OpenShift Container Platform 4.18

高级网络

安装并配置 OpenShift Container Platform 集群

Last Updated: 2025-10-07

OpenShift Container Platform 4.18 高级网络

安装并配置 OpenShift Container Platform 集群

Legal Notice

Abstract

本文档提供有关安装和配置 OpenShift Container Platform 的信息。

Table of Contents

第 1 章 验证到端点的连接

Cluster Network Operator（CNO）运行一个控制器（连接检查控制器），用于在集群的资源间执行连接健康检查。通过查看健康检查的结果，您可以诊断连接问题或解决网络连接问题，将其作为您要调查的问题的原因。

1.1. 执行的连接健康检查

要验证集群资源是否可以访问，请向以下集群 API 服务的每个服务都有一个 TCP 连接：

- Kubernetes API 服务器服务
- Kubernetes API 服务器端点
- OpenShift API 服务器服务
- OpenShift API 服务器端点
- 负载均衡器

要验证服务和服务端点是否可在集群中的每个节点上访问，请对以下每个目标都进行 TCP 连接：

- 健康检查目标服务
- 健康检查目标端点

1.2. 连接健康检查实现

在集群中，连接检查控制器编配连接验证检查。连接测试的结果存储在 **openshift-network-diagnostics** 命名空间中的 **PodNetworkConnectivity** 对象中。连接测试会每分钟以并行方式执行。

Cluster Network Operator（CNO）将几个资源部署到集群，以发送和接收连接性健康检查：

健康检查源

此程序部署在一个由 **Deployment** 对象管理的单个 pod 副本集中。程序会消耗 **PodNetworkConnectivity** 对象，并连接到每个对象中指定的 **spec.targetEndpoint**。

健康检查目标

pod 作为集群中每个节点上的守护进程集的一部分部署。pod 侦听入站健康检查。在每个节点上存在这个 pod 可以测试到每个节点的连接。

您可以使用节点选择器配置在其上运行网络连接源和目标的节点。另外，您可以为源和目标 pod 指定允许的*容限*。配置在 **config.openshift.io/v1** API 组中的 **Network** API 的单例 **cluster** 自定义资源中定义。

Pod 调度在更新了配置后发生。因此，您必须在更新配置前应用要在选择器中使用的节点标签。更新网络连接后应用的标签将忽略 pod 放置。

请参考以下 YAML 中的默认配置：

连接源和目标 pod 的默认配置

```
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. 指定网络诊断配置。如果没有指定值，或者指定了空对象，并在名为 **cluster** 的 **network.operator.openshift.io** 自定义资源中设置 **spec.disableNetworkDiagnostics=true**，则会禁用网络诊断。如果设置，这个值会覆盖 **spec.disableNetworkDiagnostics=true**。
2. 指定诊断模式。该值可以是空字符串、**All** 或 **Disabled**。空字符串等同于指定 **All**。
3. 可选：指定连接检查源 pod 的选择器。您可以使用 **nodeSelector** 和 **tolerations** 字段来进一步指定 **sourceNode** pod。对于源和目标 pod，它们都是可选的。您可以省略它们，同时使用它们，或者只使用其中一个。
4. 可选：指定连接检查目标 pod 的选择器。您可以使用 **nodeSelector** 和 **tolerations** 字段来进一步指定 **targetNode** pod。对于源和目标 pod，它们都是可选的。您可以省略它们，同时使用它们，或者只使用其中一个。

1.3. 配置 POD 连接检查放置

作为集群管理员，您可以通过修改名为 **cluster** 的 **network.config.openshift.io** 对象来配置运行连接 pod 的节点。

先决条件

- 安装 OpenShift CLI (**oc**) 。

流程

1. 输入以下命令编辑连接检查配置：

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

2. 在文本编辑器中，更新 **networkDiagnostics** 小节，以指定您要用于源和目标 pod 的节点选择器。
3. 保存更改并退出文本编辑器。

验证

- 输入以下命令验证源和目标 pod 是否在预期的节点上运行：

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

输出示例

```
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 对象字段

PodNetworkConnectivityCheck 对象字段在下表中描述。

表 1.1. PodNetworkConnectivityCheck 对象字段

字段	类型	描述
metadata.name	**字符串**	对象的名称，其格式如下：**<source>-to-<target>**。**<target>** 描述的目的地包括以下字符串之一： • **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	**字符串**	与对象关联的命名空间。此值始终为 **openshift-network-diagnostics**。

字段	类型	描述
spec.sourcePod	**字符串**	连接检查来源于的 pod 的名称，如 **network-check-source-596b4c6566-rgh92**。
spec.targetEndpoint	**字符串**	连接检查的目标，如 **api.devcluster.example.com:6443**。
spec.tlsClientCert	**对象**	要使用的 TLS 证书配置。
spec.tlsClientCert.name	**字符串**	使用的 TLS 证书的名称（若有）。默认值为空字符串。
status	**对象**	代表连接测试条件和最近连接发生和失败的日志的对象。
status.conditions	**数组**	连接检查以及任何之前的状态的最新状态。
status.failures	**数组**	连接测试日志不会失败。
status.outages	**数组**	涵盖任何中断的时间连接测试日志。
status.successes	**数组**	成功尝试的连接测试日志。

下表描述了 **status.conditions** 阵列中对象的字段：

表 1.2. status.conditions

字段	类型	描述
lastTransitionTime	**字符串**	连接条件从一个状态转换到另一个状态的时间。
message	**字符串**	有关最后一次转换的详情（人类可读的格式）。
reason	**字符串**	有关最后一次转换的详情（机器可读的格式）。
status	**字符串**	条件的状态。
type	**字符串**	条件的类型。

下表描述了 **status.conditions** 阵列中对象的字段：

表 1.3. status.outages

字段	类型	描述
end	**字符串**	连接失败时的时间戳。
endLogs	**数组**	连接日志条目，包括与成功关闭相关的日志条目。
message	**字符串**	以人类可读格式显示停机详情概述。
开始	**字符串**	第一次检测到连接失败时的时间戳。
startLogs	**数组**	连接日志条目，包括原始失败。

1.4.1. 连接日志字段

下表中描述了连接日志条目的字段。该对象用于以下字段：

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

表 1.4. 连接日志对象

字段	类型	描述
latency	**字符串**	记录操作的持续时间。
message	**字符串**	以人类可读格式提供的状态信息。
reason	**字符串**	以可读格式提供状态的原因。这个值是 **TCPConnect**、**TCPConnectError**、**DNSResolve**、**DNSError** 之一。
success	**布尔值**	指明日志条目是否成功或失败。
time	**字符串**	连接检查的开始时间。

1.5. 验证端点的网络连接

作为集群管理员，您可以验证端点的连接，如 API 服务器、负载均衡器、服务或 pod，并验证是否启用了网络诊断。

先决条件

- 安装 OpenShift CLI（**oc**）。

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 输入以下命令确认启用了网络诊断：

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

输出示例

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

2. 输入以下命令列出当前的 **PodNetworkConnectivityCheck** 对象：

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

输出示例

```
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. 查看连接测试日志：

   a. 在上一命令的输出中，标识您要查看连接日志的端点。

   b. 输入以下命令来查看对象：

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

   这里的 **<name>** 指定 **PodNetworkConnectivityCheck** 对象的名称。

   输出示例

```
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"
```

第 2 章 更改集群网络的 MTU

作为集群管理员，您可以在集群安装后更改集群网络的 MTU。这一更改具有破坏性，因为必须重启集群节点才能完成 MTU 更改。

2.1. 关于集群 MTU

在安装集群网络的最大传输单元(MTU)期间，会根据集群中节点的主网络接口的 MTU 自动检测到。您通常不需要覆盖检测到的 MTU。

您可能希望因为以下原因更改集群网络的 MTU：

- 集群安装过程中检测到的 MTU 不正确。
- 集群基础架构现在需要不同的 MTU，如添加需要不同 MTU 的节点来获得最佳性能

只有 OVN-Kubernetes 集群网络插件支持更改 MTU 值。

2.1.1. 服务中断注意事项

当您为集群启动 MTU 更改时，以下效果可能会影响服务可用性：

- 至少需要两个滚动重启才能完成迁移到新的 MTU。在此过程中，一些节点在重启时不可用。
- 部署到集群的特定应用程序带有较短的超时间隔，超过绝对 TCP 超时间隔可能会在 MTU 更改过程中造成中断。

2.1.2. MTU 值选择

在规划 MTU 迁移时，需要考虑两个相关但不同的 MTU 值。

- **Hardware MTU** ：此 MTU 值根据您的网络基础架构的具体设置。
- **Cluster network MTU** ：此 MTU 值始终小于您的硬件 MTU，以考虑集群网络覆盖开销。具体开销由您的网络插件决定。对于 OVN-Kubernetes，开销为 **100** 字节。

如果您的集群为不同的节点需要不同的 MTU 值，则必须从集群中任何节点使用的最低 MTU 值中减去网络插件的开销值。例如，如果集群中的某些节点的 MTU 为 **9001**，而某些节点的 MTU 为 **1500**，则必须将此值设置为 **1400**。

重要

为了避免选择节点无法接受的 MTU 值，请使用 **ip -d link** 命令验证网络接口接受的最大 MTU 值 (**maxmtu**)。

2.1.3. 迁移过程如何工作

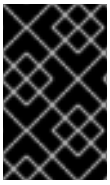
下表对迁移过程进行了概述，它分为操作中的用户发起的步骤，以及在响应过程中迁移过程要执行的操作。

表 2.1. 集群 MTU 的实时迁移

用户发起的步骤	OpenShift Container Platform 活动
在 Cluster Network Operator 配置中设置以下值： • **spec.migration.mtu.machine.to** • **spec.migration.mtu.network.from** • **spec.migration.mtu.network.to**	**Cluster Network Operator(CNO)** ：确认每个字段都设置为有效的值。 • 如果硬件的 MTU 没有改变，则 **mtu.machine.to** 必须设置为新硬件 MTU 或当前的硬件 MTU。这个值是临时的，被用作迁移过程的一部分。如果您指定了与现有硬件 MTU 值不同的硬件 MTU，您必须手动将 MTU 配置为持久，如机器配置、DHCP 设置或 Linux 内核命令行。 • **mtu.network.from** 字段必须等于 **network.status.clusterNetworkMTU** 字段，这是集群网络的当前 MTU。 • **mtu.network.to** 字段必须设置为目标集群网络 MTU，且必须小于硬件 MTU，以允许网络插件的覆盖开销。对于 OVN-Kubernetes，开销为 **100** 字节。 如果提供的值有效，CNO 会生成一个新的临时配置，它将集群网络集的 MTU 设置为 **mtu.network.to** 字段的值。 **Machine Config Operator(MCO)** ：执行集群中每个节点的滚动重启。
重新配置集群中节点的主网络接口 MTU。您可以使用各种方法完成此操作，包括： • 使用 MTU 更改部署新的 NetworkManager 连接配置集 • 通过 DHCP 服务器设置更改 MTU • 通过引导参数更改 MTU	N/A
在网络插件的 CNO 配置中设置 **mtu** 值，并将 **spec.migration** 设置为 **null**。	**Machine Config Operator(MCO)** ：使用新的 MTU 配置执行集群中每个节点的滚动重启。

2.2. 更改集群网络 MTU

作为集群管理员，您可以增加或减少集群的最大传输单元 (MTU)。



重要

您无法在 MTU 迁移过程中回滚节点的 MTU 值，但您可以在 MTU 迁移过程完成后回滚。

当 MTU 更新推出时，集群中的迁移具有破坏性且节点可能会临时不可用。

以下流程解释了如何使用机器配置、动态主机配置协议 (DHCP) 或 ISO 镜像更改集群网络 MTU。如果使用 DHCP 或 ISO 方法，则必须在安装集群后保留的配置工件来完成此流程。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 权限的账户访问集群。
- 已为集群识别目标 MTU。OVN-Kubernetes 网络插件的 MTU 必须设置为比集群中的最低硬件 MTU 值小 **100**。
- 如果您的节点是物理计算机，请确保集群网络和连接的网络交换机支持巨型帧。
- 如果您的节点是虚拟机 (VM)，请确保虚拟机监控程序和连接的网络交换机支持巨型帧。

流程

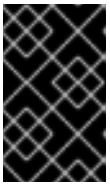
1. 要获得集群网络的当前 MTU，请输入以下命令：

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

输出示例

```
...
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. 为硬件 MTU 准备配置：



重要

可以通过多种方式为集群节点配置硬件 MTU。以下示例显示最常见的方法。在继续操作前，您必须验证您的基础架构 MTU 是否正确，并在集群节点中配置硬件 MTU 的首选方法是否生效。

- 如果您的硬件 MTU 通过 DHCP 指定，请使用以下 dnsmasq 配置更新 DHCP 配置：

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

其中：

<mtu>

指定要公告的 DHCP 服务器的硬件 MTU。

- 如果使用 PXE 的内核命令行指定硬件 MTU，请相应地更新该配置。
- 如果在 NetworkManager 连接配置中指定了硬件 MTU，请完成以下步骤。如果没有使用 DHCP、内核命令行或某种其他方法显式指定网络配置，则此方法是 OpenShift Container Platform 的默认方法。集群节点必须全部使用相同的底层网络配置，才能使以下过程未经修

改地工作。

i. 输入以下命令查找主网络接口：

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

其中：

<node_name>

指定集群中的节点的名称。

ii. 在 **<interface>-mtu.conf** 文件中创建以下 NetworkManager 配置：

NetworkManager 连接配置示例

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

其中：

<mtu>

指定新的硬件 MTU 值。

<interface>

指定主网络接口名称。

iii. 创建两个 **MachineConfig** 对象，一个用于 control plane 节点，另一个用于集群中的 worker 节点：

A. 在 **control-plane-interface.bu** 文件中创建以下 Butane 配置：

注意

您在配置文件中指定的 Butane 版本应与 OpenShift Container Platform 版本匹配，并且始终以 **0** 结尾。例如：**4.18.0**。有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

```
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) 指定主网络接口的 NetworkManager 连接名称。

2 指定上一步中更新的 NetworkManager 配置文件的本地文件名。

B. 在 **worker-interface.bu** 文件中创建以下 Butane 配置：

注意

您在配置文件中指定的 Butane 版本应与 OpenShift Container Platform 版本匹配，并且始终以 **0** 结尾。例如：**4.18.0**。有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

```
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 指定主网络接口的 NetworkManager 连接名称。

2 指定上一步中更新的 NetworkManager 配置文件的本地文件名。

C. 运行以下命令，从 Butane 配置创建 **MachineConfig** 对象：

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

警告

在此流程的稍后明确指示之前，不要应用这些机器配置。应用这些机器配置现在会导致集群的稳定性丢失。

3. 要开始 MTU 迁移，请输入以下命令指定迁移配置。Machine Config Operator 在集群中执行节点的滚动重启，以准备 MTU 更改。

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

其中：

<overlay_from>

指定当前的集群网络 MTU 值。

<overlay_to>

指定集群网络的目标 MTU。这个值相对于 **<machine_to>** 的值设置。对于 OVN-Kubernetes，这个值必须比 **<machine_to>** 的值小 **100**。

<machine_to>

指定底层主机网络上的主网络接口的 MTU。

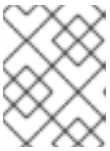
增加集群 MTU 的示例

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

4. 当 Machine Config Operator 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态： **UPDATED=true**、**UPDATING=false**、**DEGRADED=false**。



注意

默认情况下，Machine Config Operator 一次更新每个池中的一个机器，从而导致迁移总时间随着集群大小而增加。

5. 确认主机上新机器配置的状态：

   a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

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

   输出示例

```
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. 验证以下语句是否正确：

      - **machineconfiguration.openshift.io/state** 字段的值为 **Done**。
      - **machineconfiguration.openshift.io/currentConfig** 字段的值等于 **machineconfiguration.openshift.io/desiredConfig** 字段的值。

   c. 要确认机器配置正确，请输入以下命令：

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

这里的 **<config_name>** 是 **machineconfiguration.openshift.io/currentConfig** 字段中机器配置的名称。

机器配置必须包括以下对 systemd 配置的更新：

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

6. 更新底层网络接口 MTU 值：

   - 如果您要使用 NetworkManager 连接配置指定新 MTU，请输入以下命令。MachineConfig Operator 会自动执行集群中节点的滚动重启。

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

   - 如果您要使用 DHCP 服务器选项或内核命令行和 PXE 指定新 MTU，请对基础架构进行必要的更改。

7. 当 Machine Config Operator 更新每个机器配置池中的机器时，它会逐一重启每个节点。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

   ```
   $ oc get machineconfigpools
   ```

   成功更新的节点具有以下状态： **UPDATED=true**、**UPDATING=false**、**DEGRADED=false**。

   注意

   默认情况下，Machine Config Operator 一次更新每个池中的一个机器，从而导致迁移总时间随着集群大小而增加。

8. 确认主机上新机器配置的状态：

   a. 要列出机器配置状态和应用的机器配置名称，请输入以下命令：

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

      输出示例

      ```
      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
      ```

      验证以下语句是否正确：

      - **machineconfiguration.openshift.io/state** 字段的值为 **Done**。

- **machineconfiguration.openshift.io/currentConfig** 字段的值等于 **machineconfiguration.openshift.io/desiredConfig** 字段的值。

b. 要确认机器配置正确，请输入以下命令：

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

这里的 **<config_name>** 是 **machineconfiguration.openshift.io/currentConfig** 字段中机器配置的名称。

如果机器配置被成功部署，则前面的输出会包含 **/etc/NetworkManager/conf.d/99-<interface>-mtu.conf** 文件路径和 **ExecStart=/usr/local/bin/mtu-migration.sh** 行。

9. 要完成 MTU 迁移，请为 OVN-Kubernetes 网络插件输入以下命令：

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

其中：

<mtu>

指定您使用 **<overlay_to>** 指定的新集群网络 MTU。

10. 最终调整 MTU 迁移后，每个机器配置池节点都会逐个重启一个。您必须等到所有节点都已更新。输入以下命令检查机器配置池状态：

```
$ oc get machineconfigpools
```

成功更新的节点具有以下状态：**UPDATED=true**、**UPDATING=false**、**DEGRADED=false**。

验证

1. 要获得集群网络的当前 MTU，请输入以下命令：

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

2. 获取节点的主网络接口的当前 MTU：

a. 要列出集群中的节点，请输入以下命令：

```
$ oc get nodes
```

b. 要获取节点上主网络接口的当前 MTU 设置，请输入以下命令：

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

其中：

<node>

指定上一步中的输出节点。

<interface>

指定节点的主网络接口名称。

输出示例

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

2.3. 其他资源

- 使用高级网络选项进行 PXE 和 ISO 安装
- 使用密钥文件格式手动创建 NetworkManager 配置集
- 使用 nmcli 配置动态以太网连接

第 3 章 使用流控制传输协议 (SCTP)

作为集群管理员，您可以在一个裸机集群中使用集群中的流控制传输协议 (SCTP)。

3.1. 在 OPENSHIFT CONTAINER PLATFORM 上支持 SCTP

作为集群管理员，您可以在集群中的主机上启用 SCTP。在 Red Hat Enterprise Linux CoreOS (RHCOS) 上，SCTP 模块被默认禁用。

SCTP 是基于信息的可靠协议，可在 IP 网络之上运行。

启用后，您可以使用 SCTP 作为带有 pod、服务和网络策略的协议。**Service** 对象必须通过将 **type** 参数设置为 **ClusterIP** 或 **NodePort** 值来定义。

3.1.1. 使用 SCTP 协议的示例配置

您可以通过将 pod 或服务对象中的 **protocol** 参数设置为 **SCTP** 来将 pod 或服务配置为使用 SCTP。

在以下示例中，pod 被配置为使用 SCTP:

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

在以下示例中，服务被配置为使用 SCTP:

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

在以下示例中，**NetworkPolicy** 对象配置为对来自具有特定标签的任何 pod 的端口 **80** 应用 SCTP 网络流量：

```
kind: NetworkPolicy
```

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

3.2. 启用流控制传输协议 (SCTP)

作为集群管理员，您可以在集群中的 worker 节点上加载并启用列入黑名单的 SCTP 内核模块。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建名为 **load-sctp-module.yaml** 的文件，其包含以下 YAML 定义：

```
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. 运行以下命令来创建 **MachineConfig** 对象：

