   ```
   $ oc rsh sctpclient
   ```

   d. To start the SCTP client, enter the following command. Replace **<cluster_IP>** with the cluster IP address of the **sctpservice** service.

   ```
   # nc <cluster_IP> 30102 --sctp
   ```

CHAPTER 5. ASSOCIATING SECONDARY INTERFACES METRICS TO NETWORK ATTACHMENTS

Administrators can use the **pod_network_info** metric to classify and monitor secondary network interfaces. The metric does this by adding a label that identifies the interface type, typically based on the associated **NetworkAttachmentDefinition** resource.

5.1. EXTENDING SECONDARY NETWORK METRICS FOR MONITORING

Secondary devices, or interfaces, are used for different purposes. Metrics from secondary network interfaces need to be classified to allow for effective aggregation and monitoring.

Exposed metrics contain the interface but do not specify where the interface originates. This is workable when there are no additional interfaces. However, relying on interface names alone becomes problematic when secondary interfaces are added because it is difficult to identify their purpose and use their metrics effectively..

When adding secondary interfaces, their names depend on the order in which they are added. Secondary interfaces can belong to distinct networks that can each serve a different purposes.

With **pod_network_name_info** it is possible to extend the current metrics with additional information that identifies the interface type. In this way, it is possible to aggregate the metrics and to add specific alarms to specific interface types.

The network type is generated from the name of the **NetworkAttachmentDefinition** resource, which distinguishes different secondary network classes. For example, different interfaces belonging to different networks or using different CNIs use different network attachment definition names.

5.2. NETWORK METRICS DAEMON

The Network Metrics Daemon is a daemon component that collects and publishes network related metrics.

The kubelet is already publishing network related metrics you can observe. These metrics are:

- **container_network_receive_bytes_total**
- **container_network_receive_errors_total**
- **container_network_receive_packets_total**
- **container_network_receive_packets_dropped_total**
- **container_network_transmit_bytes_total**
- **container_network_transmit_errors_total**
- **container_network_transmit_packets_total**
- **container_network_transmit_packets_dropped_total**

The labels in these metrics contain, among others:

- Pod name

- Pod namespace
- Interface name (such as **eth0**)

These metrics work well until new interfaces are added to the pod, for example via Multus, as it is not clear what the interface names refer to.

The interface label refers to the interface name, but it is not clear what that interface is meant for. In case of many different interfaces, it would be impossible to understand what network the metrics you are monitoring refer to.

This is addressed by introducing the new **pod_network_name_info** described in the following section.

5.3. METRICS WITH NETWORK NAME

The Network Metrics daemonset publishes a **pod_network_name_info** gauge metric, with a fixed value of **0**.

Example of pod_network_name_info

```
pod_network_name_info{interface="net0",namespace="namespacename",network_name="nadnamespace/firstNAD",pod="podname"} 0
```

The network name label is produced using the annotation added by Multus. It is the concatenation of the namespace the network attachment definition belongs to, plus the name of the network attachment definition.

The new metric alone does not provide much value, but combined with the network related **container_network_*** metrics, it offers better support for monitoring secondary networks.

Using a **promql** query like the following ones, it is possible to get a new metric containing the value and the network name retrieved from the **k8s.v1.cni.cncf.io/network-status** annotation:

```
(container_network_receive_bytes_total) + on(namespace,pod,interface) group_left(network_name) ( pod_network_name_info )
(container_network_receive_errors_total) + on(namespace,pod,interface) group_left(network_name) ( pod_network_name_info )
(container_network_receive_packets_total) + on(namespace,pod,interface) group_left(network_name) ( pod_network_name_info )
(container_network_receive_packets_dropped_total) + on(namespace,pod,interface) group_left(network_name) ( pod_network_name_info )
(container_network_transmit_bytes_total) + on(namespace,pod,interface) group_left(network_name) ( pod_network_name_info )
(container_network_transmit_errors_total) + on(namespace,pod,interface) group_left(network_name) ( pod_network_name_info )
(container_network_transmit_packets_total) + on(namespace,pod,interface) group_left(network_name) ( pod_network_name_info )
(container_network_transmit_packets_dropped_total) + on(namespace,pod,interface) group_left(network_name)
```

CHAPTER 6. USING PTP HARDWARE

6.1. ABOUT PRECISION TIME PROTOCOL IN OPENSHIFT CLUSTER NODES

Precision Time Protocol (PTP) is used to synchronize clocks in a network. When used in conjunction with hardware support, PTP is capable of sub-microsecond accuracy, and is more accurate than Network Time Protocol (NTP).

IMPORTANT

If your **openshift-sdn** cluster with PTP uses the User Datagram Protocol (UDP) for hardware time stamping and you migrate to the OVN-Kubernetes plugin, the hardware time stamping cannot be applied to primary interface devices, such as an Open vSwitch (OVS) bridge. As a result, UDP version 4 configurations cannot work with a **br-ex** interface.

You can configure **linuxptp** services and use PTP-capable hardware in OpenShift Container Platform cluster nodes.

Use the OpenShift Container Platform web console or OpenShift CLI (**oc**) to install PTP by deploying the PTP Operator. The PTP Operator creates and manages the **linuxptp** services and provides the following features:

- Discovery of the PTP-capable devices in the cluster.
- Management of the configuration of **linuxptp** services.
- Notification of PTP clock events that negatively affect the performance and reliability of your application with the PTP Operator **cloud-event-proxy** sidecar.

NOTE

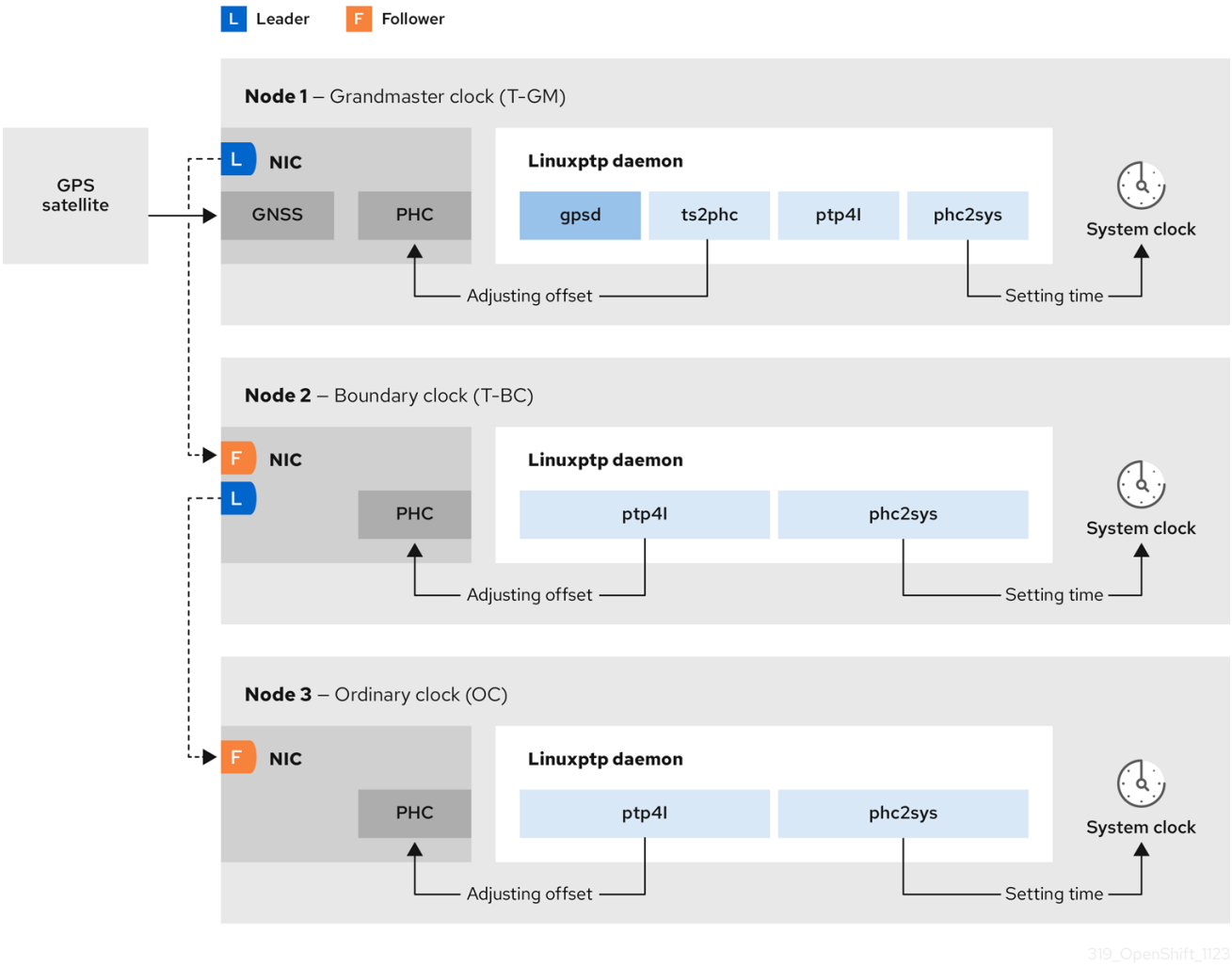
The PTP Operator works with PTP-capable devices on clusters provisioned only on bare-metal infrastructure.

6.1.1. Elements of a PTP domain

PTP uses a leader-follower hierarchy of grandmaster, boundary, and ordinary clocks to synchronize time with high precision across network nodes.

The clocks synchronized by PTP are organized in a leader-follower hierarchy. The hierarchy is created and updated automatically by the best master clock (BMC) algorithm, which runs on every clock. Follower clocks are synchronized to leader clocks, and follower clocks can themselves be the source for other downstream clocks.

Figure 6.1. PTP nodes in the network



The three primary types of PTP clocks are described in the following sections.

Grandmaster clock

The grandmaster clock provides standard time information to other clocks across the network and ensures accurate and stable synchronization. It writes time stamps and responds to time requests from other clocks. Grandmaster clocks synchronize to a global navigation satellite system (GNSS) time source. The grandmaster clock is the authoritative source of time in the network and is responsible for providing time synchronization to all other devices.

Boundary clock

The boundary clock has ports in two or more communication paths and can be a source and a destination to other destination clocks at the same time. The boundary clock works as a destination clock upstream. The destination clock receives the timing message, adjusts for delay, and then creates a new source time signal to pass down the network. The boundary clock produces a new timing packet that is still correctly synced with the source clock and can reduce the number of connected devices reporting directly to the source clock.

Ordinary clock

The ordinary clock has a single port connection that can play the role of source or destination clock, depending on its position in the network. The ordinary clock can read and write timestamps.

6.1.1.1. Advantages of PTP over NTP

One of the main advantages that PTP has over NTP is the hardware support present in various network

interface controllers (NIC) and network switches. The specialized hardware allows PTP to account for delays in message transfer and improves the accuracy of time synchronization. To achieve the best possible accuracy, it is recommended that all networking components between PTP clocks are PTP hardware enabled.

Hardware-based PTP provides optimal accuracy, since the NIC can timestamp the PTP packets at the exact moment they are sent and received. Compare this to software-based PTP, which requires additional processing of the PTP packets by the operating system.

Before enabling PTP, ensure that NTP is disabled for the required nodes. You can disable the chrony time service (**chronyd**) using a **MachineConfig** custom resource.

IMPORTANT

Although PTP provides superior accuracy over NTP, you can configure NTP as a backup time source for PTP Grandmaster (T-GM) clocks. In GNSS-to-NTP failover configurations, the system uses GNSS as the primary time source through PTP, but automatically fails over to NTP (**chronyd**) if the GNSS signal is lost or degraded. This provides resilient timekeeping even when the primary GNSS time source is temporarily unavailable. For more information about configuring GNSS-to-NTP failover, see *Configuring GNSS/NTP failover*.

Additional resources

- Disabling chrony time service

6.1.2. Overview of linuxptp and gpsd in OpenShift Container Platform nodes

OpenShift Container Platform uses the PTP Operator with **linuxptp** and **gpsd** packages for high precision network synchronization. The **linuxptp** package provides tools and daemons for PTP timing in networks. Cluster hosts with Global Navigation Satellite System (GNSS) capable NICs use **gpsd** to interface with GNSS clock sources.

The **linuxptp** package includes the **ts2phc**, **pmc**, **ptp4l**, and **phc2sys** programs for system clock synchronization.

ts2phc

ts2phc synchronizes the PTP hardware clock (PHC) across PTP devices with a high degree of precision. **ts2phc** is used in grandmaster clock configurations. It receives the precision timing signal a high precision clock source such as Global Navigation Satellite System (GNSS). GNSS provides an accurate and reliable source of synchronized time for use in large distributed networks. GNSS clocks typically provide time information with a precision of a few nanoseconds.
The **ts2phc** system daemon sends timing information from the grandmaster clock to other PTP devices in the network by reading time information from the grandmaster clock and converting it to PHC format. PHC time is used by other devices in the network to synchronize their clocks with the grandmaster clock.

pmc

pmc implements a PTP management client (**pmc**) according to IEEE standard 1588.1588. **pmc** provides basic management access for the **ptp4l** system daemon. **pmc** reads from standard input and sends the output over the selected transport, printing any replies it receives.

ptp4l

ptp4l implements the PTP boundary clock and ordinary clock and runs as a system daemon. **ptp4l** does the following:

- Synchronizes the PHC to the source clock with hardware time stamping
- Synchronizes the system clock to the source clock with software time stamping

phc2sys

phc2sys synchronizes the system clock to the PHC on the network interface controller (NIC). The **phc2sys** system daemon continuously monitors the PHC for timing information. When it detects a timing error, the PHC corrects the system clock.

The **gpsd** package includes the **ubxtool**, **gspipe**, **gpsd**, programs for GNSS clock synchronization with the host clock.

ubxtool

ubxtool CLI allows you to communicate with a u-blox GPS system. The **ubxtool** CLI uses the u-blox binary protocol to communicate with the GPS.

gpspipe

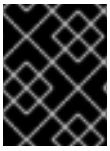
gpspipe connects to **gpsd** output and pipes it to **stdout**.

gpsd

gpsd is a service daemon that monitors one or more GPS or AIS receivers connected to the host.

6.1.3. Overview of GNSS timing for PTP grandmaster clocks

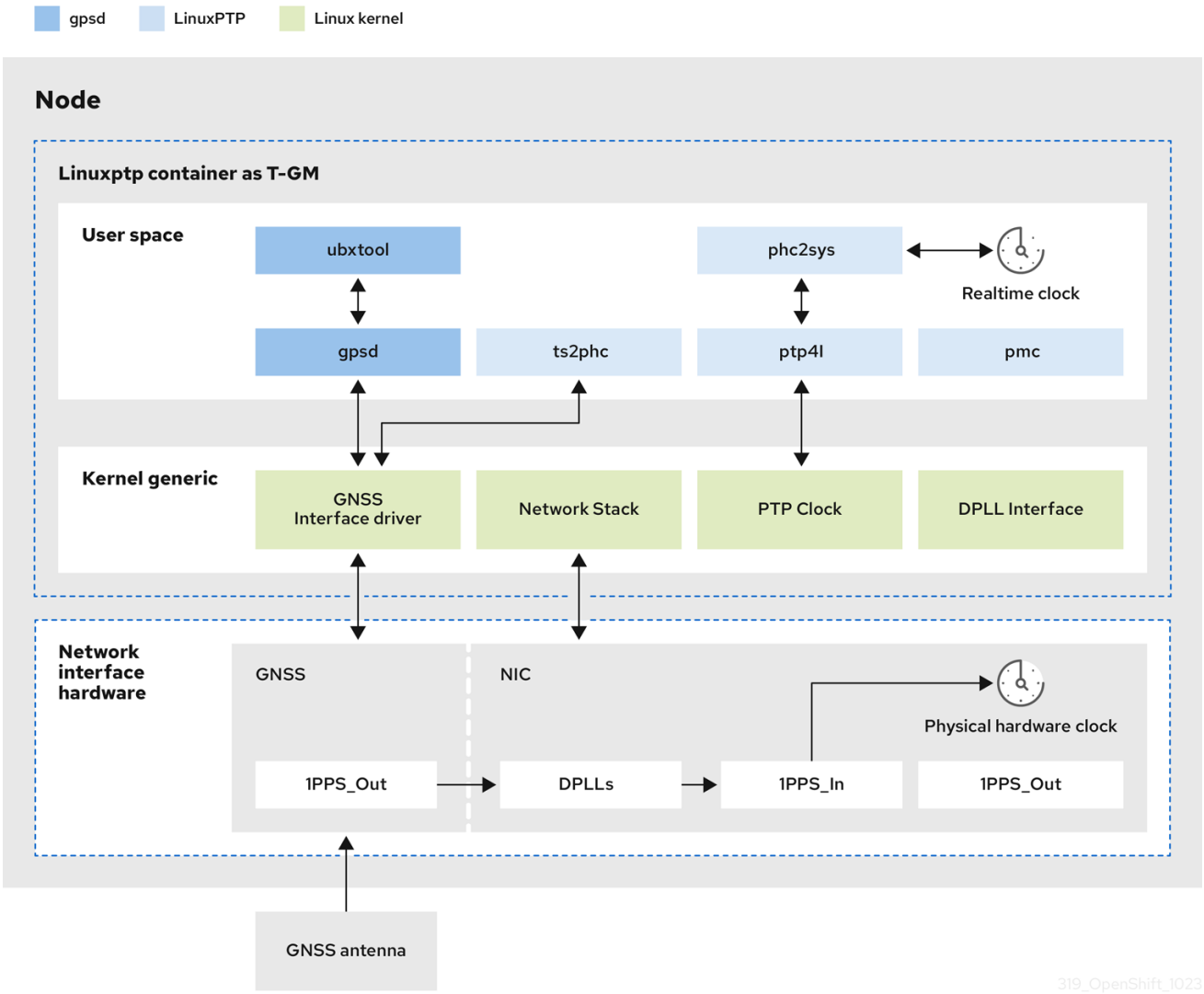
OpenShift Container Platform supports receiving precision PTP timing from Global Navigation Satellite System (GNSS) sources and grandmaster clocks (T-GM) in the cluster.



IMPORTANT

OpenShift Container Platform supports PTP timing from GNSS sources with Intel E810 Westport Channel NICs only.

Figure 6.2. Overview of Synchronization with GNSS and T-GM



Global Navigation Satellite System (GNSS)

GNSS is a satellite-based system used to provide positioning, navigation, and timing information to receivers around the globe. In PTP, GNSS receivers are often used as a highly accurate and stable reference clock source. These receivers receive signals from multiple GNSS satellites, allowing them to calculate precise time information. The timing information obtained from GNSS is used as a reference by the PTP grandmaster clock.

By using GNSS as a reference, the grandmaster clock in the PTP network can provide highly accurate timestamps to other devices, enabling precise synchronization across the entire network.

Digital Phase-Locked Loop (DPLL)

DPLL provides clock synchronization between different PTP nodes in the network. DPLL compares the phase of the local system clock signal with the phase of the incoming synchronization signal, for example, PTP messages from the PTP grandmaster clock. The DPLL continuously adjusts the local clock frequency and phase to minimize the phase difference between the local clock and the reference clock.

6.1.3.1. Handling leap second events in GNSS-synced PTP grandmaster clocks

A leap second is a one-second adjustment that is occasionally applied to Coordinated Universal Time (UTC) to keep it synchronized with International Atomic Time (TAI). UTC leap seconds are unpredictable. Internationally agreed leap seconds are listed in leap-seconds.list. This file is regularly

updated by the International Earth Rotation and Reference Systems Service (IERS). An unhandled leap second can have a significant impact on far edge RAN networks. It can cause the far edge RAN application to immediately disconnect voice calls and data sessions.

6.1.4. About PTP and clock synchronization error events

Cloud native applications such as virtual RAN (vRAN) require access to notifications about hardware timing events that are critical to the functioning of the overall network. PTP clock synchronization errors can negatively affect the performance and reliability of your low-latency application, for example, a vRAN application running in a distributed unit (DU).

Loss of PTP synchronization is a critical error for a RAN network. If synchronization is lost on a node, the radio might be shut down and the network Over the Air (OTA) traffic might be shifted to another node in the wireless network. Fast event notifications mitigate against workload errors by allowing cluster nodes to communicate PTP clock sync status to the vRAN application running in the DU.

Event notifications are available to vRAN applications running on the same DU node. A publish/subscribe REST API passes events notifications to the messaging bus. Publish/subscribe messaging, or pub-sub messaging, is an asynchronous service-to-service communication architecture where any message published to a topic is immediately received by all of the subscribers to the topic.

The PTP Operator generates fast event notifications for every PTP-capable network interface. You can access the events by using a **cloud-event-proxy** sidecar container over an HTTP message bus.

NOTE

PTP fast event notifications are available for network interfaces configured to use PTP ordinary clocks, PTP grandmaster clocks, or PTP boundary clocks.

6.1.5. 2-card E810 NIC configuration reference

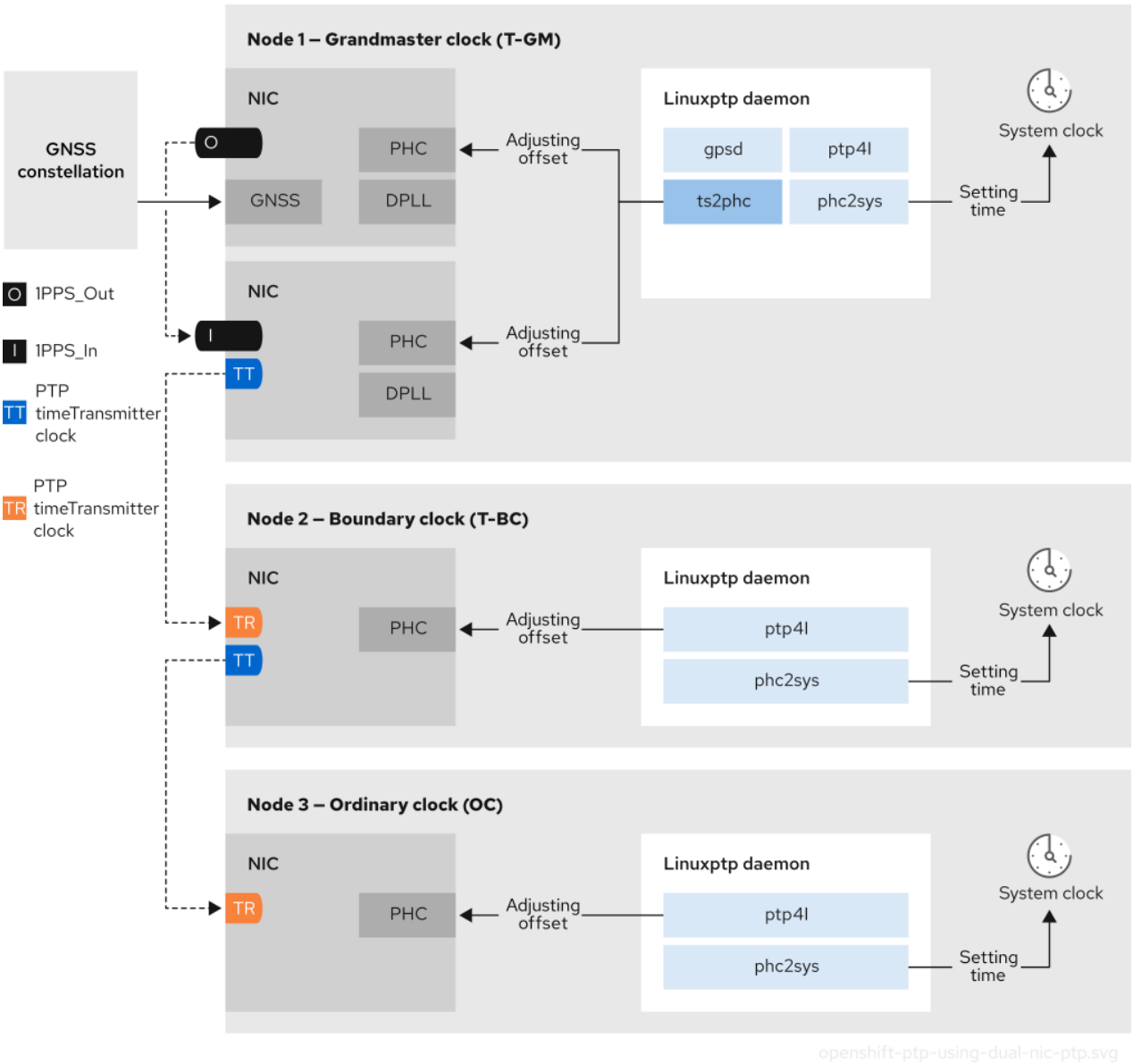
OpenShift Container Platform supports single and dual-NIC Intel E810 hardware for PTP timing in grandmaster clocks (T-GM) and boundary clocks (T-BC).

Dual NIC grandmaster clock

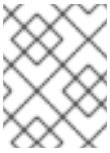
You can use a cluster host that has dual-NIC hardware as PTP grandmaster clock. One NIC receives timing information from the global navigation satellite system (GNSS). The second NIC receives the timing information from the first using the SMA1 Tx/Rx connections on the E810 NIC faceplate. The system clock on the cluster host is synchronized from the NIC that is connected to the GNSS satellite.

Dual NIC grandmaster clocks are a feature of distributed RAN (D-RAN) configurations where the Remote Radio Unit (RRU) and Baseband Unit (BBU) are located at the same radio cell site. D-RAN distributes radio functions across multiple sites, with backhaul connections linking them to the core network.

Figure 6.3. Dual NIC grandmaster clock



openshift-ptp-using-dual-nic-ptp.svg



NOTE

In a dual-NIC T-GM configuration, a single **ts2phc** program operate on two PTP hardware clocks (PHCs), one for each NIC.

Dual NIC boundary clock

For 5G telco networks that deliver mid-band spectrum coverage, each virtual distributed unit (vDU) requires connections to 6 radio units (RUs). To make these connections, each vDU host requires 2 NICs configured as boundary clocks.

Dual NIC hardware allows you to connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

Highly available system clock with dual-NIC boundary clocks

You can configure Intel E810-XXVDA4 Salem channel dual-NIC hardware as dual PTP boundary clocks that provide timing for a highly available system clock. This configuration is useful when you have multiple time sources on different NICs. High availability ensures that the node does not lose timing synchronization if one of the two timing sources is lost or disconnected.

Each NIC is connected to the same upstream leader clock. Highly available boundary clocks use

multiple PTP domains to synchronize with the target system clock. When a T-BC is highly available, the host system clock can maintain the correct offset even if one or more **ptp4l** instances syncing the NIC PHC clock fails. If any single SFP port or cable failure occurs, the boundary clock stays in sync with the leader clock.

Boundary clock leader source selection is done using the A-BMCA algorithm. For more information, see ITU-T recommendation G.8275.1.

