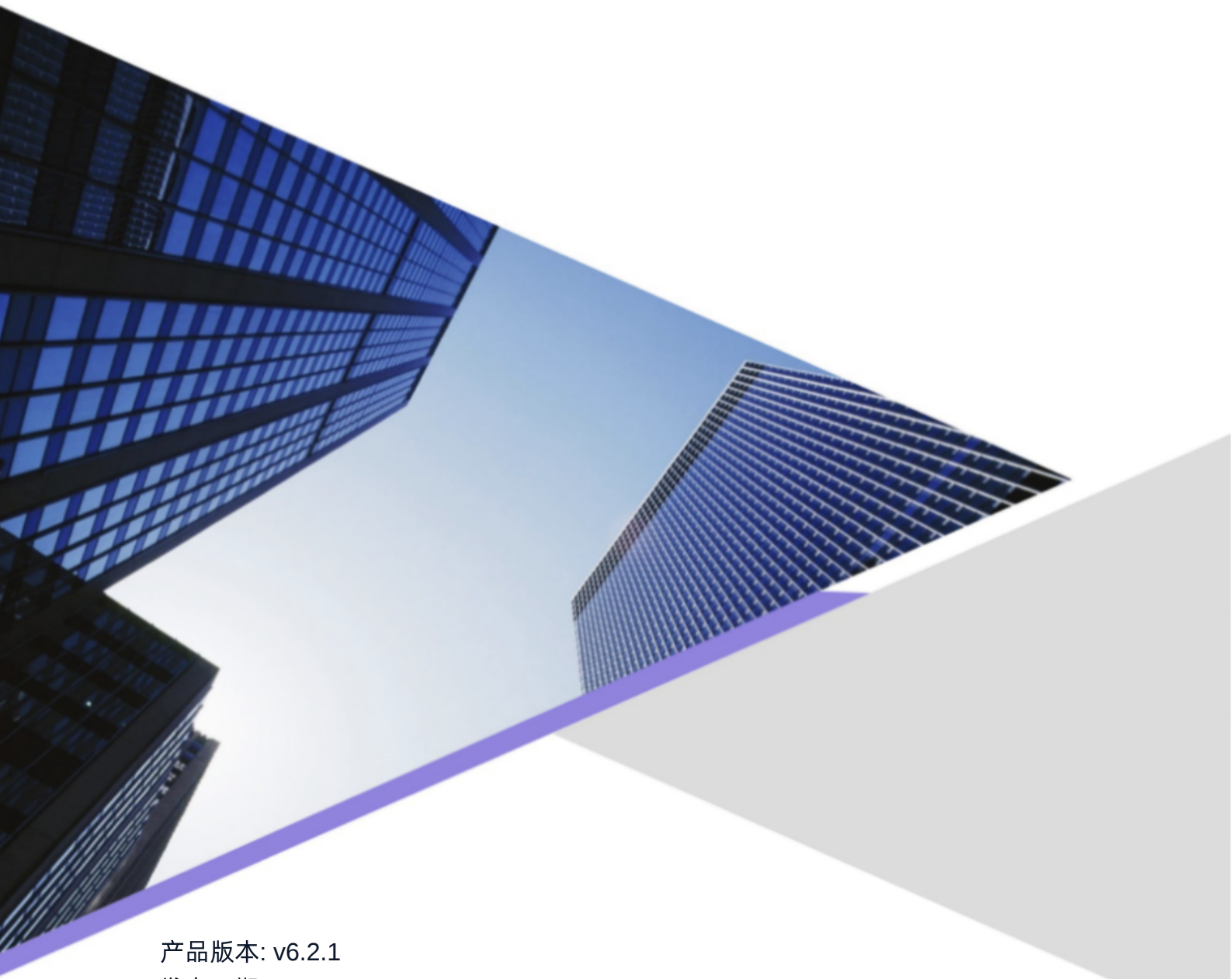
EasyStack
易捷行云

安全容器服务

最佳实践



产品版本: v6.2.1

发布日期: 2024-06-05

目录

1 最佳实践

1.1 配置SR-IOV网络

1.1.1 前置条件准备

为了使用SR-IOV功能，需要满足以下前提条件：

1. 服务器BIOS中已开启SR-IOV功能。
2. 启用Intel IOMMU功能，并以Pass-Through模式进行配置。
3. 已安装NVIDIA MLNX_OFED驱动。
4. 网卡固件启用SR-IOV功能。
5. 在MLNX_OFED驱动上启用SR-IOV。