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

3. 可选：要在 MachineConfig Operator 应用配置更改时监测节点的状态，请使用以下命令。当节点状态变为 **Ready**时，则代表配置更新已被应用。

```
$ oc get nodes
```

3.3. 验证流控制传输协议 (SCTP) 已启用

您可以通过创建一个 pod 以及侦听 SCTP 流量的应用程序，将其与服务关联，然后连接到公开的服务，来验证 SCTP 是否在集群中工作。

先决条件

- 从集群访问互联网来安装 **nc** 软件包。
- 安装 OpenShift CLI (**oc**) 。
- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建 pod 启动 SCTP 侦听程序：
   a. 创建名为 **sctp-server.yaml** 的文件，该文件使用以下 YAML 定义 pod:

```
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. 运行以下命令来创建 pod：

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

2. 为 SCTP 侦听程序 pod 创建服务。
   a. 创建名为 **sctp-service.yaml** 的文件，该文件使用以下 YAML 定义服务：

```
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. 要创建服务，请输入以下命令：

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

3. 为 SCTP 客户端创建 pod。

a. 使用以下 YAML 创建名为 **sctp-client.yaml** 的文件：

```
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. 运行以下命令来创建 **Pod** 对象：

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

4. 在服务器中运行 SCTP 侦听程序。

a. 要连接到服务器 pod，请输入以下命令：

```
$ oc rsh sctpserver
```

b. 要启动 SCTP 侦听程序，请输入以下命令：

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

5. 连接到服务器上的 SCTP 侦听程序。

a. 在终端程序里打开一个新的终端窗口或标签页。

b. 获取 **sctpservice** 服务的 IP 地址。使用以下命令：

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

c. 要连接到客户端 pod，请输入以下命令：

```
$ oc rsh sctpclient
```

d. 要启动 SCTP 客户端，请输入以下命令。将 **<cluster_IP>** 替换为 **sctpservice** 服务的集群 IP 地址。

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

第 4 章 将二级接口指标与网络附加关联

管理员可以使用 **pod_network_info** 指标来分类和监控二级网络接口。指标通过添加标识接口类型的标签（通常基于关联的 **NetworkAttachmentDefinition** 资源）来实现此目的。

4.1. 为监控扩展二级网络指标

二级设备或接口用于不同目的。需要对二级网络接口的指标进行分类，以允许有效的聚合和监控。

公开的指标会包括接口，但不会指定接口的起始位置。当没有其他接口时，这可以正常工作。但是，当添加二级接口时，依赖接口名称会变得有问题，因为很难识别其目的并有效地使用其指标。

添加二级接口时，它们的名称取决于添加它们的顺序。二级接口可以属于不同的网络，它们各自满足不同的目的。

通过使用 **pod_network_name_info**，可以使用标识接口类型的额外信息来扩展当前的指标。这样，就可以聚合指标，并为特定接口类型添加特定的警告。

网络类型从 **NetworkAttachmentDefinition** 资源的名称生成，用于区分不同的二级网络类。例如，属于不同网络或使用不同的 CNI 的不同接口使用不同的网络附加定义名称。

4.2. 网络指标守护进程

网络指标守护进程是收集并发布与网络相关的指标的守护进程组件。

kubelet 已经发布了您可以观察到的网络相关指标。这些指标是：

- **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**

这些指标中的标签包括：

- Pod 名称
- Pod 命名空间
- 接口名称（比如 **eth0**）

这些指标在为 pod 添加新接口之前（例如通过 Multus ）可以正常工作。在添加了新接口后，无法清楚地知道接口名称代表什么。

interface 标签指向接口名称，但它不知道接口的作用是什么。在有多个不同接口的情况下，无法了解您监控的指标代表什么网络。

现在引入了新的 **pod_network_name_info** 可以帮助解决这个问题。

4.3. 带有网络名称的指标

Network Metrics daemonset 发布 **pod_network_name_info** 量表指标，固定值为 **0。**

pod_network_name_info示例

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

使用 Multus 所添加的注解生成网络名称标签。它是网络附加定义所属命名空间的连接，加上网络附加定义的名称。

新的指标本身不会提供很多值，但与网络相关的 **container_network_*** 指标一起使用，可以为二集网络的监控提供更好的支持。

通过使用类似以下的 **promql** 查询，可以获取包含值的新指标，以及从 **k8s.v1.cni.cncf.io/network-status** 注解中检索的网络名称：

```
(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)
```

第 5 章 使用 PTP 硬件

5.1. 关于 OPENSHIFT 集群节点中的精确时间协议

精度时间协议(PTP)用于同步网络中的时钟。与硬件支持一起使用时，PTP 能够达到微秒级的准确性，比网络时间协议 (NTP) 更加准确。

重要

如果您的带有 PTP 的 **openshift-sdn** 集群使用 User Datagram Protocol (UDP) 进行硬件时间戳，且迁移到 OVN-Kubernetes 插件，则硬件时间戳无法应用到主接口设备，如 Open vSwitch (OVS) 网桥。因此，UDP 版本 4 配置无法使用 **br-ex** 接口。

您可以配置 **linuxptp** 服务，并在 OpenShift Container Platform 集群节点中使用具有 PTP 功能的硬件。

通过部署 PTP Operator，使用 OpenShift Container Platform Web 控制台或 OpenShift CLI (**oc**)安装 PTP。PTP Operator 会创建和管理 **linuxptp** 服务，并提供以下功能：

- 在集群中发现具有 PTP 功能的设备。
- 管理 **linuxptp** 服务的配置。
- PTP 时钟事件通知会使用 PTP Operator **cloud-event-proxy** sidecar 会对应用程序的性能和可靠性造成负面影响。

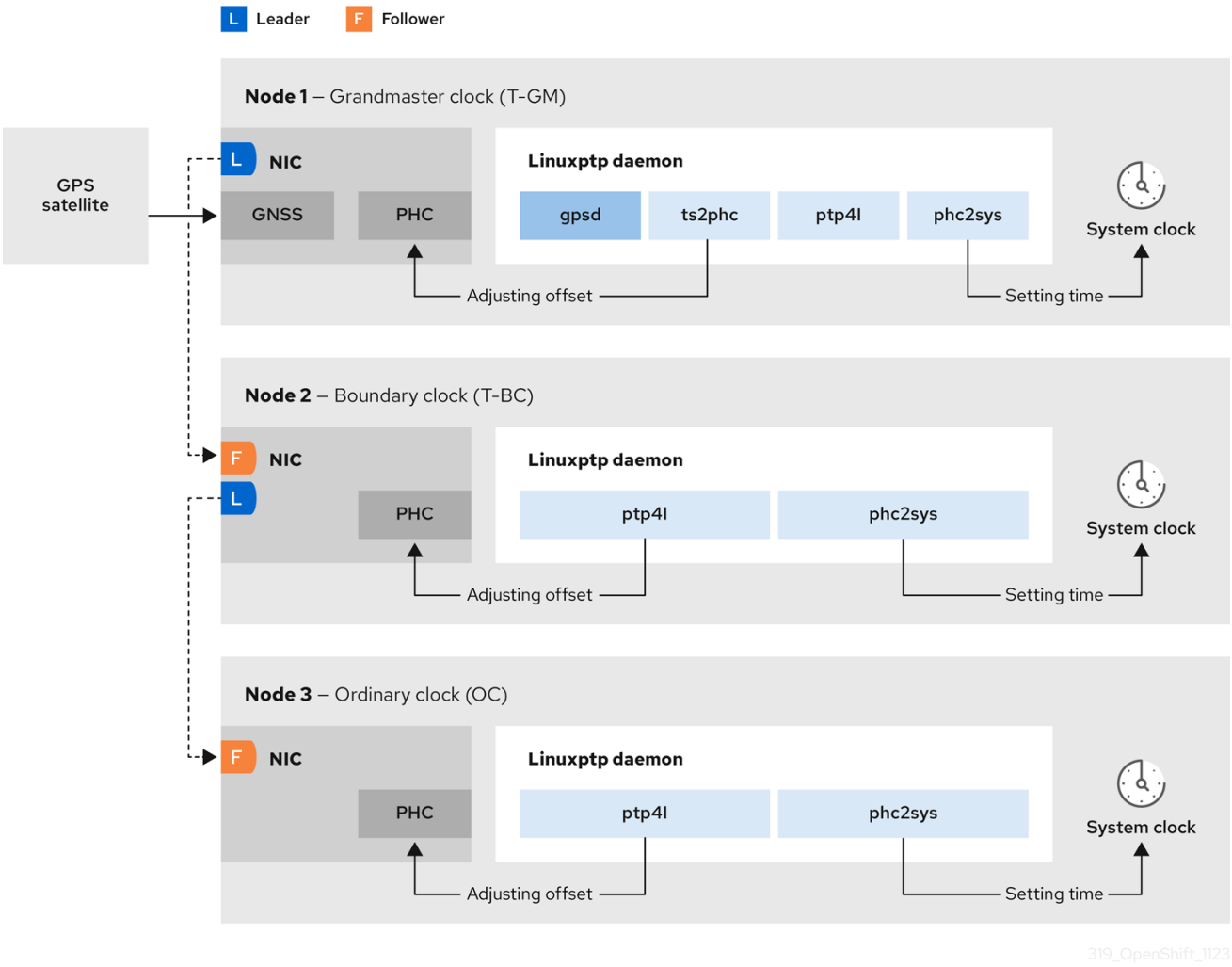
注意

PTP Operator 只适用于仅在裸机基础架构上置备的集群上具有 PTP 功能的设备。

5.1.1. PTP 域的元素

PTP 用于将网络中连接的多个节点与每个节点的时钟同步。PTP 同步时钟以领导层次结构进行组织。层次结构由最佳 master 时钟 (BMC) 算法自动创建和更新，该算法在每个时钟上运行。后续时钟与领导时钟同步，后续时钟本身可以是其他下游时钟的来源。

图 5.1. 网络中的 PTP 节点



下面描述了三种 PTP 时钟类型。

Grandmaster 时钟

grandmaster 时钟向网络上的其他时钟提供标准时间信息并确保准确和稳定的同步。它写入时间戳并响应来自其他时钟的时间间隔。grandmaster 时钟与全局导航 Satellite 系统 (GNSS) 时间源同步。Grandmaster 时钟是网络中权威时间来源，负责为所有其他设备提供时间同步。

Boundary 时钟

Boundary（边界）时钟在两个或更多个通信路径中具有端口，并且可以是指向其他目标时钟的源和目标。边界时钟作为上游目标时钟工作。目标时钟接收计时消息，针对延迟进行调整，然后创建一个新的源时间信号来传递网络。边界时钟生成一个新的计时数据包，它仍然与源时钟正确同步，并可减少直接报告到源时钟的连接设备数量。

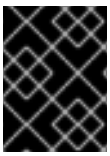
Ordinary 时钟

Ordinary（普通）时钟具有一个端口连接，可根据其在网络中的位置扮演源或目标时钟的角色。普通时钟可以读取和写入时间戳。

5.1.1.1. PTP 优于 NTP 的优点

PTP 与 NTP 相比有一个主要优势，即各种网络接口控制器 (NIC) 和网络交换机中存在的硬件支持。特殊硬件允许 PTP 考虑消息传输的延迟，并提高时间同步的准确性。为了获得最佳准确性，建议启用 PTP 时钟间的所有网络组件。

基于硬件的 PTP 提供最佳准确性，因为 NIC 可以在准确发送和接收时对 PTP 数据包进行时间戳。这与基于软件的 PTP 进行比较，这需要操作系统对 PTP 数据包进行额外的处理。



重要

在启用 PTP 前，请确保为所需节点禁用 NTP。您可以使用 **MachineConfig** 自定义资源禁用 chrony 时间服务 (**chronyd**)。如需更多信息，请参阅禁用 chrony 时间服务。

5.1.2. OpenShift Container Platform 节点中的 linuxptp 和 gpsd 概述

OpenShift Container Platform 使用带有 **linuxptp** 和 **gpsd** 软件包的 PTP Operator 进行高精度网络同步。**linuxptp** 软件包为网络中的 PTP 时间提供工具和守护进程。带有 Global Navigation Satellite System (GNSS)功能 NIC 的集群主机使用 **gpsd** 来与 GNSS 时钟源进行接口。

linuxptp 软件包包括用于系统时钟同步的 **ts2phc**、**pmc**、**ptp4l** 和 **phc2sys** 程序。

ts2phc

ts2phc 将 PTP 设备中的 PTP 硬件时钟(PHC)与高度精确度同步。 **ts2phc** 用于 grandmaster 时钟配置。它收到精度计时信号，这是一个高度精确时钟源，如 Global Navigation Satellite System (GNSS)。GNSS 提供准确可靠的同步时间源，用于大型分布式网络。GNSS 时钟通常提供时间信息，其精度为几个纳秒。
ts2phc 系统守护进程通过读取 grandmaster 时钟中的时间信息，将时间信息从 grandmaster 时钟发送到网络中的其他 PTP 设备，并将其转换为 PHC 格式。PHC 时间供网络中的其他设备用来将其时钟与 grandmaster 时钟同步。

pmc

pmc 根据 IEEE 标准 1588.1588 实现 PTP 管理客户端 (**pmc**)。**pmc** 为 **ptp4l** 系统守护进程提供基本的管理访问权限。**pmc** 从标准输入读取，并通过所选传输发送输出，打印它收到的任何回复。

ptp4l

ptp4l 实现 PTP 边界时钟和普通时钟，并作为系统守护进程运行。 **ptp4l** 执行以下操作：

- 将 PHC 同步到源时钟与硬件时间戳
- 将系统时钟与源时钟与软件时间戳同步

phc2sys

phc2sys 将系统时钟与网络接口控制器 (NIC) 上的 PHC 同步。 **phc2sys** 系统守护进程持续监控 PHC 以获取计时信息。当检测到计时错误时，LareC 会更正系统时钟。

gpsd 软件包包括 **ubxtool**、**gspipe**、**gpsd**、GNSS 时钟与主机时钟同步的程序。

ubxtool

ubxtool CLI 可让您与 u-blox GPS 系统通信。 **ubxtool** CLI 使用 u-blox 二进制协议与 GPS 通信。

gpspipe

gpspipe 连接到 **gpsd** 输出并将其传送到 **stdout**。

gpsd

gpsd 是一个服务守护进程，它监控一个或多个连接到主机的 GPS 或 AIS 接收器。

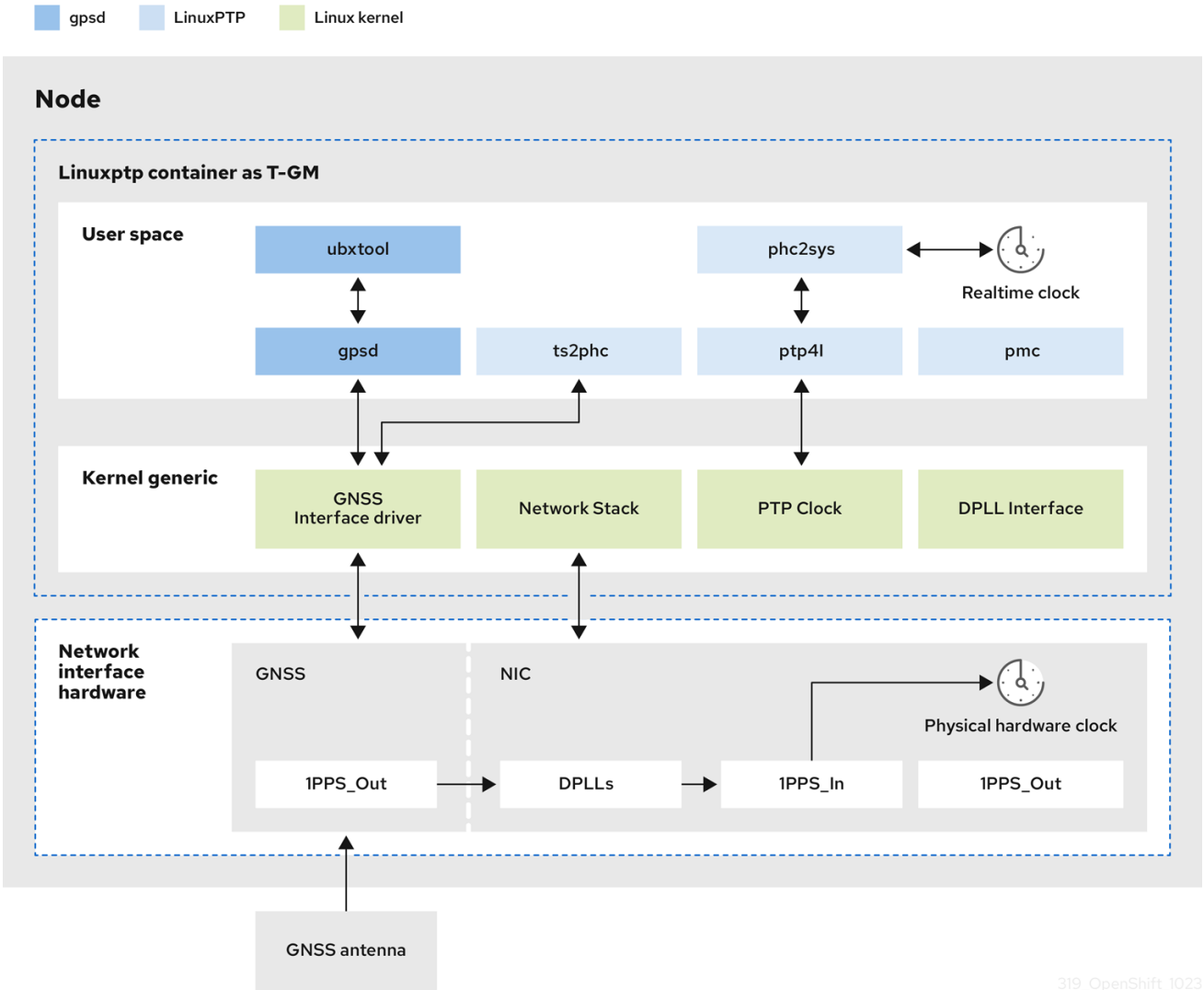
5.1.3. PTP grandmaster 时钟的 GNSS 时间概述

OpenShift Container Platform 支持从集群中的 Global Navigation Satellite 系统(GNSS)源和 grandmaster 时钟(T-GM)接收精度 PTP 时间。

重要

OpenShift Container Platform 仅支持 Intel E810 Westport Channel NIC 的 GNSS 源中的 PTP 时间。

图 5.2. 使用 GNSS 和 T-GM 同步概述



全局导航 Satellite 系统(GNSS)

GNSS 是一个基于 satellite 的系统，用来为全球范围内接收器提供定位、导航和计时信息。在 PTP 中，GNSS 接收器通常用作高度准确且稳定的参考时钟源。这些接收器从多个 GNSS satellites 接收信号，允许它们计算精确的时间信息。从 GNSS 获取的时间信息被 PTP grandmaster 时钟参考。通过将 GNSS 用作参考，PTP 网络中的 grandmaster 时钟可以为其他设备提供高度准确的时间戳，从而在整个网络中启用精确同步。

Digital Phase-Locked Loop (DPLL)

DPLL 在网络中的不同 PTP 节点之间提供时钟同步。DPLL 将本地系统时钟信号的阶段与传入同步信号的阶段进行比较，例如，来自 PTP grandmaster 时钟的 PTP 信息。DPLL 持续调整本地时钟频率和阶段，以最大程度降低本地时钟和参考时钟之间的阶段差异。

5.1.3.1. 在 GNSS-synced PTP grandmaster 时钟中处理 leap 秒事件

Leap second（闰秒）是一秒的调整，偶尔会被应用于 Coordinated Universal Time (UTC)，使其与国际原子时间 (TAI) 同步。UTC 秒是无法预计的。在国际范围内认可的闰秒信息包括在 leap-seconds.list 中。此文件通过国际 Earth Rotation 和 Reference Systems Service (IERS)定期更新。一个未处理的闰秒对边缘 RAN 网络有严重影响。可能会导致边缘 RAN 应用程序立即断开语音调用和数据会话。

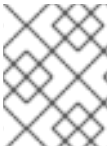
5.1.4. 关于 PTP 和时钟同步错误事件

虚拟 RAN (vRAN) 等云原生应用需要访问对整个网络运行至关重要的硬件计时事件通知。PTP 时钟同步错误可能会对低延迟应用程序的性能和可靠性造成负面影响，例如：在一个分布式单元 (DU) 中运行的 vRAN 应用程序。

丢失 PTP 同步是 RAN 网络的一个关键错误。如果在节点上丢失同步，则可能会关闭无线广播，并且网络 Over the Air (OTA) 流量可能会转移到无线网络中的另一个节点。快速事件通知允许集群节点与 DU 中运行的 vRAN 应用程序通信 PTP 时钟同步状态，从而缓解工作负载错误。

事件通知可用于在同一 DU 节点上运行的 vRAN 应用。发布/订阅 REST API 将事件通知传递到消息传递总线。发布-订阅消息传递或发布-订阅消息传递是服务通信架构的异步服务，通过服务通信架构，所有订阅者会立即收到发布到某一主题的消息。

PTP Operator 为每个支持 PTP 的网络接口生成快速事件通知。您可以通过 HTTP 消息总线使用 **cloud-event-proxy** sidecar 容器来访问事件。



注意

PTP 快速事件通知可用于配置为使用 PTP 普通时钟、PTP grandmaster 时钟或 PTP 边界时钟。

5.1.5. 2-card E810 NIC 配置参考

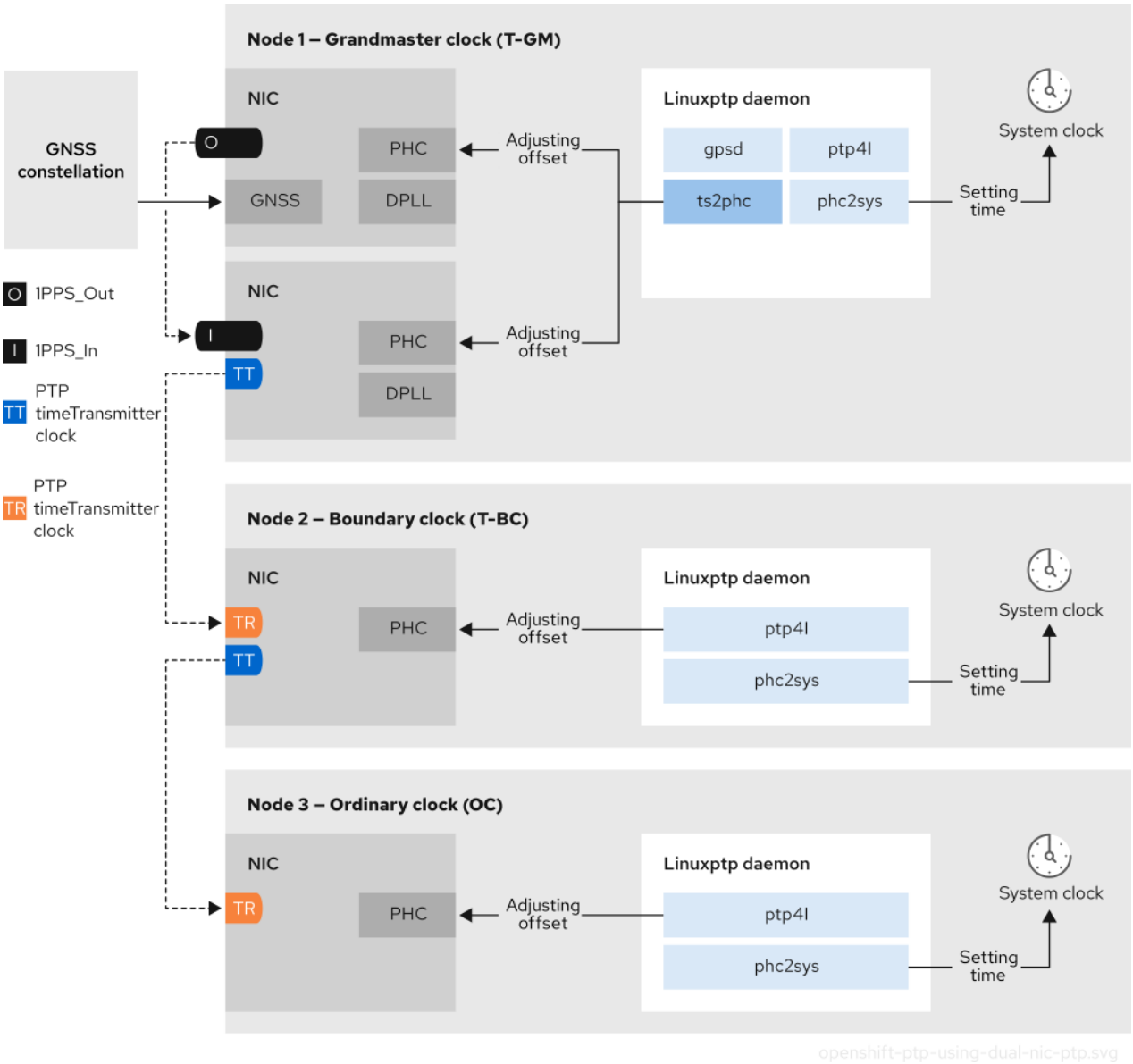
OpenShift Container Platform 支持用于 grandmaster 时钟(T-GM) 和边界时钟(T-BC) 的 PTP 时间的单和双 NIC Intel E810 硬件。

双 NIC grandmaster 时钟

您可以使用具有双 NIC 硬件作为 PTP grandmaster 时钟的集群主机。一个 NIC 从全局导航 Satellite 系统(GNSS)接收计时信息。第二个 NIC 在第一次使用 E810 NIC 上的 SMA1 Tx/Rx 连接接收时间信息。集群主机上的系统时钟从连接到 GNSS satellite 的 NIC 同步。

双 NIC grandmaster 时钟是分布式 RAN (D-RAN)配置的功能，其中远程 Radio 单元(RRU)和 Baseband 单元(BBU)位于相同的无线单元站点。d-RAN 在多个站点间分发无线功能，带有将它们链接到核心网络的连接。

图 5.3. 双 NIC grandmaster 时钟



注意

在双 NIC T-GM 配置中，单个 **ts2phc** 程序在两个 PTP 硬件时钟 (PHC)上运行，每个 NIC 对应一个。

双 NIC 边界时钟

对于提供中等范围的 5G 电信网络，每个虚拟分布式单元(vDU)需要连接到 6 个无线电单元(RU)。要使这些连接，每个 vDU 主机都需要 2 个 NIC 被配置为边界时钟。

双 NIC 硬件允许您将每个 NIC 连接到相同的上游领导时钟，并将每个 NIC 的 **ptp4l** 实例连接给下游时钟。

带有双 NIC 边界时钟的高可用性系统时钟

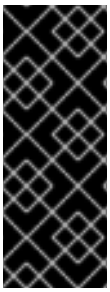
您可以将 Intel E810-XXVDA4 Salem 频道双 NIC 硬件配置为双 PTP 边界时钟，为高可用性系统时钟提供计时。如果您在不同的 NIC 上有多个时间源，此配置很有用。高可用性可确保如果两个计时源丢失或断开连接，则节点不会丢失计时同步。

每个 NIC 都连接到同一上游领导时钟。高可用性边界时钟使用多个 PTP 域与目标系统时钟同步。当 T-BC 高度可用时，主机系统时钟可以维护正确的偏移，即使一个或多个 **ptp4l** 实例同步 NIC PHC 时钟失败。如果发生任何单一 SFP 端口或电缆失败，则边界时钟会与领导时钟保持同步。

边界时钟领导源选择使用 A-BMCA 算法完成。如需更多信息，请参阅 ITU-T 建议 G.8275.1。

5.1.6. 使用双端口 NIC 来提高 PTP 普通时钟的冗余

OpenShift Container Platform 支持单和双端口网络接口卡(NIC)作为 PTP 时间的普通时钟。要提高冗余，您可以使用一个端口作为活跃端口配置双端口 NIC，另一个端口作为待机。



重要

将 linuxptp 服务配置为带有双端口 NIC 冗余的普通时钟只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

在这个配置中，双端口 NIC 中的端口按如下方式操作：

- 活动端口作为常规时钟在 **Following** 端口状态下运行。
- 待机端口保持在 **Listening** 端口状态。
- 如果活跃端口失败，待机端口转换为 active，以确保持续 PTP 时间同步。
- 如果两个端口都出现故障，时钟状态将移至 **HOLDOVER** 状态，然后在保存超时过期时 **FREERUN** 状态，然后再重新同步到领导时钟。

注意

您只能在带有双端口 NIC 的 **x86** 架构节点上配置 PTP 普通时钟。

5.1.7. 3-card Intel E810 PTP grandmaster 时钟

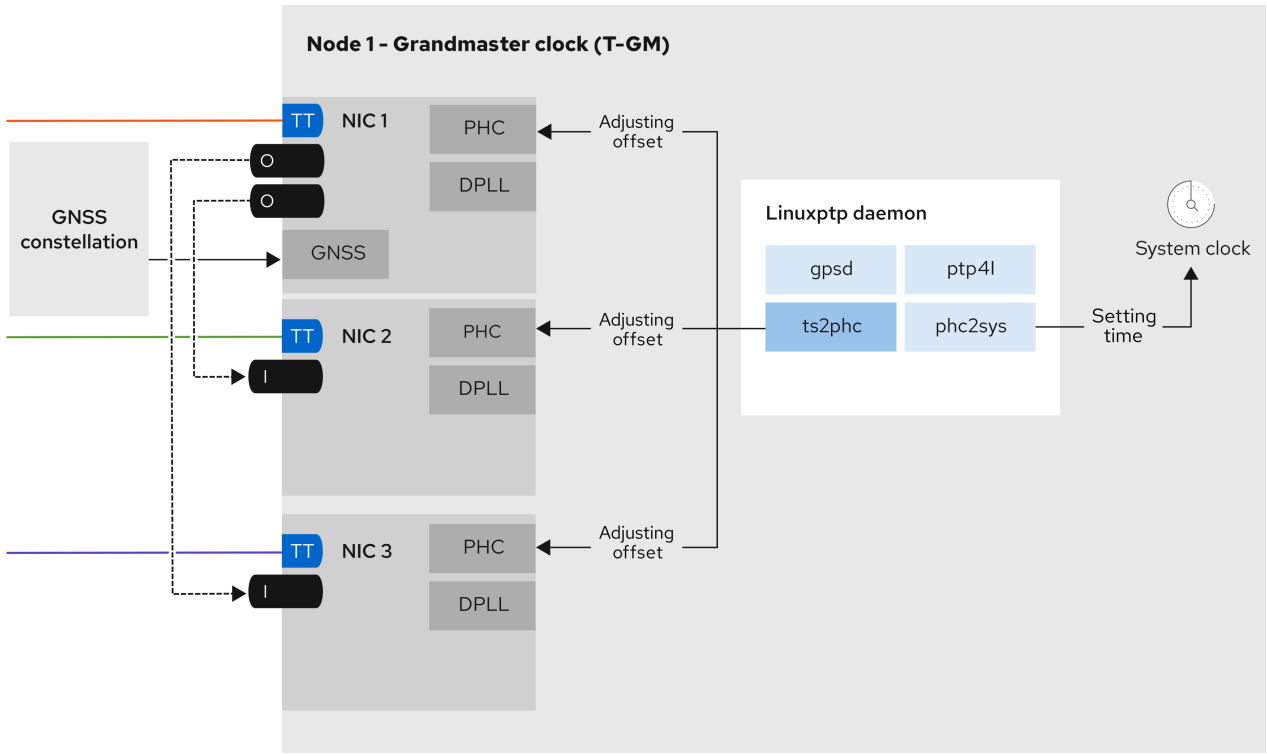
OpenShift Container Platform 支持使用 3 个 Intel E810 NIC 作为 PTP grandmaster 时钟 (T-GM) 的集群主机。

3-card grandmaster 时钟

您可以使用具有 3 个 NIC 作为 PTP grandmaster 时钟的集群主机。一个 NIC 从全局导航 Satellite 系统(GNSS)接收计时信息。第二个和第三个 NIC 使用 E810 NIC 上的 SMA1 Tx/Rx 连接从第一个接收时间信息。集群主机上的系统时钟从连接到 GNSS satellite 的 NIC 同步。

3card NIC grandmaster 时钟可用于分布式 RAN (D-RAN)配置，其中 Radio 单元(RU)直接连接到分布式单元(DU)，例如，如果 RU 和 DU 位于相同的无线单元站点。d-RAN 在多个站点间分发无线功能，带有将它们链接到核心网络的连接。

图 5.4. 3-card Intel E810 PTP grandmaster 时钟



注意

在 3-card T-GM 配置中，单个 **ts2phc** 进程报告为系统中的 3 个 **ts2phc** 实例。

5.2. 配置 PTP 设备

PTP Operator 将 **NodePtpDevice.ptp.openshift.io** 自定义资源定义（CRD）添加到 OpenShift Container Platform。

安装后，PTP Operator 会在每个节点中搜索具有 PTP 功能的网络设备。Operator 为提供兼容 PTP 的网络设备的每个节点创建并更新 **NodePtpDevice** 自定义资源(CR)对象。

带有内置 PTP 功能的网络接口控制器(NIC)硬件有时需要特定于设备的配置。您可以通过在 **PtpConfig** 自定义资源(CR)中配置插件，将特定于硬件的 NIC 功能用于 PTP Operator 支持的硬件。**linuxptp-daemon** 服务使用 **plugin** 小节中的指定参数根据特定的硬件配置启动 **linuxptp** 进程(**ptp4l** 和 **phc2sys**)。

重要

在 OpenShift Container Platform 4.18 中，通过 **PtpConfig** 插件支持 Intel E810 NIC。

5.2.1. 使用 CLI 安装 PTP Operator

作为集群管理员，您可以使用 CLI 安装 Operator。

先决条件

- 在裸机中安装有支持 PTP 硬件的节点的集群。
- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 为 PTP Operator 创建命名空间。
   a. 将以下 YAML 保存到 **ptp-namespace.yaml** 文件中：

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

   b. 创建 **Namespace** CR：

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

2. 为 PTP Operator 创建 Operator 组。
   a. 在 **ptp-operatorgroup.yaml** 文件中保存以下 YAML：

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

   b. 创建 **OperatorGroup** CR：

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

3. 订阅 PTP Operator。
   a. 将以下 YAML 保存到 **ptp-sub.yaml** 文件中：