6.1.6. Using dual-port NICs to improve redundancy for PTP ordinary clocks

OpenShift Container Platform supports single-port networking interface cards (NICs) as ordinary clocks for PTP timing. To improve redundancy, you can configure a dual-port NIC with one port as active and the other as standby.

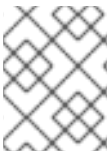
IMPORTANT

Configuring linuxptp services as an ordinary clock with dual-port NIC redundancy is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

In this configuration, the ports in a dual-port NIC operate as follows:

- The active port functions as an ordinary clock in the **Following** port state.
- The standby port remains in the **Listening** port state.
- If the active port fails, the standby port transitions to active to ensure continued PTP timing synchronization.
- If both ports become faulty, the clock state moves to the **HOLDOVER** state, then the **FREERUN** state when the holdover timeout expires, before resyncing to a leader clock.



NOTE

You can configure PTP ordinary clocks with added redundancy on **x86** architecture nodes with dual-port NICs only.

6.1.7. 3-card Intel E810 PTP grandmaster clock

OpenShift Container Platform supports cluster hosts with 3 Intel E810 NICs as PTP grandmaster clocks (T-GM).

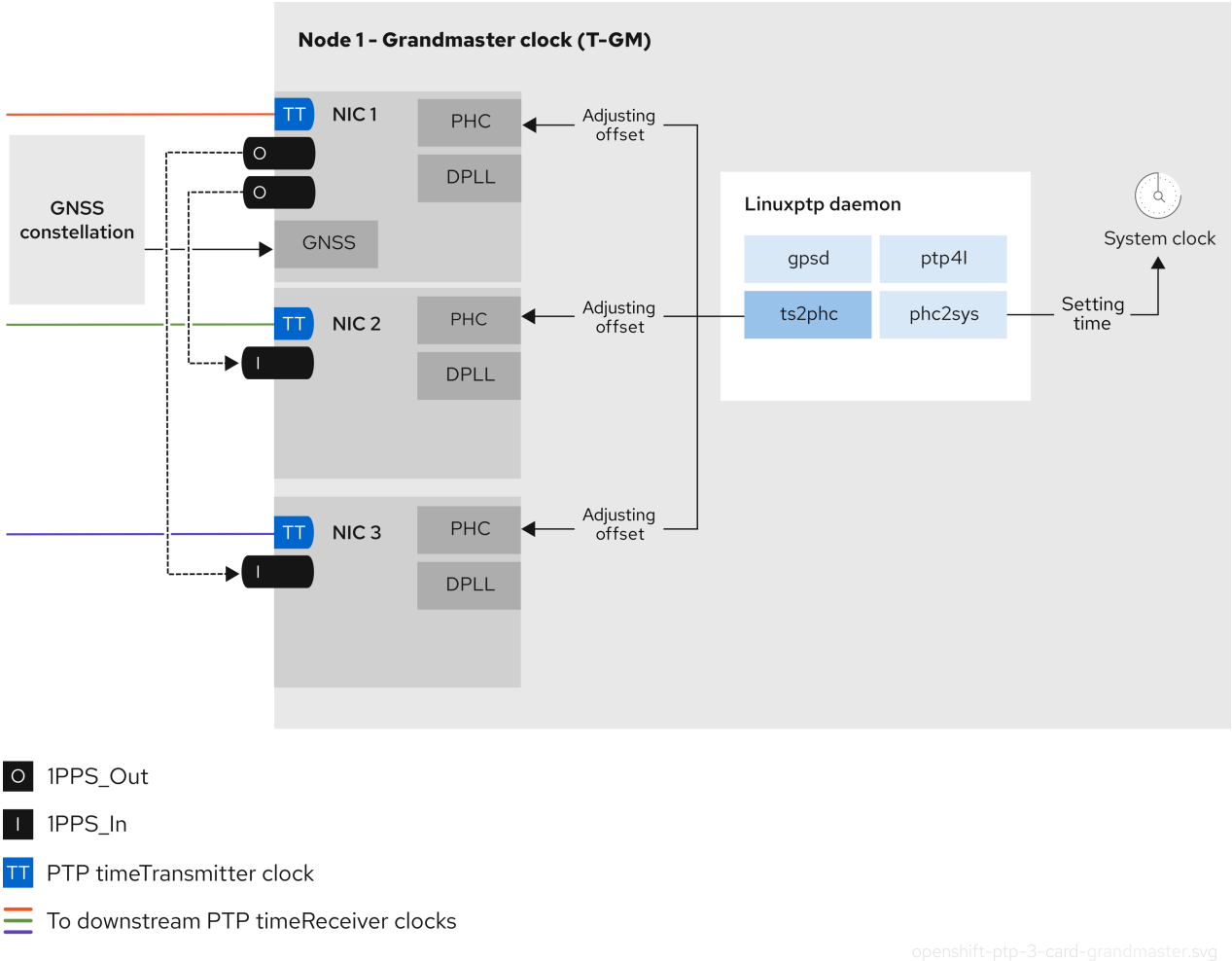
3-card grandmaster clock

You can use a cluster host that has 3 NICs as PTP grandmaster clock. One NIC receives timing information from the global navigation satellite system (GNSS). The second and third NICs receive the timing information from the first by using the SMA1 Tx/Rx connections on the E810 NIC faceplate.

The system clock on the cluster host is synchronized from the NIC that is connected to the GNSS satellite.

3-card NIC grandmaster clocks can be used for distributed RAN (D-RAN) configurations where the Radio Unit (RU) is connected directly to the distributed unit (DU) without a front haul switch, for example, if the RU and DU are located in the same radio cell site. D-RAN distributes radio functions across multiple sites, with backhaul connections linking them to the core network.

Figure 6.4. 3-card Intel E810 PTP grandmaster clock



NOTE

In a 3-card T-GM configuration, a single **ts2phc** process reports as 3 **ts2phc** instances in the system.

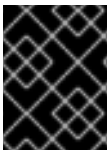
6.2. CONFIGURING PTP DEVICES

The PTP Operator adds the **NodePtpDevice.ptp.openshift.io** custom resource definition (CRD) to OpenShift Container Platform.

When installed, the PTP Operator searches your cluster for Precision Time Protocol (PTP) capable network devices on each node. The Operator creates and updates a **NodePtpDevice** custom resource (CR) object for each node that provides a compatible PTP-capable network device.

Network interface controller (NIC) hardware with built-in PTP capabilities sometimes require a device-specific configuration. You can use hardware-specific NIC features for supported hardware with the

PTP Operator by configuring a plugin in the **PtpConfig** custom resource (CR). The **linuxptp-daemon** service uses the named parameters in the **plugin** stanza to start **linuxptp** processes, **ptp4l** and **phc2sys**, based on the specific hardware configuration.



IMPORTANT

In OpenShift Container Platform 4.18, the Intel E810 NIC is supported with a **PtpConfig** plugin.

6.2.1. Installing the PTP Operator using the CLI

As a cluster administrator, you can install the Operator by using the CLI.

Prerequisites

- A cluster installed on bare-metal hardware with nodes that have hardware that supports PTP.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a namespace for the PTP Operator.
   a. Save the following YAML in the **ptp-namespace.yaml** file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-ptp
  annotations:
    workload.openshift.io/allowed: management
  labels:
    name: openshift-ptp
    openshift.io/cluster-monitoring: "true"
```

   b. Create the **Namespace** CR:

```
$ oc create -f ptp-namespace.yaml
```

2. Create an Operator group for the PTP Operator.
   a. Save the following YAML in the **ptp-operatorgroup.yaml** file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ptp-operators
  namespace: openshift-ptp
spec:
  targetNamespaces:
  - openshift-ptp
```

b. Create the **OperatorGroup** CR:

```
$ oc create -f ptp-operatorgroup.yaml
```

3. Subscribe to the PTP Operator.

a. Save the following YAML in the **ptp-sub.yaml** file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ptp-operator-subscription
  namespace: openshift-ptp
spec:
  channel: "stable"
  name: ptp-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

b. Create the **Subscription** CR:

```
$ oc create -f ptp-sub.yaml
```

4. To verify that the Operator is installed, enter the following command:

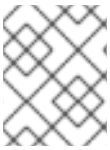
```
$ oc get csv -n openshift-ptp -o custom-
columns=Name:.metadata.name,Phase:.status.phase
```

Example output

```
Name                    Phase
4.18.0-202301261535     Succeeded
```

6.2.2. Installing the PTP Operator by using the web console

As a cluster administrator, you can install the PTP Operator by using the web console.



NOTE

You have to create the namespace and Operator group as mentioned in the previous section.

Procedure

1. Install the PTP Operator using the OpenShift Container Platform web console:

a. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.

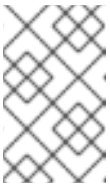
b. Choose **PTP Operator** from the list of available Operators, and then click **Install**.

c. On the **Install Operator** page, under **A specific namespace on the cluster**select **openshift-ptp**. Then, click **Install**.

2. Optional: Verify that the PTP Operator installed successfully:

   a. Switch to the **Operators → Installed Operators** page.

   b. Ensure that **PTP Operator** is listed in the **openshift-ptp** project with a **Status** of **InstallSucceeded**.



NOTE

During installation an Operator might display a **Failed** status. If the installation later succeeds with an **InstallSucceeded** message, you can ignore the **Failed** message.

If the Operator does not appear as installed, to troubleshoot further:

- Go to the **Operators → Installed Operators** page and inspect the **Operator Subscriptions** and **Install Plans** tabs for any failure or errors under **Status**.
- Go to the **Workloads → Pods** page and check the logs for pods in the **openshift-ptp** project.

6.2.3. Discovering PTP-capable network devices in your cluster

Identify PTP-capable network devices that exist in your cluster so that you can configure them

Prerequisites

- You installed the PTP Operator.

Procedure

- To return a complete list of PTP capable network devices in your cluster, run the following command:

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

Example output

```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    creationTimestamp: "2022-01-27T15:16:28Z"
    generation: 1
    name: dev-worker-0
    namespace: openshift-ptp
    resourceVersion: "6538103"
    uid: d42fc9ad-bcbf-4590-b6d8-b676c642781a
  spec: {}
  status:
    devices:
    - name: eno1
    - name: eno2
```

```
    - name: eno3
    - name: eno4
    - name: enp5s0f0
    - name: enp5s0f1
...
```

where:

-worker-0

The value for the **name** parameter is the same as the name of the parent node.

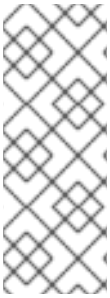
devices

The **devices** collection includes a list of the PTP capable devices that the PTP Operator discovers for the node.

6.2.4. Configuring linuxptp services as a grandmaster clock

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as grandmaster clock (T-GM) by creating a **PtpConfig** custom resource (CR) that configures the host NIC.

The **ts2phc** utility allows you to synchronize the system clock with the PTP grandmaster clock so that the node can stream precision clock signal to downstream PTP ordinary clocks and boundary clocks.



NOTE

Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as T-GM for an Intel Westport Channel E810-XXVDA4T network interface.

To configure PTP fast events, set appropriate values for **ptp4lOpts**, **ptp4lConf**, and **ptpClockThreshold**. **ptpClockThreshold** is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

Prerequisites

- For T-GM clocks in production environments, install an Intel E810 Westport Channel NIC in the bare-metal cluster host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Create the **PtpConfig** CR. For example:

   a. Depending on your requirements, use one of the following T-GM configurations for your deployment. Save the YAML in the **grandmaster-clock-ptp-config.yaml** file:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
```

```
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"
      ptp4lOpts: "-2 --summary_interval -4"
      phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s $iface_master -n 24
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      plugins:
        e810:
          enableDefaultConfig: false
          settings:
            LocalMaxHoldoverOffSet: 1500
            LocalHoldoverTimeout: 14400
            MaxInSpecOffset: 1500
          pins: $e810_pins
          #  "$iface_master":
          #    "U.FL2": "0 2"
          #    "U.FL1": "0 1"
          #    "SMA2": "0 2"
          #    "SMA1": "0 1"
          ublxCmds:
            - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
                - "-P"
                - "29.20"
                - "-z"
                - "CFG-HW-ANT_CFG_VOLTCTRL,1"
              reportOutput: false
            - args: #ubxtool -P 29.20 -e GPS
                - "-P"
                - "29.20"
                - "-e"
                - "GPS"
              reportOutput: false
            - args: #ubxtool -P 29.20 -d Galileo
                - "-P"
                - "29.20"
                - "-d"
                - "Galileo"
              reportOutput: false
            - args: #ubxtool -P 29.20 -d GLONASS
                - "-P"
                - "29.20"
                - "-d"
                - "GLONASS"
              reportOutput: false
            - args: #ubxtool -P 29.20 -d BeiDou
                - "-P"
                - "29.20"
                - "-d"
                - "BeiDou"
              reportOutput: false
            - args: #ubxtool -P 29.20 -d SBAS
```

```
            - "-P"
            - "29.20"
            - "-d"
            - "SBAS"
          reportOutput: false
        - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
            - "-P"
            - "29.20"
            - "-t"
            - "-w"
            - "5"
            - "-v"
            - "1"
            - "-e"
            - "SURVEYIN,600,50000"
          reportOutput: true
        - args: #ubxtool -P 29.20 -p MON-HW
            - "-P"
            - "29.20"
            - "-p"
            - "MON-HW"
          reportOutput: true
        - args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
            - "-P"
            - "29.20"
            - "-p"
            - "CFG-MSG,1,38,248"
          reportOutput: true
    ts2phcOpts: " "
    ts2phcConf: |
      [nmea]
      ts2phc.master 1
      [global]
      use_syslog  0
      verbose 1
      logging_level 7
      ts2phc.pulsewidth 100000000
      #cat /dev/GNSS to find available serial port
      #example value of gnss_serialport is /dev/ttyGNSS_1700_0
      ts2phc.nmea_serialport $gnss_serialport
      [$iface_master]
      ts2phc.extts_polarity rising
      ts2phc.extts_correction 0
    ptp4lConf: |
      [$iface_master]
      masterOnly 1
      [$iface_master_1]
      masterOnly 1
      [$iface_master_2]
      masterOnly 1
      [$iface_master_3]
      masterOnly 1
      [global]
      #
      # Default Data Set
      #
```

```
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
```

```
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      clock_servo pi
      sanity_freq_limit  200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type BC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0x20
  recommend:
    - profile: "grandmaster"
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

NOTE

For E810 Westport Channel NICs, set the value for **ts2phc.nmea_serialport** to **/dev/gnss0**.

b. Create the CR by running the following command:

```
$ oc create -f grandmaster-clock-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

   a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                            READY   STATUS    RESTARTS   AGE     IP           NODE
   linuxptp-daemon-74m2g           3/3     Running   3          4d15h   10.16.230.7  compute-
   1.example.com
   ptp-operator-5f4f48d7c-x7zkf    1/1     Running   1          4d15h   10.128.1.145  compute-
   1.example.com
   ```

   b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

   ```
   $ oc logs linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

   ```
   ts2phc[94980.334]: [ts2phc.0.config] nmea delay: 98690975 ns
   ts2phc[94980.334]: [ts2phc.0.config] ens3f0 extts index 0 at 1676577329.999999999 corr
   0 src 1676577330.901342528 diff -1
   ts2phc[94980.334]: [ts2phc.0.config] ens3f0 master offset         -1 s2 freq      -1
   ts2phc[94980.441]: [ts2phc.0.config] nmea sentence:
   GNRMC,195453.00,A,4233.24427,N,07126.64420,W,0.008,,160223,,,A,V
   phc2sys[94980.450]: [ptp4l.0.config] CLOCK_REALTIME phc offset       943 s2 freq  -
   89604 delay    504
   phc2sys[94980.512]: [ptp4l.0.config] CLOCK_REALTIME phc offset      1000 s2 freq  -
   89264 delay    474
   ```

6.2.4.1. Configuring linuxptp services as a grandmaster clock for dual E810 NICs

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock (T-GM) for 2 E810 NICs by creating a **PtpConfig** custom resource (CR) that configures the NICs.

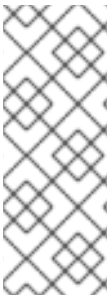
You can configure the **linuxptp** services as a T-GM for the following E810 NICs:

- Intel E810-XXVDA4T Westport Channel NIC
- Intel E810-CQDA2T Logan Beach NIC

For distributed RAN (D-RAN) use cases, you can configure PTP for 2 NICs as follows:

- NIC 1 is synced to the global navigation satellite system (GNSS) time source.
- NIC 2 is synced to the 1PPS timing output provided by NIC one. This configuration is provided by the PTP hardware plugin in the **PtpConfig** CR.

The 2-card PTP T-GM configuration uses one instance of **ptp4l** and one instance of **ts2phc**. The **ptp4l** and **ts2phc** programs are each configured to operate on two PTP hardware clocks (PHCs), one for each NIC. The host system clock is synchronized from the NIC that is connected to the GNSS time source.



NOTE

Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as T-GM for dual Intel E810 network interfaces.

To configure PTP fast events, set appropriate values for **ptp4lOpts**, **ptp4lConf**, and **ptpClockThreshold**. **ptpClockThreshold** is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

Prerequisites

- For T-GM clocks in production environments, install two Intel E810 NICs in the bare-metal cluster host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Create the **PtpConfig** CR. For example:
   a. Save the following YAML in the **grandmaster-clock-ptp-config-dual-nics.yaml** file:

```
# In this example two cards $iface_nic1 and $iface_nic2 are connected via
# SMA1 ports by a cable and $iface_nic2 receives 1PPS signals from $iface_nic1
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"
      ptp4lOpts: "-2 --summary_interval -4"
      phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s $iface_nic1 -n 24
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      plugins:
        e810:
          enableDefaultConfig: false
          settings:
            LocalMaxHoldoverOffSet: 1500
            LocalHoldoverTimeout: 14400
            MaxInSpecOffset: 1500
          pins: $e810_pins
```

```
#  "$iface_nic1":
#    "U.FL2": "0 2"
#    "U.FL1": "0 1"
#    "SMA2": "0 2"
#    "SMA1": "2 1"
#  "$iface_nic2":
#    "U.FL2": "0 2"
#    "U.FL1": "0 1"
#    "SMA2": "0 2"
#    "SMA1": "1 1"
ublxCmds:
  - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
      - "-P"
      - "29.20"
      - "-z"
      - "CFG-HW-ANT_CFG_VOLTCTRL,1"
    reportOutput: false
  - args: #ubxtool -P 29.20 -e GPS
      - "-P"
      - "29.20"
      - "-e"
      - "GPS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d Galileo
      - "-P"
      - "29.20"
      - "-d"
      - "Galileo"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d GLONASS
      - "-P"
      - "29.20"
      - "-d"
      - "GLONASS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d BeiDou
      - "-P"
      - "29.20"
      - "-d"
      - "BeiDou"
    reportOutput: false
  - args: #ubxtool -P 29.20 -d SBAS
      - "-P"
      - "29.20"
      - "-d"
      - "SBAS"
    reportOutput: false
  - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
      - "-P"
      - "29.20"
      - "-t"
      - "-w"
      - "5"
      - "-v"
      - "1"
      - "-e"
```

```
          - "SURVEYIN,600,50000"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p MON-HW
          - "-P"
          - "29.20"
          - "-p"
          - "MON-HW"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
          - "-P"
          - "29.20"
          - "-p"
          - "CFG-MSG,1,38,248"
        reportOutput: true
  ts2phcOpts: " "
  ts2phcConf: |
    [nmea]
    ts2phc.master 1
    [global]
    use_syslog  0
    verbose 1
    logging_level 7
    ts2phc.pulsewidth 100000000
    #cat /dev/GNSS to find available serial port
    #example value of gnss_serialport is /dev/ttyGNSS_1700_0
    ts2phc.nmea_serialport $gnss_serialport
    [$iface_nic1]
    ts2phc.extts_polarity rising
    ts2phc.extts_correction 0
    [$iface_nic2]
    ts2phc.master 0
    ts2phc.extts_polarity rising
    #this is a measured value in nanoseconds to compensate for SMA cable delay
    ts2phc.extts_correction -10
  ptp4lConf: |
    [$iface_nic1]
    masterOnly 1
    [$iface_nic1_1]
    masterOnly 1
    [$iface_nic1_2]
    masterOnly 1
    [$iface_nic1_3]
    masterOnly 1
    [$iface_nic2]
    masterOnly 1
    [$iface_nic2_1]
    masterOnly 1
    [$iface_nic2_2]
    masterOnly 1
    [$iface_nic2_3]
    masterOnly 1
    [global]
    #
    # Default Data Set
    #
    twoStepFlag 1
```

```
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
```

```
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      clock_servo pi
      sanity_freq_limit  200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type BC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 1
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0x20
recommend:
  - profile: "grandmaster"
    priority: 4
    match:
      - nodeLabel: "node-role.kubernetes.io/$mcp"
```

NOTE

Set the value for **ts2phc.nmea_serialport** to **/dev/gnss0**.

b. Create the CR by running the following command:

```
$ oc create -f grandmaster-clock-ptp-config-dual-nics.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

   a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                            READY   STATUS    RESTARTS   AGE     IP           NODE
   linuxptp-daemon-74m2g           3/3     Running   3          4d15h   10.16.230.7  compute-
   1.example.com
   ptp-operator-5f4f48d7c-x7zkf    1/1     Running   1          4d15h   10.128.1.145  compute-
   1.example.com
   ```

   b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

   ```
   $ oc logs linuxptp-daemon-74m2g -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

   ```
   ts2phc[509863.660]: [ts2phc.0.config] nmea delay: 347527248 ns
   ts2phc[509863.660]: [ts2phc.0.config] ens2f0 extts index 0 at 1705516553.000000000
   corr 0 src 1705516553.652499081 diff 0
   ts2phc[509863.660]: [ts2phc.0.config] ens2f0 master offset          0 s2 freq      -0
   I0117 18:35:16.000146 1633226 stats.go:57] state updated for ts2phc =s2
   I0117 18:35:16.000163 1633226 event.go:417] dpll State s2, gnss State s2, tsphc state
   s2, gm state s2,
   ts2phc[1705516516]:[ts2phc.0.config] ens2f0 nmea_status 1 offset 0 s2
   GM[1705516516]:[ts2phc.0.config] ens2f0 T-GM-STATUS s2
   ts2phc[509863.677]: [ts2phc.0.config] ens7f0 extts index 0 at 1705516553.000000010
   corr -10 src 1705516553.652499081 diff 0
   ts2phc[509863.677]: [ts2phc.0.config] ens7f0 master offset          0 s2 freq      -0
   I0117 18:35:16.016597 1633226 stats.go:57] state updated for ts2phc =s2
   phc2sys[509863.719]: [ptp4l.0.config] CLOCK_REALTIME phc offset        -6 s2 freq
   +15441 delay    510
   phc2sys[509863.782]: [ptp4l.0.config] CLOCK_REALTIME phc offset        -7 s2 freq
   +15438 delay    502
   ```

6.2.4.2. Configuring linuxptp services as a grandmaster clock for 3 E810 NICs

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock (T-GM) for 3 E810 NICs by creating a **PtpConfig** custom resource (CR) that configures the NICs.

You can configure the **linuxptp** services as a T-GM with 3 NICs for the following E810 NICs:

- Intel E810-XXVDA4T Westport Channel NIC
- Intel E810-CQDA2T Logan Beach NIC

For distributed RAN (D-RAN) use cases, you can configure PTP for 3 NICs as follows:

- NIC 1 is synced to the Global Navigation Satellite System (GNSS)

- NICs 2 and 3 are synced to NIC 1 with 1PPS faceplate connections

Use the following example **PtpConfig** CRs as the basis to configure **linuxptp** services as a 3-card Intel E810 T-GM.