这些前提条件的满足将确保系统具备使用SR-IOV功能所需的硬件支持和驱动程序。请确保在使用SR-IOV功能之前，在执行后续的配置操作之前，先完成以下准备工作：

操作步骤

1.服务器BIOS中已开启SR-IOV功能

请确保服务器BIOS中已开启SR-IOV功能。每个服务器都有不同的虚拟化BIOS配置选项。BIOS configuration examples可参考：BIOS configuration examples

2.服务器BIOS中已开启SR-IOV功能

请确保将"**intel_iommu=on**"和"**iommu=pt**"添加到grub文件的配置中。

```
# cat /boot/grub2/grub.cfg

# GRUB Environment Block
saved_entry=0
kernelopts=root=/dev/mapper/os-root ro edd=off kvm.halt_poll_ns=400000
cgroup.memory=nokmem intel_iommu=on iommu=pt pci=realloc
```

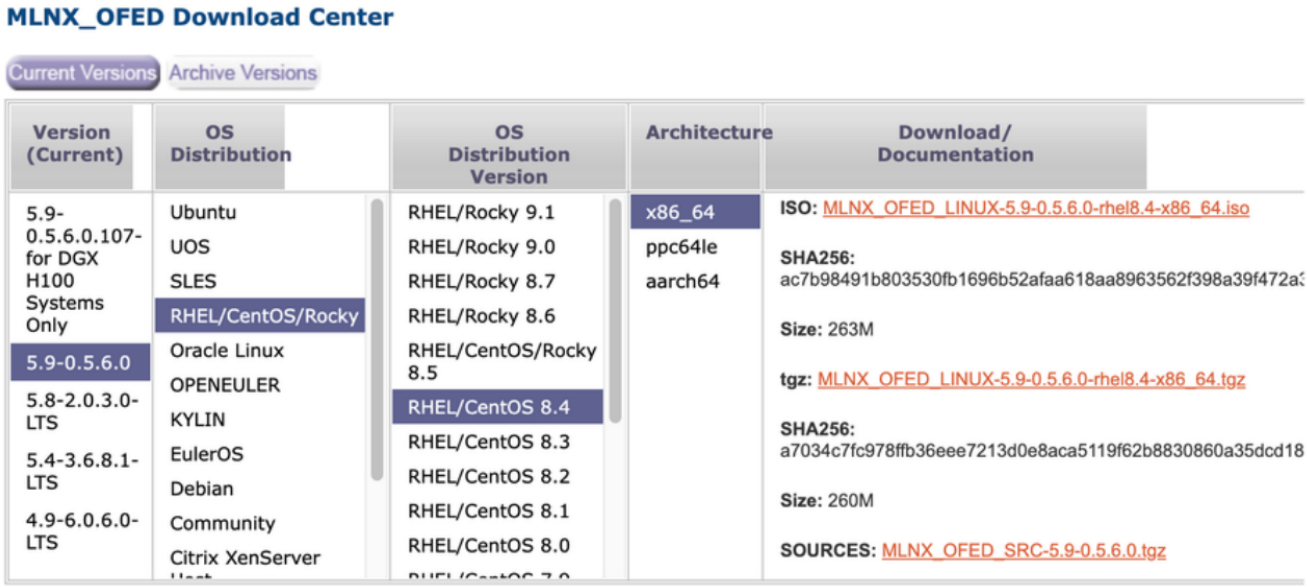
```
ixgbe.allow_unsupported_sfp=1 rootdelay=90 nomodeset intel_idle.max_cstate=0
processor.max_cstate=0 crashkernel=300M rd.lvm.lv=os/root biosdevname=0
net.ifnames=1
boot_success=0
```