```
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. 创建 **Subscription** CR：

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

4. 要验证是否已安装 Operator，请输入以下命令：

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

输出示例

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

5.2.2. 使用 Web 控制台安装 PTP Operator

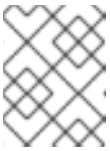
作为集群管理员，您可以使用 Web 控制台安装 PTP Operator。

注意

如上一节所述，您必须创建命名空间和 operator 组。

流程

1. 使用 OpenShift Container Platform Web 控制台安装 PTP Operator：
   a. 在 OpenShift Container Platform Web 控制台中，点击 **Operators → OperatorHub**。
   b. 从可用的 Operator 列表中选择 **PTP Operator**，然后点 **Install**。
   c. 在 **Install Operator** 页面中，在 **A specific namespace on the cluster**下选择 **openshift-ptp**。然后点击 **Install**。
2. 可选：验证是否成功安装了 PTP Operator：
   a. 切换到 **Operators → Installed Operators** 页面。
   b. 确保 **openshift-ptp** 项目中列出的 **PTP Operator** 的 **Status** 为 **InstallSucceeded**。



注意

在安装过程中，Operator 可能会显示 **Failed** 状态。如果安装过程结束后有 **InstallSucceeded** 信息，您可以忽略这个 **Failed** 信息。

如果 Operator 没有被成功安装，请按照以下步骤进行故障排除：

- 进入 **Operators → Installed Operators** 页面，检查 **Operator Subscriptions** 和 **Install Plans** 选项卡中的 **Status** 项中是否有任何错误。
- 进入 **Workloads → Pods** 页面，检查 **openshift-ptp** 项目中 pod 的日志。

5.2.3. 在集群中发现支持 PTP 的网络设备

识别集群中存在的 PTP 功能网络设备，以便您可以配置它们

先决条件

- 已安装 PTP Operator。

流程

- 要返回集群中具有 PTP 功能网络设备的完整列表，请运行以下命令：

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

输出示例

```
apiVersion: v1
items:
- apiVersion: ptp.openshift.io/v1
  kind: NodePtpDevice
  metadata:
    creationTimestamp: "2022-01-27T15:16:28Z"
    generation: 1
    name: dev-worker-0 (1)
    namespace: openshift-ptp
    resourceVersion: "6538103"
    uid: d42fc9ad-bcbf-4590-b6d8-b676c642781a
  spec: {}
  status:
    devices: (2)
    - name: eno1
    - name: eno2
    - name: eno3
    - name: eno4
    - name: enp5s0f0
    - name: enp5s0f1
...
```

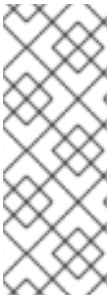
(1) **name** 参数的值与父节点的名称相同。

(2) **devices** 集合包含 PTP Operator 发现节点的 PTP 功能设备列表。

5.2.4. 将 linuxptp 服务配置为 grandmaster 时钟

您可以通过创建一个配置主机 NIC 的 **PtpConfig** 自定义资源(CR)将 **linuxptp** 服务(**ptp4l**、**phc2sys**、**ts2phc**)配置为 grandmaster 时钟(T-GM)。

ts2phc 工具允许您将系统时钟与 PTP grandmaster 时钟同步，以便节点可以将精度时钟信号流传输到下游 PTP 普通时钟和边界时钟。



注意

使用 **PtpConfig** CR 示例，将 **linuxptp** 服务配置为 Intel Westport Channel E810-XXVDA4T 网络接口的 T-GM。

要配置 PTP 快速事件，请为 **ptp4lOpts**、**ptp4lConf** 和 **ptpClockThreshold** 设置适当的值。**ptpClockThreshold** 仅在启用事件时使用。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- 对于生产环境中的 T-GM 时钟，请在裸机集群主机上安装 Intel E810 Westport Channel NIC。
- 安装 OpenShift CLI (**oc**) 。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建 **PtpConfig** CR。例如：
   a. 根据您的要求，为您的部署使用以下 T-GM 配置之一。将 YAML 保存到 **grandmaster-clock-ptp-config.yaml** 文件中：

例 5.1. E810 NIC 的 PTP grandmaster 时钟配置

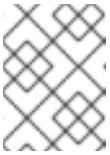
```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - name: "grandmaster"
      ptp4lOpts: "-2 --summary_interval -4"
      phc2sysOpts: -r -u 0 -m -N 8 -R 16 -s $iface_master -n 24
      ptpSchedulingPolicy: 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"
```



注意

对于 E810 Westport Channel NIC，将 **ts2phc.nmea_serialport** 的值设置为 **/dev/gnss0**。

b. 运行以下命令来创建 CR：

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

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

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

   输出示例

   ```
   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. 检查配置集是否正确。检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志。运行以下命令:

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

输出示例

```
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
```

5.2.4.1. 将 linuxptp 服务配置为双 E810 NIC 的 grandmaster 时钟

您可以通过创建一个配置 NIC 的 **PtpConfig** 自定义资源(T-GM)，将 **linuxptp** 服务(**ptp4l**、**phc2sys**、**ts2phc**) 配置为 2 E810 NIC。

您可以将 **linuxptp** 服务配置为以下 E810 NIC 的 T-GM：

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

对于分布式 RAN (D-RAN)用例，您可以为 2 个 NIC 配置 PTP，如下所示：

- NIC 1 与全局导航 satellite 系统(GNSS)时间源同步。
- NIC 2 将同步到 NIC 1 提供的 1PPS 时间输出。此配置由 **PtpConfig** CR 中的 PTP 硬件插件提供。

2-card PTP T-GM 配置使用 **ptp4l** 实例，以及一个 **ts2phc** 实例。**ptp4l** 和 **ts2phc** 程序都配置为在两个 PTP 硬件时钟(PHC)上运行，每个 NIC 对应一个。主机系统时钟与连接到 GNSS 时间源的 NIC 同步。

注意

使用以下的 **PtpConfig** CR 示例作为基础，为双 Intel E810 网络接口将 **linuxptp** 服务配置为 T-GM。

要配置 PTP 快速事件，请为 **ptp4lOpts**、**ptp4lConf** 和 **ptpClockThreshold** 设置适当的值。**ptpClockThreshold** 仅在启用事件时使用。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- 对于生产环境中的 T-GM 时钟，请在裸机集群主机上安装两个 Intel E810 NIC。
- 安装 OpenShift CLI (**oc**) 。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建 **PtpConfig** CR。例如：

   a. 将以下 YAML 保存到 **grandmaster-clock-ptp-config-dual-nics.yaml** 文件中：

例 5.2. 用于双 E810 NIC 的 PTP grandmaster 时钟配置

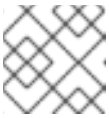
```
# In this example two cards $iface_nic1 and $iface_nic2 are connected via
# SMA1 ports by a cable and $iface_nic2 receives 1PPS signals from $iface_nic1
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: grandmaster
  namespace: openshift-ptp
  annotations: {}
spec:
  profile:
    - 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"
```



注意

将 **ts2phc.nmea_serialport** 的值设置为 **/dev/gnss0**。

b. 运行以下命令来创建 CR：

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

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

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

   输出示例

   ```
   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. 检查配置集是否正确。检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志。运行以下命令:

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

   输出示例

```
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
```

5.2.4.2. 将 linuxptp 服务配置为 3 E810 NIC 的 grandmaster 时钟

您可以通过创建一个配置 NIC 的 **PtpConfig** 自定义资源(T-GM)，将 **linuxptp** 服务 (**ptp4l**、**phc2sys**、**ts2phc**) 配置为 3 E810 NIC。

您可以为以下 E810 NIC 将 **linuxptp** 服务配置为带有 3 个 NIC 的 T-GM：

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

对于分布式 RAN (D-RAN)用例，您可以为 3 个 NIC 配置 PTP，如下所示：

- NIC 1 同步到全局导航 Satellite 系统(GNSS)
- NIC 2 和 3 与带有 1PPS 的 NIC 1 同步连接

使用 **PtpConfig** CR 示例，将 **linuxptp** 服务配置为 3card Intel E810 T-GM。

先决条件

- 对于生产环境中的 T-GM 时钟，在裸机集群主机上安装 3 个 Intel E810 NIC。
- 安装 OpenShift CLI (**oc**) 。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建 **PtpConfig** CR。例如：
   a. 将以下 YAML 保存到 **three-nic-grandmaster-clock-ptp-config.yaml** 文件中：

例 5.3. 3 E810 NIC 的 PTP grandmaster 时钟配置

```
# In this example, the three cards are connected via SMA cables:
```

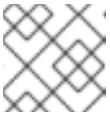
```
# - $iface_timeTx1 has the GNSS signal input
# - $iface_timeTx2 SMA1 is connected to $iface_timeTx1 SMA1
# - $iface_timeTx3 SMA1 is connected to $iface_timeTx1 SMA2
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: gm-3card
  namespace: openshift-ptp
  annotations:
    ran.openshift.io/ztp-deploy-wave: "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
```



注意

将 **ts2phc.nmea_serialport** 的值设置为 **/dev/gnss0**。

b. 运行以下命令来创建 CR：

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

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

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

   输出示例

   ```
   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. 检查配置集是否正确。运行以下命令，并检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志：

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

   输出示例

   ```
   ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
   /dev/ptp11 (1)
   ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
   /dev/ptp7 (2)
   ts2phc[2527.586]: [ts2phc.0.config:7] adding tstamp 1742826342.000000000 to clock
   /dev/ptp14 (3)
   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 (4)
   ts2phc[2527.587]: [ts2phc.0.config:6] /dev/ptp7 offset          0 s2 freq      +0 (5)
   ts2phc[2527.587]: [ts2phc.0.config:6] /dev/ptp11 offset         0 s2 freq      -0 (6)
   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 (7)
   ```

   (1) (2) (3) **ts2phc** 更新 PTP 硬件时钟。

   (4) (5) (6) PTP 设备和参考时钟之间预计 PTP 设备偏移是 0 nanoseconds。PTP 设备与领导时钟同步。

   (7) T-GM 处于锁定状态 (s2)。

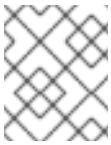
其他资源

- 配置 PTP 快速事件通知发布程序

5.2.5. grandmaster clock PtpConfig 配置参考

以下参考信息描述了 **PtpConfig** 自定义资源(CR)的配置选项，将 **linuxptp** 服务(**ptp4l**、**phc2sys**、**ts2phc**)配置为 grandmaster 时钟。

表 5.1. PTP Grandmaster 时钟的 PtpConfig 配置选项

PtpConfig CR 字段	描述
plugins	指定一组 **.exec.cmdline** 选项来为 grandmaster 时钟操作配置 NIC。grandmaster 时钟配置需要禁用某些 PTP pin。 插件机制允许 PTP Operator 进行自动硬件配置。对于 Intel Westport Channel NIC 或 Intel Logan Beach NIC，当 **enableDefaultConfig** 字段被设置为 **true** 时，PTP Operator 会运行一个硬编码的脚本来为 NIC 执行所需的配置。
ptp4lOpts	为 **ptp4l** 服务指定系统配置选项。该选项不应包含网络接口名称 **-i <interface>** 和服务配置文件 **-f /etc/ptp4l.conf**，因为网络接口名称和服务配置文件会被自动附加。
ptp4lConf	指定启动 **ptp4l** 作为 grandmaster 时钟所需的配置。例如，**ens2f1** 接口同步下游连接的设备。对于 grandmaster 时钟，将 **clockClass** 设置为 **6**，并将 **clockAccuracy** 设置为 **0x27**。将 **timeSource** 设置为 **0x20**，以便在从全局导航 Satellite 系统 (GNSS) 接收计时信号时。
tx_timestamp_timeout	指定丢弃数据前从发送方等待传输 (TX) 时间戳的最长时间。
boundary_clock_jbod	指定 JBOD 边界时钟时间延迟值。这个值用于更正网络时间设备之间传递的时间值。
phc2sysOpts	为 **phc2sys** 服务指定系统配置选项。如果此字段为空，PTP Operator 不会启动 **phc2sys** 服务。  **注意** 确保此处列出的网络接口配置为 grandmaster，并在 **ts2phcConf** 和 **ptp4lConf** 字段中根据需要引用。
ptpSchedulingPolicy	为 **ptp4l** 和 **phc2sys** 进程配置调度策略。默认值为 **SCHED_OTHER**。在支持 FIFO 调度的系统上使用 **SCHED_FIFO**。
ptpSchedulingPriority	当 **ptpSchedulingPolicy** 设置为 **SCHED_FIFO** 时，设置 1-65 的整数值来为 **ptp4l** 和 **phc2sys** 进程配置 FIFO 优先级。当 **ptpSchedulingPolicy** 设置为 **SCHED_OTHER** 时，不使用 **ptpSchedulingPriority** 字段。

PtpConfig CR 字段	描述
ptpClockThreshold	可选。如果 **ptpClockThreshold** 小节不存在，则使用 **ptpClockThreshold** 字段的默认值。小节显示默认的 **ptpClockThreshold** 值。**ptpClockThreshold** 值配置 PTP master 时钟在触发 PTP 事件前的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。
ts2phcConf	设置 **ts2phc** 命令的配置。 **leapfile** 是 PTP Operator 容器镜像中当前 leap 秒定义文件的默认路径。 **ts2phc.nmea_serialport** 是连接到 NMEA GPS 时钟源的串行端口设备。配置后，GNSS 接收器可在 **/dev/gnss<id>** 上访问。如果主机有多个 GNSS 接收器，您可以通过枚举以下设备之一来查找正确的设备： ● **/sys/class/net/<eth_port>/device/gnss/** ● **/sys/class/gnss/gnss<id>/device/**
ts2phcOpts	为 **ts2phc** 命令设置选项。
建议	指定包括一个或多个 **recommend** 对象的数组，该数组定义了如何将**配置集**应用到节点的规则。
.recommend.profile	指定在 **profile** 部分中定义的 **.recommend.profile** 对象名称。
.recommend.priority	使用 **0** 到 **99** 之间的一个整数值指定 **priority**。大数值的优先级较低，因此优先级 **99** 低于优先级 **10**。如果节点可以根据 **match** 字段中定义的规则与多个配置集匹配，则优先级较高的配置集会应用到该节点。
.recommend.match	使用 **nodeLabel** 或 **nodeName** 值指定 **.recommend.match** 规则。
.recommend.match.nodeLabel	通过 **oc get nodes --show-labels** 命令，使用来自节点对象的 **node.Labels** 的 **key** 设置 **nodeLabel**。例如，**node-role.kubernetes.io/worker**。
.recommend.match.nodeName	使用 **oc get nodes** 命令，将 **nodeName** 设置为来自节点对象的 **node.Name** 值。例如，**compute-1.example.com**。

5.2.5.1. grandmaster 时钟类同步状态参考

下表描述了 PTP grandmaster 时钟(T-GM) **gm.ClockClass** 状态。时钟类状态根据其准确性和稳定性根据主要参考时间时钟(PRTC)或其他计时来源对 T-GM 时钟进行分类。

holdover 规格是 PTP 时钟可以维护同步的时间，而无需从主时间源接收更新。

表 5.2. T-GM 时钟类状态

时钟类状态	描述
gm.ClockClass 6	T-GM 时钟在 **LOCKED** 模式中连接到 PRTC。例如，PRTC 可以追溯到 GNSS 时间源。
gm.ClockClass 7	T-GM 时钟处于 **HOLDOVER** 模式，在既存的规格内。时钟源可能无法追溯到类别 1 频率源。
gm.ClockClass 248	T-GM 时钟处于 **FREERUN** 模式。

如需更多信息，请参阅 "Phase/time traceability information", ITU-T G.8275.1/Y.1369.1 Recommendations.

5.2.5.2. Intel E810 NIC 硬件配置参考

使用此信息了解如何使用 Intel E810 硬件插件将 E810 网络接口配置为 PTP grandmaster 时钟。硬件固定配置决定了网络接口如何与系统中的其他组件和设备进行交互。对于外部 1PPS 信号，Intel E810 NIC 有四个连接器： **SMA1**、**SMA** 2、**U.FL1** 和 **U.FL2**。

表 5.3. Intel E810 NIC 硬件连接器配置

硬件固定	推荐的设置	描述
U.FL1	**0 1**	禁用 **U.FL1** 连接器输入。**U.FL1** 连接器是仅用于输出的。
U.FL2	**0 2**	禁用 **U.FL2** 连接器输出。**U.FL2** 连接器是仅限输入的。
SMA1	**0 1**	禁用 **SMA1** 连接器输入。**SMA1** 连接器是双向的。
SMA2	**0 2**	禁用 **SMA2** 连接器输出。**SMA2** 连接器是双向的。

您可以使用 **spec.profile.plugins.e810.pins** 参数在 Intel E810 NIC 上设置 pin 配置，如下例所示：

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

其中：

<function>: 指定 pin 的角色。以下值与 pin 角色关联：

- **0**: 禁用
- **1**: Rx（接收时间戳）

- **2**: Tx（传输时间戳）

<channel number> : 与物理连接器关联的数字。以下频道号与物理连接器关联：

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

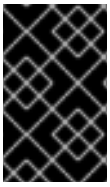
示例：

- **0 1**: 禁用 pin 映射到 **SMA1** 或 **U.FL1**。
- **1 2**: 将 Rx 功能分配给 **SMA2** 或 **U.FL2**。

注意

SMA1 和 **U.FL1** 连接器共享通道。**SMA2** 和 **U.FL2** 连接器共享通道二。

设置 **spec.profile.plugins.e810.ublxCmds** 参数，以在 **PtpConfig** 自定义资源(CR) 中配置 GNSS 时钟。



重要

您必须配置偏移值来补偿 T-GM GPS atenna 电缆信号延迟。要配置最佳 T-GM antenna offset 值，请对 GNSS atenna 电缆信号延迟进行精确测量。红帽无法协助进行这个测量，或为所需的延迟偏移提供任何值。

这些 **ublxCmds** 小节各自对应于使用 **ubxtool** 命令应用到主机 NIC 的配置。例如：

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

1 以纳秒为单位测量 T-GM 延迟偏移。要获得所需的延迟偏移值，您必须使用外部测试设备测量电缆延迟。

下表描述了等效的 **ubxtool** 命令：

表 5.4. Intel E810 ublxCmds 配置

ubxtool 命令	描述
ubxtool -P 29.20 -z CFG-HW-ANT_CFG_VOLTCTRL,1 -z CFG-TP-ANT_CABLEDELAY,<antenna_delay_offset>	启用一个tenna voltage 控制，允许在 **UBX-MON-RF** 和 **UBX-INF-NOTICE** 日志消息中报告tenna 状态，并设置一个 **<antenna_delay_offset>** 值，以纳秒表示偏移 GPS atenna 电缆信号延迟。

ubxtool 命令	描述
ubxtool -P 29.20 -e GPS	启用 atenna 接收 GPS 信号。
ubxtool -P 29.20 -d Galileo	配置 atenna 以接收来自 Galileo GPS satellite 的信号。
ubxtool -P 29.20 -d GLONASS	禁用 atenna 从 GLONASS GPS satellite 接收信号。
ubxtool -P 29.20 -d BeiDou	禁用 atenna 从 BeiDou GPS satellite 接收信号。
ubxtool -P 29.20 -d SBAS	禁用 atenna 从 SBAS GPS satellite 接收信号。
ubxtool -P 29.20 -t -w 5 -v 1 -e SURVEYIN,600,50000	配置 GNSS 接收器调查进程，以提高其初始位置估算。这可能需要 24 小时才能获得最佳结果。
ubxtool -P 29.20 -p MON-HW	对硬件运行单个自动扫描，并报告 NIC 状态和配置设置。

5.2.5.3. 双 E810 NIC 配置参考

使用这些信息了解如何使用 Intel E810 硬件插件将 E810 网络接口配置为 PTP grandmaster 时钟 (T-GM)。

在配置双 NIC 集群主机前，您必须使用 1PPS faceplace 连接将两个 NIC 与 SMA1 电缆连接。

当您配置双 NIC T-GM 时，您需要补补使用 SMA1 连接端口连接 NIC 时发生的 1PPS 信号延迟。电缆长度、基线温度、组件和制造容错等各种因素可能会影响信号延迟。要满足延迟要求，您必须计算用于偏移信号延迟的特定值。

表 5.5. E810 dual-NIC T-GM PtpConfig CR 参考

PtpConfig 字段	描述
spec.profile.plugins.e810.pins	使用 PTP Operator E810 硬件插件配置 E810 硬件固定。 • 固定 **2 1** 为 NIC 1 上的**SMA1** 启用 **1PPS OUT** 连接。 • 固定 **1 1** 为 NIC 2 上的**SMA1** 启用 **1PPS IN** 连接。
spec.profile.ts2phcConf	使用 **ts2phcConf** 字段为 NIC 1 和 NIC 2 配置参数。为 NIC 2 设置 **ts2phc.master 0**。这会配置来自 1PPS 输入的 NIC 2 的计时源，而不是 GNSS。为 NIC 2 配置 **ts2phc.extts_correction** 值，以补偿您所使用的特定 SMA 电缆和电缆长度的延迟。您配置的值取决于您的特定测量和 SMA1 电缆长度。
spec.profile.ptp4lConf	将 **boundary_clock_jbod** 的值设置为 1，以启用对多个 NIC 的支持。

spec.profile.plugins.e810.pins 列表中的每个值都遵循 **<function> <channel_number>** 格式。

其中：

<function> : 指定 pin 角色。以下值与 pin 角色关联：

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

<channel_number> : 与物理连接器关联的数字。以下频道号与物理连接器关联：

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

示例：

- **2 1**: 在 **SMA1** 中启用 **1PPS OUT** (Tx)。
- **1 1**: 在 **SMA1** 中启用 **1PPS IN** (Rx)

PTP Operator 将这些值传递给 Intel E810 硬件插件，并将其写入每个 NIC 上的 sysfs pin 配置接口。

5.2.5.4. 3-card E810 NIC 配置参考

使用此信息了解如何将 3 E810 NIC 配置为 PTP grandmaster 时钟 (T-GM)。

在配置 3card 集群主机前，您必须使用 1PPS faceplate 连接 3 个 NIC。主 NIC **1PPS_out** 输出提供其他 2 NIC。

当您配置 3 个卡 T-GM 时，您需要使用 SMA1 连接端口连接 NIC 时发生 1PPS 信号延迟。电缆长度、基线温度、组件和制造容错等各种因素可能会影响信号延迟。要满足延迟要求，您必须计算用于偏移信号延迟的特定值。

表 5.6. 3-card E810 T-GM PtpConfig CR 参考

PtpConfig 字段	描述
spec.profile.plugins.e810.pins	使用 PTP Operator E810 硬件插件配置 E810 硬件固定。 • **$iface_timeTx1.SMA1** 为 NIC 1 上的**SMA1** 启用 **1PPS OUT** 连接。 • **$iface_timeTx1.SMA2** 为 NIC 1 上的**SMA2** 启用 **1PPS OUT** 连接。 • **$iface_timeTx2.SMA1** 和 **$iface_timeTx3.SMA1** 为 NIC 2 和 NIC 3 启用 **1PPS IN** 连接。 • **$iface_timeTx2.SMA2** 和 **$iface_timeTx3.SMA2** 为 NIC 2 和 NIC 3 禁用 **SMA2** 连接。

PtpConfig 字段	描述
spec.profile.ts2phcConf	使用 **ts2phcConf** 字段为 NIC 配置参数。为 NIC 2 和 NIC 3 设置 **ts2phc.master 0**。这会从 1PPS 输入（而不是 GNSS）为 NIC 2 和 NIC 3 配置计时源。为 NIC 2 和 NIC 3 配置 **ts2phc.extts_correction** 值，以为因为使用的特定 SMA 电缆和电缆长度造成的延迟进行补偿。您配置的值取决于您的特定测量和 SMA1 电缆长度。
spec.profile.ptp4lConf	将 **boundary_clock_jbod** 的值设置为 1，以启用对多个 NIC 的支持。

5.2.6. 在有 GNSS 作为源时在 grandmaster 时钟中延续

holdover 允许 grandmaster (T-GM)时钟在全局导航 satellite 系统(GNSS)源不可用时维护同步性能。在此期间，T-GM 时钟依赖于其内部 oscillator 并保存参数，以减少时间中断。

您可以通过在 **PTPConfig** 自定义资源(CR)中配置以下 holdover 参数来定义保存的行为：

MaxInSpecOffset

以纳秒为单位指定允许的最大偏移量。如果 T-GM 时钟超过 **MaxInSpecOffset** 值，它将过渡到 **FREERUN** 状态（时钟状态 **gm.ClockClass 248**）。

LocalHoldoverTimeout

指定在过渡到 **FREERUN** 状态前 T-GM 时钟保持处于保存状态的最长持续时间（以秒为单位）。

LocalMaxHoldoverOffSet

指定 T-GM 时钟可在延续状态以纳秒为单位访问的最大偏移量。

如果 **MaxInSpecOffset** 值小于 **LocalMaxHoldoverOffset** 值，并且 T-GM 时钟超过最大偏移值，T-GM 时钟将从 holdover 状态转换为 **FREERUN** 状态。

重要

如果 **LocalMaxHoldoverOffSet** 值小于 **MaxInSpecOffset** 值，则在时钟达到最大偏移前发生 holdover 超时。要解决这个问题，将 **MaxInSpecOffset** 字段和 **LocalMaxHoldoverOffset** 字段设置为相同的值。

有关时钟类状态的详情，请参考 "Grandmaster 时钟类同步状态参考" 文档。

T-GM 时钟使用 holdover 参数 **LocalMaxHoldoverOffSet** 和 **LocalHoldoverTimeout** 来计算 slope。slope 是阶段偏移随时间变化的速度。它衡量每秒的偏移（纳秒为单位），其中设置的值代表在一个给定时间段内偏移量增加了多少。

T-GM 时钟使用 slope 值来预测和补偿时间偏差，从而减少了维护期间的时间中断。T-GM 时钟使用以下公式来计算 slope：

- Slope = **localMaxHoldoverOffSet** / **localHoldoverTimeout**
  例如，如果 **LocalHoldOverTimeout** 参数设置为 60 秒，并且 **LocalMaxHoldoverOffset** 参数设置为 3000 纳秒，则 slope 计算如下：

  slope = 3000 纳秒 / 60 秒 = 每秒 50纳秒

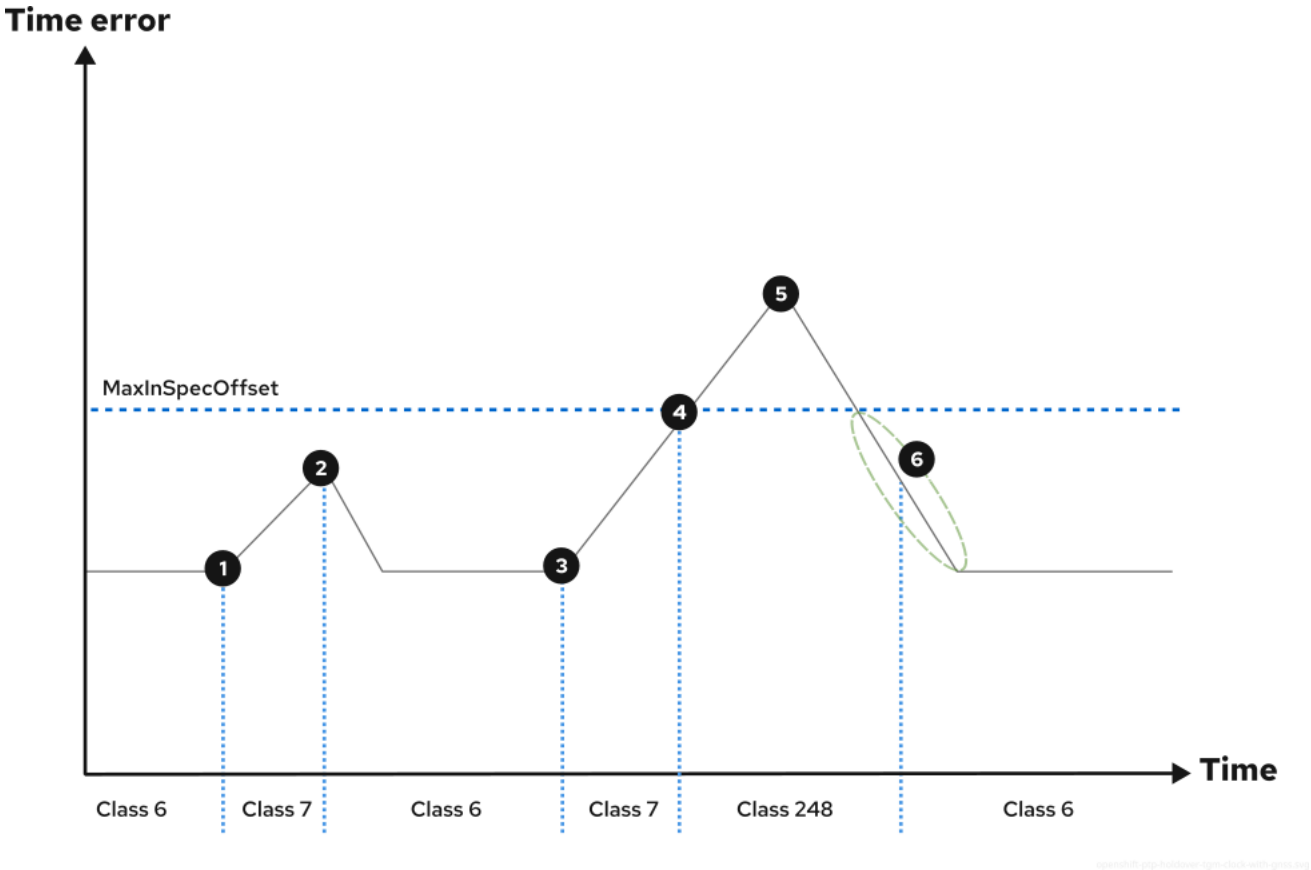
  T-GM 时钟在 60 秒内达到最大偏移量。

注意

阶段偏移从 picoseconds 转换为纳秒。因此，在 holdover 过程中计算的阶段偏移以纳秒表示，生成的 slope 以每秒纳秒表示。

下图演示了 T-GM 时钟中的 holdover 行为，使用 GNSS 作为源：

图 5.5. 保存在 T-GM 时钟中，使用 GNSS 作为源



1 GNSS 信号丢失，从而导致 T-GM 时钟进入 **HOLDOVER** 模式。T-GM 时钟使用其内部时钟保持时间准确性。

2 GNSS 信号已被恢复，T-GM 时钟重新进入 **LOCKED** 模式。当恢复了 GNSS 信号时，T-GM 时钟仅在同步链中的所有依赖组件，如 **ts2phc** 偏移、数字阶段锁定循环 (DPLL) 阶段偏移和 GNSS 偏移都进入一个稳定的 **LOCKED** 模式后，才会重新进入 **LOCKED** 模式。

3 GNSS 信号再次丢失，T-GM 时钟重新进入 **HOLDOVER** 模式。时间错误开始增加。

4 由于可追溯性丢失，时间错误超过 **MaxInSpecOffset** 阈值。

5 GNSS 信号已被恢复，T-GM 时钟会恢复同步。时间错误将开始减少。

6 时间错误减少了，并返回到 **MaxInSpecOffset** 阈值内。

其他资源

- grandmaster 时钟类同步状态参考

5.2.7. 为 PTP grandmaster 时钟配置动态秒处理

PTP Operator 容器镜像包含最新的 **leap-seconds.list** 文件，该文件在发布时可用。您可以使用全局位置系统(GPS)公告将 PTP Operator 配置为自动更新闰秒文件。

闰秒信息存储在 **openshift-ptp** 命名空间中的名为 **leap-configmap** 的自动生成的 **ConfigMap** 资源中。PTP Operator 将 **leap-configmap** 资源挂载为 **linuxptp-daemon** pod 中的卷，该 pod 可以被 **ts2phc** 进程访问。

如果 GPS satellite 广播新的闰秒数据，PTP Operator 会使用新数据更新 **leap-configmap** 资源。**ts2phc** 进程自动获取更改。

注意

以下步骤作为参考提供。PTP Operator 的 4.18 版本默认启用自动 leap 秒管理。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 您已在集群中安装 PTP Operator 并配置了 PTP grandmaster 时钟 (T-GM)。

流程

1. 在 **PtpConfig** CR 的 **phc2sysOpts** 部分中配置自动步处理。设置以下选项：