Prerequisites

- For T-GM clocks in production environments, install 3 Intel E810 NICs in the bare-metal cluster host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Create the **PtpConfig** CR. For example:
   a. Save the following YAML in the **three-nic-grandmaster-clock-ptp-config.yaml** file:

```
# In this example, the three cards are connected via SMA cables:
# - $iface_timeTx1 has the GNSS signal input
# - $iface_timeTx2 SMA1 is connected to $iface_timeTx1 SMA1
# - $iface_timeTx3 SMA1 is connected to $iface_timeTx1 SMA2
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: gm-3card
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
  - name: grandmaster
    ptp4lOpts: -2 --summary_interval -4
    phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s $iface_timeTx1 -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"
    plugins:
      e810:
        enableDefaultConfig: false
        settings:
          LocalHoldoverTimeout: 14400
          LocalMaxHoldoverOffSet: 1500
          MaxInSpecOffset: 1500
        pins:
          # Syntax guide:
          # - The 1st number in each pair must be one of:
          #    0 - Disabled
          #    1 - RX
          #    2 - TX
          # - The 2nd number in each pair must match the channel number
```

```
    $iface_timeTx1:
      SMA1: 2 1
      SMA2: 2 2
      U.FL1: 0 1
      U.FL2: 0 2
    $iface_timeTx2:
      SMA1: 1 1
      SMA2: 0 2
      U.FL1: 0 1
      U.FL2: 0 2
    $iface_timeTx3:
      SMA1: 1 1
      SMA2: 0 2
      U.FL1: 0 1
      U.FL2: 0 2
  ublxCmds:
    - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
        - "-P"
        - "29.20"
        - "-z"
        - "CFG-HW-ANT_CFG_VOLTCTRL,1"
      reportOutput: false
    - args: #ubxtool -P 29.20 -e GPS
        - "-P"
        - "29.20"
        - "-e"
        - "GPS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d Galileo
        - "-P"
        - "29.20"
        - "-d"
        - "Galileo"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d GLONASS
        - "-P"
        - "29.20"
        - "-d"
        - "GLONASS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d BeiDou
        - "-P"
        - "29.20"
        - "-d"
        - "BeiDou"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d SBAS
        - "-P"
        - "29.20"
        - "-d"
        - "SBAS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
        - "-P"
        - "29.20"
        - "-t"
```

```
            - "-w"
            - "5"
            - "-v"
            - "1"
            - "-e"
            - "SURVEYIN,600,50000"
          reportOutput: true
        - args: #ubxtool -P 29.20 -p MON-HW
            - "-P"
            - "29.20"
            - "-p"
            - "MON-HW"
          reportOutput: true
        - args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
            - "-P"
            - "29.20"
            - "-p"
            - "CFG-MSG,1,38,248"
          reportOutput: true
    ts2phcOpts: " "
    ts2phcConf: |
      [nmea]
      ts2phc.master 1
      [global]
      use_syslog  0
      verbose 1
      logging_level 7
      ts2phc.pulsewidth 100000000
      #example value of nmea_serialport is /dev/gnss0
      ts2phc.nmea_serialport (?<gnss_serialport>[/\w\s/]+)
      leapfile /usr/share/zoneinfo/leap-seconds.list
      [$iface_timeTx1]
      ts2phc.extts_polarity rising
      ts2phc.extts_correction 0
      [$iface_timeTx2]
      ts2phc.master 0
      ts2phc.extts_polarity rising
      #this is a measured value in nanoseconds to compensate for SMA cable delay
      ts2phc.extts_correction -10
      [$iface_timeTx3]
      ts2phc.master 0
      ts2phc.extts_polarity rising
      #this is a measured value in nanoseconds to compensate for SMA cable delay
      ts2phc.extts_correction -10
    ptp4lConf: |
      [$iface_timeTx1]
      masterOnly 1
      [$iface_timeTx1_1]
      masterOnly 1
      [$iface_timeTx1_2]
      masterOnly 1
      [$iface_timeTx1_3]
      masterOnly 1
      [$iface_timeTx2]
      masterOnly 1
      [$iface_timeTx2_1]
```

```
masterOnly 1
[$iface_timeTx2_2]
masterOnly 1
[$iface_timeTx2_3]
masterOnly 1
[$iface_timeTx3]
masterOnly 1
[$iface_timeTx3_1]
masterOnly 1
[$iface_timeTx3_2]
masterOnly 1
[$iface_timeTx3_3]
masterOnly 1
[global]
#
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
```

```
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
delay_filter_length 10
egressLatency 0
ingressLatency 0
boundary_clock_jbod 1
#
# Clock description
#
productDescription ;;
revisionData ;;
manufacturerIdentity 00:00:00
userDescription ;
timeSource 0x20
```

```
    ptpClockThreshold:
      holdOverTimeout: 5
      maxOffsetThreshold: 1500
      minOffsetThreshold: -1500
  recommend:
  - profile: grandmaster
    priority: 4
    match:
    - nodeLabel: node-role.kubernetes.io/$mcp
```

NOTE

Set the value for **ts2phc.nmea_serialport** to **/dev/gnss0**.

b. Create the CR by running the following command:

```
$ oc create -f three-nic-grandmaster-clock-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

   a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                     READY   STATUS    RESTARTS   AGE     IP           NODE
   linuxptp-daemon-74m3q    3/3     Running   3          4d15h   10.16.230.7  compute-
   1.example.com
   ptp-operator-5f4f48d7c-x6zkn  1/1     Running   1          4d15h   10.128.1.145  compute-
   1.example.com
   ```

   b. Check that the profile is correct. Run the following command, and examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile:

   ```
   $ oc logs linuxptp-daemon-74m3q -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

   ```
   ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
   /dev/ptp11
   ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
   /dev/ptp7
   ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
   /dev/ptp14
   ts2phc[2527.586]: [ts2phc.0.config:7] nmea delay: 56308811 ns
   ts2phc[2527.586]: [ts2phc.0.config:6] /dev/ptp14 offset          0 s2 freq      +0
   ts2phc[2527.587]: [ts2phc.0.config:6] /dev/ptp7 offset          0 s2 freq      +0
   ts2phc[2527.587]: [ts2phc.0.config:6] /dev/ptp11 offset          0 s2 freq      -0
   I0324 14:25:05.000439  106907 stats.go:61] state updated for ts2phc =s2
   I0324 14:25:05.000504  106907 event.go:419] dpll State s2, gnss State s2, tsphc state
   ```

```
s2, gm state s2,
I0324 14:25:05.000906  106907 stats.go:61] state updated for ts2phc =s2
I0324 14:25:05.001059  106907 stats.go:61] state updated for ts2phc =s2
ts2phc[1742826305]:[ts2phc.0.config] ens4f0 nmea_status 1 offset 0 s2
GM[1742826305]:[ts2phc.0.config] ens4f0 T-GM-STATUS s2
```

where:

adding tstamp <timestamp> to clock /dev/ptp<N>

Indicates **ts2phc** is actively synchronizing the PTP hardware clock (PHC) by applying a specific timestamp.

/dev/ptp<N> offset 0 s2 freq +0

Displays the estimated offset between the PTP device and the reference; an offset of 0 and state **s2** signifies full synchronization.

T-GM-STATUS s2

Confirms the Telecom Grandmaster (T-GM) is in a locked state (**s2**), providing a stable time reference for the network.

Additional resources

- Configuring the PTP fast event notifications publisher

6.2.5. Grandmaster clock PtpConfig configuration reference

The following reference information describes the configuration options for the **PtpConfig** custom resource (CR) that configures the **linuxptp** services (**ptp4l**, **phc2sys**, **ts2phc**) as a grandmaster clock.

Table 6.1. PtpConfig configuration options for PTP Grandmaster clock

PtpConfig CR field	Description
plugins	Specify an array of **.exec.cmdline** options that configure the NIC for grandmaster clock operation. Grandmaster clock configuration requires certain PTP pins to be disabled. The plugin mechanism allows the PTP Operator to do automated hardware configuration. For the Intel Westport Channel NIC or the Intel Logan Beach NIC, when the **enableDefaultConfig** field is set to **true**, the PTP Operator runs a hard-coded script to do the required configuration for the NIC.
ptp4lOpts	Specify system configuration options for the **ptp4l** service. The options should not include the network interface name **-i <interface>** and service config file**-f /etc/ptp4l.conf** because the network interface name and the service config file are automatically appended.
ptp4lConf	Specify the required configuration to start **ptp4l** as a grandmaster clock. For example, the **ens2f1** interface synchronizes downstream connected devices. For grandmaster clocks, set **clockClass** to **6** and set **clockAccuracy** to **0x27**. Set **timeSource** to **0x20** for when receiving the timing signal from a Global navigation satellite system (GNSS).

PtpConfig CR field	Description
tx_timestamp_timeout	Specify the maximum amount of time to wait for the transmit (TX) timestamp from the sender before discarding the data.
boundary_clock_jbod	Specify the JBOD boundary clock time delay value. This value is used to correct the time values that are passed between the network time devices.
phc2sysOpts	Specify system config options for the **phc2sys** service. If this field is empty the PTP Operator does not start the **phc2sys** service.  NOTE Ensure that the network interface listed here is configured as grandmaster and is referenced as required in the **ts2phcConf** and **ptp4lConf** fields.
ptpSchedulingPolicy	Configure the scheduling policy for **ptp4l** and **phc2sys** processes. Default value is **SCHED_OTHER**. Use **SCHED_FIFO** on systems that support FIFO scheduling.
ptpSchedulingPriority	Set an integer value from 1-65 to configure FIFO priority for **ptp4l** and **phc2sys** processes when **ptpSchedulingPolicy** is set to **SCHED_FIFO**. The **ptpSchedulingPriority** field is not used when **ptpSchedulingPolicy** is set to **SCHED_OTHER**.
ptpClockThreshold	Optional. If **ptpClockThreshold** stanza is not present, default values are used for **ptpClockThreshold** fields. Stanza shows default **ptpClockThreshold** values. **ptpClockThreshold** values configure how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.

PtpConfig CR field	Description
ts2phcConf	Sets the configuration for the **ts2phc** command. **leapfile** is the default path to the current leap seconds definition file in the PTP Operator container image. **ts2phc.nmea_serialport** is the serial port device that is connected to the NMEA GPS clock source. When configured, the GNSS receiver is accessible on **/dev/gnss<id>**. If the host has multiple GNSS receivers, you can find the correct device by enumerating either of the following devices: • **/sys/class/net/<eth_port>/device/gnss/** • **/sys/class/gnss/gnss<id>/device/**
ts2phcOpts	Set options for the **ts2phc** command.
recommend	Specify an array of one or more **recommend** objects that define rules on how the **profile** should be applied to nodes.
.recommend.profile	Specify the **.recommend.profile** object name that is defined in the **profile** section.
.recommend.priority	Specify the **priority** with an integer value between **0** and **99**. A larger number gets lower priority, so a priority of **99** is lower than a priority of **10**. If a node can be matched with multiple profiles according to rules defined in the **match** field, the profile with the higher priority is applied to that node.
.recommend.match	Specify **.recommend.match** rules with **nodeLabel** or **nodeName** values.
.recommend.match.nodeLabel	Set **nodeLabel** with the **key** of the **node.Labels** field from the node object by using the **oc get nodes --show-labels** command. For example, **node-role.kubernetes.io/worker**.
.recommend.match.nodeName	Set **nodeName** with the value of the **node.Name** field from the node object by using the **oc get nodes** command. For example, **compute-1.example.com**.

6.2.5.1. Grandmaster clock class sync state reference

The following table describes the PTP grandmaster clock (T-GM) **gm.ClockClass** states.

Clock class states categorize T-GM clocks based on their accuracy and stability with regard to the Primary Reference Time Clock (PRTC) or other timing source.

Holdover specification is the amount of time a PTP clock can maintain synchronization without receiving updates from the primary time source.

Table 6.2. T-GM clock class states

Clock class state	Description
gm.ClockClass 6	T-GM clock is connected to a PRTC in **LOCKED** mode. For example, the PRTC is traceable to a GNSS time source.
gm.ClockClass 7	T-GM clock is in **HOLDOVER** mode, and within holdover specification. The clock source might not be traceable to a category 1 frequency source.
gm.ClockClass 248	T-GM clock is in **FREERUN** mode.

For more information, see "Phase/time traceability information", ITU-T G.8275.1/Y.1369.1 Recommendations.

6.2.5.2. Intel E810 NIC hardware configuration reference

Use this information to understand how to use the Intel E810 hardware plugin to configure the E810 network interface as PTP grandmaster clock.

Hardware pin configuration determines how the network interface interacts with other components and devices in the system. The Intel E810 NIC has four connectors for external 1PPS signals: **SMA1**, **SMA2**, **U.FL1**, and **U.FL2**.

Table 6.3. Intel E810 NIC hardware connectors configuration

Hardware pin	Recommended setting	Description
U.FL1	**0 1**	Disables the **U.FL1** connector input. The **U.FL1** connector is output-only.
U.FL2	**0 2**	Disables the **U.FL2** connector output. The **U.FL2** connector is input-only.
SMA1	**0 1**	Disables the **SMA1** connector input. The **SMA1** connector is bidirectional.
SMA2	**0 2**	Disables the **SMA2** connector output. The **SMA2** connector is bidirectional.

You can set the pin configuration on the Intel E810 NIC by using the **spec.profile.plugins.e810.pins** parameters as shown in the following example:

```
pins:
    <interface_name>:
      <connector_name>: <function> <channel_number>
```

Where:

<function>: Specifies the role of the pin. The following values are associated with the pin role:

- **0**: Disabled
- **1**: Rx (Receive timestamping)
- **2**: Tx (Transmit timestamping)

<channel number>: A number associated with the physical connector. The following channel numbers are associated with the physical connectors:

- **1**: **SMA1** or **U.FL1**
- **2**: **SMA2** or **U.FL2**

Examples:

- **0 1**: Disables the pin mapped to **SMA1** or **U.FL1**.
- **1 2**: Assigns the Rx function to **SMA2** or **U.FL2**.

NOTE

SMA1 and **U.FL1** connectors share channel one. **SMA2** and **U.FL2** connectors share channel two.

Set **spec.profile.plugins.e810.ublxCmds** parameters to configure the GNSS clock in the **PtpConfig** custom resource (CR).

IMPORTANT

You must configure an offset value to compensate for T-GM GPS antenna cable signal delay. To configure the optimal T-GM antenna offset value, make precise measurements of the GNSS antenna cable signal delay. Red Hat cannot assist in this measurement or provide any values for the required delay offsets.

Each of these **ublxCmds** stanzas correspond to a configuration that is applied to the host NIC by using **ubxtool** commands. For example:

```
ublxCmds:
  - args:
      - "-P"
      - "29.20"
      - "-z"
      - "CFG-HW-ANT_CFG_VOLTCTRL,1"
      - "-z"
      - "CFG-TP-ANT_CABLEDELAY,<antenna_delay_offset>"
    reportOutput: false
```

where:

"CFG-TP-ANT_CABLEDELAY,<antenna_delay_offset>"

Measured T-GM antenna delay offset in nanoseconds. To get the required delay offset value, you must measure the cable delay using external test equipment.

The following table describes the equivalent **ubxtool** commands:

Table 6.4. Intel E810 ublxCmds configuration

ubxtool command	Description
ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1 -z CFG-TP-ANT_CABLEDELAY,<antenna_delay_offset>	Enables antenna voltage control, allows antenna status to be reported in the **UBX-MON-RF** and **UBX-INF-NOTICE** log messages, and sets a **<antenna_delay_offset>** value in nanoseconds that offsets the GPS antenna cable signal delay.
ubxtool -P 29.20 -e GPS	Enables the antenna to receive GPS signals.
ubxtool -P 29.20 -d Galileo	Configures the antenna to receive signal from the Galileo GPS satellite.
ubxtool -P 29.20 -d GLONASS	Disables the antenna from receiving signal from the GLONASS GPS satellite.
ubxtool -P 29.20 -d BeiDou	Disables the antenna from receiving signal from the BeiDou GPS satellite.
ubxtool -P 29.20 -d SBAS	Disables the antenna from receiving signal from the SBAS GPS satellite.
ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000	Configures the GNSS receiver survey-in process to improve its initial position estimate. This can take up to 24 hours to achieve an optimal result.
ubxtool -P 29.20 -p MON-HW	Runs a single automated scan of the hardware and reports on the NIC state and configuration settings.

6.2.5.3. Dual E810 NIC configuration reference

Use this information to understand how to use the Intel E810 hardware plugin to configure a pair of E810 network interfaces as PTP grandmaster clock (T-GM).

Before you configure the dual-NIC cluster host, you must connect the two NICs with an SMA1 cable using the 1PPS faceplace connections.

When you configure a dual-NIC T-GM, you need to compensate for the 1PPS signal delay that occurs when you connect the NICs using the SMA1 connection ports. Various factors such as cable length, ambient temperature, and component and manufacturing tolerances can affect the signal delay. To compensate for the delay, you must calculate the specific value that you use to offset the signal delay.

Table 6.5. E810 dual-NIC T-GM PtpConfig CR reference

PtpConfig field	Description
spec.profile.plugins.e810.pins	Configure the E810 hardware pins using the PTP Operator E810 hardware plugin. • Pin **2 1** enables the **1PPS OUT** connection for **SMA1** on NIC one. • Pin **1 1** enables the **1PPS IN** connection for **SMA1** on NIC two.
spec.profile.ts2phcConf	Use the **ts2phcConf** field to configure parameters for NIC one and NIC two. Set **ts2phc.master 0** for NIC two. This configures the timing source for NIC two from the 1PPS input, not GNSS. Configure the **ts2phc.extts_correction** value for NIC two to compensate for the delay that is incurred for the specific SMA cable and cable length that you use. The value that you configure depends on your specific measurements and SMA1 cable length.
spec.profile.ptp4lConf	Set the value of **boundary_clock_jbod** to 1 to enable support for multiple NICs.

Each value in the **spec.profile.plugins.e810.pins** list follows the **<function> <channel_number>** format.

Where:

<function>: Specifies the pin role. The following values are associated with the pin role:

- **0**: Disabled
- **1**: Receive (Rx) – for 1PPS IN
- **2**: Transmit (Tx) – for 1PPS OUT

<channel_number>: A number associated with the physical connector. The following channel numbers are associated with the physical connectors:

- **1**: **SMA1** or **U.FL1**
- **2**: **SMA2** or **U.FL2**

Examples:

- **2 1**: Enables **1PPS OUT** (Tx) on **SMA1**.
- **1 1**: Enables **1PPS IN** (Rx) on **SMA1**.

The PTP Operator passes these values to the Intel E810 hardware plugin and writes them to the sysfs pin configuration interface on each NIC.

6.2.5.4. 3-card E810 NIC configuration reference

Use this information to understand how to configure 3 E810 NICs as PTP grandmaster clock (T-GM).

Before you configure the 3-card cluster host, you must connect the 3 NICs by using the 1PPS faceplate connections. The primary NIC **1PPS_out** outputs feed the other 2 NICs.

When you configure a 3-card T-GM, you need to compensate for the 1PPS signal delay that occurs when you connect the NICs by using the SMA1 connection ports. Various factors such as cable length, ambient temperature, and component and manufacturing tolerances can affect the signal delay. To compensate for the delay, you must calculate the specific value that you use to offset the signal delay.

Table 6.6. 3-card E810 T-GM PtpConfig CR reference

PtpConfig field	Description
spec.profile.plugins.e810.pins	Configure the E810 hardware pins with the PTP Operator E810 hardware plugin. • **$iface_timeTx1.SMA1** enables the **1PPS OUT** connection for **SMA1** on NIC 1. • **$iface_timeTx1.SMA2** enables the **1PPS OUT** connection for **SMA2** on NIC 1. • **$iface_timeTx2.SMA1** and **$iface_timeTx3.SMA1** enables the **1PPS IN** connection for **SMA1** on NIC 2 and NIC 3. • **$iface_timeTx2.SMA2** and **$iface_timeTx3.SMA2** disables the **SMA2** connection on NIC 2 and NIC 3.
spec.profile.ts2phcConf	Use the **ts2phcConf** field to configure parameters for the NICs. Set **ts2phc.master 0** for NIC 2 and NIC 3. This configures the timing source for NIC 2 and NIC 3 from the 1PPS input, not GNSS. Configure the **ts2phc.extts_correction** value for NIC 2 and NIC 3 to compensate for the delay that is incurred for the specific SMA cable and cable length that you use. The value that you configure depends on your specific measurements and SMA1 cable length.
spec.profile.ptp4lConf	Set the value of **boundary_clock_jbod** to 1 to enable support for multiple NICs.

6.2.6. Holdover in a grandmaster clock with GNSS as the source

Holdover allows the grandmaster (T-GM) clock to maintain synchronization performance when the global navigation satellite system (GNSS) source is unavailable. During this period, the T-GM clock relies on its internal oscillator and holdover parameters to reduce timing disruptions.

You can define the holdover behavior by configuring the following holdover parameters in the **PTPConfig** custom resource (CR):

MaxInSpecOffset

Specifies the maximum allowed offset in nanoseconds. If the T-GM clock exceeds the **MaxInSpecOffset** value, it transitions to the **FREERUN** state (clock class state **gm.ClockClass 248**).

LocalHoldoverTimeout

Specifies the maximum duration, in seconds, for which the T-GM clock remains in the holdover state before transitioning to the **FREERUN** state.

LocalMaxHoldoverOffSet

Specifies the maximum offset that the T-GM clock can reach during the holdover state in nanoseconds.

If the **MaxInSpecOffset** value is less than the **LocalMaxHoldoverOffset** value, and the T-GM clock exceeds the maximum offset value, the T-GM clock transitions from the holdover state to the **FREERUN** state.

IMPORTANT

If the **LocalMaxHoldoverOffSet** value is less than the **MaxInSpecOffset** value, the holdover timeout occurs before the clock reaches the maximum offset. To resolve this issue, set the **MaxInSpecOffset** field and the **LocalMaxHoldoverOffset** field to the same value.

For information about clock class states, see "Grandmaster clock class sync state reference" document.

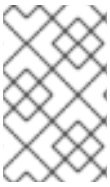
The T-GM clock uses the holdover parameters **LocalMaxHoldoverOffSet** and **LocalHoldoverTimeout** to calculate the slope. Slope is the rate at which the phase offset changes over time. It is measured in nanoseconds per second, where the set value indicates how much the offset increases over a given time period.

The T-GM clock uses the slope value to predict and compensate for time drift, so reducing timing disruptions during holdover. The T-GM clock uses the following formula to calculate the slope:

- Slope = **localMaxHoldoverOffSet** / **localHoldoverTimeout**
  For example, if the **LocalHoldOverTimeout** parameter is set to 60 seconds, and the **LocalMaxHoldoverOffset** parameter is set to 3000 nanoseconds, the slope is calculated as follows:

  Slope = 3000 nanoseconds / 60 seconds = 50 nanoseconds per second

  The T-GM clock reaches the maximum offset in 60 seconds.

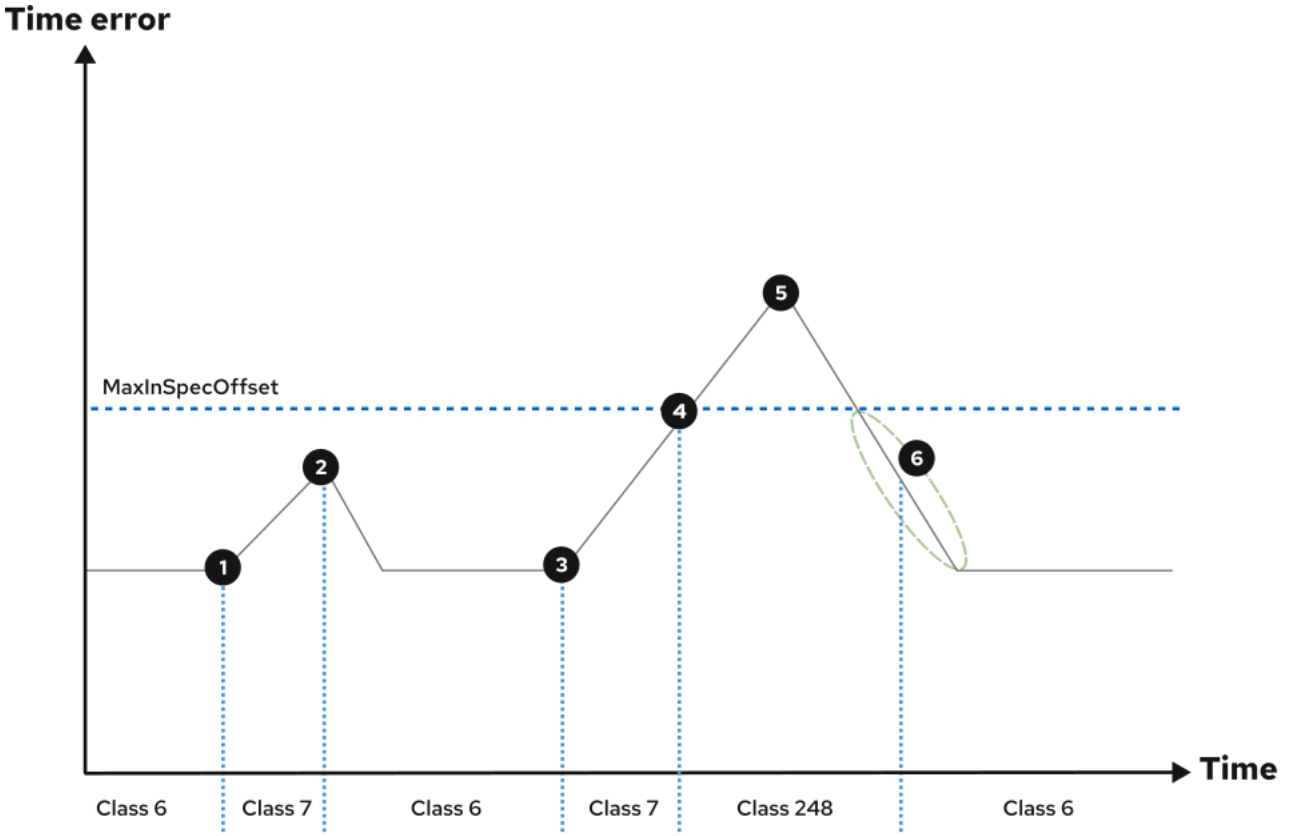


NOTE

The phase offset is converted from picoseconds to nanoseconds. As a result, the calculated phase offset during holdover is expressed in nanoseconds, and the resulting slope is expressed in nanoseconds per second.

The following figure illustrates the holdover behavior in a T-GM clock with GNSS as the source:

Figure 6.5. Holdover in a T-GM clock with GNSS as the source



1 The GNSS signal is lost, causing the T-GM clock to enter the **HOLDOVER** mode. The T-GM clock maintains time accuracy by using its internal clock.

2 The GNSS signal is restored and the T-GM clock re-enters the **LOCKED** mode. When the GNSS signal is restored, the T-GM clock re-enters the **LOCKED** mode only after all dependent components in the synchronization chain, such as **ts2phc** offset, digital phase-locked loop (DPLL) phase offset, and GNSS offset, reach a stable **LOCKED** mode.

3 The GNSS signal is lost again, and the T-GM clock re-enters the **HOLDOVER** mode. The time error begins to increase.

4 The time error exceeds the **MaxInSpecOffset** threshold due to prolonged loss of traceability.

5 The GNSS signal is restored, and the T-GM clock resumes synchronization. The time error starts to decrease.

6 The time error decreases and falls back within the **MaxInSpecOffset** threshold.

Additional resources

- Grandmaster clock class sync state reference

6.2.7. Configuring dynamic leap seconds handling for PTP grandmaster clocks

The PTP Operator container image includes the latest **leap-seconds.list** file that is available at the time of release.

You can configure the PTP Operator to automatically update the leap second file by using Global Positioning System (GPS) announcements.

Leap second information is stored in an automatically generated **ConfigMap** resource named **leap-configmap** in the **openshift-ptp** namespace. The PTP Operator mounts the **leap-configmap** resource as a volume in the **linuxptp-daemon** pod that is accessible by the **ts2phc** process.

If the GPS satellite broadcasts new leap second data, the PTP Operator updates the **leap-configmap** resource with the new data. The **ts2phc** process picks up the changes automatically.

NOTE

The following procedure is provided as reference. The 4.18 version of the PTP Operator enables automatic leap second management by default.

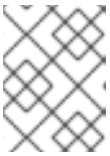
Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the PTP Operator and configured a PTP grandmaster clock (T-GM) in the cluster.

Procedure

1. Configure automatic leap second handling in the **phc2sysOpts** section of the **PtpConfig** CR. Set the following options:

```
phc2sysOpts: -r -u 0 -m -N 8 -R 16 -S 2 -s ens2f0 -n 24
```



NOTE

Previously, the T-GM required an offset adjustment in the **phc2sys** configuration (**-O -37**) to account for historical leap seconds. This is no longer needed.

2. Configure the Intel e810 NIC to enable periodical reporting of **NAV-TIMELS** messages by the GPS receiver in the **spec.profile.plugins.e810.ublxCmds** section of the **PtpConfig** CR. For example:

```
- args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
  - "-P"
  - "29.20"
  - "-p"
  - "CFG-MSG,1,38,248"
```

Verification

1. Validate that the configured T-GM is receiving **NAV-TIMELS** messages from the connected GPS. Run the following command:

```
$ oc -n openshift-ptp -c linuxptp-daemon-container exec -it $(oc -n openshift-ptp get pods -o
name | grep daemon) -- ubxtool -t -p NAV-TIMELS -P 29.20
```

Example output

```
1722509534.4417
UBX-NAV-STATUS:
  iTOW 384752000 gpsFix 5 flags 0xdd fixStat 0x0 flags2 0x8
  ttff 18261, msss 1367642864

1722509534.4419
UBX-NAV-TIMELS:
  iTOW 384752000 version 0 reserved2 0 0 0 srcOfCurrLs 2
  currLs 18 srcOfLsChange 2 lsChange 0 timeToLsEvent 70376866
  dateOfLsGpsWn 2441 dateOfLsGpsDn 7 reserved2 0 0 0
  valid x3