要了解更多关于iommu grub参数的信息，请参阅:Understanding the iommu Linux grub File Configuration

3.安装NVIDIA MLNX_OFED驱动

3.1 下载驱动包

NVIDIA驱动官方下载：Linux InfiniBand Drivers 请根据您所选择的操作系统和版本，以及CPU架构，下载相应的tar包。例如，选择MLNX_OFED_LINUX-5.9-0.5.6.0-rhel8.4-x86_64.tgz进行下载。



3.2 执行安装操作

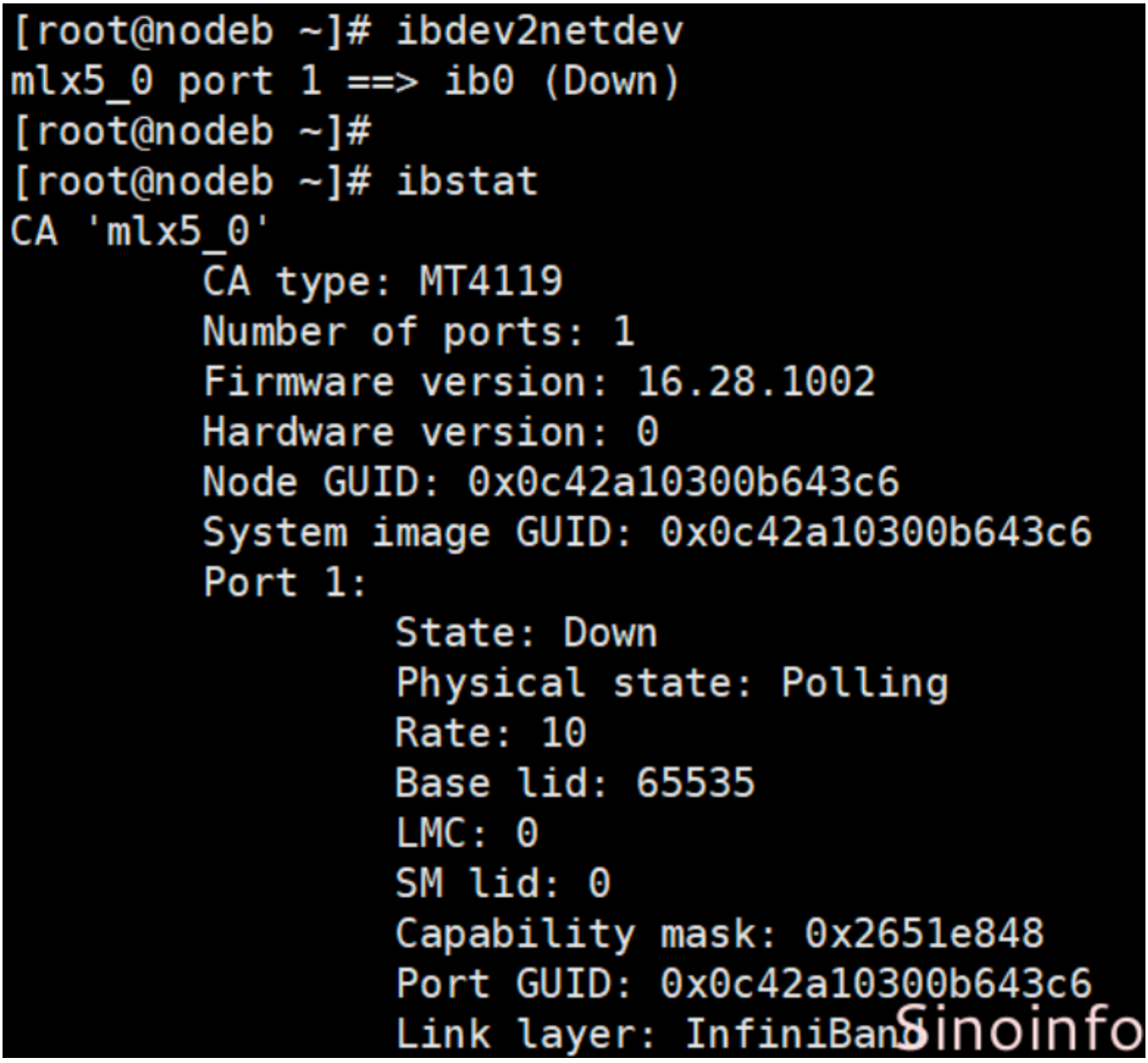
```
# ./mlnxofedinstall
```

3.3 查看驱动版本信息

```
# ofed_info -s
MLNX_OFED_LINUX-5.9-0.5.6.0:
```

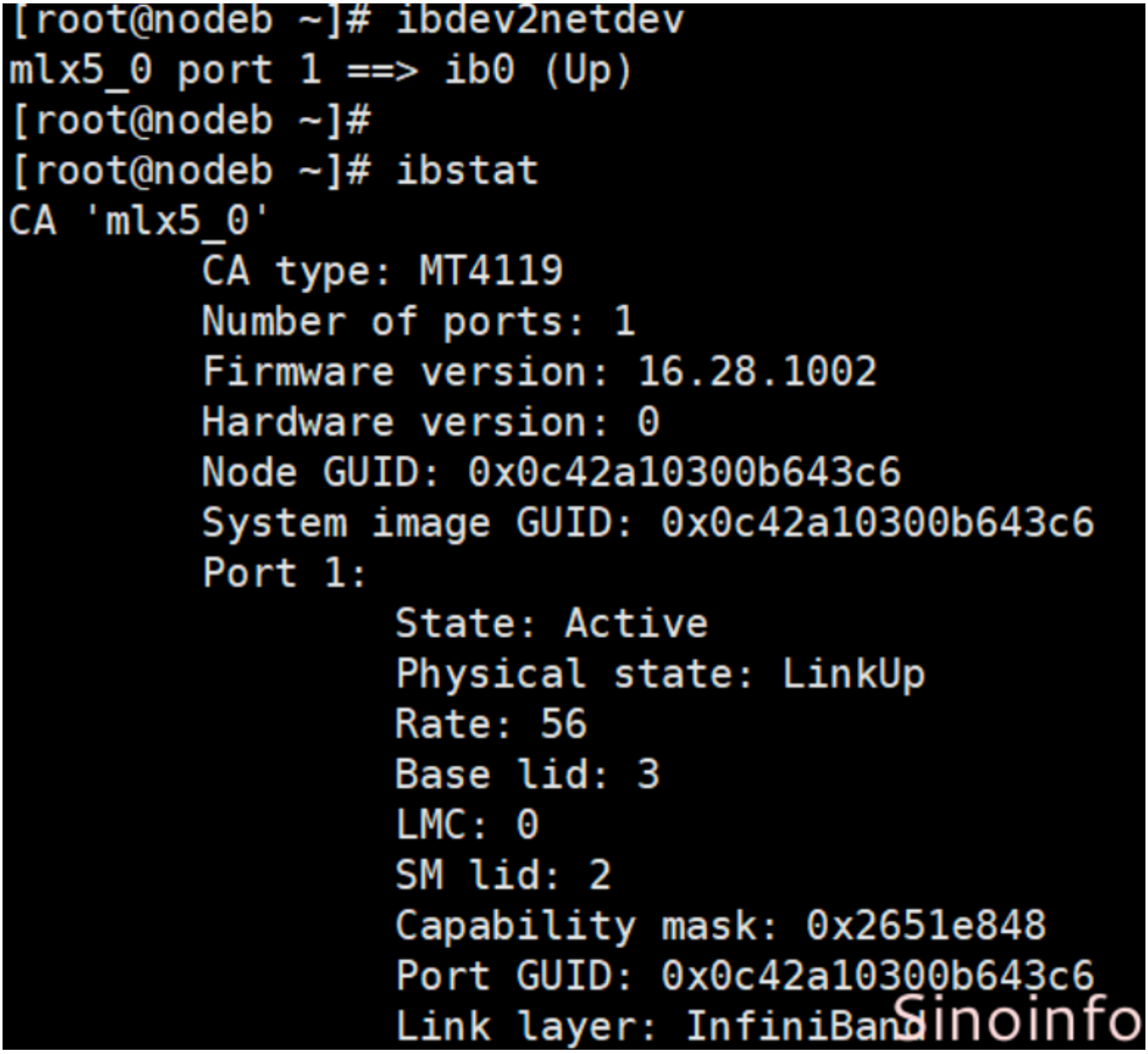
3.4 网卡状态

3.4.1 首次安装网卡时，如果未连接到交换机，网卡的状态将如下所示：



（如上图，ib0状态Down（双口卡还有ib1），网卡的SM lid为0，Base lid为65535，Link layer为IB模式）

3.4.2 使用线缆将网卡连接到交换机，大概30秒，端口UP，如下所示：



（如上图，网卡ib0物理状态LinkUp, 逻辑状态Active，子网管理器SM lid为 2，网卡Base lid为 3）

> 说明：
> 本案例交换机SB7800已经开启子网管理器功能，交换机端口速率为EDR 100Gb/s，网卡CX5也是 EDR 100Gb/s，由于线缆是FDR 56G，所以上图的Rate：56，并没有达到100

3.4.3 启动opensmd服务（视交换机型号选装）

如果交换机是SB7890或者是其它不带子网管理器功能的交换机（SB7890，SB7790，SX6025，IS5025），必须在服务器上开启子网管理器， 安装好网卡驱动后就可以使用下面命令：