```
phc2sysOpts: -r -u 0 -m -N 8 -R 16 -S 2 -s ens2f0 -n 24 (1)
```

注意

在以前的版本中，T-GM 需要 **phc2sys** 配置 (**-O -37**) 中的偏移调整才能考虑历史的秒。这已不再需要。

2. 配置 Intel e810 NIC，以便由 **PtpConfig** CR 的 **spec.profile.plugins.e810.ublxCmds** 部分中的 GPS 接收器启用定期报告 **NAV-TIMELS** 消息。例如：

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

验证

1. 验证配置的 T-GM 是否收到来自连接的 GPS 的 **NAV-TIMELS** 消息。运行以下命令:

```
$ 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
```

输出示例

```
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. 验证 **leap-configmap** 资源是否已由 PTP Operator 成功生成，并且使用最新版本的 leap-seconds.list 保持最新状态。运行以下命令：

```
$ oc -n openshift-ptp get configmap leap-configmap -o jsonpath='{.data.<node_name>}' 1
```

1 1 将 **<node_name>** 替换为您安装并配置 PTP T-GM 时钟的节点，使用自动闰秒管理。在节点名称中转义特殊字符。例如，**node-1\.example\.com**。

输出示例

```
# 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
```

5.2.8. 将 linuxptp 服务配置为边界时钟

您可以通过创建 **PtpConfig** 自定义资源(CR)对象将 **linuxptp** 服务（**ptp4l**、**phc2sys**）配置为边界时钟。

注意

使用 **PtpConfig** CR 示例，将 **linuxptp** 服务配置为特定硬件和环境的边界时钟。这个示例 CR 没有配置 PTP 快速事件。要配置 PTP 快速事件，请为 **ptp4lOpts**、**ptp4lConf** 和 **ptpClockThreshold** 设置适当的值。**ptpClockThreshold** 仅在启用事件时使用。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建以下 **PtpConfig** CR，然后在 **boundaries-clock-ptp-config.yaml** 文件中保存 YAML。

   PTP 边界时钟配置示例

```
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"
```

表 5.7. PTP 边界时钟 CR 配置选项

CR 字段	描述
name	**PtpConfig** CR 的名称。
配置集	指定包括一个或多个 **profile** 的数组。
name	指定唯一标识配置集对象的配置集对象的名称。
ptp4lOpts	为 **ptp4l** 服务指定系统配置选项。该选项不应包含网络接口名称 **-i <interface>** 和服务配置文件 **-f /etc/ptp4l.conf**，因为网络接口名称和服务配置文件会被自动附加。
ptp4lConf	指定启动 **ptp4l** 作为边界时钟所需的配置。例如，**ens1f0** 同步来自 Pumaster 时钟，**ens1f3** 同步连接的设备。
<interface_1>	接收同步时钟的接口。
<interface_2>	发送同步时钟的接口。
tx_timestamp_timeout	对于 Intel Columbiaville 800 系列 NIC，将 **tx_timestamp_timeout** 设置为 **50**。
boundary_clock_jbod	对于 Intel Columbiaville 800 系列 NIC，请确保 **boundary_clock_jbod** 设置为 **0**。对于 Intel Fortville X710 系列 NIC，请确保 **boundary_clock_jbod** 设置为 **1**。
phc2sysOpts	为 **phc2sys** 服务指定系统配置选项。如果此字段为空，PTP Operator 不会启动 **phc2sys** 服务。
ptpSchedulingPolicy	ptp4l 和 phc2sys 进程的调度策略。默认值为 **SCHED_OTHER**。在支持 FIFO 调度的系统上使用 **SCHED_FIFO**。
ptpSchedulingPriority	当 **ptpSchedulingPolicy** 设置为 **SCHED_FIFO** 时，用于为 **ptp4l** 和 **phc2sys** 进程设置 FIFO 优先级的整数值（1 到 65）。当 **ptpSchedulingPolicy** 设置为 **SCHED_OTHER** 时，不使用 **ptpSchedulingPriority** 字段。

CR 字段	描述
ptpClockThreshold	可选。如果没有 **ptpClockThreshold**，用于 **ptpClockThreshold** 字段的默认值。**ptpClockThreshold** 配置在触发 PTP 时间前，PTP master 时钟已断开连接的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。
建议	指定包括一个或多个 **recommend** 对象的数组，该数组定义了如何将**配置集**应用到节点的规则。
.recommend.profile	指定在 **profile** 部分定义的 **.recommend.profile** 对象名称。
.recommend.priority	使用 **0** 到 **99** 之间的一个整数值指定 **priority**。大数值的优先级较低，因此优先级 **99** 低于优先级 **10**。如果节点可以根据 **match** 字段中定义的规则与多个配置集匹配，则优先级较高的配置集会应用到该节点。
.recommend.match	使用 **nodeLabel** 或 **nodeName** 值指定 **.recommend.match** 规则。
.recommend.match.nodeLabel	通过 **oc get nodes --show-labels** 命令，使用来自节点对象的 **node.Labels** 的 **key** 设置 **nodeLabel**。例如，**node-role.kubernetes.io/worker**。
.recommend.match.nodeName	使用 **oc get nodes** 命令，将 **nodeName** 设置为来自节点对象的 **node.Name** 值。例如，**compute-1.example.com**。

2. 运行以下命令来创建 CR：

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

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

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

   输出示例

   ```
   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. 检查配置集是否正确。检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志。运行以下命令:

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

输出示例

```
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] ------------------------------------
```

其他资源

- 为 PTP 硬件配置 FIFO 优先级调度
- 配置 PTP 快速事件通知发布程序

5.2.8.1. 将 linuxptp 服务配置为双 NIC 硬件的边界时钟

您可以通过为每个 NIC 创建一个 **PtpConfig** 自定义资源(CR)对象，将 **linuxptp** 服务（**ptp4l**、**phc2sys**）配置为双 NIC 硬件的边界时钟。

双 NIC 硬件允许您将每个 NIC 连接到相同的上游领导时钟，并将每个 NIC 的 **ptp4l** 实例连接给下游时钟。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建两个单独的 **PtpConfig** CR，每个 NIC 使用 "Configuring linuxptp 服务作为边界时钟"中的引用 CR，作为每个 CR 的基础。例如：

   a. 创建 **boundary-clock-ptp-config-nic1.yaml**，为 **phc2sysOpts** 指定值：

```
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: | 1
      [ens5f1]
      masterOnly 1
      [ens5f0]
      masterOnly 0
    ...
    phc2sysOpts: "-a -r -m -n 24 -N 8 -R 16" 2
```

1 指定所需的接口来启动 **ptp4l** 作为一个边境时钟。例如，**ens5f0** 从 grandmaster 时钟同步，**ens5f1** 同步连接的设备。

2 所需的 **phc2sysOpts** 值。**-m** 将消息输出到 **stdout**。**linuxptp-daemon DaemonSet** 解析日志并生成 Prometheus 指标。

b. 创建 **boundary-clock-ptp-config-nic2.yaml**，删除 **phc2sysOpts** 字段，以完全禁用第二个 NIC 的 **phc2sys** 服务：

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

1 在第二个 NIC上 指定所需的接口来启动 **ptp4l** 作为一个边境时钟。

注意

您必须从第二个 **PtpConfig** CR 中完全删除 **phc2sysOpts** 字段，以禁用第二个 NIC 上的 **phc2sys** 服务。

2. 运行以下命令来创建双 NIC **PtpConfig** CR：

a. 创建 CR 来为第一个 NIC 配置 PTP：

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

b. 创建 CR 来为第二个 NIC 配置 PTP：

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

验证

- 检查 PTP Operator 是否为两个 NIC 应用了 **PtpConfig** CR。检查与安装了双 NIC 硬件的节点对应的 **linuxptp** 守护进程的日志。例如，运行以下命令：

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

输出示例

```
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
```

5.2.8.2. 将 linuxptp 配置为双 NIC Intel E810 PTP 边界时钟的高可用性系统时钟

您可以将 **linuxptp** 服务 **ptp4l** 和 **phc2sys** 配置为双 PTP 边界时钟 (T-BC) 的高可用性 (HA) 系统时钟。

高可用性系统时钟使用来自双 NIC Intel E810 Salem 频道硬件的多个时间源，配置为两个边界时钟。两个边界时钟实例参与 HA 设置，每个设置都有自己的配置 profile。您可以将每个 NIC 连接到相同的上游领导时钟，每个 NIC 为下游时钟提供单独的 **ptp4l** 实例。

创建两个 **PtpConfig** 自定义资源 (CR) 对象，将 NIC 配置为 T-BC 和第三个 **PtpConfig** CR，以配置两个 NIC 之间的高可用性。

重要

您可以在配置 HA 的 **PtpConfig** CR 中设置 **phc2SysOpts** 选项。在 PtpConfig CR 中将 **phc2sysOpts** 字段设置为配置两个 NIC 的 **PtpConfig** CR 中的空字符串。这可防止为两个配置集设置单独的 **phc2sys** 进程。

第三个 **PtpConfig** CR 配置高度可用的系统时钟服务。CR 将 **ptp4lOpts** 字段设置为空字符串，以防止 **ptp4l** 进程运行。CR 在 **spec.profile.ptpSettings.haProfiles** 键下添加 **ptp4l** 配置的配置集，并将这些配置集的内核套接字路径传递给 **phc2sys** 服务。当出现 **ptp4l** 失败时，**phc2sys** 服务将切换到备份 **ptp4l** 配置。当主配置集再次激活时，**phc2sys** 服务将恢复到原始状态。

重要

确保将 **spec.recommend.priority** 设置为您用来配置 HA 的所有三个 **PtpConfig** CR 的值。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

- 使用 Intel E810 Salem 频道双 NIC 配置集群节点。

流程

1. 创建两个单独的 **PtpConfig** CR，每个 NIC 使用"将 linuxptp 服务配置为双 NIC 硬件边界时钟"中的 CR 作为每个 CR 的引用。

   a. 创建 **ha-ptp-config-nic1.yaml** 文件，为 **phc2sysOpts** 字段指定一个空字符串。例如：

```
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: | (1)
      [ens5f1]
      masterOnly 1
      [ens5f0]
      masterOnly 0
    #...
    phc2sysOpts: "" (2)
```

   (1) 指定所需的接口来启动 **ptp4l** 作为一个边境时钟。例如，**ens5f0** 从 grandmaster 时钟同步，**ens5f1** 同步连接的设备。

   (2) 使用空字符串设置 **phc2sysOpts**。这些值从配置高可用性的 **PtpConfig** CR 的 **spec.profile.ptpSettings.haProfiles** 字段填充。

   b. 运行以下命令，为 NIC 1 应用 **PtpConfig** CR：

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

   c. 创建 **ha-ptp-config-nic2.yaml** 文件，为 **phc2sysOpts** 字段指定一个空字符串。例如：

```
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. 运行以下命令，为 NIC 2 应用 **PtpConfig** CR：

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

2. 创建配置 HA 系统时钟的 **PtpConfig** CR。例如：

a. 创建 **ptp-config-for-ha.yaml** 文件。将 **haProfiles** 设置为与配置两个 NIC 的 **PtpConfig** CR 中设置的 **metadata.name** 字段匹配。例如：**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: "" 1
      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"
```

1 将 **ptp4lOpts** 字段设置为空字符串。如果它不为空，**p4ptl** 进程开始时会带有一个严重错误。

重要

在配置单个 NIC 的 **PtpConfig** CR 前，不要应用高可用性 **PtpConfig** CR。

a. 运行以下命令来应用 HA **PtpConfig** CR：

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

验证

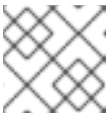
- 验证 PTP Operator 是否已正确应用 **PtpConfig** CR。执行以下步骤：

a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

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

输出示例

```
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
```



注意

应该只有一个 **linuxptp-daemon** pod。

b. 运行以下命令，检查配置集是否正确。检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志。

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

输出示例

```
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] ------------------------------------
```

5.2.9. 将 linuxptp 服务配置为常规时钟

您可以通过创建 **PtpConfig** 自定义资源(CR)对象将 **linuxptp** 服务（**ptp4l**、**phc2sys**）配置为常规时钟。

注意

使用 **PtpConfig** CR 示例，将 **linuxptp** 服务配置为特定硬件和环境的普通时钟。这个示例 CR 没有配置 PTP 快速事件。要配置 PTP 快速事件，请为 **ptp4lOpts**、**ptp4lConf** 和 **ptpClockThreshold** 设置适当的值。只有在启用事件时才需要 **ptpClockThreshold**。如需更多信息，请参阅"配置 PTP 快速事件通知发布程序"。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

流程

1. 创建以下 **PtpConfig** CR，然后在 **ordinary-clock-ptp-config.yaml** 文件中保存 YAML。

PTP 普通时钟配置示例