1722509534.4421
UBX-NAV-CLOCK:
  iTOW 384752000 clkB 784281 clkD 435 tAcc 3 fAcc 215

1722509535.4477
UBX-NAV-STATUS:
  iTOW 384753000 gpsFix 5 flags 0xdd fixStat 0x0 flags2 0x8
  ttff 18261, msss 1367643864

1722509535.4479
UBX-NAV-CLOCK:
  iTOW 384753000 clkB 784716 clkD 435 tAcc 3 fAcc 218
```

2. Validate that the **leap-configmap** resource has been successfully generated by the PTP Operator and is up to date with the latest version of the leap-seconds.list. Run the following command:

```
$ oc -n openshift-ptp get configmap leap-configmap -o jsonpath='{.data.<node_name>}'
```

Replace **<node_name>** with the node where you have installed and configured the PTP T-GM clock with automatic leap second management. Escape special characters in the node name. For example, **node-1\.example\.com**.

Example output

```
# Do not edit
# This file is generated automatically by linuxptp-daemon
#$  3913697179
#@  4291747200
2272060800     10   # 1 Jan 1972
2287785600     11   # 1 Jul 1972
2303683200     12   # 1 Jan 1973
2335219200     13   # 1 Jan 1974
2366755200     14   # 1 Jan 1975
2398291200     15   # 1 Jan 1976
2429913600     16   # 1 Jan 1977
2461449600     17   # 1 Jan 1978
```

```
2492985600    18    # 1 Jan 1979
2524521600    19    # 1 Jan 1980
2571782400    20    # 1 Jul 1981
2603318400    21    # 1 Jul 1982
2634854400    22    # 1 Jul 1983
2698012800    23    # 1 Jul 1985
2776982400    24    # 1 Jan 1988
2840140800    25    # 1 Jan 1990
2871676800    26    # 1 Jan 1991
2918937600    27    # 1 Jul 1992
2950473600    28    # 1 Jul 1993
2982009600    29    # 1 Jul 1994
3029443200    30    # 1 Jan 1996
3076704000    31    # 1 Jul 1997
3124137600    32    # 1 Jan 1999
3345062400    33    # 1 Jan 2006
3439756800    34    # 1 Jan 2009
3550089600    35    # 1 Jul 2012
3644697600    36    # 1 Jul 2015
3692217600    37    # 1 Jan 2017

#h  e65754d4 8f39962b aa854a61 661ef546 d2af0bfa
```

6.2.8. Configuring linuxptp services as a boundary clock

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**) as boundary clock by creating a **PtpConfig** custom resource (CR) object.

NOTE

Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as the boundary clock for your particular hardware and environment. This example CR does not configure PTP fast events. To configure PTP fast events, set appropriate values for **ptp4lOpts**, **ptp4lConf**, and **ptpClockThreshold**. **ptpClockThreshold** is used only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **boundary-clock-ptp-config.yaml** file.

   Example PTP boundary clock configuration

   ```
   apiVersion: ptp.openshift.io/v1
   kind: PtpConfig
   ```

```
metadata:
  name: boundary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: boundary-clock
      ptp4lOpts: "-2"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
      ptp4lConf: |
        # The interface name is hardware-specific
        [$iface_slave]
        masterOnly 0
        [$iface_master_1]
        masterOnly 1
        [$iface_master_2]
        masterOnly 1
        [$iface_master_3]
        masterOnly 1
        [global]
        #
        # Default Data Set
        #
        twoStepFlag 1
        slaveOnly 0
        priority1 128
        priority2 128
        domainNumber 24
        #utc_offset 37
        clockClass 248
        clockAccuracy 0xFE
        offsetScaledLogVariance 0xFFFF
        free_running 0
        freq_est_interval 1
        dscp_event 0
        dscp_general 0
        dataset_comparison G.8275.x
        G.8275.defaultDS.localPriority 128
        #
        # Port Data Set
        #
        logAnnounceInterval -3
        logSyncInterval -4
        logMinDelayReqInterval -4
        logMinPdelayReqInterval -4
        announceReceiptTimeout 3
        syncReceiptTimeout 0
        delayAsymmetry 0
        fault_reset_interval -4
        neighborPropDelayThresh 20000000
        masterOnly 0
        G.8275.portDS.localPriority 128
```

```
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval 0
kernel_leap 1
check_fup_sync 0
clock_class_threshold 135
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
max_frequency 900000000
clock_servo pi
sanity_freq_limit 200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
udp6_scope 0x0E
uds_address /var/run/ptp4l
#
# Default interface options
#
clock_type BC
network_transport L2
delay_mechanism E2E
time_stamping hardware
tsproc_mode filter
delay_filter moving_median
```

```
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  recommend:
    - profile: boundary-clock
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

Table 6.7. PTP boundary clock CR configuration options

CR field	Description
name	The name of the **PtpConfig** CR.
profile	Specify an array of one or more **profile** objects.
name	Specify the name of a profile object which uniquely identifies a profile object.
ptp4lOpts	Specify system config options for the **ptp4l** service. The options should not include the network interface name **-i <interface>** and service config file **-f /etc/ptp4l.conf** because the network interface name and the service config file are automatically appended.
ptp4lConf	Specify the required configuration to start **ptp4l** as boundary clock. For example, **ens1f0** synchronizes from a grandmaster clock and**ens1f3** synchronizes connected devices.
<interface_1>	The interface that receives the synchronization clock.
<interface_2>	The interface that sends the synchronization clock.
tx_timestamp_timeout	For Intel Columbiaville 800 Series NICs, set **tx_timestamp_timeout** to **50**.
boundary_clock_jbod	For Intel Columbiaville 800 Series NICs, ensure **boundary_clock_jbod** is set to **0**. For Intel Fortville X710 Series NICs, ensure **boundary_clock_jbod** is set to **1**.

CR field	Description
phc2sysOpts	Specify system config options for the **phc2sys** service. If this field is empty, the PTP Operator does not start the **phc2sys** service.
ptpSchedulingPolicy	Scheduling policy for ptp4l and phc2sys processes. Default value is **SCHED_OTHER**. Use **SCHED_FIFO** on systems that support FIFO scheduling.
ptpSchedulingPriority	Integer value from 1-65 used to set FIFO priority for **ptp4l** and **phc2sys** processes when **ptpSchedulingPolicy** is set to **SCHED_FIFO**. The **ptpSchedulingPriority** field is not used when **ptpSchedulingPolicy** is set to **SCHED_OTHER**.
ptpClockThreshold	Optional. If **ptpClockThreshold** is not present, default values are used for the **ptpClockThreshold** fields. **ptpClockThreshold** configures how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.
recommend	Specify an array of one or more **recommend** objects that define rules on how the **profile** should be applied to nodes.
.recommend.profile	Specify the **.recommend.profile** object name defined in the **profile** section.
.recommend.priority	Specify the **priority** with an integer value between **0** and **99**. A larger number gets lower priority, so a priority of **99** is lower than a priority of **10**. If a node can be matched with multiple profiles according to rules defined in the **match** field, the profile with the higher priority is applied to that node.
.recommend.match	Specify **.recommend.match** rules with **nodeLabel** or **nodeName** values.
.recommend.match.nodeLabel	Set **nodeLabel** with the **key** of the **node.Labels** field from the node object by using the **oc get nodes --show-labels** command. For example, **node-role.kubernetes.io/worker**.
.recommend.match.nodeName	Set **nodeName** with the value of the **node.Name** field from the node object by using the **oc get nodes** command. For example, **compute-1.example.com**.

2. Create the CR by running the following command:

```
$ oc create -f boundary-clock-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

   a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                            READY   STATUS    RESTARTS   AGE   IP            NODE
   linuxptp-daemon-4xkbb           1/1     Running   0          43m   10.1.196.24   compute-
   0.example.com
   linuxptp-daemon-tdspf           1/1     Running   0          43m   10.1.196.25   compute-
   1.example.com
   ptp-operator-657bbb64c8-2f8sj   1/1     Running   0          43m   10.129.0.61   control-
   plane-1.example.com
   ```

   b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

   ```
   $ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

   ```
   I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
   I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
   I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
   I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
   I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
   I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2
   I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
   I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
   ```

Additional resources

- Configuring FIFO priority scheduling for PTP hardware
- Configuring the PTP fast event notifications publisher

6.2.8.1. Configuring linuxptp services as boundary clocks for dual-NIC hardware

You can configure the **linuxptp** services (**ptp4l**, **phc2sys**) as boundary clocks for dual-NIC hardware by creating a **PtpConfig** custom resource (CR) object for each NIC.

Dual NIC hardware allows you to connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Create two separate **PtpConfig** CRs, one for each NIC, using the reference CR in "Configuring linuxptp services as a boundary clock" as the basis for each CR. For example:

   a. Create **boundary-clock-ptp-config-nic1.yaml**, specifying values for **phc2sysOpts**:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic1
  namespace: openshift-ptp
spec:
  profile:
  - name: "profile1"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: |
      [ens5f1]
      masterOnly 1
      [ens5f0]
      masterOnly 0
    ...
    phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16"
```

   where:

   ptp4lConf

   Specifies the required interfaces to start **ptp4l** as a boundary clock. For example, **ens5f0** synchronizes from a grandmaster clock and **ens5f1** synchronizes connected devices.

   phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16"

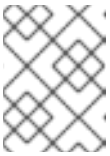
   Sets the required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

   b. Create **boundary-clock-ptp-config-nic2.yaml**, removing the **phc2sysOpts** field altogether to disable the **phc2sys** service for the second NIC:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-clock-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
  - name: "profile2"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: |
```

```
    [ens7f1]
    masterOnly 1
    [ens7f0]
    masterOnly 0
...
```

Specify the required interfaces to start **ptp4l** as a boundary clock on the second NIC.



NOTE

You must completely remove the **phc2sysOpts** field from the second **PtpConfig** CR to disable the **phc2sys** service on the second NIC.

2. Create the dual-NIC **PtpConfig** CRs by running the following commands:

   a. Create the CR that configures PTP for the first NIC:

   ```
   $ oc create -f boundary-clock-ptp-config-nic1.yaml
   ```

   b. Create the CR that configures PTP for the second NIC:

   ```
   $ oc create -f boundary-clock-ptp-config-nic2.yaml
   ```

Verification

- Check that the PTP Operator has applied the **PtpConfig** CRs for both NICs. Examine the logs for the **linuxptp** daemon corresponding to the node that has the dual-NIC hardware installed. For example, run the following command:

  ```
  $ oc logs linuxptp-daemon-cvgr6 -n openshift-ptp -c linuxptp-daemon-container
  ```

  Example output

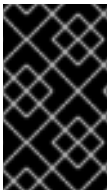
  ```
  ptp4l[80828.335]: [ptp4l.1.config] master offset          5 s2 freq   -5727 path delay       519
  ptp4l[80828.343]: [ptp4l.0.config] master offset         -5 s2 freq  -10607 path delay       533
  phc2sys[80828.390]: [ptp4l.0.config] CLOCK_REALTIME phc offset         1 s2 freq  -87239
  delay    539
  ```

6.2.8.2. Configuring linuxptp as a highly available system clock for dual-NIC Intel E810 PTP boundary clocks

You can configure the **linuxptp** services **ptp4l** and **phc2sys** as a highly available (HA) system clock for dual PTP boundary clocks (T-BC).

The highly available system clock uses multiple time sources from dual-NIC Intel E810 Salem channel hardware configured as two boundary clocks. Two boundary clocks instances participate in the HA setup, each with its own configuration profile. You connect each NIC to the same upstream leader clock with separate **ptp4l** instances for each NIC feeding the downstream clocks.

Create two **PtpConfig** custom resource (CR) objects that configure the NICs as T-BC and a third **PtpConfig** CR that configures high availability between the two NICs.



IMPORTANT

You set **phc2SysOpts** options once in the **PtpConfig** CR that configures HA. Set the **phc2sysOpts** field to an empty string in the **PtpConfig** CRs that configure the two NICs. This prevents individual **phc2sys** processes from being set up for the two profiles.

The third **PtpConfig** CR configures a highly available system clock service. The CR sets the **ptp4lOpts** field to an empty string to prevent the **ptp4l** process from running. The CR adds profiles for the **ptp4l** configurations under the **spec.profile.ptpSettings.haProfiles** key and passes the kernel socket path of those profiles to the **phc2sys** service. When a **ptp4l** failure occurs, the **phc2sys** service switches to the backup **ptp4l** configuration. When the primary profile becomes active again, the **phc2sys** service reverts to the original state.

IMPORTANT

Ensure that you set **spec.recommend.priority** to the same value for all three **PtpConfig** CRs that you use to configure HA.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.
- Configure a cluster node with Intel E810 Salem channel dual-NIC.

Procedure

1. Create two separate **PtpConfig** CRs, one for each NIC, using the CRs in "Configuring linuxptp services as boundary clocks for dual-NIC hardware" as a reference for each CR.

   a. Create the **ha-ptp-config-nic1.yaml** file, specifying an empty string for the **phc2sysOpts** field. For example:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ha-ptp-config-nic1
  namespace: openshift-ptp
spec:
  profile:
  - name: "ha-ptp-config-profile1"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: |
      [ens5f1]
      masterOnly 1
      [ens5f0]
      masterOnly 0
    #...
    phc2sysOpts: ""
```

   where:

ptp4lConf

Specifies the required interfaces to start **ptp4l** as a boundary clock. For example, **ens5f0** synchronizes from a grandmaster clock and **ens5f1** synchronizes connected devices.

phc2sysOpts: ""

Sets **phc2sysOpts** with an empty string. These values are populated from the **spec.profile.ptpSettings.haProfiles** field of the **PtpConfig** CR that configures high availability.

b. Apply the **PtpConfig** CR for NIC 1 by running the following command:

```
$ oc create -f ha-ptp-config-nic1.yaml
```

c. Create the **ha-ptp-config-nic2.yaml** file, specifying an empty string for the **phc2sysOpts** field. For example:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ha-ptp-config-nic2
  namespace: openshift-ptp
spec:
  profile:
  - name: "ha-ptp-config-profile2"
    ptp4lOpts: "-2 --summary_interval -4"
    ptp4lConf: |
      [ens7f1]
      masterOnly 1
      [ens7f0]
      masterOnly 0
    #...
    phc2sysOpts: ""
```

d. Apply the **PtpConfig** CR for NIC 2 by running the following command:

```
$ oc create -f ha-ptp-config-nic2.yaml
```

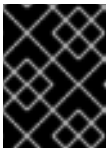
2. Create the **PtpConfig** CR that configures the HA system clock. For example:

a. Create the **ptp-config-for-ha.yaml** file. Set **haProfiles** to match the **metadata.name** fields that are set in the **PtpConfig** CRs that configure the two NICs. For example: **haProfiles: ha-ptp-config-nic1,ha-ptp-config-nic2**

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: boundary-ha
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "boundary-ha"
      ptp4lOpts: ""
      phc2sysOpts: "-a -r -n 24"
```

```
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
        haProfiles: "$profile1,$profile2"
    recommend:
      - profile: "boundary-ha"
        priority: 4
        match:
          - nodeLabel: "node-role.kubernetes.io/$mcp"
```

Set the **ptp4lOpts** field to an empty string. If it is not empty, the **p4ptl** process starts with a critical error.



IMPORTANT

Do not apply the high availability **PtpConfig** CR before the **PtpConfig** CRs that configure the individual NICs.

b. Apply the HA **PtpConfig** CR by running the following command:

```
$ oc create -f ptp-config-for-ha.yaml
```

Verification

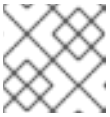
- Verify that the PTP Operator has applied the **PtpConfig** CRs correctly. Perform the following steps:

  a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

  ```
  $ oc get pods -n openshift-ptp -o wide
  ```

  Example output

  ```
  NAME                            READY   STATUS    RESTARTS   AGE   IP             NODE
  linuxptp-daemon-4xkrb           1/1     Running   0          43m   10.1.196.24    compute-
  0.example.com
  ptp-operator-657bbq64c8-2f8sj   1/1     Running   0          43m   10.129.0.61    control-
  plane-1.example.com
  ```

  NOTE

  There should be only one **linuxptp-daemon** pod.

  b. Check that the profile is correct by running the following command. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile.

  ```
  $ oc logs linuxptp-daemon-4xkrb -n openshift-ptp -c linuxptp-daemon-container
  ```

  Example output

  ```
  I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
  ```

```
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: ha-ptp-config-profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface:
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
```

6.2.9. Configuring linuxptp services as an ordinary clock

You can configure **linuxptp** services (**ptp4l**, **phc2sys**) as ordinary clock by creating a **PtpConfig** custom resource (CR) object.

NOTE

Use the following example **PtpConfig** CR as the basis to configure **linuxptp** services as an ordinary clock for your particular hardware and environment. This example CR does not configure PTP fast events. To configure PTP fast events, set appropriate values for **ptp4lOpts**, **ptp4lConf**, and **ptpClockThreshold**. **ptpClockThreshold** is required only when events are enabled. See "Configuring the PTP fast event notifications publisher" for more information.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **ordinary-clock-ptp-config.yaml** file.

   Example PTP ordinary clock configuration

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ordinary-clock
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: ordinary-clock
      # The interface name is hardware-specific
      interface: $interface
      ptp4lOpts: "-2 -s"
      phc2sysOpts: "-a -r -n 24"
      ptpSchedulingPolicy: SCHED_FIFO
      ptpSchedulingPriority: 10
      ptpSettings:
        logReduce: "true"
```

```
    ptp4lConf: |
      [global]
      #
      # Default Data Set
      #
      twoStepFlag 1
      slaveOnly 1
      priority1 128
      priority2 128
      domainNumber 24
      #utc_offset 37
      clockClass 255
      clockAccuracy 0xFE
      offsetScaledLogVariance 0xFFFF
      free_running 0
      freq_est_interval 1
      dscp_event 0
      dscp_general 0
      dataset_comparison G.8275.x
      G.8275.defaultDS.localPriority 128
      #
      # Port Data Set
      #
      logAnnounceInterval -3
      logSyncInterval -4
      logMinDelayReqInterval -4
      logMinPdelayReqInterval -4
      announceReceiptTimeout 3
      syncReceiptTimeout 0
      delayAsymmetry 0
      fault_reset_interval -4
      neighborPropDelayThresh 20000000
      masterOnly 0
      G.8275.portDS.localPriority 128
      #
      # Run time options
      #
      assume_two_step 0
      logging_level 6
      path_trace_enabled 0
      follow_up_info 0
      hybrid_e2e 0
      inhibit_multicast_service 0
      net_sync_monitor 0
      tc_spanning_tree 0
      tx_timestamp_timeout 50
      unicast_listen 0
      unicast_master_table 0
      unicast_req_duration 3600
      use_syslog 1
      verbose 0
      summary_interval 0
      kernel_leap 1
      check_fup_sync 0
      clock_class_threshold 7
      #
```

```
      # Servo Options
      #
      pi_proportional_const 0.0
      pi_integral_const 0.0
      pi_proportional_scale 0.0
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      max_frequency 900000000
      clock_servo pi
      sanity_freq_limit 200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type OC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0xA0
  recommend:
    - profile: ordinary-clock
      priority: 4
      match:
        - nodeLabel: "node-role.kubernetes.io/$mcp"
```

Table 6.8. PTP ordinary clock CR configuration options

CR field	Description
name	The name of the **PtpConfig** CR.
profile	Specify an array of one or more **profile** objects. Each profile must be uniquely named.
interface	Specify the network interface to be used by the **ptp4l** service, for example **ens787f1**.
ptp4lOpts	Specify system config options for the **ptp4l** service, for example **-2** to select the IEEE 802.3 network transport. The options should not include the network interface name **-i <interface>** and service config file **-f /etc/ptp4l.conf** because the network interface name and the service config file are automatically appended. Append **--summary_interval -4** to use PTP fast events with this interface.
phc2sysOpts	Specify system config options for the **phc2sys** service. If this field is empty, the PTP Operator does not start the **phc2sys** service. For Intel Columbiaville 800 Series NICs, set **phc2sysOpts** options to **-a -r -m -n 24 -N 8 -R 16**. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.
ptp4lConf	Specify a string that contains the configuration to replace the default **/etc/ptp4l.conf** file. To use the default configuration, leave the field empty.
tx_timestamp_timeout	For Intel Columbiaville 800 Series NICs, set **tx_timestamp_timeout** to **50**.
boundary_clock_jbod	For Intel Columbiaville 800 Series NICs, set **boundary_clock_jbod** to **0**.
ptpSchedulingPolicy	Scheduling policy for **ptp4l** and **phc2sys** processes. Default value is **SCHED_OTHER**. Use **SCHED_FIFO** on systems that support FIFO scheduling.
ptpSchedulingPriority	Integer value from 1-65 used to set FIFO priority for **ptp4l** and **phc2sys** processes when **ptpSchedulingPolicy** is set to **SCHED_FIFO**. The **ptpSchedulingPriority** field is not used when **ptpSchedulingPolicy** is set to **SCHED_OTHER**.

CR field	Description
ptpClockThreshold	Optional. If **ptpClockThreshold** is not present, default values are used for the **ptpClockThreshold** fields. **ptpClockThreshold** configures how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.
recommend	Specify an array of one or more **recommend** objects that define rules on how the **profile** should be applied to nodes.
.recommend.profile	Specify the **.recommend.profile** object name defined in the **profile** section.
.recommend.priority	Set **.recommend.priority** to **0** for ordinary clock.
.recommend.match	Specify **.recommend.match** rules with **nodeLabel** or **nodeName** values.
.recommend.match.nodeLabel	Set **nodeLabel** with the **key** of the **node.Labels** field from the node object by using the **oc get nodes --show-labels** command. For example, **node-role.kubernetes.io/worker**.
.recommend.match.nodeName	Set **nodeName** with the value of the **node.Name** field from the node object by using the **oc get nodes** command. For example, **compute-1.example.com**.

2. Create the **PtpConfig** CR by running the following command:

```
$ oc create -f ordinary-clock-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.
   a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

Example output

```
NAME                    READY   STATUS    RESTARTS   AGE   IP          NODE
```

```
linuxptp-daemon-4xkbb          1/1   Running   0          43m   10.1.196.24   compute-
0.example.com
linuxptp-daemon-tdspf          1/1   Running   0          43m   10.1.196.25   compute-
1.example.com
ptp-operator-657bbb64c8-2f8sj  1/1   Running   0          43m   10.129.0.61   control-
plane-1.example.com
```

b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

```
$ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
```

Example output

```
I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: profile1
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2 -s
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24
I1115 09:41:17.117626 4143292 daemon.go:116] ------------------------------------
```

Additional resources

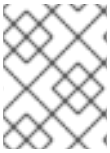
- Configuring FIFO priority scheduling for PTP hardware
- Configuring the PTP fast event notifications publisher

6.2.9.1. Intel Columbiaville E800 series NIC as PTP ordinary clock reference

The following table describes the changes that you must make to the reference PTP configuration to use Intel Columbiaville E800 series NICs as ordinary clocks. Make the changes in a **PtpConfig** custom resource (CR) that you apply to the cluster.

Table 6.9. Recommended PTP settings for Intel Columbiaville NIC

PTP configuration	Recommended setting
phc2sysOpts	**-a -r -m -n 24 -N 8 -R 16**
tx_timestamp_timeout	**50**
boundary_clock_jbod	**0**



NOTE

For **phc2sysOpts**, **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

6.2.9.2. Configuring linuxptp services as an ordinary clock with dual-port NIC redundancy

You can configure **linuxptp** services (**ptp4l**, **phc2sys**) as an ordinary clock with dual-port NIC redundancy by creating a **PtpConfig** custom resource (CR) object.

In a dual-port NIC configuration for an ordinary clock, if one port fails, the standby port takes over, maintaining PTP timing synchronization.

IMPORTANT

Configuring linuxptp services as an ordinary clock with dual-port NIC redundancy is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.
- The node uses an x86_64 architecture with a dual-port NIC.

Procedure

1. Create the following **PtpConfig** CR, and then save the YAML in the **oc-dual-port-ptp-config.yaml** file.

   Example PTP ordinary clock dual-port configuration

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: ordinary-clock-1
  namespace: openshift-ptp
spec:
  profile:
  - name: oc-dual-port
    phc2sysOpts: -a -r -n 24 -N 8 -R 16 -u 0
    ptp4lConf: |-
      [ens3f2]
      masterOnly 0
      [ens3f3]
      masterOnly 0

      [global]
      #
      # Default Data Set
```

```
    #
    slaveOnly 1
#...
```

where:

phc2sysOpts: -a -r -n 24 -N 8 -R 16 -u 0

Specifies the system config options for the **phc2sys** service.

ptp4lConf

Specifies the interface configuration for the **ptp4l** service. In this example, setting **masterOnly 0** for the **ens3f2** and **ens3f3** interfaces enables both ports on the **ens3** interface to run as leader or follower clocks. In combination with the **slaveOnly 1** specification, this configuration ensures one port operates as the active ordinary clock, and the other port operates as a standby ordinary clock in the **Listening** port state.

slaveOnly 1

Configures **ptp4l** to run as an ordinary clock only.

2. Create the **PtpConfig** CR by running the following command:

```
$ oc create -f oc-dual-port-ptp-config.yaml
```

Verification

1. Check that the **PtpConfig** profile is applied to the node.

   a. Get the list of pods in the **openshift-ptp** namespace by running the following command:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                          READY   STATUS    RESTARTS   AGE   IP            NODE
   linuxptp-daemon-4xkbb         1/1     Running   0          43m   10.1.196.24   compute-
   0.example.com
   linuxptp-daemon-tdspf         1/1     Running   0          43m   10.1.196.25   compute-
   1.example.com
   ptp-operator-657bbb64c8-2f8sj 1/1     Running   0          43m   10.129.0.61   control-
   plane-1.example.com
   ```

   b. Check that the profile is correct. Examine the logs of the **linuxptp** daemon that corresponds to the node you specified in the **PtpConfig** profile. Run the following command:

   ```
   $ oc logs linuxptp-daemon-4xkbb -n openshift-ptp -c linuxptp-daemon-container
   ```

   Example output

   ```
   I1115 09:41:17.117596 4143292 daemon.go:107] in applyNodePTPProfile
   I1115 09:41:17.117604 4143292 daemon.go:109] updating NodePTPProfile to:
   I1115 09:41:17.117607 4143292 daemon.go:110] ------------------------------------
   I1115 09:41:17.117612 4143292 daemon.go:102] Profile Name: oc-dual-port
   ```

```
I1115 09:41:17.117616 4143292 daemon.go:102] Interface: ens787f1
I1115 09:41:17.117620 4143292 daemon.go:102] Ptp4lOpts: -2 --summary_interval -4
I1115 09:41:17.117623 4143292 daemon.go:102] Phc2sysOpts: -a -r -n 24 -N 8 -R 16 -u
0
I1115 09:41:17.117626 4143292 daemon.go:116] -----------------------------------
```

Additional resources

- Configuring linuxptp services as ordinary clock
- Using dual-port NICs to improve redundancy for PTP ordinary clocks
- Using dual-port NICs to improve redundancy for PTP ordinary clocks

6.2.10. Configuring FIFO priority scheduling for PTP hardware

In telco or other deployment types that require low latency performance, PTP daemon threads run in a constrained CPU footprint alongside the rest of the infrastructure components. By default, PTP threads run with the **SCHED_OTHER** policy. Under high load, these threads might not get the scheduling latency they require for error-free operation.

To mitigate against potential scheduling latency errors, you can configure the PTP Operator **linuxptp** services to allow threads to run with a **SCHED_FIFO** policy. If **SCHED_FIFO** is set for a **PtpConfig** CR, then **ptp4l** and **phc2sys** will run in the parent container under **chrt** with a priority set by the **ptpSchedulingPriority** field of the **PtpConfig** CR.

NOTE

Setting **ptpSchedulingPolicy** is optional, and is only required if you are experiencing latency errors.

Procedure

1. Edit the **PtpConfig** CR profile:

```
$ oc edit PtpConfig -n openshift-ptp
```

2. Change the **ptpSchedulingPolicy** and **ptpSchedulingPriority** fields:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
  - name: "profile1"
...
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
```

where:

ptpSchedulingPolicy: SCHED_FIFO

Sets the scheduling policy for **ptp4l** and **phc2sys** processes. Use **SCHED_FIFO** on systems that support FIFO scheduling.

ptpSchedulingPriority: 10

Sets the integer value 1-65 used to configure FIFO priority for **ptp4l** and **phc2sys** processes.

3. Save and exit to apply the changes to the **PtpConfig** CR.

Verification

1. Get the name of the **linuxptp-daemon** pod and corresponding node where the **PtpConfig** CR has been applied:

```
$ oc get pods -n openshift-ptp -o wide
```

Example output

```
NAME                           READY   STATUS    RESTARTS   AGE     IP           NODE
linuxptp-daemon-gmv2n          3/3     Running   0          1d17h   10.1.196.24   compute-
0.example.com
linuxptp-daemon-lgm55          3/3     Running   0          1d17h   10.1.196.25   compute-
1.example.com
ptp-operator-3r4dcvf7f4-zndk7  1/1     Running   0          1d7h    10.129.0.61   control-plane-
1.example.com
```

2. Check that the **ptp4l** process is running with the updated **chrt** FIFO priority:

```
$ oc -n openshift-ptp logs linuxptp-daemon-lgm55 -c linuxptp-daemon-container|grep chrt
```

Example output

```
I1216 19:24:57.091872 1600715 daemon.go:285] /bin/chrt -f 65 /usr/sbin/ptp4l -f
/var/run/ptp4l.0.config -2  --summary_interval -4 -m
```

6.2.11. Configuring log filtering for linuxptp services

Modify the **PtpConfig** custom resource (CR) to configure basic log filtering and exclude log messages that report the master offset value.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

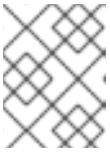
Procedure

1. Edit the **PtpConfig** CR:

```
$ oc edit PtpConfig -n openshift-ptp
```

2. In **spec.profile**, add the **ptpSettings.logReduce** specification and set the value to **true**:

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
  - name: "profile1"
...
    ptpSettings:
      logReduce: "true"
```



NOTE

For debugging purposes, you can revert this specification to **False** to include the master offset messages.

3. Save and exit to apply the changes to the **PtpConfig** CR.

Verification

1. Get the name of the **linuxptp-daemon** pod and corresponding node where the **PtpConfig** CR has been applied:

```
$ oc get pods -n openshift-ptp -o wide
```

Example output

```
NAME                           READY   STATUS    RESTARTS   AGE     IP           NODE
linuxptp-daemon-gmv2n          3/3     Running   0          1d17h   10.1.196.24   compute-
0.example.com
linuxptp-daemon-lgm55          3/3     Running   0          1d17h   10.1.196.25   compute-
1.example.com
ptp-operator-3r4dcvf7f4-zndk7  1/1     Running   0          1d7h    10.129.0.61   control-plane-
1.example.com
```

2. Verify that master offset messages are excluded from the logs by running the following command:

```
$ oc -n openshift-ptp logs <linux_daemon_container> -c linuxptp-daemon-container | grep
"master offset"
```

- **<linux_daemon_container>** is the name of the **linuxptp-daemon** pod, for example **linuxptp-daemon-gmv2n**.
  When you configure the **logReduce** specification, this command does not report any instances of **master offset** in the logs of the **linuxptp** daemon.

6.2.12. Configuring GNSS failover to NTP for time synchronization continuity

Automatic failover from global navigation satellite system (GNSS) to Network Time Protocol (NTP) maintains time synchronization continuity when the primary signal is lost, ensuring system stability for telco operations.

Telco operators require time source redundancy to ensure time synchronization continuity and system stability.

OpenShift Container Platform provides automatic failover capabilities to maintain synchronization. The system utilizes GNSS (delivered by **phc2sys**) as the primary time source. To protect against primary signal loss, such as jamming or antenna failure, the system automatically transitions to the secondary time source, NTP delivered by **chronyd**. Upon signal recovery, the system automatically switches back to and resumes synchronization with **phc2sys**.

You can control the resilience of the time synchronization by setting the **ts2phc.holdover** parameter in seconds. This value dictates the maximum time the internal control algorithm can continue synchronizing the PHC after the main time of day (ToD) source such as a GNSS receiver is lost. The algorithm can only continue if it remains in a stable state (SERVO_LOCKED_STABLE). When the process exits this configured holdover period, it signifies an unrecoverable primary signal loss. The system then allows failover to a secondary source such as NTP.

6.2.12.1. Creating a PTP Grandmaster configuration with GNSS failover

Configure a Precision Time Protocol (PTP) Telecom Grandmaster clock with automatic failover from global navigation satellite system (GNSS) to Network Time Protocol (NTP) when satellite signals are unavailable.

This procedure configures a T-GM (Telecom Grandmaster) clock that uses an Intel E810 Westport Channel NIC as the PTP grandmaster clock with GNSS to NTP failover capabilities.

Prerequisites

- For T-GM clocks in production environments, install an Intel E810 Westport Channel NIC in the bare-metal cluster host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Verify the PTP Operator installation by running the following command:

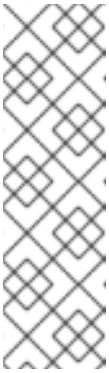
```
$ oc get pods -n openshift-ptp -o wide
```

The output is similar to the following listing the PTP Operator pod and the **linuxptp-daemon** pods:

```
NAME                          READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
linuxptp-daemon-4xk9m         2/2     Running   0          15m   192.168.1.101   worker-
0.cluster.local     <none>         <none>
```

```
linuxptp-daemon-7bv2n          2/2     Running   0          15m   192.168.1.102   worker-
1.cluster.local     <none>          <none>
linuxptp-daemon-9cp4r          2/2     Running   0          15m   192.168.1.103   worker-
2.cluster.local     <none>          <none>
linuxptp-daemon-kw8h5          2/2     Running   0          15m   192.168.1.104   worker-
3.cluster.local     <none>          <none>
linuxptp-daemon-m3j7t          2/2     Running   0          15m   192.168.1.105   worker-
4.cluster.local     <none>          <none>
ptp-operator-75c77dbf86-xm9kl  1/1     Running   0          20m   10.129.0.45     master-
1.cluster.local     <none>          <none>
```

- **ptp-operator-***: The PTP Operator pod (one instance in the cluster)
- **linuxptp-daemon-***: The linuxptp daemon pods. A daemon pod runs on each node that matches the PtpConfig profile. Each daemon pod should show **2/2** in the READY column, indicating both containers (**linuxptp-daemon-container** and **kube-rbac-proxy**) are running.



NOTE

The number of **linuxptp-daemon** pods is determined by the node labels defined in the **PtpOperatorConfig** which controls the DaemonSet deployment. The PtpConfig profile matching, as shown in Step 4, only determines which specific PTP settings are applied on the running daemons. In this example, the operator configuration targets all 5 worker nodes. For single-node OpenShift clusters, you will see only one **linuxptp-daemon** pod, as the configuration targets only the control plane node which acts as the worker.

2. Check which network interfaces support hardware timestamping by running the following command:

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

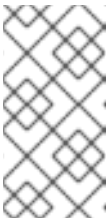
The output is similar to the following showing NodePtpDevice resources for nodes with PTP-capable network interfaces:

```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    name: worker-0.cluster.local
    namespace: openshift-ptp
  spec: {}
  status:
    devices:
    - name: ens7f0
      hwConfig:
        phcIndex: 0
    - name: ens7f1
      hwConfig:
        phcIndex: 1
- apiVersion: ptp.openshift.io/v1
```

```
    kind: NodePtpDevice
    metadata:
      name: worker-1.cluster.local
      namespace: openshift-ptp
    spec: {}
    status:
      devices:
      - name: ens7f0
        hwConfig:
          phcIndex: 0
      - name: ens7f1
        hwConfig:
          phcIndex: 1
  kind: List
  metadata:
    resourceVersion: ""
```

In this example output:

- **ens7f0** and **ens7f1** are PTP-capable interfaces (Intel E810 NIC ports).
- **phcIndex** indicates the PTP Hardware Clock number (maps to **/dev/ptp0**, **/dev/ptp1**, and so on).



NOTE

The output shows one NodePtpDevice resource for each node with PTP-capable interfaces. In this example, five worker nodes have Intel E810 NICs. For single-node OpenShift clusters, you would see only one NodePtpDevice resource.

3. The PTP profile uses node labels for matching. Check your machine config pool (MCP) to find the node labels by running the following command:

```
$ oc get mcp
```

The output is similar to the following:

```
NAME     CONFIG                 UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT   UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master   rendered-master-a1b1**  True     False     False     3            3                3
0                   45d
worker   rendered-worker-f6e5**  True     False     False     5            5                5
0                   45d
```

NOTE

The CONFIG column shows a truncated hash of the rendered MachineConfig. In actual output, this will be a full 64-character hash such as **rendered-master-a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6**.

- In this example, the **<MCP-name>** is **worker** for worker nodes and **master** for control plane nodes. Most T-GM deployments use worker nodes, so you would use **worker** as the **<MCP-name>**.
- For single-node OpenShift clusters, the **<MCP-name>** is **master** (the worker MCP will show **MACHINECOUNT** of 0).

4. Create a **PtpConfig** custom resource (CR) that configures the T-GM clock with GNSS to NTP failover. Save the following YAML configuration to a file named **ptp-config-gnss-ntp-failover.yaml**, replacing **<MCP-name>** with the name of your machine config pool from the previous step.

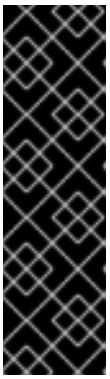
```
# The grandmaster profile is provided for testing only
# It is not installed on production clusters
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s ens7f0 -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"

    # --- FAILOVER CONFIGURATION ---
    # Holdover time: 14400 seconds (4 hours) before switching to NTP
    ts2phcOpts: "--ts2phc.holdover 14400"

    # Configure Chronyd (Secondary Time Source)
    chronydOpts: "-d"
    chronydConf: |
      server time.nist.gov iburst
      makestep 1.0 -1
      pidfile /var/run/chronyd.pid

    plugins:
      # E810 Hardware-Specific Configuration
      e810:
        enableDefaultConfig: false
        settings:
          LocalHoldoverTimeout: 14400
          LocalMaxHoldoverOffSet: 1500
          MaxInSpecOffset: 1500
        pins:
          # Syntax guide:
          # - The 1st number in each pair must be one of:
          #    0 - Disabled
          #    1 - RX
          #    2 - TX
```

```
    # - The 2nd number in each pair must match the channel number
    ens7f0:
      SMA1: 0 1
      SMA2: 0 2
      U.FL1: 0 1
      U.FL2: 0 2
  ublxCmds:
    - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
        - "-P"
        - "29.20"
        - "-z"
        - "CFG-HW-ANT_CFG_VOLTCTRL,1"
      reportOutput: false
    - args: #ubxtool -P 29.20 -e GPS
        - "-P"
        - "29.20"
        - "-e"
        - "GPS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d Galileo
        - "-P"
        - "29.20"
        - "-d"
        - "Galileo"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d GLONASS
        - "-P"
        - "29.20"
        - "-d"
        - "GLONASS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d BeiDou
        - "-P"
        - "29.20"
        - "-d"
        - "BeiDou"
      reportOutput: false
    - args: #ubxtool -P 29.20 -d SBAS
        - "-P"
        - "29.20"
        - "-d"
        - "SBAS"
      reportOutput: false
    - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
        - "-P"
        - "29.20"
        - "-t"
        - "-w"
        - "5"
        - "-v"
        - "1"
        - "-e"
        - "SURVEYIN,600,50000"
      reportOutput: true
    - args: #ubxtool -P 29.20 -p MON-HW
        - "-P"
```

```
          - "29.20"
          - "-p"
          - "MON-HW"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
          - "-P"
          - "29.20"
          - "-p"
          - "CFG-MSG,1,38,248"
        reportOutput: true

  # NTP Failover Plugin
  ntpfailover:
    gnssFailover: true

# --- GNSS (ts2phc) CONFIGURATION (Primary Source) ---
ts2phcConf: |
  [nmea]
  ts2phc.master 1
  [global]
  use_syslog  0
  verbose 1
  logging_level 7
  ts2phc.pulsewidth 100000000
  ts2phc.nmea_serialport /dev/ttyGNSS_1700_0
  leapfile  /usr/share/zoneinfo/leap-seconds.list
  [ens7f0]
  ts2phc.extts_polarity rising
  ts2phc.extts_correction 0

# --- PTP4L CONFIGURATION (Grandmaster Role) ---
ptp4lConf: |
  [ens7f0]
  masterOnly 1
  [ens7f1]
  masterOnly 1
  [global]
  #
  # Default Data Set
  #
  twoStepFlag 1
  priority1 128
  priority2 128
  domainNumber 24
  #utc_offset 37
  clockClass 6
  clockAccuracy 0x27
  offsetScaledLogVariance 0xFFFF
  free_running 0
  freq_est_interval 1
  dscp_event 0
  dscp_general 0
  dataset_comparison G.8275.x
  G.8275.defaultDS.localPriority 128
  #
  # Port Data Set
```

```
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
pi_integral_const 0.0
pi_proportional_scale 0.0
pi_proportional_exponent -0.3
pi_proportional_norm_max 0.7
pi_integral_scale 0.0
pi_integral_exponent 0.4
pi_integral_norm_max 0.3
step_threshold 2.0
first_step_threshold 0.00002
clock_servo pi
sanity_freq_limit  200000000
ntpshm_segment 0
#
# Transport options
#
transportSpecific 0x0
ptp_dst_mac 01:1B:19:00:00:00
p2p_dst_mac 01:80:C2:00:00:0E
udp_ttl 1
```

```
    udp6_scope 0x0E
    uds_address /var/run/ptp4l
    #
    # Default interface options
    #
    clock_type BC
    network_transport L2
    delay_mechanism E2E
    time_stamping hardware
    tsproc_mode filter
    delay_filter moving_median
    delay_filter_length 10
    egressLatency 0
    ingressLatency 0
    boundary_clock_jbod 0
    #
    # Clock description
    #
    productDescription ;;
    revisionData ;;
    manufacturerIdentity 00:00:00
    userDescription ;
    timeSource 0x20
  ptpClockThreshold:
    holdOverTimeout: 5
    maxOffsetThreshold: 100
    minOffsetThreshold: -100
recommend:
- profile: "grandmaster"
  priority: 4
  match:
  - nodeLabel: node-role.kubernetes.io/<MCP-name>
```



IMPORTANT

Replace the example interface names (**ens7f0**, **ens7f1**) with your actual E810 NIC interface names found in step 2. Common E810 interface naming patterns include **ens7f0**, **ens8f0**, **eth0**, **enp2s0f0**, and so on. The exact name depends on your system firmware settings and Linux network device naming conventions. Also replace **/dev/ttyGNSS_1700_0** with your actual GNSS serial port device path. For single-node OpenShift clusters, replace **<MCP-name>** with **master** in the nodeLabel match. For multi-node clusters using worker nodes as T-GM, use **worker**.

The configuration includes the following components:

- **PTP4L options:**
  - **-2**: Use PTP version 2
  - **--summary_interval -4**: Log summary every 2^(-4) = 0.0625 seconds
- **PHC2SYS options:**
  - **-r**: Synchronize system clock from PTP hardware clock

- **-u 0**: Update rate multiplier
- **-m**: Print messages to stdout
- **-N 8**: Domain number for ptp4l
- **-R 16**: Update rate
- **-s ens7f0**: Source interface (replace with your E810 interface name)
- **-n 24**: Domain number

- Failover configuration:
  - **ts2phcOpts --ts2phc.holdover 14400**: 4-hour holdover before switching to NTP
  - **chronydConf**: NTP server configuration for failover replace **time.nist.gov** with your preferred NTP server
  - **ntpfailover plugin**: Enables automatic GNSS-to-NTP switching with **gnssFailover: true**
- E810 plugin configuration:
  - **LocalHoldoverTimeout: 14400**: E810 hardware holdover timeout (4 hours)
  - **pins**: Configuration for 1PPS input on E810 physical pins (U.FL2, SMA1, SMA2, U.FL1)
  - **ublxCmds**: Commands to configure u-blox GNSS receiver (enable GPS, disable other constellations, set survey-in mode)
- GNSS (ts2phc) configuration:
  - **ts2phc.nmea_serialport /dev/ttyGNSS_1700_0**: GNSS serial port device path (replace with your actual GNSS device)
  - **ts2phc.extts_polarity rising**: 1PPS signal on rising edge
  - **ts2phc.pulsewidth 100000000**: 1PPS pulse width in nanoseconds
- PTP4L configuration:
  - **masterOnly 1**: Interface acts only as PTP master
  - **clockClass 6**: GPS-synchronized quality level
  - **domainNumber 24**: PTP domain
  - **clock_type BC**: Boundary Clock mode
  - **time_stamping hardware**: Use hardware timestamps from E810 NIC

5. Apply the **PtpConfig** CR by running the following command:

```
$ oc apply -f ptp-config-gnss-ntp-failover.yaml
```

The output is similar to the following:

```
ptpconfig.ptp.openshift.io/grandmaster created
```

Verification

1. The PTP daemon checks for profile updates every 30 seconds. Wait approximately 30 seconds, then verify by running the following command:

   ```
   $ oc get ptpconfig -n openshift-ptp
   ```

   The output is similar to the following:

   ```
   NAME          AGE
   grandmaster   2m
   ```

2. Check the NodePtpDevice to see if the profile is applied by running the following command, replacing **<node_name>** with your node hostname:

   ```
   $ oc describe nodeptpdevice <node_name> -n openshift-ptp
   ```

   For example, on a multi-node cluster with worker nodes: **worker-0.cluster.local**

   For single-node OpenShift clusters, use the control plane node name, which you can find by running:

   ```
   $ oc get nodes
   ```

3. Check if the profile is being loaded by monitoring the daemon logs:

   ```
   $ oc get pods -n openshift-ptp | grep linuxptp-daemon
   ```

   Then check the logs, replacing **<linuxptp-daemon-pod>** with the actual pod name from the previous command:

   ```
   $ oc logs -n openshift-ptp <linuxptp-daemon-pod> -c linuxptp-daemon-container --tail=100
   ```

   Success indicators in the logs are:

   - **load profiles** - Profile is being loaded
   - **in applyNodePTPProfiles** - Profile is being applied
   - No **ptp profile doesn't exist for node** errors

4. Check **chronyd** status to verify NTP is running as the secondary time source by running the following command:

   ```
   $ oc logs -n openshift-ptp <linuxptp-daemon-pod> -c linuxptp-daemon-container | grep chronyd
   ```

   The output is similar to the following:

```
chronyd version 4.5 starting
Added source ID#0000000001 (time.nist.gov)
```

5. Check GNSS/gpsd by running the following command:

```
$ oc logs -n openshift-ptp <linuxptp-daemon-pod> -c linuxptp-daemon-container | grep gpsd
```

The output shows the following when GNSS is functioning correctly:

- **gpsd** starting successfully
- No **No such file or directory** errors exist

6. Check **ts2phc** (GNSS synchronization) status by running the following command:

```
$ oc logs -n openshift-ptp <linuxptp-daemon-pod> -c linuxptp-daemon-container | grep ts2phc
```

7. Check **phc2sys** (system clock sync) status by running the following command:

```
$ oc logs -n openshift-ptp <linuxptp-daemon-pod> -c linuxptp-daemon-container | grep
phc2sys
```

The output shows synchronization status messages for **phc2sys**.

```
phc2sys[xxx]: CLOCK_REALTIME phc offset -17 s2 freq -13865 delay 2305
```

6.2.12.2. Creating a PTP Grandmaster configuration with GNSS failover on Single Node OpenShift

This procedure configures a T-GM (Telecom Grandmaster) clock on single-node OpenShift that uses an Intel E810 Westport Channel NIC as the PTP grandmaster clock with GNSS to NTP failover capabilities.

Prerequisites

- For T-GM clocks in production environments, install an Intel E810 Westport Channel NIC in the bare metal single-node OpenShift host.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator.

Procedure

1. Verify the PTP Operator installation by running the following command:

```
$ oc get pods -n openshift-ptp -o wide
```

The output is similar to the following listing the PTP Operator pod and the single **linuxptp-daemon** pod:

```
NAME                        READY   STATUS    RESTARTS   AGE   IP           NODE
NOMINATED NODE   READINESS GATES
linuxptp-daemon-xz8km       2/2     Running   0          15m   192.168.1.50   mysno-
sno.demo.lab     <none>         <none>
ptp-operator-75c77dbf86-xm9kl   1/1     Running   0          20m   10.129.0.45    mysno-
sno.demo.lab     <none>         <none>
```

- **ptp-operator-***: The PTP Operator pod (one instance in the cluster).
- **linuxptp-daemon-***: The linuxptp daemon pod. On single-node OpenShift, there is only one daemon pod running on the master node. The daemon pod should show **2/2** in the READY column, indicating both containers (**linuxptp-daemon-container** and **kube-rbac-proxy**) are running.

2. Check which network interfaces support hardware timestamping by running the following command:

```
$ oc get NodePtpDevice -n openshift-ptp -o yaml
```

The output is similar to the following one, showing the NodePtpDevice resource for the single-node OpenShift node with PTP-capable network interfaces:

```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    name: mysno-sno.demo.lab
    namespace: openshift-ptp
  spec: {}
  status:
    devices:
    - name: ens7f0
      hwConfig:
        phcIndex: 0
    - name: ens7f1
      hwConfig:
        phcIndex: 1
kind: List
metadata:
  resourceVersion: ""
```

In this example output:

- **ens7f0** and **ens7f1** are PTP-capable interfaces (Intel E810 NIC ports).
- **phcIndex** indicates the PTP Hardware Clock number (maps to **/dev/ptp0**, **/dev/ptp1**, etc.)

NOTE

On single-node OpenShift clusters, you will see only one NodePtpDevice resource for the single master node.

3. The PTP profile we applied labels the node with `ptp/slave`, so the MCP (machine config pool) that manages it will show `UPDATED=True` and `MACHINECOUNT=1`. Verify this by running the following command:

3. The PTP profile uses node labels for matching. Check your machine config pool (MCP) to verify the master MCP by running the following command:

```
$ oc get mcp
```

The output is similar to the following:

```
NAME     CONFIG              UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT DEGRADEDMACHINECOUNT
AGE
master   rendered-master-a1b1*  True     False     False     1          1              1
0                  45d
worker   rendered-worker-f6e5*  True     False     False     0          0              0
0                  45d
```

NOTE

The CONFIG column shows a truncated hash of the rendered MachineConfig. In actual output, this will be a full 64-character hash like **rendered-master-a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6**.

On single-node OpenShift clusters, the master MCP shows **MACHINECOUNT** of 1 (the single node), and the worker MCP shows **MACHINECOUNT** of 0. The PTP profile must target the **master** node label.