```
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"
```

表 5.8. PTP 普通时钟 CR 配置选项

CR 字段	描述
name	**PtpConfig** CR 的名称。
配置集	指定包括一个或多个 **profile** 的数组。每个配置集的名称都需要是唯一的。
interface	指定 **ptp4l** 服务要使用的网络接口，如 **ens787f1**。
ptp4lOpts	为 **ptp4l** 服务指定系统配置选项，例如 **-2** 来选择 IEEE 802.3 网络传输。该选项不应包含网络接口名称 **-i <interface>** 和服务配置文件 **-f /etc/ptp4l.conf**，因为网络接口名称和服务配置文件会被自动附加。附加 **--summary_interval -4** 来对此接口使用 PTP 快速事件。
phc2sysOpts	为 **phc2sys** 服务指定系统配置选项。如果此字段为空，PTP Operator 不会启动 **phc2sys** 服务。对于 Intel Columbiaville 800 Series NIC，将 **phc2sysOpts** 选项设置为 **-a -r -m -n 24 -N 8 -R 16**.**-m** 将消息输出到 **stdout**。**linuxptp-daemon DaemonSet** 解析日志并生成 Prometheus 指标。
ptp4lConf	指定一个字符串，其中包含要替换默认的 **/etc/ptp4l.conf** 文件的配置。要使用默认配置，请将字段留空。
tx_timestamp_timeout	对于 Intel Columbiaville 800 系列 NIC，将 **tx_timestamp_timeout** 设置为 **50**。
boundary_clock_jbod	对于 Intel Columbiaville 800 系列 NIC，将 **boundary_clock_jbod** 设置为 **0**。
ptpSchedulingPolicy	**ptp4l** 和 **phc2sys** 进程的调度策略。默认值为 **SCHED_OTHER**。在支持 FIFO 调度的系统上使用 **SCHED_FIFO**。

CR 字段	描述
ptpSchedulingPriority	当 **ptpSchedulingPolicy** 设置为 **SCHED_FIFO** 时，用于为 **ptp4l** 和 **phc2sys** 进程设置 FIFO 优先级的整数值（1 到 65）。当 **ptpSchedulingPolicy** 设置为 **SCHED_OTHER** 时，不使用 **ptpSchedulingPriority** 字段。
ptpClockThreshold	可选。如果没有 **ptpClockThreshold**，用于 **ptpClockThreshold** 字段的默认值。**ptpClockThreshold** 配置在触发 PTP 时间前，PTP master 时钟已断开连接的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。
建议	指定包括一个或多个 **recommend** 对象的数组，该数组定义了如何将**配置集**应用到节点的规则。
.recommend.profile	指定在 **profile** 部分定义的 **.recommend.profile** 对象名称。
.recommend.priority	对于普通时钟，将 **.recommend.priority** 设置为 **0**。
.recommend.match	使用 **nodeLabel** 或 **nodeName** 值指定 **.recommend.match** 规则。
.recommend.match.nodeLabel	通过 **oc get nodes --show-labels** 命令，使用来自节点对象的 **node.Labels** 的 **key** 设置 **nodeLabel**。例如，**node-role.kubernetes.io/worker**。
.recommend.match.nodeName	使用 **oc get nodes** 命令，将 **nodeName** 设置为来自节点对象的 **node.Name** 值。例如，**compute-1.example.com**。

2. 运行以下命令来创建 **PtpConfig** CR：

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

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

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

   输出示例

```
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. 检查配置集是否正确。检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志。运行以下命令:

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

输出示例

```
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] ------------------------------------
```

其他资源

- 为 PTP 硬件配置 FIFO 优先级调度
- 配置 PTP 快速事件通知发布程序

5.2.9.1. Intel Columbiaville E800 series NIC 作为 PTP 常规时钟参考

下表描述了您必须对引用 PTP 配置所做的更改，以使用 Intel Columbiaville E800 系列 NIC 作为普通时钟。在应用到集群的 **PtpConfig** 自定义资源(CR)中进行更改。

表 5.9. Intel Columbiaville NIC 的推荐 PTP 设置

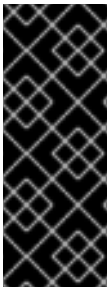
PTP 配置	推荐的设置
phc2sysOpts	**-a -r -m -n 24 -N 8 -R 16**
tx_timestamp_timeout	**50**
boundary_clock_jbod	**0**

注意

对于 **phc2sysOpts**，**-m** 会将信息输出到 **stdout**。**linuxptp-daemon DaemonSet** 解析日志并生成 Prometheus 指标。

5.2.9.2. 将 linuxptp 服务配置为具有双端口 NIC 冗余的普通时钟

您可以通过创建 **PtpConfig** 自定义资源(CR)对象，将 **linuxptp** 服务(**ptp4l**、**phc2sys**)配置为带有双端口 NIC 冗余的普通时钟。在普通时钟的双端口 NIC 配置中，如果一个端口失败，待机端口会接管，维护 PTP 时间同步。



重要

将 linuxptp 服务配置为带有双端口 NIC 冗余的普通时钟只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。
- 节点使用带有双端口 NIC 的 x86_64 架构。

流程

1. 创建以下 **PtpConfig** CR，然后在 **oc-dual-port-ptp-config.yaml** 文件中保存 YAML。

PTP 普通时钟双端口配置示例

```
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 (1)
    ptp4lConf: |- (2)
      [ens3f2]
      masterOnly 0
      [ens3f3]
      masterOnly 0

      [global]
      #
      # Default Data Set
      #
      slaveOnly 1 (3)
#...
```

(1) 为 **ptp4l** 服务指定系统配置选项。

(2) 为 **ptp4l** 服务指定接口配置。在本例中，为 **ens3f2** 和 **ens3f3** 接口设置 **masterOnly 0** 可让 **ens3** 接口上的两个端口作为领导或后续时钟运行。与 **slaveOnly 1** 规范相结合，此配置可

确保一个端口作为活跃普通时钟运行，另一个端口作为 **Listening** 端口状态的待机时钟运行。

3 将 **ptp4l** 配置为仅作为普通时钟运行。

2. 运行以下命令来创建 **PtpConfig** CR：

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

验证

1. 检查 **PtpConfig** 配置集是否已应用到节点。

   a. 运行以下命令，获取 **openshift-ptp** 命名空间中的 pod 列表：

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

   输出示例

   ```
   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. 检查配置集是否正确。检查与 **PtpConfig** 配置集中指定的节点对应的 **linuxptp** 守护进程的日志。运行以下命令:

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

   输出示例

   ```
   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] ------------------------------------
   ```

其他资源

- 有关将 **linuxptp** 服务配置为具有 PTP 快速事件的普通时钟的完整示例 CR，请参阅 将 linuxptp 服务配置为普通时钟。
- 使用双端口 NIC 来提高 PTP 普通时钟的冗余
- 使用双端口 NIC 来提高 PTP 普通时钟的冗余

5.2.10. 为 PTP 硬件配置 FIFO 优先级调度

在需要低延迟性能的电信或其他部署类型中，PTP 守护进程线程在受限的 CPU 占用空间以及剩余的基础架构组件一起运行。默认情况下，PTP 线程使用 **SCHED_OTHER** 策略运行。在高负载下，这些线程可能没有获得无错操作所需的调度延迟。

要缓解潜在的调度延迟错误，您可以将 PTP Operator **linuxptp** 服务配置为允许线程使用 **SCHED_FIFO** 策略运行。如果为 **PtpConfig** CR 设置了 **SCHED_FIFO**，则 **ptp4l** 和 **phc2sys** 将在 **chrt** 的父容器中运行，且由 **PtpConfig** CR 的 **ptpSchedulingPriority** 字段设置。

注意

设置 **ptpSchedulingPolicy** 是可选的，只有在遇到延迟错误时才需要。

流程

1. 编辑 **PtpConfig** CR 配置集：

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

2. 更改 **ptpSchedulingPolicy** 和 **ptpSchedulingPriority** 字段：

```
apiVersion: ptp.openshift.io/v1
kind: PtpConfig
metadata:
  name: <ptp_config_name>
  namespace: openshift-ptp
...
spec:
  profile:
  - name: "profile1"
...
    ptpSchedulingPolicy: SCHED_FIFO (1)
    ptpSchedulingPriority: 10 (2)
```

(1) **ptp4l** 和 **phc2sys** 进程的调度策略。在支持 FIFO 调度的系统上使用 **SCHED_FIFO**。

(2) 必需。设置整数值 1-65，用于为 **ptp4l** 和 **phc2sys** 进程配置 FIFO 优先级。

3. 保存并退出，以将更改应用到 **PtpConfig** CR。

验证

1. 获取 **linuxptp-daemon** pod 的名称以及应用 **PtpConfig** CR 的对应节点：

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

输出示例

```
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. 检查 **ptp4l** 进程是否使用更新的 **chrt** FIFO 运行：

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

输出示例

```
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
```

5.2.11. 为 linuxptp 服务配置日志过滤

linuxptp 守护进程生成可用于调试目的的日志。在具有有限存储容量的电信或其他部署类型中，这些日志可以添加到存储要求中。

要减少数量日志消息，您可以配置 **PtpConfig** 自定义资源 (CR) 来排除报告 **master offset** 值的日志消息。**master offset** 日志消息以纳秒为单位报告当前节点时钟和 master 时钟之间的区别。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 安装 PTP Operator。

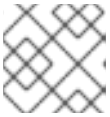
流程

1. 编辑 **PtpConfig** CR：

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

2. 在 **spec.profile** 中，添加 **ptpSettings.logReduce** 规格，并将值设为 **true** ：

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



注意

为了进行调试，您可以将此规格恢复到 **False**，使其包含 master 偏移消息。

3. 保存并退出，以将更改应用到 **PtpConfig** CR。

验证

1. 获取 **linuxptp-daemon** pod 的名称以及应用 **PtpConfig** CR 的对应节点：

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

输出示例

```
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. 运行以下命令，验证 master 偏移信息是否不包括在日志中：

```
$ oc -n openshift-ptp logs <linux_daemon_container> -c linuxptp-daemon-container | grep
"master offset" 1
```

1 <linux_daemon_container> 是 **linuxptp-daemon** pod 的名称，如 **linuxptp-daemon-gmv2n**。

当您配置 **logReduce** 规格时，这个命令会在 **linuxptp** 守护进程日志中报告任何 **master offset** 实例。

5.2.12. 常见 PTP Operator 故障排除

通过执行以下步骤排除 PTP Operator 中的常见问题。

先决条件

- 安装 OpenShift Container Platform CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 使用支持 PTP 的主机在裸机集群中安装 PTP Operator。

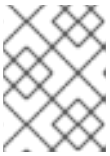
流程

1. 检查集群中为配置的节点成功部署了 Operator 和操作对象。

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

输出示例

```
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
```



注意

当启用 PTP fast 事件总线时，就绪的 **linuxptp-daemon** pod 的数量是 **3/3**。如果没有启用 PTP fast 事件总线，则会显示 **2/2**。

2. 检查集群中是否已找到支持的硬件。

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io
```

输出示例

```
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. 检查节点的可用 PTP 网络接口：

```
$ oc -n openshift-ptp get nodeptpdevices.ptp.openshift.io <node_name> -o yaml
```

其中：

<node_name>

指定您要查询的节点，例如 **compute-0.example.com**。

输出示例

```
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. 通过访问对应节点的 **linuxptp-daemon** pod，检查 PTP 接口是否已与主时钟成功同步。

   a. 运行以下命令来获取 **linuxptp-daemon** pod 的名称以及您要排除故障的对应节点：

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

   输出示例

   ```
   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. 在远程 shell 到所需的 **linuxptp-daemon** 容器：

   ```
   $ oc rsh -n openshift-ptp -c linuxptp-daemon-container <linux_daemon_container>
   ```

   其中：

   <linux_daemon_container>

   您要诊断的容器，如 **linuxptp-daemon-lmvgn**。

   c. 在与 **linuxptp-daemon** 容器的远程 shell 连接中，使用 PTP Management Client (**pmc**) 工具诊断网络接口。运行以下 **pmc** 命令，以检查 PTP 设备的同步状态，如 **ptp4l**。

   ```
   # pmc -u -f /var/run/ptp4l.0.config -b 0 'GET PORT_DATA_SET'
   ```

   当节点成功同步到主时钟时的输出示例

   ```
   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. 对于 GNSS-sourced grandmaster 时钟，运行以下命令来验证 in-tree NIC ice 驱动程序是否正确，例如：

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-74m2g ethtool -i ens7f0
```

输出示例

```
driver: ice
version: 5.14.0-356.bz2232515.el9.x86_64
firmware-version: 4.20 0x8001778b 1.3346.0
```

6. 对于 GNSS-sourced grandmaster 时钟，请验证 **linuxptp-daemon** 容器是否从 GNSS antenna 接收信号。如果容器没有收到 GNSS 信号，则不会填充 **/dev/gnss0** 文件。要验证，请运行以下命令：

```
$ oc rsh -n openshift-ptp -c linuxptp-daemon-container linuxptp-daemon-jnz6r cat /dev/gnss0
```

输出示例

```
$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
```

5.2.13. 在 Intel 800 系列 NIC 中获取 CGU 的 DPLL 固件版本

您可以通过打开 debug shell 到集群节点并查询 NIC 硬件，在 Intel 800 系列 NIC 中获取 Clock Generation Unit (CGU) 的数字签名循环 (DPLL) 固件版本。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 您已在集群主机中安装了 Intel 800 系列 NIC。
- 您已在带有支持 PTP 的主机的裸机集群中安装 PTP Operator。

流程

1. 运行以下命令来启动 debug pod：

```
$ oc debug node/<node_name>
```

其中：

<node_name>
是安装 Intel 800 系列 NIC 的节点。

2. 使用 **devlink** 工具以及安装 NIC 的总线和设备名称，检查 NIC 中的 CGU 固件版本。例如，运行以下命令：

```
sh-4.4# devlink dev info <bus_name>/<device_name> | grep cgu
```

其中：

<bus_name>
是安装 NIC 的总线。例如，**pci**。

<device_name>
是 NIC 设备名称。例如，**0000:51:00.0**。

输出示例

```
cgu.id 36 1
fw.cgu 8032.16973825.6021 2
```

1 CGU 硬件修订号

2 在 CGU 中运行的 DPLL 固件版本，DPLL 固件版本为 **6201**，DPLL 模型是 **8032**。字符串 **16973825** 是 DPLL 固件版本的二进制版本的简写形式（**1.3.0.1**）。

注意

固件版本的每个版本号部分都包括了前导和 3 个八位字节位。数字 **16973825** 的二进制格式是 **0001 0000 0011 0000 0000 0000 0001**。使用二进制值来解码固件版本。例如：

表 5.10. DPLL 固件版本

二进制部分	十进制值
0001	1
0000 0011	3
0000 0000	0
0000 0001	1

5.2.14. 收集 PTP Operator 数据

您可以使用 **oc adm must-gather** 命令收集有关集群的信息，包括与 PTP Operator 关联的功能和对象。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 PTP Operator。

流程

- 要使用 **must-gather** 来收集 PTP Operator 数据，您必须指定 PTP Operator **must-gather** 镜像。

```
$ oc adm must-gather --image=registry.redhat.io/openshift4/ptp-must-gather-rhel9:v4.18
```

5.3. 使用 REST API V2 开发 PTP 事件消费者应用程序

在裸机集群节点上开发使用 Precision Time Protocol (PTP)事件的消费者应用程序时，您可以在单独的应用程序 pod 中部署消费者应用程序。消费者应用程序使用 PTP 事件 REST API v2 订阅 PTP 事件。

注意

以下信息提供了开发使用 PTP 事件的消费者应用程序的一般指导。完整的事件消费者应用示例超出了此信息的范围。

其他资源

- PTP 事件 REST API v2 参考

5.3.1. 关于 PTP 快速事件通知框架

使用 Precision Time Protocol (PTP) 快速事件 REST API v2 将集群应用程序订阅到裸机集群节点生成的 PTP 事件。

注意

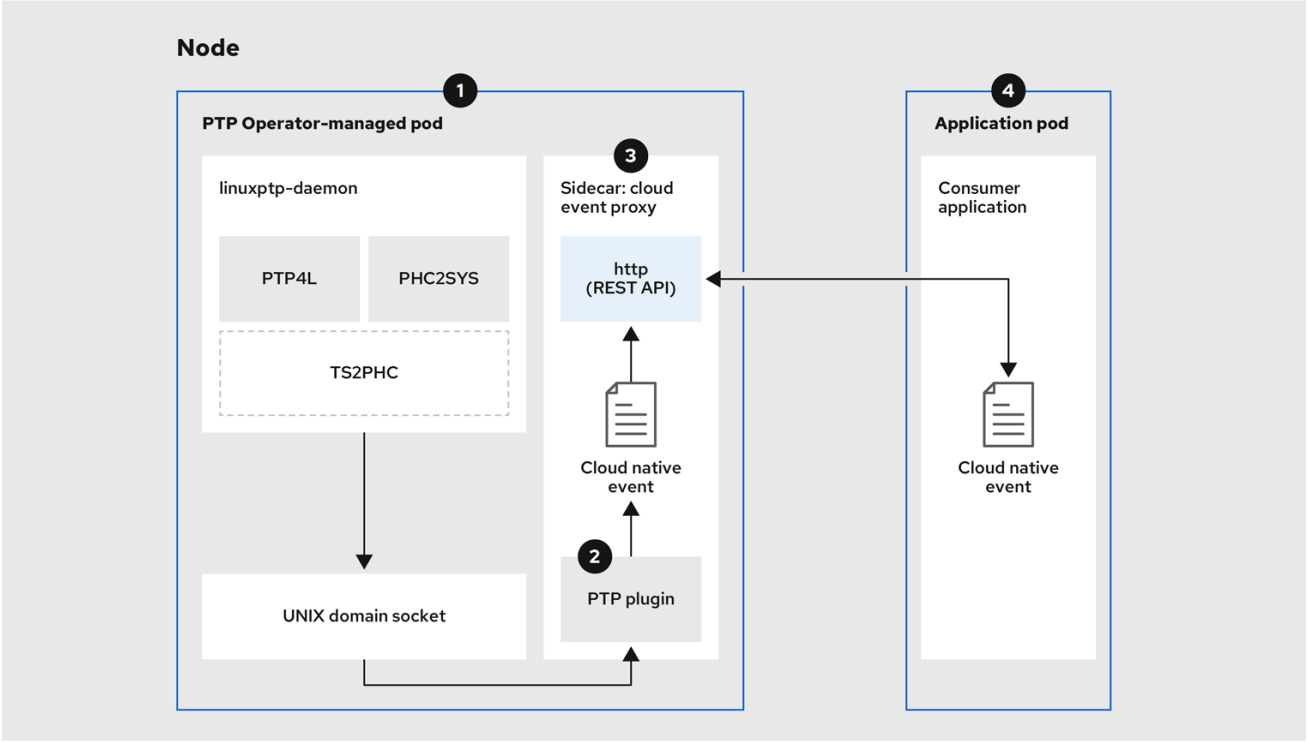
快速事件通知框架使用 REST API 进行通信。PTP 事件 REST API v1 和 v2 基于 *O-RAN O-Cloud Notification API Specification for Event Consumers 4.0*，它包括在 O-RAN ALLIANCE Specifications 中。

只有 PTP 事件 REST API v2 符合 O-RAN v4。

5.3.2. 使用 PTP 事件 REST API v2 检索 PTP 事件

应用程序在生成者云事件代理 sidecar 中使用 O-RAN v4 兼容 REST API 订阅 PTP 事件。**cloud-event-proxy** sidecar 容器可以访问与主应用程序容器相同的资源，而无需使用主应用程序的任何资源，且没有大量延迟。

图 5.6. 从 PTP 事件制作者 REST API v2 中消耗 PTP 快速事件概述



319_OpenShift_0323

1 **事件在集群主机上生成**

PTP Operator 管理的 pod 中的 **linuxptp-daemon** 进程作为 Kubernetes **DaemonSet** 运行，并管理各种 **linuxptp** 进程 (**ptp4l**、**phc2sys**，以及可选的用于 grandmaster 时钟 **ts2phc**)。**linuxptp-daemon** 将事件传递给 UNIX 域套接字。

2 **事件传递给 cloud-event-proxy sidecar**

PTP 插件从 UNIX 域套接字读取事件，并将其传递给 PTP Operator 管理的 pod 中的 **cloud-event-proxy** sidecar。**cloud-event-proxy** 将 Kubernetes 基础架构的事件提供给具有低延迟的 Cloud-Native Network Function (CNF)。

3 **事件已发布**

PTP Operator 管理的 pod 中的 **cloud-event-proxy** sidecar 处理事件，并使用 PTP 事件 REST API v2 发布事件。

4 **消费者应用程序请求订阅并接收订阅的事件**

消费者应用程序向制作者 **cloud-event-proxy** sidecar 发送 API 请求，以创建 PTP 事件订阅。订阅后，消费者应用程序会侦听资源限定符中指定的地址，并接收和处理 PTP 事件。

5.3.3. 配置 PTP 快速事件通知发布程序

要为集群中的网络接口启动使用 PTP fast 事件通知，您必须在 PTP Operator **PtpOperatorConfig** 自定义资源 (CR) 中启用快速事件发布程序，并在您创建的 **PtpConfig** CR 中配置 **ptpClockThreshold** 值。

先决条件

- 已安装 OpenShift Container Platform CLI (**oc**)。

- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 已安装 PTP Operator。

流程

1. 修改默认 PTP Operator 配置以启用 PTP 快速事件。

   a. 在 **ptp-operatorconfig.yaml** 文件中保存以下 YAML：

```
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 通过将 **ptpEventConfig.apiVersion** 设置为 "2.0"，为 PTP 事件生成者启用 PTP 事件 REST API v2。默认值为 "1.0"。