4. Create a **PtpConfig** custom resource (CR) that configures the T-GM clock with GNSS to NTP failover. Save the following YAML configuration to a file named **ptp-config-gnss-ntp-failover-sno.yaml**.

```
# The grandmaster profile is provided for testing only
# It is not installed on production clusters
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "10"
spec:
  profile:
  - name: "grandmaster"
    ptp4lOpts: "-2 --summary_interval -4"
    phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s ens7f0 -n 24
    ptpSchedulingPolicy: SCHED_FIFO
    ptpSchedulingPriority: 10
    ptpSettings:
      logReduce: "true"

    # --- FAILOVER CONFIGURATION ---
    # Holdover time: 14400 seconds (4 hours) before switching to NTP
    ts2phcOpts: "--ts2phc.holdover 14400"

    # Configure Chronyd (Secondary Time Source)
    chronydOpts: "-d"
```

```
chronydConf: |
  server time.nist.gov iburst
  makestep 1.0 -1
  pidfile /var/run/chronyd.pid

plugins:
  # E810 Hardware-Specific Configuration
  e810:
    enableDefaultConfig: false
    settings:
      LocalHoldoverTimeout: 14400
      LocalMaxHoldoverOffSet: 1500
      MaxInSpecOffset: 1500
    pins:
      # Syntax guide:
      # - The 1st number in each pair must be one of:
      #    0 - Disabled
      #    1 - RX
      #    2 - TX
      # - The 2nd number in each pair must match the channel number
      ens7f0:
        SMA1: 0 1
        SMA2: 0 2
        U.FL1: 0 1
        U.FL2: 0 2
    ublxCmds:
      - args: #ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1
          - "-P"
          - "29.20"
          - "-z"
          - "CFG-HW-ANT_CFG_VOLTCTRL,1"
        reportOutput: false
      - args: #ubxtool -P 29.20 -e GPS
          - "-P"
          - "29.20"
          - "-e"
          - "GPS"
        reportOutput: false
      - args: #ubxtool -P 29.20 -d Galileo
          - "-P"
          - "29.20"
          - "-d"
          - "Galileo"
        reportOutput: false
      - args: #ubxtool -P 29.20 -d GLONASS
          - "-P"
          - "29.20"
          - "-d"
          - "GLONASS"
        reportOutput: false
      - args: #ubxtool -P 29.20 -d BeiDou
          - "-P"
          - "29.20"
          - "-d"
          - "BeiDou"
        reportOutput: false
```

```
      - args: #ubxtool -P 29.20 -d SBAS
          - "-P"
          - "29.20"
          - "-d"
          - "SBAS"
        reportOutput: false
      - args: #ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000
          - "-P"
          - "29.20"
          - "-t"
          - "-w"
          - "5"
          - "-v"
          - "1"
          - "-e"
          - "SURVEYIN,600,50000"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p MON-HW
          - "-P"
          - "29.20"
          - "-p"
          - "MON-HW"
        reportOutput: true
      - args: #ubxtool -P 29.20 -p CFG-MSG,1,38,248
          - "-P"
          - "29.20"
          - "-p"
          - "CFG-MSG,1,38,248"
        reportOutput: true

  # NTP Failover Plugin
  ntpfailover:
    gnssFailover: true

# --- GNSS (ts2phc) CONFIGURATION (Primary Source) ---
ts2phcConf: |
  [nmea]
  ts2phc.master 1
  [global]
  use_syslog  0
  verbose 1
  logging_level 7
  ts2phc.pulsewidth 100000000
  ts2phc.nmea_serialport /dev/ttyGNSS_1700_0
  leapfile  /usr/share/zoneinfo/leap-seconds.list
  [ens7f0]
  ts2phc.extts_polarity rising
  ts2phc.extts_correction 0

# --- PTP4L CONFIGURATION (Grandmaster Role) ---
ptp4lConf: |
  [ens7f0]
  masterOnly 1
  [ens7f1]
  masterOnly 1
  [global]
```

```
#
# Default Data Set
#
twoStepFlag 1
priority1 128
priority2 128
domainNumber 24
#utc_offset 37
clockClass 6
clockAccuracy 0x27
offsetScaledLogVariance 0xFFFF
free_running 0
freq_est_interval 1
dscp_event 0
dscp_general 0
dataset_comparison G.8275.x
G.8275.defaultDS.localPriority 128
#
# Port Data Set
#
logAnnounceInterval -3
logSyncInterval -4
logMinDelayReqInterval -4
logMinPdelayReqInterval 0
announceReceiptTimeout 3
syncReceiptTimeout 0
delayAsymmetry 0
fault_reset_interval -4
neighborPropDelayThresh 20000000
masterOnly 0
G.8275.portDS.localPriority 128
#
# Run time options
#
assume_two_step 0
logging_level 6
path_trace_enabled 0
follow_up_info 0
hybrid_e2e 0
inhibit_multicast_service 0
net_sync_monitor 0
tc_spanning_tree 0
tx_timestamp_timeout 50
unicast_listen 0
unicast_master_table 0
unicast_req_duration 3600
use_syslog 1
verbose 0
summary_interval -4
kernel_leap 1
check_fup_sync 0
clock_class_threshold 7
#
# Servo Options
#
pi_proportional_const 0.0
```

```
      pi_integral_const 0.0
      pi_proportional_scale 0.0
      pi_proportional_exponent -0.3
      pi_proportional_norm_max 0.7
      pi_integral_scale 0.0
      pi_integral_exponent 0.4
      pi_integral_norm_max 0.3
      step_threshold 2.0
      first_step_threshold 0.00002
      clock_servo pi
      sanity_freq_limit  200000000
      ntpshm_segment 0
      #
      # Transport options
      #
      transportSpecific 0x0
      ptp_dst_mac 01:1B:19:00:00:00
      p2p_dst_mac 01:80:C2:00:00:0E
      udp_ttl 1
      udp6_scope 0x0E
      uds_address /var/run/ptp4l
      #
      # Default interface options
      #
      clock_type BC
      network_transport L2
      delay_mechanism E2E
      time_stamping hardware
      tsproc_mode filter
      delay_filter moving_median
      delay_filter_length 10
      egressLatency 0
      ingressLatency 0
      boundary_clock_jbod 0
      #
      # Clock description
      #
      productDescription ;;
      revisionData ;;
      manufacturerIdentity 00:00:00
      userDescription ;
      timeSource 0x20
    ptpClockThreshold:
      holdOverTimeout: 5
      maxOffsetThreshold: 100
      minOffsetThreshold: -100
  recommend:
  - profile: "grandmaster"
    priority: 4
    match:
    - nodeLabel: node-role.kubernetes.io/master
```

IMPORTANT

Replace the example interface names (**ens7f0**, **ens7f1**) with your actual E810 NIC interface names found in step 2. Common E810 interface naming patterns include **ens7f0**, **ens8f0**, **eth0**, **enp2s0f0**, and so on. The exact name depends on your system BIOS settings and Linux network device naming conventions. Also, replace **/dev/ttyGNSS_1700_0** with your actual GNSS serial port device path. The **nodeLabel** is set to **node-role.kubernetes.io/master** to target the single-node OpenShift master node which serves all roles.

The configuration includes the following components:

- **PTP4L options**:
  - **-2**: Use PTP version 2
  - **--summary_interval -4**: Log summary every 2^(-4) = 0.0625 seconds
- **PHC2SYS options:**
  - **-r**: Synchronize system clock from PTP hardware clock
  - **-u 0**: Update rate multiplier
  - **-m**: Print messages to stdout
  - **-N 8**: Domain number for ptp4l
  - **-R 16**: Update rate
  - **-s ens7f0**: Source interface (replace with your E810 interface name)
  - **-n 24**: Domain number
- **Failover configuration:**
  - **ts2phcOpts --ts2phc.holdover 14400**: 4-hour holdover before switching to NTP
  - **chronydConf**: NTP server configuration for failover replace **time.nist.gov** with your preferred NTP server
  - **ntpfailover plugin**: Enables automatic GNSS-to-NTP switching with **gnssFailover: true**.
- **E810 plugin configuration:**
  - **LocalHoldoverTimeout: 14400**: E810 hardware holdover timeout (4 hours)
  - **pins**: Configuration for 1PPS input on E810 physical pins (U.FL2, SMA1, SMA2, U.FL1)
  - **ublxCmds**: Commands to configure u-blox GNSS receiver (enable GPS, disable other constellations, set survey-in mode)
- **GNSS (ts2phc) configuration:**
  - **ts2phc.nmea_serialport /dev/ttyGNSS_1700_0**: GNSS serial port device path (replace with your actual GNSS device)

- **ts2phc.extts_polarity rising**: 1PPS signal on rising edge
- **ts2phc.pulsewidth 100000000**: 1PPS pulse width in nanoseconds

- PTP4L configuration:
  - **masterOnly 1**: Interface acts only as PTP master
  - **clockClass 6**: GPS-synchronized quality level
  - **domainNumber 24**: PTP domain
  - **clock_type BC**: Boundary Clock mode
  - **time_stamping hardware**: Use hardware timestamps from E810 NIC

5. Apply the **PtpConfig** CR by running the following command:

```
$ oc apply -f ptp-config-gnss-ntp-failover-sno.yaml
```

The output is similar to the following:

```
ptpconfig.ptp.openshift.io/grandmaster created
```

Verification

1. The PTP daemon checks for profile updates every 30 seconds. Wait approximately 30 seconds, then verify by running the following command:

```
$ oc get ptpconfig -n openshift-ptp
```

The output is similar to the following:

```
NAME          AGE
grandmaster   2m
```

2. Check the NodePtpDevice to see if the profile is applied. First, get your single-node OpenShift node name:

```
$ oc get nodes
```

The output is similar to the following:

```
NAME                 STATUS   ROLES                         AGE     VERSION
mysno-sno.demo.lab   Ready    control-plane,master,worker   4h19m   v1.34.1
```

Then describe the NodePtpDevice using your node name:

```
$ oc describe nodeptpdevice mysno-sno.demo.lab -n openshift-ptp
```

3. Check if the profile is being loaded by monitoring the daemon logs. First, get the daemon pod name:

```
$ oc get pods -n openshift-ptp | grep linuxptp-daemon
```

The output shows the single linuxptp-daemon pod:

```
linuxptp-daemon-xz8km          2/2     Running   0          15m
```

Then check the logs using the pod name:

```
$ oc logs -n openshift-ptp linuxptp-daemon-xz8km -c linuxptp-daemon-container --tail=100
```

Success indicators in the logs are:

- **load profiles** - Profile is being loaded
- **in applyNodePTPProfiles** - Profile is being applied
- No **ptp profile doesn't exist for node** errors

4. Check **chronyd** status to verify NTP is running as the secondary time source by running the following command:

```
$ oc logs -n openshift-ptp linuxptp-daemon-xz8km -c linuxptp-daemon-container | grep chronyd
```

The output is similar to the following:

```
chronyd version 4.5 starting
Added source ID#0000000001 (time.nist.gov)
```

5. Check GNSS/gpsd by running the following command:

```
$ oc logs -n openshift-ptp linuxptp-daemon-xz8km -c linuxptp-daemon-container | grep gpsd
```

The output shows the following when GNSS is functioning correctly:

- **gpsd** starting successfully
- No **No such file or directory** errors exist

6. Check **ts2phc** (GNSS synchronization) status by running the following command:

```
$ oc logs -n openshift-ptp linuxptp-daemon-xz8km -c linuxptp-daemon-container | grep ts2phc
```

7. Check **phc2sys** (system clock sync) status by running the following command:

```
$ oc logs -n openshift-ptp linuxptp-daemon-xz8km -c linuxptp-daemon-container | grep phc2sys
```

The output shows synchronization status messages for **phc2sys**.

```
phc2sys[xxx]: CLOCK_REALTIME phc offset -17 s2 freq -13865 delay 2305
```

6.2.13. Troubleshooting common PTP Operator issues

Troubleshoot common problems with the PTP Operator by performing the following steps.

Prerequisites

- Install the OpenShift Container Platform CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Install the PTP Operator on a bare-metal cluster with hosts that support PTP.

Procedure

1. Check the Operator and operands are successfully deployed in the cluster for the configured nodes.

```
$ oc get pods -n openshift-ptp -o wide
```

Example output

```
NAME                           READY   STATUS    RESTARTS   AGE     IP           NODE
linuxptp-daemon-lmvgn          3/3     Running   0          4d17h   10.1.196.24  compute-
0.example.com
linuxptp-daemon-qhfg7          3/3     Running   0          4d17h   10.1.196.25  compute-
1.example.com
ptp-operator-6b8dcbf7f4-zndk7  1/1     Running   0          5d7h    10.129.0.61  control-plane-
1.example.com
```

NOTE

When the PTP fast event bus is enabled, the number of ready **linuxptp-daemon** pods is **3/3**. If the PTP fast event bus is not enabled, **2/2** is displayed.

2. Check that supported hardware is found in the cluster.

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io
```

Example output

```
NAME                           AGE
control-plane-0.example.com    10d
control-plane-1.example.com    10d
compute-0.example.com          10d
compute-1.example.com          10d
compute-2.example.com          10d
```

3. Check the available PTP network interfaces for a node:

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io <node_name> -o yaml
```

where:

<node_name>

Specifies the node you want to query, for example, **compute-0.example.com**.

Example output

```
apiVersion: ptp.openshift.io/v1
kind: NodePtpDevice
metadata:
  creationTimestamp: "2021-09-14T16:52:33Z"
  generation: 1
  name: compute-0.example.com
  namespace: openshift-ptp
  resourceVersion: "177400"
  uid: 30413db0-4d8d-46da-9bef-737bacd548fd
spec: {}
status:
  devices:
  - name: eno1
  - name: eno2
  - name: eno3
  - name: eno4
  - name: enp5s0f0
  - name: enp5s0f1
```

4. Check that the PTP interface is successfully synchronized to the primary clock by accessing the **linuxptp-daemon** pod for the corresponding node.

   a. Get the name of the **linuxptp-daemon** pod and corresponding node you want to troubleshoot by running the following command:

   ```
   $ oc get pods -n openshift-ptp -o wide
   ```

   Example output

   ```
   NAME                          READY   STATUS    RESTARTS   AGE     IP           NODE
   linuxptp-daemon-lmvgn         3/3     Running   0          4d17h   10.1.196.24  compute-
   0.example.com
   linuxptp-daemon-qhfg7         3/3     Running   0          4d17h   10.1.196.25  compute-
   1.example.com
   ptp-operator-6b8dcbf7f4-zndk7 1/1     Running   0          5d7h    10.129.0.61  control-
   plane-1.example.com
   ```

   b. Remote shell into the required **linuxptp-daemon** container:

   ```
   $ oc rsh -n openshift-ptp -c linuxptp-daemon-container <linux_daemon_container>
   ```

   where:

   <linux_daemon_container>

   is the container you want to diagnose, for example **linuxptp-daemon-lmvgn**.

c. In the remote shell connection to the **linuxptp-daemon** container, use the PTP Management Client (**pmc**) tool to diagnose the network interface. Run the following **pmc** command to check the sync status of the PTP device, for example **ptp4l**.

```
# pmc -u -f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
```

Example output when the node is successfully synced to the primary clock

```
sending: GET PORT_DATA_SET
   40a6b7.fffe.166ef0-1 seq 0 RESPONSE MANAGEMENT PORT_DATA_SET
      portIdentity            40a6b7.fffe.166ef0-1
      portState               SLAVE
      logMinDelayReqInterval  -4
      peerMeanPathDelay       0
      logAnnounceInterval     -3
      announceReceiptTimeout  3
      logSyncInterval         -4
      delayMechanism          1
      logMinPdelayReqInterval -4
      versionNumber           2
```

5. For GNSS-sourced grandmaster clocks, verify that the in-tree NIC ice driver is correct by running the following command, for example:

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-74m2g ethtool -i ens7f0
```

Example output

```
driver: ice
version: 5.14.0-356.bz2232515.el9.x86_64
firmware-version: 4.20 0x8001778b 1.3346.0
```

6. For GNSS-sourced grandmaster clocks, verify that the **linuxptp-daemon** container is receiving signal from the GNSS antenna. If the container is not receiving the GNSS signal, the **/dev/gnss0** file is not populated. To verify, run the following command:

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-jnz6r cat /dev/gnss0
```

Example output

```
$GNRMC,125223.00,A,4233.24463,N,07126.64561,W,0.000,,300823,,,A,V*0A
$GNVTG,,T,,M,0.000,N,0.000,K,A*3D
$GNGGA,125223.00,4233.24463,N,07126.64561,W,1,12,99.99,98.6,M,-33.1,M,,*7E
$GNGSA,A,3,25,17,19,11,12,06,05,04,09,20,,,99.99,99.99,99.99,1*37
$GPGSV,3,1,10,04,12,039,41,05,31,222,46,06,50,064,48,09,28,064,42,1*62
```

6.2.14. Getting the DPLL firmware version for the CGU in an Intel 800 series NIC

You can get the digital phase-locked loop (DPLL) firmware version for the Clock Generation Unit (CGU) in an Intel 800 series NIC by opening a debug shell to the cluster node and querying the NIC hardware.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed an Intel 800 series NIC in the cluster host.
- You have installed the PTP Operator on a bare-metal cluster with hosts that support PTP.

Procedure

1. Start a debug pod by running the following command:

```
$ oc debug node/<node_name>
```

   where:

   <node_name>
   Is the node where you have installed the Intel 800 series NIC.

2. Check the CGU firmware version in the NIC by using the **devlink** tool and the bus and device name where the NIC is installed. For example, run the following command:

```
sh-4.4# devlink dev info <bus_name>/<device_name> | grep cgu
```

   where:

   <bus_name>
   Is the bus where the NIC is installed. For example, **pci**.

   <device_name>
   Is the NIC device name. For example, **0000:51:00.0**.

   Example output

```
cgu.id 36
fw.cgu 8032.16973825.6021
```

   where:

   cgu.id 36
   CGU hardware revision number.

   fw.cgu 8032.16973825.6021
   DPLL firmware version running in the CGU, where the DPLL firmware version is **6201**, and the DPLL model is **8032**. The string **16973825** is a shorthand representation of the binary version of the DPLL firmware version (**1.3.0.1**).

NOTE

The firmware version has a leading nibble and 3 octets for each part of the version number. The number **16973825** in binary is **0001 0000 0011 0000 0000 0000 0001**. Use the binary value to decode the firmware version. For example:

Table 6.10. DPLL firmware version

Binary part	Decimal value
0001	1
0000 0011	3
0000 0000	0
0000 0001	1

6.2.15. Collecting PTP Operator data

You can use the **oc adm must-gather** command to collect information about your cluster, including features and objects associated with PTP Operator.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).
- You have installed the PTP Operator.

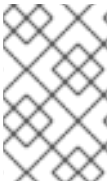
Procedure

- To collect PTP Operator data with **must-gather**, you must specify the PTP Operator **must-gather** image.

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/ptp-must-gather-rhel9:v4.18
```

6.3. DEVELOPING PTP EVENTS CONSUMER APPLICATIONS WITH THE REST API V2

When developing consumer applications that make use of Precision Time Protocol (PTP) events on a bare-metal cluster node, you deploy your consumer application in a separate application pod. The consumer application subscribes to PTP events by using the PTP events REST API v2.



NOTE

The following information provides general guidance for developing consumer applications that use PTP events. A complete events consumer application example is outside the scope of this information.

Additional resources

- PTP events REST API v2 reference

6.3.1. About the PTP fast event notifications framework

Use the Precision Time Protocol (PTP) fast event REST API v2 to subscribe cluster applications to PTP events that the bare-metal cluster node generates.

NOTE

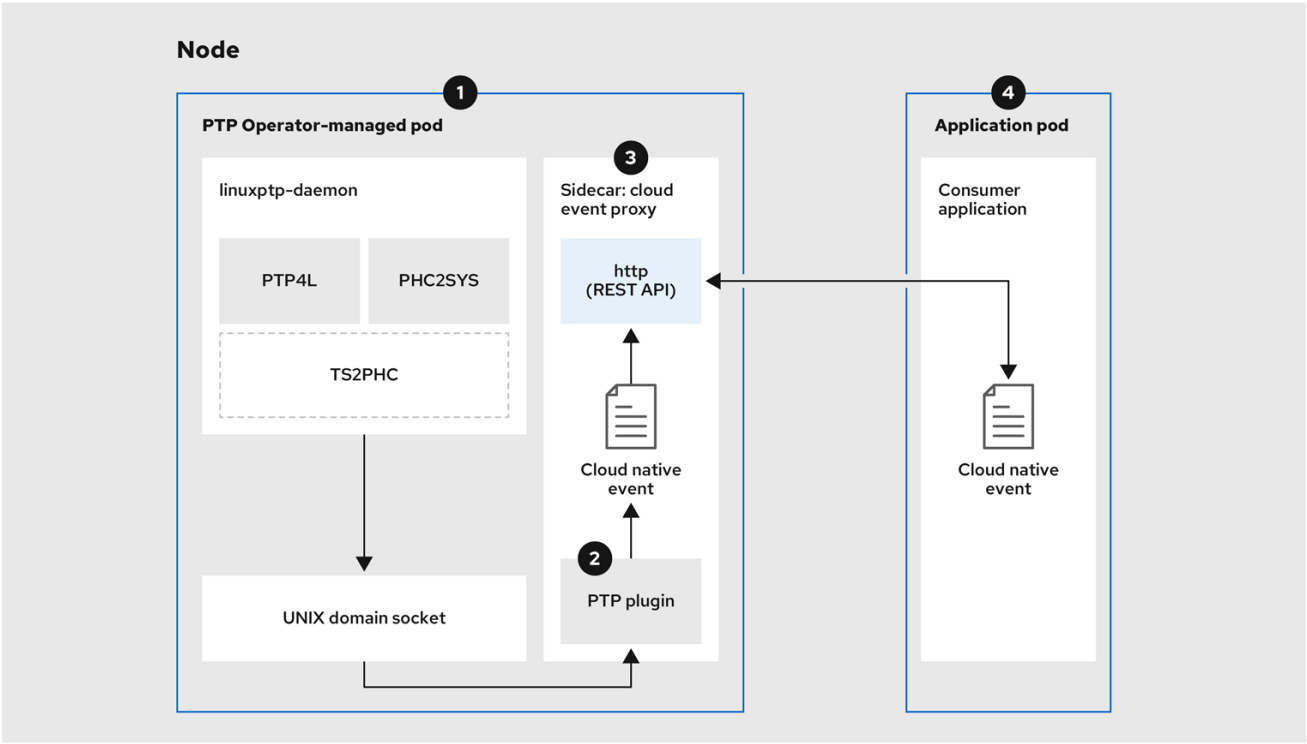
The fast events notifications framework uses a REST API for communication. The PTP events REST API v1 and v2 are based on the *O-RAN O-Cloud Notification API Specification for Event Consumers 4.0* that is available from O-RAN ALLIANCE Specifications.

Only the PTP events REST API v2 is O-RAN v4 compliant.

6.3.2. Retrieving PTP events with the PTP events REST API v2

Applications subscribe to PTP events by using an O-RAN v4 compatible REST API in the producer-side cloud event proxy sidecar. The **cloud-event-proxy** sidecar container can access the same resources as the primary application container without using any of the resources of the primary application and with no significant latency.

Figure 6.6. Overview of consuming PTP fast events from the PTP event producer REST API v2



1 Event is generated on the cluster host

The **linuxptp-daemon** process in the PTP Operator-managed pod runs as a Kubernetes **DaemonSet** and manages the various **linuxptp** processes (**ptp4l**, **phc2sys**, and optionally for grandmaster clocks, **ts2phc**). The **linuxptp-daemon** passes the event to the UNIX domain socket.

2 **Event is passed to the cloud-event-proxy sidecar**

The PTP plugin reads the event from the UNIX domain socket and passes it to the **cloud-event-proxy** sidecar in the PTP Operator-managed pod. **cloud-event-proxy** delivers the event from the Kubernetes infrastructure to Cloud-Native Network Functions (CNFs) with low latency.

3 **Event is published**

The **cloud-event-proxy** sidecar in the PTP Operator-managed pod processes the event and publishes the event by using the PTP events REST API v2.

4 **Consumer application requests a subscription and receives the subscribed event**

The consumer application sends an API request to the producer **cloud-event-proxy** sidecar to create a PTP events subscription. Once subscribed, the consumer application listens to the address specified in the resource qualifier and receives and processes the PTP events.

6.3.3. Configuring the PTP fast event notifications publisher

To start using PTP fast event notifications for a network interface in your cluster, you must enable the fast event publisher in the PTP Operator **PtpOperatorConfig** custom resource (CR) and configure **ptpClockThreshold** values in a **PtpConfig** CR that you create.

Prerequisites

- You have installed the OpenShift Container Platform CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the PTP Operator.

Procedure

1. Modify the default PTP Operator config to enable PTP fast events.
   a. Save the following YAML in the **ptp-operatorconfig.yaml** file:

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    apiVersion: "2.0" 1
    enableEventPublisher: true 2
```

1 Enable the PTP events REST API v2 for the PTP event producer by setting the **ptpEventConfig.apiVersion** to "2.0". The default value is "1.0".

2 Enable PTP fast event notifications by setting **enableEventPublisher** to **true**.

NOTE

In OpenShift Container Platform 4.13 or later, you do not need to set the **spec.ptpEventConfig.transportHost** field in the **PtpOperatorConfig** resource when you use HTTP transport for PTP events.

b. Update the **PtpOperatorConfig** CR:

```
$ oc apply -f ptp-operatorconfig.yaml
```

2. Create a **PtpConfig** custom resource (CR) for the PTP enabled interface, and set the required values for **ptpClockThreshold** and **ptp4lOpts**. The following YAML illustrates the required values that you must set in the **PtpConfig** CR:

```
spec:
  profile:
  - name: "profile1"
    interface: "enp5s0f0"
    ptp4lOpts: "-2 -s --summary_interval -4" 1
    phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
    ptp4lConf: "" 3
    ptpClockThreshold: 4
      holdOverTimeout: 5
      maxOffsetThreshold: 100
      minOffsetThreshold: -100
```

1 Append **--summary_interval -4** to use PTP fast events.

2 Required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

3 Specify a string that contains the configuration to replace the default **/etc/ptp4l.conf** file. To use the default configuration, leave the field empty.

4 Optional. If the **ptpClockThreshold** stanza is not present, default values are used for the **ptpClockThreshold** fields. The stanza shows default **ptpClockThreshold** values. The **ptpClockThreshold** values configure how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.

Additional resources

- Configuring linuxptp services as ordinary clock

6.3.4. PTP events REST API v2 consumer application reference

PTP event consumer applications require the following features:

1. A web service running with a **POST** handler to receive the cloud native PTP events JSON payload
2. A **createSubscription** function to subscribe to the PTP events producer
3. A **getCurrentState** function to poll the current state of the PTP events producer

The following example Go snippets illustrate these requirements:

Example PTP events consumer server function in Go

```
func server() {
  http.HandleFunc("/event", getEvent)
  http.ListenAndServe(":9043", nil)
}

func getEvent(w http.ResponseWriter, req *http.Request) {
  defer req.Body.Close()
  bodyBytes, err := io.ReadAll(req.Body)
  if err != nil {
    log.Errorf("error reading event %v", err)
  }
  e := string(bodyBytes)
  if e != "" {
    processEvent(bodyBytes)
    log.Infof("received event %s", string(bodyBytes))
  }
  w.WriteHeader(http.StatusNoContent)
}
```

Example PTP events createSubscription function in Go

```
import (
"github.com/redhat-cne/sdk-go/pkg/pubsub"
"github.com/redhat-cne/sdk-go/pkg/types"
v1pubsub "github.com/redhat-cne/sdk-go/v1/pubsub"
)

// Subscribe to PTP events using v2 REST API
s1,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/sync-state")
s2,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/lock-state")
s3,_:=createsubscription("/cluster/node/<node_name>/sync/gnss-status/gnss-sync-status")
s4,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state")
s5,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/clock-class")

// Create PTP event subscriptions POST
func createSubscription(resourceAddress string) (sub pubsub.PubSub, err error) {
  var status int
  apiPath := "/api/ocloudNotifications/v2/"
  localAPIAddr := "consumer-events-subscription-service.cloud-events.svc.cluster.local:9043" // vDU
service API address
```

```
  apiAddr := "ptp-event-publisher-service-<node_name>.openshift-ptp.svc.cluster.local:9043" 1
  apiVersion := "2.0"

  subURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: apiAddr
    Path: fmt.Sprintf("%s%s", apiPath, "subscriptions")}}
  endpointURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: localAPIAddr,
    Path: "event"}}

  sub = v1pubsub.NewPubSub(endpointURL, resourceAddress, apiVersion)
  var subB []byte

  if subB, err = json.Marshal(&sub); err == nil {
    rc := restclient.New()
    if status, subB = rc.PostWithReturn(subURL, subB); status != http.StatusCreated {
      err = fmt.Errorf("error in subscription creation api at %s, returned status %d", subURL, status)
    } else {
      err = json.Unmarshal(subB, &sub)
    }
  } else {
    err = fmt.Errorf("failed to marshal subscription for %s", resourceAddress)
  }
  return
}
```

1 Replace **<node_name>** with the FQDN of the node that is generating the PTP events. For example, **compute-1.example.com**.

Example PTP events consumer getCurrentState function in Go

```
//Get PTP event state for the resource
func getCurrentState(resource string) {
  //Create publisher
  url := &types.URI{URL: url.URL{Scheme: "http",
    Host: "ptp-event-publisher-service-<node_name>.openshift-ptp.svc.cluster.local:9043", 1
    Path: fmt.SPrintf("/api/ocloudNotifications/v2/%s/CurrentState",resource}}
  rc := restclient.New()
  status, event := rc.Get(url)
  if status != http.StatusOK {
    log.Errorf("CurrentState:error %d from url %s, %s", status, url.String(), event)
  } else {
    log.Debugf("Got CurrentState: %s ", event)
  }
}
```

1 Replace **<node_name>** with the FQDN of the node that is generating the PTP events. For example, **compute-1.example.com**.