   2 通过将 **enableEventPublisher** 设置为 **true** 来启用 PTP 快速事件通知。

注意

在 OpenShift Container Platform 4.13 或更高版本中，当将 HTTP 传输用于 PTP 事件时，您不需要在 **PtpOperatorConfig** 资源中设置 **spec.ptpEventConfig.transportHost** 字段。

   b. 更新 **PtpOperatorConfig** CR：

```
$ oc apply -f ptp-operatorconfig.yaml
```

2. 为 PTP 启用接口创建 **PtpConfig** 自定义资源(CR)，并设置 **ptpClockThreshold** 和 **ptp4lOpts** 所需的值。以下 YAML 演示了您必须在 **PtpConfig** CR 中设置的必要值：

```
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. 附加 **--summary_interval -4** 以使用 PTP 快速事件。
2. 所需的 **phc2sysOpts** 值。**-m** 将消息输出到 **stdout**。**linuxptp-daemon DaemonSet** 解析日志并生成 Prometheus 指标。
3. 指定一个字符串，其中包含要替换默认的 **/etc/ptp4l.conf** 文件的配置。要使用默认配置，请将字段留空。
4. 可选。如果 **ptpClockThreshold** 小节不存在，则默认值用于 **ptpClockThreshold** 字段。小节显示默认的 **ptpClockThreshold** 值。**ptpClockThreshold** 值配置 PTP master 时钟在触发 PTP 事件前的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。

其他资源

- 有关将 **linuxptp** 服务配置为具有 PTP 快速事件的普通时钟的完整示例 CR，请参阅 将 linuxptp 服务配置为普通时钟。

5.3.4. PTP 事件 REST API v2 消费者应用程序参考

PTP 事件消费者应用程序需要以下功能：

1. 使用 **POST** 处理程序运行的 Web 服务，以接收云原生 PTP 事件 JSON 有效负载
2. 订阅 PTP 事件制作者的 **createSubscription** 功能
3. **getCurrentState** 功能轮询 PTP 事件制作者的当前状态

以下示例 Go 片断演示了这些要求：

Go 中的 PTP 事件消费者服务器功能示例

```
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)
}
```

–

Go 中的 PTP 事件 createSubscription 功能示例

```
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) 将 **<node_name>** 替换为正在生成 PTP 事件的节点 FQDN。例如， **compute-1.example.com**。

Go 中的 PTP 事件消费者 getCurrentState 功能示例

```
//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 将 **<node_name>** 替换为正在生成 PTP 事件的节点 FQDN。例如， **compute-1.example.com**。

5.3.5. 使用 PTP 事件 REST API v2 引用事件消费者部署和服务 CR

在部署您的 PTP 事件消费者应用程序来与 PTP 事件 REST API v2 一起使用时，使用 PTP 事件消费者自定义资源 (CR) 示例作为一个参考。

仓库云事件消费者命名空间

```
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
```

参考云事件消费者部署

```
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: {}
```

参考云事件消费者服务帐户

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: consumer-sa
  namespace: cloud-events
```

参考云事件消费者服务

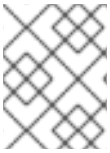
```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/scrape: "true"
  name: consumer-events-subscription-service
  namespace: cloud-events
  labels:
    app: consumer-service
spec:
  ports:
    - name: sub-port
      port: 9043
  selector:
```

```
  app: consumer
sessionAffinity: None
type: ClusterIP
```

5.3.6. 使用 REST API v2 订阅 PTP 事件

部署 **cloud-event-consumer** 应用程序容器，并将 **cloud-event-consumer** 应用程序订阅到 PTP 事件，这些事件由 PTP Operator 管理的 pod 中的 **cloud-event-proxy** 容器发布。

通过将 **POST** 请求发送到 **http://ptp-event-publisher-service-NODE_NAME.openshift-ptp.svc.cluster.local:9043/api/ocloudNotifications/v2/subscriptions** 传递适当的订阅请求有效负载，将消费者应用程序订阅到 PTP 事件。



注意

9043 是 PTP 事件制作者 pod 中部署的 **cloud-event-proxy** 容器的默认端口。您可以根据需要为应用程序配置不同的端口。

其他资源

- api/ocloudNotifications/v2/subscriptions

5.3.7. 验证 PTP 事件 REST API v2 消费者应用程序是否收到事件

验证应用程序 pod 中的 **cloud-event-consumer** 容器是否正在接收 Precision Time Protocol (PTP) 事件。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 已安装并配置了 PTP Operator。
- 您已部署了云事件应用程序 pod 和 PTP 事件消费者应用程序。

流程

1. 检查部署的事件消费者应用的日志。例如，运行以下命令：

```
$ oc -n cloud-events logs -f deployment/cloud-consumer-deployment
```

输出示例

```
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. 可选。使用 **linuxptp-daemon** 部署中的 **oc** 和 port-forwarding 端口 **9043** 来测试 REST API。例如，运行以下命令：

```
$ oc port-forward -n openshift-ptp ds/linuxptp-daemon 9043:9043
```

输出示例

```
Forwarding from 127.0.0.1:9043 -> 9043
Forwarding from [::1]:9043 -> 9043
Handling connection for 9043
```

打开新的 shell 提示符并测试 REST API v2 端点：

```
$ curl -X GET http://localhost:9043/api/ocloudNotifications/v2/health
```

输出示例

```
OK
```

5.3.8. 监控 PTP 快速事件指标

您可以从运行 **linuxptp-daemon** 的集群节点监控 PTP 快速事件指标。您还可以使用预先配置和自我更新的 Prometheus 监控堆栈来监控 OpenShift Container Platform Web 控制台中的 PTP 快速事件指标。

先决条件

- 安装 OpenShift Container Platform CLI **oc**。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 在具有 PTP 功能硬件的节点上安装和配置 PTP Operator。

流程

1. 运行以下命令，为节点启动 debug pod：

```
$ oc debug node/<node_name>
```

2. 检查 **linuxptp-daemon** 容器公开的 PTP 指标。例如，运行以下命令：

```
sh-4.4# curl http://localhost:9091/metrics
```

输出示例

```
# 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. 可选。您还可以在 **cloud-event-proxy** 容器的日志中找到 PTP 事件。例如，运行以下命令：

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

4. 要在 OpenShift Container Platform web 控制台中查看 PTP 事件，请复制您要查询的 PTP 指标的名称，如 **openshift_ptp_offset_ns**。

5. 在 OpenShift Container Platform web 控制台中点 **Observe → Metrics**。

6. 将 PTP 指标名称粘贴到 **Expression** 字段中，然后点 **Run query**。

其他资源

- 以开发者身份访问指标

5.3.9. PTP 快速事件指标参考

下表描述了运行 **linuxptp-daemon** 服务的集群节点可用的 PTP 快速事件指标。

表 5.11. PTP 快速事件指标

指标	描述	Example

指标	描述	Example
openshift_ptp_clock_class	返回接口的 PTP 时钟类。对于 PTP 时钟类的可能值为：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**), 或 255 (**SLAVE ONLY CLOCK**)。	**{node="compute-1.example.com",process="ptp4l"} 6**
openshift_ptp_clock_state	返回接口的当前 PTP 时钟状态。PTP 时钟状态的可能值为 **FREERUN**、**LOCKED** 或 **HOLDOVER**。	**{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1**
openshift_ptp_delay_ns	返回主时钟发送计时数据包和接收计时数据包之间的延迟（以纳秒为单位）。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0**
openshift_ptp_ha_profile_status	当不同 NIC 上有多个时间源时，返回高可用性系统时钟的当前状态。可能的值为 0 (**INACTIVE**) 和 1 (**ACTIVE**)。	**{node="node1",process="phc2sys",profile="profile1"} 1{node="node1",process="phc2sys",profile="profile2"} 0**
openshift_ptp_frequency_adjustment_ns	以纳秒为单位返回 2 PTP 时钟之间的频率调整。例如，在上游时钟和 NIC 之间，系统时钟和 NIC 之间，或在 PTP 硬件时钟(**phc**)和 NIC 之间。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -6768**
openshift_ptp_interface_role	返回为接口配置的 PTP 时钟角色。可能的值包括 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	返回 2 时钟或接口之间的最大偏移量（以纳秒为单位）。例如，在上游 GNSS 时钟和 NIC (**ts2phc**)之间，或在 PTP 硬件时钟(**phc**)和系统时钟(**phc2sys**)之间。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 1.038099569e+09**
openshift_ptp_offset_ns	返回 DPLL 时钟或 GNSS 时钟源和 NIC 硬件时钟之间的偏移量。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -9**
openshift_ptp_process_restart_count	返回 **ptp4l** 和 **ts2phc** 进程重启的次数。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**

指标	描述	Example
openshift_ptp_process_status	返回显示 PTP 进程是否正在运行的状态代码。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_threshold	为 **HoldOverTimeout**, **MaxOffsetThreshold**, 和 **MinOffsetThreshold** 返回值。 • **holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。 • **maxOffsetThreshold** 和 **minOffsetThreshold** 是以纳秒为单位的偏移值，它比较 **CLOCK_REALTIME** (**phc2sys**) 的值，或您在 **PtpConfig** CR 中为 NIC 配置的 r master offset (**ptp4l**) 的值。	**{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5**

5.3.9.1. 只有在启用 T-GM 时，PTP 快速事件指标

下表描述了仅在启用 PTP grandmaster 时钟 (T-GM) 时可用的 PTP 快速事件指标。

表 5.12. 启用 T-GM 时的 PTP 快速事件指标

指标	描述	Example
openshift_ptp_frequency_status	返回 NIC 的数字阶段锁定循环(DPLL)频率的当前状态。可能的值为 -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), 或 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_nmea_status	返回 NMEA 连接的当前状态。NMEA 是 1PPS NIC 连接使用的协议。可能的值有 0 (**UNAVAILABLE**) 和 1 (**AVAILABLE**)。	**{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1**
openshift_ptp_phase_status	返回 NIC 的 DPLL 阶段的状态。可能的值为 -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), 或 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_pps_status	返回 NIC 1PPS 连接的当前状态。您可以使用 1PPS 连接在连接的 NIC 之间同步计时。可能的值有 0 (**UNAVAILABLE**) 和 1 (**AVAILABLE**)。	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1**

指标	描述	Example
openshift_ptp_gnss_status	返回全局导航 Satellite 系统(GNSS)连接的当前状态。GNSS 在全局范围内提供基于 satellite 的位置、导航和计时服务。可能的值包括 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**

5.4. PTP 事件 REST API V2 参考

使用以下 REST API v2 端点，将 **cloud-event-consumer** 应用程序订阅到 Precision Time Protocol (PTP) 事件，在 PTP 事件制作者 pod 中的 **http://localhost:9043/api/ocloudNotifications/v2** 中发布。

- **api/ocloudNotifications/v2/subscriptions**
  - **POST** ：创建新订阅
  - **GET** ：删除订阅列表
  - **DELETE** ：删除所有订阅
- **api/ocloudNotifications/v2/subscriptions/{subscription_id}**
  - **GET** ：返回指定订阅 ID 的详情
  - **DELETE** ：删除与指定订阅 ID 关联的订阅
- **api/ocloudNotifications/v2/health**
  - **GET**：返回 **ocloudNotifications** API 的健康状况
- **api/ocloudNotifications/v2/publishers**
  - **GET**：返回集群节点的 PTP 事件发布程序列表
- **api/ocloudnotifications/v2/{resource_address}/CurrentState**
  - **GET**：返回由 **{resouce_address}** 指定的事件类型的当前状态。

5.4.1. PTP 事件 REST API v2 端点

5.4.1.1. api/ocloudNotifications/v2/subscriptions

HTTP 方法
GET api/ocloudNotifications/v2/subscriptions

描述
返回订阅列表。如果订阅存在，则返回 **200 OK** 状态代码以及订阅列表。

API 响应示例

```
[
 {
  "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 方法
POST api/ocloudNotifications/v2/subscriptions

描述
通过传递适当的有效负载来为所需的事件创建新订阅。

您可以订阅以下 PTP 事件：

- **sync-state** 事件
- **lock-state** 事件
- **gnss-sync-status events** 事件
- **os-clock-sync-state** 事件
- **clock-class** 事件

表 5.13. 查询参数

参数	类型
subscription	data

sync-state 订阅有效负载示例

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/sync-status/sync-state"
}
```

PTP lock-state 事件订阅有效负载示例

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/ptp-status/lock-state"
}
```

PTP gnss-sync-status 事件订阅有效负载示例

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-
events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/gnss-status/gnss-sync-status"
}
```

PTP os-clock-sync-state 事件订阅有效负载示例

```
{
"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"
}
```

PTP clock-class 事件订阅有效负载示例

```
{
"EndpointUri": "http://consumer-events-subscription-service.cloud-
```

```
events.svc.cluster.local:9043/event",
"ResourceAddress": "/cluster/node/{node_name}/sync/ptp-status/clock-class"
}
```

API 响应示例

```
{
    "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"
}
```

可能会出现以下订阅状态事件：

表 5.14. PTP 事件 REST API v2 订阅状态代码

状态代码	描述
201 created	表示已创建了订阅
400 错误请求	表示服务器无法处理请求，因为它是不正确的或无效
404 not Found	表示订阅资源不可用
409 冲突	表示订阅已存在

HTTP 方法
DELETE api/ocloudNotifications/v2/subscriptions

描述
删除所有订阅。

API 响应示例

```
{
"status": "deleted all subscriptions"
}
```

5.4.1.2. api/ocloudNotifications/v2/subscriptions/{subscription_id}

HTTP 方法
GET api/ocloudNotifications/v2/subscriptions/{subscription_id}

描述
返回 ID 为 **subscription_id** 的订阅详情。

表 5.15. 全局路径参数

参数	类型
subscription_id	string

API 响应示例

```
{
   "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 方法
DELETE api/ocloudNotifications/v2/subscriptions/{subscription_id}

描述
使用 ID **subscription_id** 删除订阅。

表 5.16. 全局路径参数

参数	类型
subscription_id	string

表 5.17. HTTP 响应代码

HTTP 响应	描述
204 无内容	成功

5.4.1.3. api/ocloudNotifications/v2/health

HTTP 方法
GET api/ocloudNotifications/v2/health/

描述
返回 **ocloudNotifications** REST API 的健康状况。

表 5.18. HTTP 响应代码

HTTP 响应	描述
200 OK	成功

5.4.1.4. api/ocloudNotifications/v2/publishers

HTTP 方法
GET api/ocloudNotifications/v2/publishers

描述
返回集群节点的发布者详情列表。当相关的设备状态改变时，系统会生成通知。

您可以组合使用设备同步状态订阅，以提供有关系统总体同步健康状况的详细视图。

API 响应示例


```
[
  {
    "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"
  }
]
```


表 5.19. HTTP 响应代码

HTTP 响应	描述
200 OK	成功

5.4.1.5. api/ocloudNotifications/v2/{resource_address}/CurrentState

HTTP 方法
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

描述
返回集群节点的 **os-clock-sync-state**,**clock-class**,**lock-state**,**gnss-sync-status**, 或 **sync-state** 事件的当前状态。

- **os-clock-sync-state** 通知描述了主机操作系统时钟同步状态。可以是 **LOCKED** 或 **FREERUN** 状态。
- **clock-class** 通知描述了 PTP 时钟类的当前状态。
- **lock-state** 通知描述了 PTP 设备锁定状态的当前状态。可以处于 **LOCKED**、**HOLDOVER** 或 **FREERUN** 状态。
- **sync-state** 通知描述了最少 PTP 时钟 **lock-state** 和 **os-clock-sync-state** 状态同步的当前状态。
- **GNSS-sync-status** 通知描述了 GNSS 时钟同步状态。

表 5.20. 全局路径参数

参数	类型
resource_address	string

lock-state API 响应示例

```
{
  "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"
      }
    ]
  }
}
```

os-clock-sync-state API 响应示例

```
{
  "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"
      }
    ]
  }
}
```

clock-class API 响应示例

```
{
  "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"
      }
    ]
  }
}
```

sync-state API 响应示例

```
{
   "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"
      }]
   }
}
```

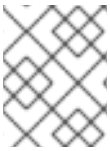
gnss-sync-state API 响应示例

```
{
  "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"
      }
```

```
    ]
  }
}
```

5.5. 使用 REST API V1 开发 PTP 事件消费者应用程序

在裸机集群节点上开发使用 Precision Time Protocol (PTP)事件的消费者应用程序时，您可以在单独的应用程序 pod 中部署消费者应用程序。消费者应用程序使用 PTP 事件 REST API v1 订阅 PTP 事件。



注意

以下信息提供了开发使用 PTP 事件的消费者应用程序的一般指导。完整的事件消费者应用示例超出了此信息的范围。

重要

PTP 事件 REST API v1 和事件消费者应用程序 sidecar 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中 *已弃用和删除的功能*部分。

其他资源

- PTP events REST API v1 参考

5.5.1. 关于 PTP 快速事件通知框架

使用 Precision Time Protocol (PTP) 快速事件 REST API v2 将集群应用程序订阅到裸机集群节点生成的 PTP 事件。

注意

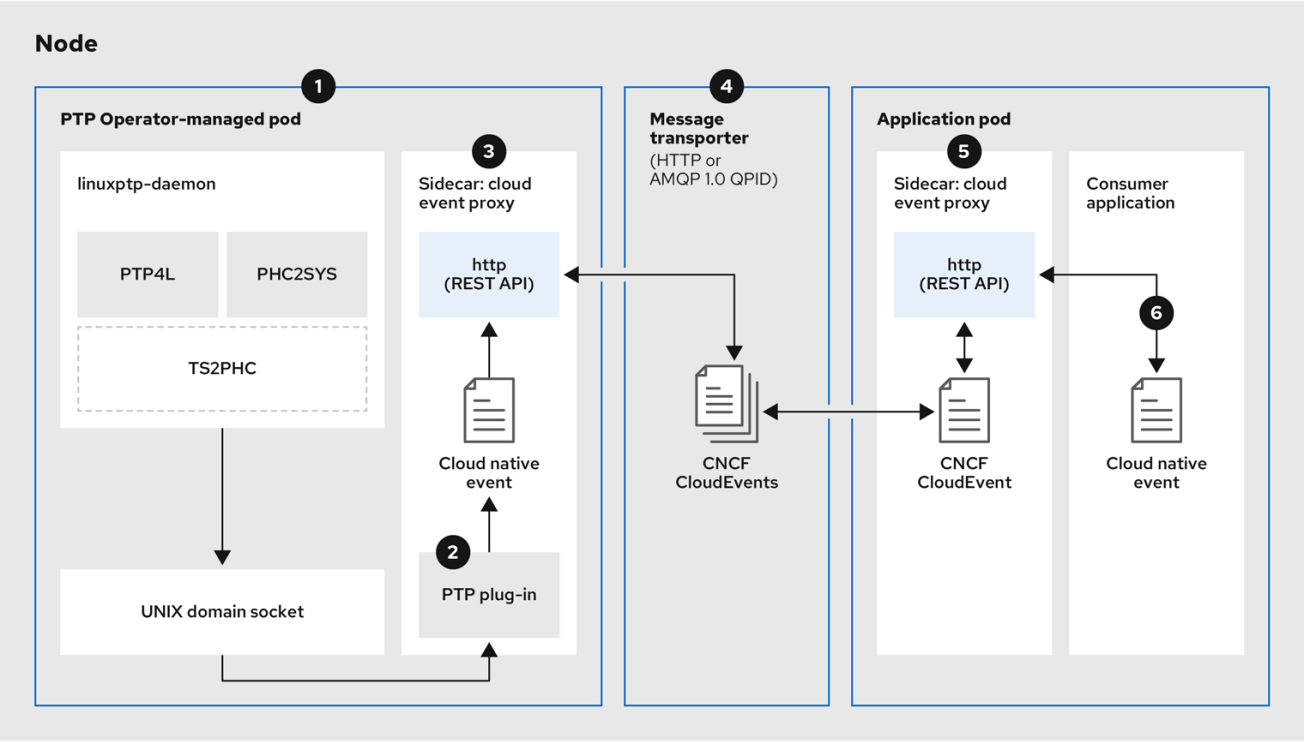
快速事件通知框架使用 REST API 进行通信。PTP 事件 REST API v1 和 v2 基于 *O-RAN O-Cloud Notification API Specification for Event Consumers 4.0*，它包括在 O-RAN ALLIANCE Specifications 中。

只有 PTP 事件 REST API v2 符合 O-RAN v4。

5.5.2. 使用 PTP 事件 REST API v1 检索 PTP 事件

应用程序以 sidecar 模式运行 **cloud-event-proxy** 容器，以订阅 PTP 事件。 **cloud-event-proxy** sidecar 容器可以访问与主应用程序容器相同的资源，而无需使用主应用程序的任何资源，且没有大量延迟。

图 5.7. 带有消费者 sidecar 和 HTTP 消息传输的 PTP 快速事件概述



319_OpenShift_0323

1 事件在集群主机上生成

PTP Operator 管理的 pod 中的 **linuxptp-daemon** 作为 Kubernetes **DaemonSet** 运行，并管理各种 **linuxptp** 进程 (**ptp4l**、**phc2sys**，以及可选的用于 grandmaster 时钟 **ts2phc**)。**linuxptp-daemon** 将事件传递给 UNIX 域套接字。

2 事件传递给 cloud-event-proxy sidecar

PTP 插件从 UNIX 域套接字读取事件，并将其传递给 PTP Operator 管理的 pod 中的 **cloud-event-proxy** sidecar。**cloud-event-proxy** 将 Kubernetes 基础架构的事件提供给具有低延迟的 Cloud-Native Network Function (CNF)。

3 事件是持久的

PTP Operator 管理的 pod 中的 **cloud-event-proxy** sidecar 处理事件，并使用 REST API 发布云原生事件。

4 消息已传输

消息传输程序通过 HTTP 将事件传送到应用程序 pod 中的 **cloud-event-proxy** sidecar。

5 来自 REST API 的事件

Application pod 中的 **cloud-event-proxy** sidecar 处理事件并使用 REST API 使其可用。

6 消费者应用程序请求订阅并接收订阅的事件

消费者应用程序向应用程序 pod 中的 **cloud-event-proxy** sidecar 发送 API 请求，以创建 PTP 事件订阅。**cloud-event-proxy** sidecar 为订阅中指定的资源创建一个 HTTP 消息传递监听程序协议。

应用程序 pod 中的 **cloud-event-proxy** sidecar 接收来自 PTP Operator 管理的 pod 的事件，取消封装云事件对象以检索数据，并将事件发布到消费者应用程序。消费者应用程序侦听资源限定符中指定的地址，并接收和处理 PTP 事件。

5.5.3. 配置 PTP 快速事件通知发布程序

要为集群中的网络接口启动使用 PTP fast 事件通知，您必须在 PTP Operator **PtpOperatorConfig** 自定义资源 (CR) 中启用快速事件发布程序，并在您创建的 **PtpConfig** CR 中配置 **ptpClockThreshold** 值。

先决条件

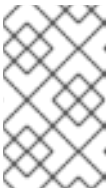
- 已安装 OpenShift Container Platform CLI (**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 已安装 PTP Operator。

流程

1. 修改默认 PTP Operator 配置以启用 PTP 快速事件。
   a. 在 **ptp-operatorconfig.yaml** 文件中保存以下 YAML：