6.3.5. Reference event consumer deployment and service CRs using PTP events REST API v2

Use the following example PTP event consumer custom resources (CRs) as a reference when deploying your PTP events consumer application for use with the PTP events REST API v2.

Reference cloud event consumer namespace

```
apiVersion: v1
kind: Namespace
metadata:
  name: cloud-events
  labels:
    security.openshift.io/scc.podSecurityLabelSync: "false"
    pod-security.kubernetes.io/audit: "privileged"
    pod-security.kubernetes.io/enforce: "privileged"
    pod-security.kubernetes.io/warn: "privileged"
    name: cloud-events
    openshift.io/cluster-monitoring: "true"
  annotations:
    workload.openshift.io/allowed: management
```

Reference cloud event consumer deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cloud-consumer-deployment
  namespace: cloud-events
  labels:
    app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      annotations:
        target.workload.openshift.io/management: '{"effect": "PreferredDuringScheduling"}'
      labels:
        app: consumer
    spec:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      serviceAccountName: consumer-sa
      containers:
        - name: cloud-event-consumer
          image: cloud-event-consumer
          imagePullPolicy: Always
          args:
            - "--local-api-addr=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
            - "--api-path=/api/ocloudNotifications/v2/"
            - "--api-addr=127.0.0.1:8089"
            - "--api-version=2.0"
            - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043"
          env:
```

```
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
        - name: CONSUMER_TYPE
          value: "PTP"
        - name: ENABLE_STATUS_CHECK
          value: "true"
    volumes:
      - name: pubsubstore
        emptyDir: {}
```

Reference cloud event consumer service account

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: consumer-sa
  namespace: cloud-events
```

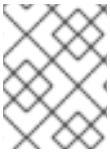
Reference cloud event consumer service

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
    app: consumer
  sessionAffinity: None
  type: ClusterIP
```

6.3.6. Subscribing to PTP events with the REST API v2

Deploy your **cloud-event-consumer** application container and subscribe the **cloud-event-consumer** application to PTP events posted by the **cloud-event-proxy** container in the pod managed by the PTP Operator.

Subscribe consumer applications to PTP events by sending a **POST** request to **http://ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions** passing the appropriate subscription request payload.



NOTE

9043 is the default port for the **cloud-event-proxy** container deployed in the PTP event producer pod. You can configure a different port for your application as required.

Additional resources

- api/ocloudNotifications/v2/subscriptions

6.3.7. Verifying that the PTP events REST API v2 consumer application is receiving events

Verify that the **cloud-event-consumer** container in the application pod is receiving Precision Time Protocol (PTP) events.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed and configured the PTP Operator.
- You have deployed a cloud events application pod and PTP events consumer application.

Procedure

1. Check the logs for the deployed events consumer application. For example, run the following command:

```
$ oc -n cloud-events logs -f deployment/cloud-consumer-deployment
```

Example output

```
time = "2024-09-02T13:49:01Z"
level = info msg = "transport host path is set to  ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043"
time = "2024-09-02T13:49:01Z"
level = info msg = "apiVersion=2.0, updated apiAddr=ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043, apiPath=/api/ocloudNotifications/v2/"
time = "2024-09-02T13:49:01Z"
level = info msg = "Starting local API listening to :9043"
time = "2024-09-02T13:49:06Z"
level = info msg = "transport host path is set to  ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043"
time = "2024-09-02T13:49:06Z"
level = info msg = "checking for rest service health"
time = "2024-09-02T13:49:06Z"
level = info msg = "health check http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/health"
time = "2024-09-02T13:49:07Z"
level = info msg = "rest service returned healthy status"
time = "2024-09-02T13:49:07Z"
level = info msg = "healthy publisher; subscribing to events"
```

```
time = "2024-09-02T13:49:07Z"
level = info msg = "received event {\"specversion\":\"1.0\",\"id\":\"ab423275-f65d-4760-97af-
5b0b846605e4\",\"source\":\"/sync/ptp-status/clock-class\",\"type\":\"event.sync.ptp-
status.ptp-clock-class-change\",\"time\":\"2024-09-02T13:49:07.226494483Z\",\"data\":
{\"version\":\"1.0\",\"values\":[{\"ResourceAddress\":\"/cluster/node/compute-
1.example.com/ptp-not-
set\",\"data_type\":\"metric\",\"value_type\":\"decimal64.3\",\"value\":\"0\"}]}}"
```

2. Optional. Test the REST API by using **oc** and port-forwarding port **9043** from the **linuxptp-daemon** deployment. For example, run the following command:

```
$ oc port-forward -n openshift-ptp ds/linuxptp-daemon 9043:9043
```

Example output

```
Forwarding from 127.0.0.1:9043 -> 9043
Forwarding from [::1]:9043 -> 9043
Handling connection for 9043
```

Open a new shell prompt and test the REST API v2 endpoints:

```
$ curl -X GET http://localhost:9043/api/ocloudNotifications/v2/health
```

Example output

```
OK
```

6.3.8. Monitoring PTP fast event metrics

You can monitor PTP fast events metrics from cluster nodes where the **linuxptp-daemon** is running. You can also monitor PTP fast event metrics in the OpenShift Container Platform web console by using the preconfigured and self-updating Prometheus monitoring stack.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in as a user with **cluster-admin** privileges.
- Install and configure the PTP Operator on a node with PTP-capable hardware.

Procedure

1. Start a debug pod for the node by running the following command:

```
$ oc debug node/<node_name>
```

2. Check for PTP metrics exposed by the **linuxptp-daemon** container. For example, run the following command:

```
sh-4.4# curl http://localhost:9091/metrics
```

Example output

```
# HELP cne_api_events_published Metric to get number of events published by the rest api
# TYPE cne_api_events_published gauge
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",status="success"} 1
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",status="success"} 94
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-status/class-change",status="success"} 18
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",status="success"} 27
```

3. Optional. You can also find PTP events in the logs for the **cloud-event-proxy** container. For example, run the following command:

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

4. To view the PTP event in the OpenShift Container Platform web console, copy the name of the PTP metric you want to query, for example, **openshift_ptp_offset_ns**.

5. In the OpenShift Container Platform web console, click **Observe → Metrics**.

6. Paste the PTP metric name into the **Expression** field, and click **Run queries**.

Additional resources

- Accessing metrics as a developer

6.3.9. PTP fast event metrics reference

The following table describes the PTP fast events metrics that are available from cluster nodes where the **linuxptp-daemon** service is running.

Table 6.11. PTP fast event metrics

Metric	Description	Example
openshift_ptp_clock_class	Returns the PTP clock class for the interface. Possible values for PTP clock class are 6 (**LOCKED**), 7 (**PRC UNLOCKED IN-SPEC**), 52 (**PRC UNLOCKED OUT-OF-SPEC**), 187 (**PRC UNLOCKED OUT-OF-SPEC**), 135 (**T-BC HOLDOVER IN-SPEC**), 165 (**T-BC HOLDOVER OUT-OF-SPEC**), 248 (**DEFAULT**), or 255 (**SLAVE ONLY CLOCK**).	**{node="compute-1.example.com",process="ptp4l"} 6**
openshift_ptp_clock_state	Returns the current PTP clock state for the interface. Possible values for PTP clock state are **FREERUN**, **LOCKED**, or **HOLDOVER**.	**{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1**

Metric	Description	Example
openshift_ptp_delay_ns	Returns the delay in nanoseconds between the primary clock sending the timing packet and the secondary clock receiving the timing packet.	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0**
openshift_ptp_ha_profile_status	Returns the current status of the highly available system clock when there are multiple time sources on different NICs. Possible values are 0 (**INACTIVE**) and 1 (**ACTIVE**).	**{node="node1",process="phc2sys",profile="profile1"} 1{node="node1",process="phc2sys",profile="profile2"} 0**
openshift_ptp_frequency_adjustment_ns	Returns the frequency adjustment in nanoseconds between 2 PTP clocks. For example, between the upstream clock and the NIC, between the system clock and the NIC, or between the PTP hardware clock (**phc**) and the NIC.	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -6768**
openshift_ptp_interface_role	Returns the configured PTP clock role for the interface. Possible values are 0 (**PASSIVE**), 1 (**SLAVE**), 2 (**MASTER**), 3 (**FAULTY**), 4 (**UNKNOWN**), or 5 (**LISTENING**).	**{iface="ens2f0", node="compute-1.example.com", process="ptp4l"} 2**
openshift_ptp_max_offset_ns	Returns the maximum offset in nanoseconds between 2 clocks or interfaces. For example, between the upstream GNSS clock and the NIC (**ts2phc**), or between the PTP hardware clock (**phc**) and the system clock (**phc2sys**).	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 1.038099569e+09**
openshift_ptp_offset_ns	Returns the offset in nanoseconds between the DPLL clock or the GNSS clock source and the NIC hardware clock.	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -9**
openshift_ptp_process_restart_count	Returns a count of the number of times the **ptp4l** and **ts2phc** processes were restarted.	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_process_status	Returns a status code that shows whether the PTP processes are running or not.	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**

Metric	Description	Example
openshift_ptp_threshold	Returns values for **HoldOverTimeout**, **MaxOffsetThreshold**, and **MinOffsetThreshold**. • **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. • **maxOffsetThreshold** and **minOffsetThreshold** are offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**) values that you configure in the **PtpConfig** CR for the NIC.	**{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5**

6.3.9.1. PTP fast event metrics only when T-GM is enabled

The following table describes the PTP fast event metrics that are available only when PTP grandmaster clock (T-GM) is enabled.

Table 6.12. PTP fast event metrics when T-GM is enabled

Metric	Description	Example
openshift_ptp_frequency_status	Returns the current status of the digital phase-locked loop (DPLL) frequency for the NIC. Possible values are -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), or 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_nmea_status	Returns the current status of the NMEA connection. NMEA is the protocol that is used for 1PPS NIC connections. Possible values are 0 (**UNAVAILABLE**) and 1 (**AVAILABLE**).	**{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1**
openshift_ptp_phase_status	Returns the status of the DPLL phase for the NIC. Possible values are -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), or 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_pps_status	Returns the current status of the NIC 1PPS connection. You use the 1PPS connection to synchronize timing between connected NICs. Possible values are 0 (**UNAVAILABLE**) and 1 (**AVAILABLE**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1**

Metric	Description	Example
openshift_ptp_gnss_status	Returns the current status of the global navigation satellite system (GNSS) connection. GNSS provides satellite-based positioning, navigation, and timing services globally. Possible values are 0 (**NOFIX**), 1 (**DEAD RECKONING ONLY**), 2 (**2D-FIX**), 3 (**3D-FIX**), 4 (**GPS+DEAD RECKONING FIX**), 5, (**TIME ONLY FIX**).	**{from="gnss",iface="ens2fx",node="compute-1.example.com",process="gnss"} 3**

6.4. PTP EVENTS REST API V2 REFERENCE

Use the following REST API v2 endpoints to subscribe the **cloud-event-consumer** application to Precision Time Protocol (PTP) events posted at **http://localhost:9043/api/ocloudNotifications/v2** in the PTP events producer pod.

- **api/ocloudNotifications/v2/subscriptions**
  - **POST**: Creates a new subscription
  - **GET**: Retrieves a list of subscriptions
  - **DELETE**: Deletes all subscriptions
- **api/ocloudNotifications/v2/subscriptions/{subscription_id}**
  - **GET**: Returns details for the specified subscription ID
  - **DELETE**: Deletes the subscription associated with the specified subscription ID
- **api/ocloudNotifications/v2/health**
  - **GET**: Returns the health status of **ocloudNotifications** API
- **api/ocloudNotifications/v2/publishers**
  - **GET**: Returns a list of PTP event publishers for the cluster node
- **api/ocloudnotifications/v2/{resource_address}/CurrentState**
  - **GET**: Returns the current state of the event type specified by the **{resouce_address}**.

6.4.1. PTP events REST API v2 endpoints

6.4.1.1. api/ocloudNotifications/v2/subscriptions

HTTP method
GET api/ocloudNotifications/v2/subscriptions

Description

Returns a list of subscriptions. If subscriptions exist, a **200 OK** status code is returned along with the list of subscriptions.

Example API response


```
[
 {
  "ResourceAddress": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
  "SubscriptionId": "ccedbf08-3f96-4839-a0b6-2eb0401855ed",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/ccedbf08-3f96-4839-a0b6-2eb0401855ed"
 },
 {
  "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/clock-class",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
  "SubscriptionId": "a939a656-1b7d-4071-8cf1-f99af6e931f2",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/a939a656-1b7d-4071-8cf1-f99af6e931f2"
 },
 {
  "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
  "SubscriptionId": "ba4564a3-4d9e-46c5-b118-591d3105473c",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/ba4564a3-4d9e-46c5-b118-591d3105473c"
 },
 {
  "ResourceAddress": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
  "SubscriptionId": "ea0d772e-f00a-4889-98be-51635559b4fb",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/ea0d772e-f00a-4889-98be-51635559b4fb"
 },
 {
  "ResourceAddress": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
  "EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
  "SubscriptionId": "762999bf-b4a0-4bad-abe8-66e646b65754",
  "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/762999bf-b4a0-4bad-abe8-66e646b65754"
 }
]
```


HTTP method
POST api/ocloudNotifications/v2/subscriptions

Description
Creates a new subscription for the required event by passing the appropriate payload.

You can subscribe to the following PTP events:

- **sync-state** events
- **lock-state** events
- **gnss-sync-status events** events
- **os-clock-sync-state** events
- **clock-class** events

Table 6.13. Query parameters

Parameter	Type
subscription	data

Example sync-state subscription payload

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/sync-status/sync-state"
}
```

Example PTP lock-state events subscription payload

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/ptp-status/lock-state"
}
```

Example PTP gnss-sync-status events subscription payload

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/gnss-status/gnss-sync-status"
}
```

Example PTP os-clock-sync-state events subscription payload

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/sync-status/os-clock-sync-state"
}
```

Example PTP clock-class events subscription payload

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/ptp-status/clock-class"
}
```

Example API response

```
{
   "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
   "EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
   "SubscriptionId": "620283f3-26cd-4a6d-b80a-bdc4b614a96a",
   "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/620283f3-26cd-4a6d-b80a-bdc4b614a96a"
}
```

The following subscription status events are possible:

Table 6.14. PTP events REST API v2 subscription status codes

Status code	Description
201 Created	Indicates that the subscription is created
400 Bad Request	Indicates that the server could not process the request because it was malformed or invalid
404 Not Found	Indicates that the subscription resource is not available
409 Conflict	Indicates that the subscription already exists

HTTP method
DELETE api/ocloudNotifications/v2/subscriptions

Description
Deletes all subscriptions.

Example API response

```
{
"status": "deleted all subscriptions"
}
```

6.4.1.2. api/ocloudNotifications/v2/subscriptions/{subscription_id}

HTTP method
GET api/ocloudNotifications/v2/subscriptions/{subscription_id}

Description
Returns details for the subscription with ID **subscription_id**.

Table 6.15. Global path parameters

Parameter	Type
subscription_id	string

Example API response


```
{
   "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
   "EndpointUri": "http://consumer-events-subscription-service.cloud-events.svc.cluster.local:9043/event",
   "SubscriptionId": "620283f3-26cd-4a6d-b80a-bdc4b614a96a",
   "UriLocation": "http://ptp-event-publisher-service-compute-1.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions/620283f3-26cd-4a6d-b80a-bdc4b614a96a"
}
```


HTTP method
DELETE api/ocloudNotifications/v2/subscriptions/{subscription_id}

Description
Deletes the subscription with ID **subscription_id**.

Table 6.16. Global path parameters

Parameter	Type
subscription_id	string

Table 6.17. HTTP response codes

HTTP response	Description
204 No Content	Success

6.4.1.3. api/ocloudNotifications/v2/health

HTTP method
GET api/ocloudNotifications/v2/health/

Description
Returns the health status for the **ocloudNotifications** REST API.

Table 6.18. HTTP response codes

HTTP response	Description
200 OK	Success

6.4.1.4. api/ocloudNotifications/v2/publishers

HTTP method
GET api/ocloudNotifications/v2/publishers

Description
Returns a list of publisher details for the cluster node. The system generates notifications when the relevant equipment state changes.

You can use equipment synchronization status subscriptions together to deliver a detailed view of the overall synchronization health of the system.

Example API response


```
[
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "4ea72bfa-185c-4703-9694-cdd0434cd570",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/4ea72bfa-185c-4703-9694-cdd0434cd570"
  },
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "71fbb38e-a65d-41fc-823b-d76407901087",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/71fbb38e-a65d-41fc-823b-d76407901087"
  },
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/clock-class",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "7bc27cad-03f4-44a9-8060-a029566e7926",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/7bc27cad-03f4-44a9-8060-a029566e7926"
  },
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "6e7b6736-f359-46b9-991c-fbaed25eb554",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/6e7b6736-f359-46b9-991c-fbaed25eb554"
  },
  {
    "ResourceAddress": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",
    "EndpointUri": "http://localhost:9043/api/ocloudNotifications/v2/dummy",
    "SubscriptionId": "31bb0a45-7892-45d4-91dd-13035b13ed18",
    "UriLocation": "http://localhost:9043/api/ocloudNotifications/v2/publishers/31bb0a45-7892-45d4-
```

```
91dd-13035b13ed18"
  }
]
```

Table 6.19. HTTP response codes

HTTP response	Description
200 OK	Success

6.4.1.5. api/ocloudNotifications/v2/{resource_address}/CurrentState

HTTP method
GET api/ocloudNotifications/v2/cluster/node/{node_name}/sync/ptp-status/lock-state/CurrentState

GET api/ocloudNotifications/v2/cluster/node/{node_name}/sync/sync-status/os-clock-sync-state/CurrentState

GET api/ocloudNotifications/v2/cluster/node/{node_name}/sync/ptp-status/clock-class/CurrentState

GET api/ocloudNotifications/v2/cluster/node/{node_name}/sync/sync-status/sync-state/CurrentState

GET api/ocloudNotifications/v2/cluster/node/{node_name}/sync/gnss-status/gnss-sync-state/CurrentState

Description
Returns the current state of the **os-clock-sync-state**, **clock-class**, **lock-state**, **gnss-sync-status**, or **sync-state** events for the cluster node.

- **os-clock-sync-state** notifications describe the host operating system clock synchronization state. Can be in **LOCKED** or **FREERUN** state.
- **clock-class** notifications describe the current state of the PTP clock class.
- **lock-state** notifications describe the current status of the PTP equipment lock state. Can be in **LOCKED**, **HOLDOVER** or **FREERUN** state.
- **sync-state** notifications describe the current status of the least synchronized of the PTP clock **lock-state** and **os-clock-sync-state** states.
- **gnss-sync-status** notifications describe the GNSS clock synchronization state.

Table 6.20. Global path parameters

Parameter	Type
resource_address	string

Example lock-state API response

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```


Example os-clock-sync-state API response


```
{
  "specversion": "0.3",
  "id": "4f51fe99-feaa-4e66-9112-66c5c9b9afcb",
  "source": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "subject": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "datacontenttype": "application/json",
  "time": "2022-11-29T17:44:22.202Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "27"
      }
    ]
  }
}
```


Example clock-class API response

```
{
  "id": "064c9e67-5ad4-4afb-98ff-189c6aa9c205",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/clock-class",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
```


Example sync-state API response


```
{
    "specversion": "0.3",
    "id": "8c9d6ecb-ae9f-4106-82c4-0a778a79838d",
    "source": "/sync/sync-status/sync-state",
    "type": "event.sync.sync-status.synchronization-state-change",
    "subject": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
    "datacontenttype": "application/json",
    "time": "2024-08-28T14:50:57.327585316Z",
    "data":
    {
        "version": "1.0",
        "values": [
        {
            "ResourceAddress": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
            "data_type": "notification",
            "value_type": "enumeration",
            "value": "LOCKED"
        }]
    }
}
```


Example gnss-sync-state API response


```
{
  "id": "435e1f2a-6854-4555-8520-767325c087d7",
  "type": "event.sync.gnss-status.gnss-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",
  "dataContentType": "application/json",
  "time": "2023-09-27T19:35:33.42347206Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens2fx/master",
```

```
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "SYNCHRONIZED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens2fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "5"
      }
    ]
  }
}
```

6.5. DEVELOPING PTP EVENTS CONSUMER APPLICATIONS WITH THE REST API V1

When developing consumer applications that make use of Precision Time Protocol (PTP) events on a bare-metal cluster node, you deploy your consumer application in a separate application pod. The consumer application subscribes to PTP events by using the PTP events REST API v1.

NOTE

The following information provides general guidance for developing consumer applications that use PTP events. A complete events consumer application example is outside the scope of this information.

IMPORTANT

PTP events REST API v1 and events consumer application sidecar is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

Additional resources

- PTP events REST API v1 reference

6.5.1. About the PTP fast event notifications framework

Use the Precision Time Protocol (PTP) fast event REST API v2 to subscribe cluster applications to PTP events that the bare-metal cluster node generates.

NOTE

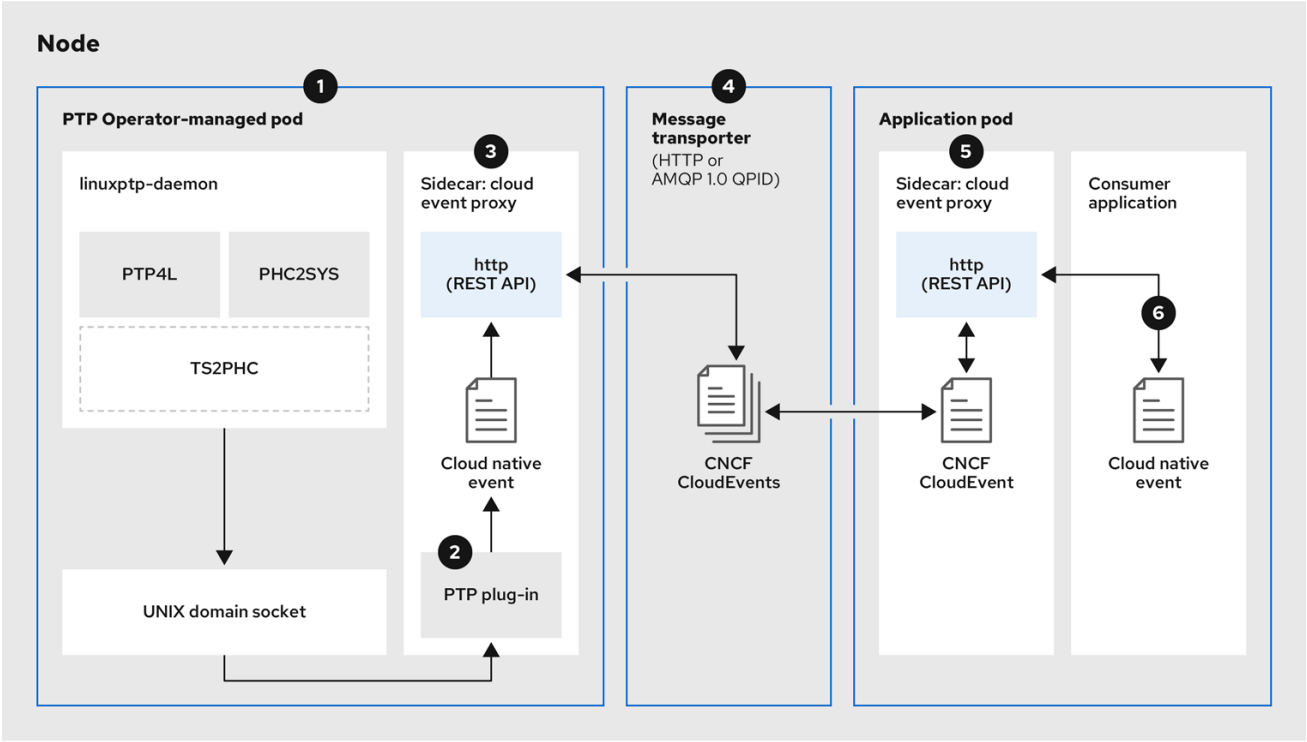
The fast events notifications framework uses a REST API for communication. The PTP events REST API v1 and v2 are based on the *O-RAN O-Cloud Notification API Specification for Event Consumers 4.0* that is available from O-RAN ALLIANCE Specifications.

Only the PTP events REST API v2 is O-RAN v4 compliant.

6.5.2. Retrieving PTP events with the PTP events REST API v1

Applications run the **cloud-event-proxy** container in a sidecar pattern to subscribe to PTP events. The **cloud-event-proxy** sidecar container can access the same resources as the primary application container without using any of the resources of the primary application and with no significant latency.

Figure 6.7. Overview of PTP fast events with consumer sidecar and HTTP message transport



1 **Event is generated on the cluster host**

linuxptp-daemon in the PTP Operator-managed pod runs as a Kubernetes **DaemonSet** and manages the various **linuxptp** processes (**ptp4l**, **phc2sys**, and optionally for grandmaster clocks, **ts2phc**). The **linuxptp-daemon** passes the event to the UNIX domain socket.

2 **Event is passed to the cloud-event-proxy sidecar**

The PTP plugin reads the event from the UNIX domain socket and passes it to the **cloud-event-proxy** sidecar in the PTP Operator-managed pod. **cloud-event-proxy** delivers the event from the Kubernetes infrastructure to Cloud-Native Network Functions (CNFs) with low latency.

3 **Event is persisted**

The **cloud-event-proxy** sidecar in the PTP Operator-managed pod processes the event and publishes the cloud-native event by using a REST API.

4 **Message is transported**

The message transporter transports the event to the **cloud-event-proxy** sidecar in the application pod over HTTP.

5 **Event is available from the REST API**

The **cloud-event-proxy** sidecar in the Application pod processes the event and makes it available by using the REST API.

6 **Consumer application requests a subscription and receives the subscribed event**

The consumer application sends an API request to the **cloud-event-proxy** sidecar in the application pod to create a PTP events subscription. The **cloud-event-proxy** sidecar creates an HTTP messaging listener protocol for the resource specified in the subscription.

The **cloud-event-proxy** sidecar in the application pod receives the event from the PTP Operator-managed pod, unwraps the cloud events object to retrieve the data, and posts the event to the consumer application. The consumer application listens to the address specified in the resource qualifier and receives and processes the PTP event.

6.5.3. Configuring the PTP fast event notifications publisher

To start using PTP fast event notifications for a network interface in your cluster, you must enable the fast event publisher in the PTP Operator **PtpOperatorConfig** custom resource (CR) and configure **ptpClockThreshold** values in a **PtpConfig** CR that you create.

Prerequisites

- You have installed the OpenShift Container Platform CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed the PTP Operator.

Procedure

1. Modify the default PTP Operator config to enable PTP fast events.
   a. Save the following YAML in the **ptp-operatorconfig.yaml** file:

```
apiVersion: ptp.openshift.io/v1
kind: PtpOperatorConfig
metadata:
  name: default
  namespace: openshift-ptp
spec:
  daemonNodeSelector:
    node-role.kubernetes.io/worker: ""
  ptpEventConfig:
    enableEventPublisher: true (1)
```

(1) Enable PTP fast event notifications by setting **enableEventPublisher** to **true**.

NOTE

In OpenShift Container Platform 4.13 or later, you do not need to set the **spec.ptpEventConfig.transportHost** field in the **PtpOperatorConfig** resource when you use HTTP transport for PTP events.

b. Update the **PtpOperatorConfig** CR:

```
$ oc apply -f ptp-operatorconfig.yaml
```

2. Create a **PtpConfig** custom resource (CR) for the PTP enabled interface, and set the required values for **ptpClockThreshold** and **ptp4lOpts**. The following YAML illustrates the required values that you must set in the **PtpConfig** CR:

```
spec:
  profile:
  - name: "profile1"
    interface: "enp5s0f0"
    ptp4lOpts: "-2 -s --summary_interval -4" 1
    phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
    ptp4lConf: "" 3
    ptpClockThreshold: 4
      holdOverTimeout: 5
      maxOffsetThreshold: 100
      minOffsetThreshold: -100
```

1 Append **--summary_interval -4** to use PTP fast events.

2 Required **phc2sysOpts** values. **-m** prints messages to **stdout**. The **linuxptp-daemon DaemonSet** parses the logs and generates Prometheus metrics.

3 Specify a string that contains the configuration to replace the default **/etc/ptp4l.conf** file. To use the default configuration, leave the field empty.

4 Optional. If the **ptpClockThreshold** stanza is not present, default values are used for the **ptpClockThreshold** fields. The stanza shows default **ptpClockThreshold** values. The **ptpClockThreshold** values configure how long after the PTP master clock is disconnected before PTP events are triggered. **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. The **maxOffsetThreshold** and **minOffsetThreshold** settings configure offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**). When the **ptp4l** or **phc2sys** offset value is outside this range, the PTP clock state is set to **FREERUN**. When the offset value is within this range, the PTP clock state is set to **LOCKED**.

Additional resources

- For a complete example CR that configures **linuxptp** services as an ordinary clock with PTP fast events, see Configuring linuxptp services as ordinary clock .

6.5.4. PTP events consumer application reference

PTP event consumer applications require the following features:

1. A web service running with a **POST** handler to receive the cloud native PTP events JSON payload
2. A **createSubscription** function to subscribe to the PTP events producer
3. A **getCurrentState** function to poll the current state of the PTP events producer

The following example Go snippets illustrate these requirements:

Example PTP events consumer server function in Go

```
func server() {
  http.HandleFunc("/event", getEvent)
  http.ListenAndServe("localhost:8989", nil)
}

func getEvent(w http.ResponseWriter, req *http.Request) {
  defer req.Body.Close()
  bodyBytes, err := io.ReadAll(req.Body)
  if err != nil {
    log.Errorf("error reading event %v", err)
  }
  e := string(bodyBytes)
  if e != "" {
    processEvent(bodyBytes)
    log.Infof("received event %s", string(bodyBytes))
  } else {
    w.WriteHeader(http.StatusNoContent)
  }
}
```

Example PTP events createSubscription function in Go

```
import (
"github.com/redhat-cne/sdk-go/pkg/pubsub"
"github.com/redhat-cne/sdk-go/pkg/types"
v1pubsub "github.com/redhat-cne/sdk-go/v1/pubsub"
)

// Subscribe to PTP events using REST API
s1,_:=createsubscription("/cluster/node/<node_name>/sync/sync-status/os-clock-sync-state") (1)
s2,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/class-change")
s3,_:=createsubscription("/cluster/node/<node_name>/sync/ptp-status/lock-state")

// Create PTP event subscriptions POST
func createSubscription(resourceAddress string) (sub pubsub.PubSub, err error) {
  var status int
     apiPath:= "/api/ocloudNotifications/v1/"
     localAPIAddr:=localhost:8989 // vDU service API address
     apiAddr:= "localhost:8089" // event framework API address

  subURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: apiAddr
```

```
    Path: fmt.Sprintf("%s%s", apiPath, "subscriptions")}}
  endpointURL := &types.URI{URL: url.URL{Scheme: "http",
    Host: localAPIAddr,
    Path: "event"}}

  sub = v1pubsub.NewPubSub(endpointURL, resourceAddress)
  var subB []byte

  if subB, err = json.Marshal(&sub); err == nil {
    rc := restclient.New()
    if status, subB = rc.PostWithReturn(subURL, subB); status != http.StatusCreated {
      err = fmt.Errorf("error in subscription creation api at %s, returned status %d", subURL, status)
    } else {
      err = json.Unmarshal(subB, &sub)
    }
  } else {
    err = fmt.Errorf("failed to marshal subscription for %s", resourceAddress)
  }
  return
}
```

1 Replace **<node_name>** with the FQDN of the node that is generating the PTP events. For example, **compute-1.example.com**.

Example PTP events consumer getCurrentState function in Go

```
//Get PTP event state for the resource
func getCurrentState(resource string) {
  //Create publisher
  url := &types.URI{URL: url.URL{Scheme: "http",
    Host: localhost:8989,
    Path: fmt.SPrintf("/api/ocloudNotifications/v1/%s/CurrentState",resource}}
  rc := restclient.New()
  status, event := rc.Get(url)
  if status != http.StatusOK {
    log.Errorf("CurrentState:error %d from url %s, %s", status, url.String(), event)
  } else {
    log.Debugf("Got CurrentState: %s ", event)
  }
}
```

6.5.5. Reference cloud-event-proxy deployment and service CRs

Use the following example **cloud-event-proxy** deployment and subscriber service CRs as a reference when deploying your PTP events consumer application.

Reference cloud-event-proxy deployment with HTTP transport

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: event-consumer-deployment
  namespace: <namespace>
  labels:
```

```
    app: consumer
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
    spec:
      serviceAccountName: sidecar-consumer-sa
      containers:
        - name: event-subscriber
          image: event-subscriber-app
        - name: cloud-event-proxy-as-sidecar
          image: openshift4/ose-cloud-event-proxy
          args:
            - "--metrics-addr=127.0.0.1:9091"
            - "--store-path=/store"
            - "--transport-host=consumer-events-subscription-service.cloud-events.svc.cluster.local:9043"
            - "--http-event-publishers=ptp-event-publisher-service-NODE_NAME.openshift-
ptp.svc.cluster.local:9043"
            - "--api-port=8089"
          env:
            - name: NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          volumeMounts:
            - name: pubsubstore
              mountPath: /store
          ports:
            - name: metrics-port
              containerPort: 9091
            - name: sub-port
              containerPort: 9043
          volumes:
            - name: pubsubstore
              emptyDir: {}
```

Reference cloud-event-proxy subscriber service

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
    service.alpha.openshift.io/serving-cert-secret-name: sidecar-consumer-secret
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
```

```
    port: 9043
selector:
  app: consumer
clusterIP: None
sessionAffinity: None
type: ClusterIP
```

6.5.6. Subscribing to PTP events with the REST API v1

Deploy your **cloud-event-consumer** application container and **cloud-event-proxy** sidecar container in a separate application pod.

Subscribe the **cloud-event-consumer** application to PTP events posted by the **cloud-event-proxy** container at **http://localhost:8089/api/ocloudNotifications/v1/** in the application pod.

NOTE

9089 is the default port for the **cloud-event-consumer** container deployed in the application pod. You can configure a different port for your application as required.

Additional resources

- api/ocloudNotifications/v1/subscriptions

6.5.7. Verifying that the PTP events REST API v1 consumer application is receiving events

Verify that the **cloud-event-proxy** container in the application pod is receiving PTP events.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have installed and configured the PTP Operator.

Procedure

1. Get the list of active **linuxptp-daemon** pods. Run the following command:

```
$ oc get pods -n openshift-ptp
```

Example output

```
NAME                 READY   STATUS    RESTARTS   AGE
linuxptp-daemon-2t78p   3/3     Running   0          8h
linuxptp-daemon-k8n88   3/3     Running   0          8h
```

2. Access the metrics for the required consumer-side **cloud-event-proxy** container by running the following command:

```
$ oc exec -it <linuxptp-daemon> -n openshift-ptp -c cloud-event-proxy -- curl
```

```
127.0.0.1:9091/metrics
```

where:

<linuxptp-daemon>
: Specifies the pod you want to query, for example, **linuxptp-daemon-2t78p**.

Example output

```
# HELP cne_transport_connections_resets Metric to get number of connection resets
# TYPE cne_transport_connections_resets gauge
cne_transport_connection_reset 1
# HELP cne_transport_receiver Metric to get number of receiver created
# TYPE cne_transport_receiver gauge
cne_transport_receiver{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 2
cne_transport_receiver{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 2
# HELP cne_transport_sender Metric to get number of sender created
# TYPE cne_transport_sender gauge
cne_transport_sender{address="/cluster/node/compute-
1.example.com/ptp",status="active"} 1
cne_transport_sender{address="/cluster/node/compute-
1.example.com/redfish/event",status="active"} 1
# HELP cne_events_ack Metric to get number of events produced
# TYPE cne_events_ack gauge
cne_events_ack{status="success",type="/cluster/node/compute-1.example.com/ptp"} 18
cne_events_ack{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP cne_events_transport_published Metric to get number of events published by the
transport
# TYPE cne_events_transport_published gauge
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="failed"} 1
cne_events_transport_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_transport_received Metric to get number of events received  by the
transport
# TYPE cne_events_transport_received gauge
cne_events_transport_received{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 18
cne_events_transport_received{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 18
# HELP cne_events_api_published Metric to get number of events published by the rest
api
# TYPE cne_events_api_published gauge
cne_events_api_published{address="/cluster/node/compute-
1.example.com/ptp",status="success"} 19
cne_events_api_published{address="/cluster/node/compute-
1.example.com/redfish/event",status="success"} 19
# HELP cne_events_received Metric to get number of events received
```

```
# TYPE cne_events_received gauge
cne_events_received{status="success",type="/cluster/node/compute-1.example.com/ptp"}
18
cne_events_received{status="success",type="/cluster/node/compute-
1.example.com/redfish/event"} 18
# HELP promhttp_metric_handler_requests_in_flight Current number of scrapes being
served.
# TYPE promhttp_metric_handler_requests_in_flight gauge
promhttp_metric_handler_requests_in_flight 1
# HELP promhttp_metric_handler_requests_total Total number of scrapes by HTTP status
code.
# TYPE promhttp_metric_handler_requests_total counter
promhttp_metric_handler_requests_total{code="200"} 4
promhttp_metric_handler_requests_total{code="500"} 0
promhttp_metric_handler_requests_total{code="503"} 0
```

6.5.8. Monitoring PTP fast event metrics

You can monitor PTP fast events metrics from cluster nodes where the **linuxptp-daemon** is running. You can also monitor PTP fast event metrics in the OpenShift Container Platform web console by using the preconfigured and self-updating Prometheus monitoring stack.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in as a user with **cluster-admin** privileges.
- Install and configure the PTP Operator on a node with PTP-capable hardware.

Procedure

1. Start a debug pod for the node by running the following command:

```
$ oc debug node/<node_name>
```

2. Check for PTP metrics exposed by the **linuxptp-daemon** container. For example, run the following command:

```
sh-4.4# curl http://localhost:9091/metrics
```

Example output

```
# HELP cne_api_events_published Metric to get number of events published by the rest api
# TYPE cne_api_events_published gauge
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/gnss-
status/gnss-sync-status",status="success"} 1
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-
status/lock-state",status="success"} 94
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/ptp-
status/class-change",status="success"} 18
cne_api_events_published{address="/cluster/node/compute-1.example.com/sync/sync-
status/os-clock-sync-state",status="success"} 27
```

3. Optional. You can also find PTP events in the logs for the **cloud-event-proxy** container. For example, run the following command:

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

4. To view the PTP event in the OpenShift Container Platform web console, copy the name of the PTP metric you want to query, for example, **openshift_ptp_offset_ns**.

5. In the OpenShift Container Platform web console, click **Observe → Metrics**.

6. Paste the PTP metric name into the **Expression** field, and click **Run queries**.

Additional resources

- Accessing metrics as a developer

6.5.9. PTP fast event metrics reference

The following table describes the PTP fast events metrics that are available from cluster nodes where the **linuxptp-daemon** service is running.

Table 6.21. PTP fast event metrics

Metric	Description	Example
openshift_ptp_clock_class	Returns the PTP clock class for the interface. Possible values for PTP clock class are 6 (**LOCKED**), 7 (**PRC UNLOCKED IN-SPEC**), 52 (**PRC UNLOCKED OUT-OF-SPEC**), 187 (**PRC UNLOCKED OUT-OF-SPEC**), 135 (**T-BC HOLDOVER IN-SPEC**), 165 (**T-BC HOLDOVER OUT-OF-SPEC**), 248 (**DEFAULT**), or 255 (**SLAVE ONLY CLOCK**).	**{node="compute-1.example.com",process="ptp4l"} 6**
openshift_ptp_clock_state	Returns the current PTP clock state for the interface. Possible values for PTP clock state are **FREERUN**, **LOCKED**, or **HOLDOVER**.	**{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1**
openshift_ptp_delay_ns	Returns the delay in nanoseconds between the primary clock sending the timing packet and the secondary clock receiving the timing packet.	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0**
openshift_ptp_ha_profile_status	Returns the current status of the highly available system clock when there are multiple time sources on different NICs. Possible values are 0 (**INACTIVE**) and 1 (**ACTIVE**).	**{node="node1",process="phc2sys",profile="profile1"} 1{node="node1",process="phc2sys",profile="profile2"} 0**

Metric	Description	Example
openshift_ptp_frequency_adjustment_ns	Returns the frequency adjustment in nanoseconds between 2 PTP clocks. For example, between the upstream clock and the NIC, between the system clock and the NIC, or between the PTP hardware clock (**phc**) and the NIC.	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -6768**
openshift_ptp_interface_role	Returns the configured PTP clock role for the interface. Possible values are 0 (**PASSIVE**), 1 (**SLAVE**), 2 (**MASTER**), 3 (**FAULTY**), 4 (**UNKNOWN**), or 5 (**LISTENING**).	**{iface="ens2f0", node="compute-1.example.com", process="ptp4l"} 2**
openshift_ptp_max_offset_ns	Returns the maximum offset in nanoseconds between 2 clocks or interfaces. For example, between the upstream GNSS clock and the NIC (**ts2phc**), or between the PTP hardware clock (**phc**) and the system clock (**phc2sys**).	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 1.038099569e+09**
openshift_ptp_offset_ns	Returns the offset in nanoseconds between the DPLL clock or the GNSS clock source and the NIC hardware clock.	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -9**
openshift_ptp_process_restart_count	Returns a count of the number of times the **ptp4l** and **ts2phc** processes were restarted.	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_process_status	Returns a status code that shows whether the PTP processes are running or not.	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_threshold	Returns values for **HoldOverTimeout**, **MaxOffsetThreshold**, and **MinOffsetThreshold**. • **holdOverTimeout** is the time value in seconds before the PTP clock event state changes to **FREERUN** when the PTP master clock is disconnected. • **maxOffsetThreshold** and **minOffsetThreshold** are offset values in nanoseconds that compare against the values for **CLOCK_REALTIME** (**phc2sys**) or master offset (**ptp4l**) values that you configure in the **PtpConfig** CR for the NIC.	**{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5**

6.5.9.1. PTP fast event metrics only when T-GM is enabled

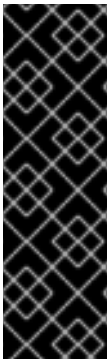
The following table describes the PTP fast event metrics that are available only when PTP grandmaster clock (T-GM) is enabled.

Table 6.22. PTP fast event metrics when T-GM is enabled

Metric	Description	Example
openshift_ptp_frequency_status	Returns the current status of the digital phase-locked loop (DPLL) frequency for the NIC. Possible values are -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), or 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_nmea_status	Returns the current status of the NMEA connection. NMEA is the protocol that is used for 1PPS NIC connections. Possible values are 0 (**UNAVAILABLE**) and 1 (**AVAILABLE**).	**{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1**
openshift_ptp_phase_status	Returns the status of the DPLL phase for the NIC. Possible values are -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), or 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_pps_status	Returns the current status of the NIC 1PPS connection. You use the 1PPS connection to synchronize timing between connected NICs. Possible values are 0 (**UNAVAILABLE**) and 1 (**AVAILABLE**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1**
openshift_ptp_gnss_status	Returns the current status of the global navigation satellite system (GNSS) connection. GNSS provides satellite-based positioning, navigation, and timing services globally. Possible values are 0 (**NOFIX**), 1 (**DEAD RECKONING ONLY**), 2 (**2D-FIX**), 3 (**3D-FIX**), 4 (**GPS+DEAD RECKONING FIX**), 5, (**TIME ONLY FIX**).	**{from="gnss",iface="ens2fx",node="compute-1.example.com",process="gnss"} 3**

6.6. PTP EVENTS REST API V1 REFERENCE

Use the following Precision Time Protocol (PTP) fast event REST API v1 endpoints to subscribe the **cloud-event-consumer** application to PTP events posted by the **cloud-event-proxy** container at **http://localhost:8089/api/ocloudNotifications/v1/** in the application pod.



IMPORTANT

PTP events REST API v1 and events consumer application sidecar is a deprecated feature. Deprecated functionality is still included in OpenShift Container Platform and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Container Platform, refer to the *Deprecated and removed features* section of the OpenShift Container Platform release notes.

The following API endpoints are available:

- **api/ocloudNotifications/v1/subscriptions**
  - **POST**: Creates a new subscription
  - **GET**: Retrieves a list of subscriptions
  - **DELETE**: Deletes all subscriptions
- **api/ocloudNotifications/v1/subscriptions/{subscription_id}**
  - **GET**: Returns details for the specified subscription ID
  - **DELETE**: Deletes the subscription associated with the specified subscription ID
- **api/ocloudNotifications/v1/health**
  - **GET**: Returns the health status of **ocloudNotifications** API
- **api/ocloudNotifications/v1/publishers**
  - **GET**: Returns a list of PTP event publishers for the cluster node
- **api/ocloudnotifications/v1/{resource_address}/CurrentState**
  - **GET**: Returns the current state of one the following event types: **sync-state**, **os-clock-sync-state**, **clock-class**, **lock-state**, or **gnss-sync-status** events

6.6.1. PTP events REST API v1 endpoints

6.6.1.1. api/ocloudNotifications/v1/subscriptions

HTTP method
GET api/ocloudNotifications/v1/subscriptions

Description
Returns a list of subscriptions. If subscriptions exist, a **200 OK** status code is returned along with the list of subscriptions.

Example API response

```
[
 {
```

```
  "id": "75b1ad8f-c807-4c23-acf5-56f4b7ee3826",
  "endpointUri": "http://localhost:9089/event",
  "uriLocation": "http://localhost:8089/api/ocloudNotifications/v1/subscriptions/75b1ad8f-c807-4c23-acf5-56f4b7ee3826",
  "resource": "/cluster/node/compute-1.example.com/ptp"
 }
]
```


HTTP method
POST api/ocloudNotifications/v1/subscriptions

Description
Creates a new subscription for the required event by passing the appropriate payload. If a subscription is successfully created, or if it already exists, a **201 Created** status code is returned. You can subscribe to the following PTP events:

- **lock-state** events
- **os-clock-sync-state** events
- **clock-class** events
- **gnss-sync-status** events
- **sync-state** events

Table 6.23. Query parameters

Parameter	Type
subscription	data

Example PTP events subscription payload


```
{
  "endpointUri": "http://localhost:8989/event",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```


Example PTP lock-state events subscription payload


```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/ptp-status/lock-state"
}
```


Example PTP os-clock-sync-state events subscription payload


```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/sync-status/os-clock-sync-state"
}
```

Example PTP clock-class events subscription payload

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/ptp-status/clock-class"
}
```

Example PTP gnss-sync-status events subscription payload

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/gnss-status/gnss-sync-status"
}
```

Example sync-state subscription payload

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/sync-status/sync-state"
}
```

HTTP method
DELETE api/ocloudNotifications/v1/subscriptions

Description
Deletes all subscriptions.

Example API response

```
{
"status": "deleted all subscriptions"
}
```

6.6.1.2. api/ocloudNotifications/v1/subscriptions/{subscription_id}

HTTP method
GET api/ocloudNotifications/v1/subscriptions/{subscription_id}

Description
Returns details for the subscription with ID **subscription_id**.

Table 6.24. Global path parameters

Parameter	Type
subscription_id	string

Example API response

```
{
  "id":"48210fb3-45be-4ce0-aa9b-41a0e58730ab",
```

```
  "endpointUri": "http://localhost:9089/event",
  "uriLocation":"http://localhost:8089/api/ocloudNotifications/v1/subscriptions/48210fb3-45be-4ce0-aa9b-41a0e58730ab",
  "resource":"/cluster/node/compute-1.example.com/ptp"
}
```


HTTP method
DELETE api/ocloudNotifications/v1/subscriptions/{subscription_id}

Description
Deletes the subscription with ID **subscription_id**.

Table 6.25. Global path parameters

Parameter	Type
subscription_id	string

Example API response


```
{
"status": "OK"
}
```


6.6.1.3. api/ocloudNotifications/v1/health

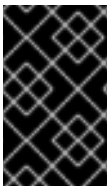
HTTP method
GET api/ocloudNotifications/v1/health/

Description
Returns the health status for the **ocloudNotifications** REST API.

Example API response

```
OK
```

6.6.1.4. api/ocloudNotifications/v1/publishers



IMPORTANT

The **api/ocloudNotifications/v1/publishers** endpoint is only available from the cloud-event-proxy container in the PTP Operator managed pod. It is not available for consumer applications in the application pod.

HTTP method
GET api/ocloudNotifications/v1/publishers

Description
Returns a list of publisher details for the cluster node. The system generates notifications when the relevant equipment state changes.

You can use equipment synchronization status subscriptions together to deliver a detailed view of the overall synchronization health of the system.

Example API response


```
[
  {
    "id": "0fa415ae-a3cf-4299-876a-589438bacf75",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/0fa415ae-a3cf-4299-876a-589438bacf75",
    "resource": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state"
  },
  {
    "id": "28cd82df-8436-4f50-bbd9-7a9742828a71",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/28cd82df-8436-4f50-bbd9-7a9742828a71",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/clock-class"
  },
  {
    "id": "44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/44aa480d-7347-48b0-a5b0-e0af01fa9677",
    "resource": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state"
  },
  {
    "id": "778da345d-4567-67b0-a43f0-rty885a456",
    "endpointUri": "http://localhost:9085/api/ocloudNotifications/v1/dummy",
    "uriLocation": "http://localhost:9085/api/ocloudNotifications/v1/publishers/778da345d-4567-67b0-a43f0-rty885a456",
    "resource": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status"
  }
]
```


6.6.1.5. api/ocloudNotifications/v1/{resource_address}/CurrentState

HTTP method

GET api/ocloudNotifications/v1/cluster/node/{node_name}/sync/ptp-status/lock-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/{node_name}/sync/sync-status/os-clock-sync-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/{node_name}/sync/ptp-status/clock-class/CurrentState

GET api/ocloudNotifications/v1/cluster/node/{node_name}/sync/sync-status/sync-state/CurrentState

GET api/ocloudNotifications/v1/cluster/node/{node_name}/sync/gnss-status/gnss-sync-state/CurrentState

Description

Returns the current state of the **os-clock-sync-state**, **clock-class**, **lock-state**, **gnss-sync-status**, or **sync-state** events for the cluster node.

- **os-clock-sync-state** notifications describe the host operating system clock synchronization state. Can be in **LOCKED** or **FREERUN** state.
- **clock-class** notifications describe the current state of the PTP clock class.
- **lock-state** notifications describe the current status of the PTP equipment lock state. Can be in **LOCKED**, **HOLDOVER** or **FREERUN** state.
- **sync-state** notifications describe the current status of the least synchronized of the **ptp-status/lock-state** and **sync-status/os-clock-sync-state** endpoints.
- **gnss-sync-status** notifications describe the GNSS clock synchronization state.

Table 6.26. Global path parameters

Parameter	Type
resource_address	string

Example lock-state API response

```
{
  "id": "c1ac3aa5-1195-4786-84f8-da0ea4462921",
  "type": "event.sync.ptp-status.ptp-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/lock-state",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:57.094981478Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "29"
      }
    ]
  }
}
```

Example os-clock-sync-state API response

```
{
  "specversion": "0.3",
```

```
  "id": "4f51fe99-feaa-4e66-9112-66c5c9b9afcb",
  "source": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "type": "event.sync.sync-status.os-clock-sync-state-change",
  "subject": "/cluster/node/compute-1.example.com/sync/sync-status/os-clock-sync-state",
  "datacontenttype": "application/json",
  "time": "2022-11-29T17:44:22.202Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/CLOCK_REALTIME",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "27"
      }
    ]
  }
}
```


Example clock-class API response


```
{
  "id": "064c9e67-5ad4-4afb-98ff-189c6aa9c205",
  "type": "event.sync.ptp-status.ptp-clock-class-change",
  "source": "/cluster/node/compute-1.example.com/sync/ptp-status/clock-class",
  "dataContentType": "application/json",
  "time": "2023-01-10T02:41:56.785673989Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens5fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "165"
      }
    ]
  }
}
```


Example sync-state API response


```
{
   "specversion": "0.3",
   "id": "8c9d6ecb-ae9f-4106-82c4-0a778a79838d",
   "source": "/sync/sync-status/sync-state",
   "type": "event.sync.sync-status.synchronization-state-change",
   "subject": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
   "datacontenttype": "application/json",
```

```
    "time": "2024-08-28T14:50:57.327585316Z",
    "data":
    {
        "version": "1.0",
        "values": [
        {
            "ResourceAddress": "/cluster/node/compute-1.example.com/sync/sync-status/sync-state",
            "data_type": "notification",
            "value_type": "enumeration",
            "value": "LOCKED"
        }]
    }
}
```


Example gnss-sync-status API response


```
{
  "id": "435e1f2a-6854-4555-8520-767325c087d7",
  "type": "event.sync.gnss-status.gnss-state-change",
  "source": "/cluster/node/compute-1.example.com/sync/gnss-status/gnss-sync-status",
  "dataContentType": "application/json",
  "time": "2023-09-27T19:35:33.42347206Z",
  "data": {
    "version": "1.0",
    "values": [
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens2fx/master",
        "data_type": "notification",
        "value_type": "enumeration",
        "value": "LOCKED"
      },
      {
        "ResourceAddress": "/cluster/node/compute-1.example.com/ens2fx/master",
        "data_type": "metric",
        "value_type": "decimal64.3",
        "value": "5"
      }
    ]
  }
}
```