```
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) 通过将 **enableEventPublisher** 设置为 **true** 来启用 PTP 快速事件通知。



注意

在 OpenShift Container Platform 4.13 或更高版本中，当将 HTTP 传输用于 PTP 事件时，您不需要在 **PtpOperatorConfig** 资源中设置 **spec.ptpEventConfig.transportHost** 字段。

   b. 更新 **PtpOperatorConfig** CR：

```
$ oc apply -f ptp-operatorconfig.yaml
```

2. 为 PTP 启用接口创建 **PtpConfig** 自定义资源(CR)，并设置 **ptpClockThreshold** 和 **ptp4lOpts** 所需的值。以下 YAML 演示了您必须在 **PtpConfig** CR 中设置的必要值：

```
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. 附加 **--summary_interval -4** 以使用 PTP 快速事件。
2. 所需的 **phc2sysOpts** 值。**-m** 将消息输出到 **stdout**。**linuxptp-daemon DaemonSet** 解析日志并生成 Prometheus 指标。
3. 指定一个字符串，其中包含要替换默认的 **/etc/ptp4l.conf** 文件的配置。要使用默认配置，请将字段留空。
4. 可选。如果 **ptpClockThreshold** 小节不存在，则默认值用于 **ptpClockThreshold** 字段。小节显示默认的 **ptpClockThreshold** 值。**ptpClockThreshold** 值配置 PTP master 时钟在触发 PTP 事件前的时长。**holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。**maxOffsetThreshold** 和 **minOffsetThreshold** 设置以纳秒为单位，它们与 **CLOCK_REALTIME** (**phc2sys**) 或 master 偏移 (**ptp4l**) 的值进行比较。当 **ptp4l** 或 **phc2sys** 偏移值超出这个范围时，PTP 时钟状态被设置为 **FREERUN**。当偏移值在这个范围内时，PTP 时钟状态被设置为 **LOCKED**。

其他资源

- 有关将 **linuxptp** 服务配置为具有 PTP 快速事件的普通时钟的完整示例 CR，请参阅 将 linuxptp 服务配置为普通时钟。

5.5.4. PTP 事件消费者应用程序参考

PTP 事件消费者应用程序需要以下功能：

1. 使用 **POST** 处理程序运行的 Web 服务，以接收云原生 PTP 事件 JSON 有效负载
2. 订阅 PTP 事件制作者的 **createSubscription** 功能
3. **getCurrentState** 功能轮询 PTP 事件制作者的当前状态

以下示例 Go 片断演示了这些要求：

Go 中的 PTP 事件消费者服务器功能示例

```
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)
  }
}
```

Go 中的 PTP 事件 createSubscription 功能示例

```
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) 将 **<node_name>** 替换为正在生成 PTP 事件的节点 FQDN。例如， **compute-1.example.com**。

Go 中的 PTP 事件消费者 getCurrentState 功能示例

```
//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)
  }
}
```

5.5.5. 引用 cloud-event-proxy 部署和服务 CR

在部署 PTP 事件消费者应用程序时，使用 **cloud-event-proxy** 部署和订阅者服务 CR 示例作为参考。

使用 HTTP 传输引用 cloud-event-proxy 部署

```
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
        - name: NODE_IP
          valueFrom:
            fieldRef:
              fieldPath: status.hostIP
          volumeMounts:
            - name: pubsubstore
              mountPath: /store
      ports:
        - name: metrics-port
          containerPort: 9091
        - name: sub-port
          containerPort: 9043
      volumes:
        - name: pubsubstore
          emptyDir: {}
```

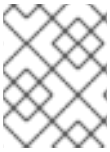
参考 cloud-event-proxy 订阅者服务

```
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
```

5.5.6. 使用 REST API v1 订阅 PTP 事件

在单独的应用程序 pod 中部署 **cloud-event-consumer** 应用程序容器和 **cloud-event-proxy** sidecar 容器。

将 **cloud-event-consumer** 应用程序订阅到 PTP 事件，这些事件由 **cloud-event-proxy** 容器发布，位于应用程序 pod 中的 **http://localhost:8089/api/ocloudNotifications/v1/**。



注意

9089 是在应用程序 Pod 中部署的 **cloud-event-consumer** 容器的默认端口。您可以根据需要为应用程序配置不同的端口。

其他资源

- api/ocloudNotifications/v1/subscriptions

5.5.7. 验证 PTP 事件 REST API v1 消费者应用程序是否收到事件

验证应用程序 pod 中的 **cloud-event-proxy** 容器是否接收 PTP 事件。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已以具有 **cluster-admin** 权限的用户身份登录。
- 已安装并配置了 PTP Operator。

流程

1. 获取活跃的 **linuxptp-daemon** pod 列表。运行以下命令:

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

输出示例

```
NAME                 READY   STATUS    RESTARTS   AGE
linuxptp-daemon-2t78p   3/3     Running   0          8h
linuxptp-daemon-k8n88   3/3     Running   0          8h
```

2. 运行以下命令，访问所需的消费者端的 **cloud-event-proxy** 容器的指标：

```
$ oc exec -it <linuxptp-daemon> -n openshift-ptp -c cloud-event-proxy -- curl
127.0.0.1:9091/metrics
```

其中：

<linuxptp-daemon>
指定您要查询的 pod，例如 **linuxptp-daemon-2t78p**。

输出示例

```
# 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
```

5.5.8. 监控 PTP 快速事件指标

您可以从运行 **linuxptp-daemon** 的集群节点监控 PTP 快速事件指标。您还可以使用预先配置和自我更新的 Prometheus 监控堆栈来监控 OpenShift Container Platform Web 控制台中的 PTP 快速事件指标。

先决条件

- 安装 OpenShift Container Platform CLI **oc**。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 在具有 PTP 功能硬件的节点上安装和配置 PTP Operator。

流程

1. 运行以下命令，为节点启动 debug pod：

```
$ oc debug node/<node_name>
```

2. 检查 **linuxptp-daemon** 容器公开的 PTP 指标。例如，运行以下命令：

```
sh-4.4# curl http://localhost:9091/metrics
```

输出示例

```
# 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. 可选。您还可以在 **cloud-event-proxy** 容器的日志中找到 PTP 事件。例如，运行以下命令：

```
$ oc logs -f linuxptp-daemon-cvgr6 -n openshift-ptp -c cloud-event-proxy
```

4. 要在 OpenShift Container Platform web 控制台中查看 PTP 事件，请复制您要查询的 PTP 指标的名称，如 **openshift_ptp_offset_ns**。
5. 在 OpenShift Container Platform web 控制台中点 **Observe → Metrics**。
6. 将 PTP 指标名称粘贴到 **Expression** 字段中，然后点 **Run query**。

其他资源

- 以开发者身份访问指标

5.5.9. PTP 快速事件指标参考

下表描述了运行 **linuxptp-daemon** 服务的集群节点可用的 PTP 快速事件指标。

表 5.21. PTP 快速事件指标

指标	描述	Example
openshift_ptp_clock_class	返回接口的 PTP 时钟类。对于 PTP 时钟类的可能值为：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**), 或 255 (**SLAVE ONLY CLOCK**)。	**{node="compute-1.example.com",process="ptp4l"} 6**
openshift_ptp_clock_state	返回接口的当前 PTP 时钟状态。PTP 时钟状态的可能值为 **FREERUN**、**LOCKED** 或 **HOLDOVER**。	**{iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} 1**
openshift_ptp_delay_ns	返回主时钟发送计时数据包和接收计时数据包之间的延迟（以纳秒为单位）。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 0**
openshift_ptp_ha_profile_status	当不同 NIC 上有多个时间源时，返回高可用性系统时钟的当前状态。可能的值为 0 (**INACTIVE**) 和 1 (**ACTIVE**)。	**{node="node1",process="phc2sys",profile="profile1"} 1{node="node1",process="phc2sys",profile="profile2"} 0**
openshift_ptp_frequency_adjustment_ns	以纳秒为单位返回 2 PTP 时钟之间的频率调整。例如，在上游时钟和 NIC 之间，系统时钟和 NIC 之间，或在 PTP 硬件时钟(**phc**)和 NIC 之间。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -6768**
openshift_ptp_interface_role	返回为接口配置的 PTP 时钟角色。可能的值包括 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	返回 2 时钟或接口之间的最大偏移量（以纳秒为单位）。例如，在上游 GNSS 时钟和 NIC (**ts2phc**)之间，或在 PTP 硬件时钟(**phc**)和系统时钟(**phc2sys**)之间。	**{from="master", iface="ens2fx", node="compute-1.example.com", process="ts2phc"} 1.038099569e+09**
openshift_ptp_offset_ns	返回 DPLL 时钟或 GNSS 时钟源和 NIC 硬件时钟之间的偏移量。	**{from="phc", iface="CLOCK_REALTIME", node="compute-1.example.com", process="phc2sys"} -9**

指标	描述	Example
openshift_ptp_process_restart_count	返回 **ptp4l** 和 **ts2phc** 进程重启的次数。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_process_status	返回显示 PTP 进程是否正在运行的状态代码。	**{config="ptp4l.0.config", node="compute-1.example.com",process="phc2sys"} 1**
openshift_ptp_threshold	为 **HoldOverTimeout**, **MaxOffsetThreshold**, 和 **MinOffsetThreshold** 返回值。 • **holdOverTimeout** 是在 PTP master clock 断开连接时，PTP 时钟事件状态更改为 **FREERUN** 前的时间值（以秒为单位）。 • **maxOffsetThreshold** 和 **minOffsetThreshold** 是以纳秒为单位的偏移值，它比较 **CLOCK_REALTIME** (**phc2sys**) 的值，或您在 **PtpConfig** CR 中为 NIC 配置的 r master offset (**ptp4l**) 的值。	**{node="compute-1.example.com", profile="grandmaster", threshold="HoldOverTimeout"} 5**

5.5.9.1. 只有在启用 T-GM 时，PTP 快速事件指标

下表描述了仅在启用 PTP grandmaster 时钟 (T-GM) 时可用的 PTP 快速事件指标。

表 5.22. 启用 T-GM 时的 PTP 快速事件指标

指标	描述	Example
openshift_ptp_frequency_status	返回 NIC 的数字阶段锁定循环(DPLL)频率的当前状态。可能的值为 -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), 或 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**
openshift_ptp_nmea_status	返回 NMEA 连接的当前状态。NMEA 是 1PPS NIC 连接使用的协议。可能的值有 0 (**UNAVAILABLE**) 和 1 (**AVAILABLE**)。	**{iface="ens2fx",node="compute-1.example.com",process="ts2phc"} 1**
openshift_ptp_phase_status	返回 NIC 的 DPLL 阶段的状态。可能的值为 -1 (**UNKNOWN**), 0 (**INVALID**), 1 (**FREERUN**), 2 (**LOCKED**), 3 (**LOCKED_HO_ACQ**), 或 4 (**HOLDOVER**).	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 3**

指标	描述	Example
openshift_ptp_pps_status	返回 NIC 1PPS 连接的当前状态。您可以使用 1PPS 连接在连接的 NIC 之间同步计时。可能的值有 0 (**UNAVAILABLE**) 和 1 (**AVAILABLE**)。	**{from="dpll",iface="ens2fx",node="compute-1.example.com",process="dpll"} 1**
openshift_ptp_gnss_status	返回全局导航 Satellite 系统(GNSS)连接的当前状态。GNSS 在全局范围内提供基于 satellite 的位置、导航和计时服务。可能的值包括 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**

5.6. PTP EVENTS REST API V1 参考

使用以下 Precision Time Protocol (PTP) 快速事件 REST API v1 端点，将 **cloud-event-consumer** 应用程序订阅到 PTP 事件，由 **cloud-event-proxy** 容器发布，位于应用程序 pod 中的 **http://localhost:8089/api/ocloudNotifications/v1/**。

重要

PTP 事件 REST API v1 和事件消费者应用程序 sidecar 是一个已弃用的功能。弃用的功能仍然包含在 OpenShift Container Platform 中，并将继续被支持。但是，这个功能会在以后的发行版本中被删除，且不建议在新的部署中使用。

有关 OpenShift Container Platform 中已弃用或删除的主要功能的最新列表，请参阅 OpenShift Container Platform 发行注记中 *已弃用和删除的功能*部分。

可用的 API 端点如下：

- **api/ocloudNotifications/v1/subscriptions**
  - **POST** ：创建新订阅
  - **GET** ：删除订阅列表
  - **DELETE** ：删除所有订阅
- **api/ocloudNotifications/v1/subscriptions/{subscription_id}**
  - **GET** ：返回指定订阅 ID 的详情
  - **DELETE** ：删除与指定订阅 ID 关联的订阅
- **api/ocloudNotifications/v1/health**
  - **GET**：返回 **ocloudNotifications** API 的健康状况
- **api/ocloudNotifications/v1/publishers**

- **GET**：返回集群节点的 PTP 事件发布程序列表

- **api/ocloudnotifications/v1/{resource_address}/CurrentState**
  - **GET**：返回以下事件类型的当前状态： **sync-state**, **os-clock-sync-state**, **clock-class**, **lock-state**, 或 **gnss-sync-status** 事件

5.6.1. PTP 事件 REST API v1 端点

5.6.1.1. api/ocloudNotifications/v1/subscriptions

HTTP 方法
GET api/ocloudNotifications/v1/subscriptions

描述
返回订阅列表。如果订阅存在，则返回 **200 OK** 状态代码以及订阅列表。

API 响应示例

```
[
 {
  "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 方法
POST api/ocloudNotifications/v1/subscriptions

描述
通过传递适当的有效负载来为所需的事件创建新订阅。如果订阅成功创建，或者已存在，则返回 **201 Created** 状态代码。您可以订阅以下 PTP 事件：

- **lock-state** 事件
- **os-clock-sync-state** 事件
- **clock-class** 事件
- **gnss-sync-status** 事件
- **sync-state** 事件

表 5.23. 查询参数

参数	类型
subscription	data

PTP 事件订阅有效负载示例

```
{
  "endpointUri": "http://localhost:8989/event",
  "resource": "/cluster/node/compute-1.example.com/ptp"
}
```

PTP lock-state 事件订阅有效负载示例

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/ptp-status/lock-state"
}
```

PTP os-clock-sync-state 事件订阅有效负载示例

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/sync-status/os-clock-sync-state"
}
```

PTP clock-class 事件订阅有效负载示例

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/ptp-status/clock-class"
}
```

PTP gnss-sync-status 事件订阅有效负载示例

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/gnss-status/gnss-sync-status"
}
```

sync-state 订阅有效负载示例

```
{
"endpointUri": "http://localhost:8989/event",
"resource": "/cluster/node/{node_name}/sync/sync-status/sync-state"
}
```

HTTP 方法
DELETE api/ocloudNotifications/v1/subscriptions

描述
删除所有订阅。

API 响应示例

```
{
"status": "deleted all subscriptions"
}
```

5.6.1.2. api/ocloudNotifications/v1/subscriptions/{subscription_id}

HTTP 方法
GET api/ocloudNotifications/v1/subscriptions/{subscription_id}

描述
返回 ID 为 **subscription_id** 的订阅详情。

表 5.24. 全局路径参数

参数	类型
subscription_id	string

API 响应示例

```
{
  "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 方法
DELETE api/ocloudNotifications/v1/subscriptions/{subscription_id}

描述
使用 ID **subscription_id** 删除订阅。

表 5.25. 全局路径参数

参数	类型
subscription_id	string

API 响应示例

```
{
"status": "OK"
}
```

5.6.1.3. api/ocloudNotifications/v1/health

HTTP 方法
GET api/ocloudNotifications/v1/health/

描述
返回 **ocloudNotifications** REST API 的健康状况。

API 响应示例

OK

5.6.1.4. api/ocloudNotifications/v1/publishers

重要

api/ocloudNotifications/v1/publishers 端点仅对于 PTP Operator 管理的 pod 中的 cloud-event-proxy 容器可用。它对于应用程序 Pod 中的消费者应用程序不可用。

HTTP 方法
GET api/ocloudNotifications/v1/publishers

描述
返回集群节点的发布者详情列表。当相关的设备状态改变时，系统会生成通知。

您可以组合使用设备同步状态订阅，以提供有关系统总体同步健康状况的详细视图。

API 响应示例

```
[
  {
    "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"
  }
]
```

5.6.1.5. api/ocloudNotifications/v1/{resource_address}/CurrentState

HTTP 方法

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

描述
返回集群节点的 **os-clock-sync-state**,**clock-class**,**lock-state**,**gnss-sync-status**, 或 **sync-state** 事件的当前状态。

- **os-clock-sync-state** 通知描述了主机操作系统时钟同步状态。可以是 **LOCKED** 或 **FREERUN** 状态。
- **clock-class** 通知描述了 PTP 时钟类的当前状态。
- **lock-state** 通知描述了 PTP 设备锁定状态的当前状态。可以处于 **LOCKED**、**HOLDOVER** 或 **FREERUN** 状态。
- **sync-state** 通知描述了至少 **ptp-status/lock-state** 和 **sync-status/os-clock-sync-state** 端点同步的当前状态。
- **GNSS-sync-status** 通知描述了 GNSS 时钟同步状态。

表 5.26. 全局路径参数

参数	类型
resource_address	string

lock-state API 响应示例

```
{
  "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"
      }
    ]
  }
}
```

os-clock-sync-state API 响应示例

```
{
  "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"
      }
    ]
  }
}
```

clock-class API 响应示例

```
{
  "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"
      }
    ]
  }
}
```

sync-state API 响应示例

```
{
   "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"
      }]
   }
}
```

gnss-sync-status API 响应示例

```
{
  "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"
      }
    ]
  }
}
```