```
# systemctl start opensmd
# systemctl enable opensmd
```

3.4.4 网卡运行环境监测

完成以上步骤，使用命令'mlnx_tune'检查，所有状态都正常，则网卡运行环境正常。

```
# mlnx_tune
```

4.网卡固件启用SR-IOV功能

4.1 下载Mellanox Management Tools (MFT)工具

下载链接：Mellanox Management Tools (MFT)

4.2 使用Mellanox Firmware Tools包在固件中启用和配置SR-IOV

```
# mst start
Starting MST (Mellanox Software Tools) driver set
Loading MST PCI module - Success
Loading MST PCI configuration module - Success
Create devices
```

4.2.1 在需要的PCI插槽上找到Connect-IB设备

```
# mst status
MST modules:
------------
    MST PCI module loaded
    MST PCI configuration module loaded
MST devices:
------------
/dev/mst/mt4115_pciconf0         - PCI configuration cycles access.
...
```

4.2.2 查询设备状态

```
# mlxconfig -d /dev/mst/mt4115_pciconf0 q
```

4.2.3 开启SR-IOV，设置可以切分的vf数量上限

```
# mlxconfig -d /dev/mst/mt4115_pciconf0 set SRIOV_EN=1 NUM_OF_VFS=8
...
Apply new Configuration? ? (y/n) [n] : y
Applying... Done!
-I- Please reboot machine to load new configurations.
```

4.2.4 设置网卡的工作模式。

Infiniband卡支持两种工作模式：IB模式和Ethernet模式

```
# mlxconfig -d /dev/mst/mt4115_pciconf0 q

Configurations:                                Next Boot
         ...
         LINK_TYPE_P1                          IB(1)    #当前工作模式 IB
         ...
```

修改网卡的工作模式：

```
Ethernet模式:  mlxconfig -d /dev/mst/mt4115_pciconf0 set LINK_TYPE_P1=2
IB模式:  mlxconfig -d /dev/mst/mt4115_pciconf0 set LINK_TYPE_P1=1
```

重启机器生效

```
# reboot
```

> 此时，通过lspci无法看到vf。只有当SR-IOV在MLNX_OFED驱动上启用时，您才能看到它们。

4.2.5 查看节点物理网卡是否支持SR-IOV

```
查看所有的pci设备
# lspci -nnn |grep -i mellanox
```

```
61:00.0 Infiniband controller [0207]: Mellanox Technologies MT27700 Family
[ConnectX-4] [15b3:1013]

查看设备详情
# lspci -vvs
61:00.0 Infiniband controller: Mellanox Technologies MT27700 Family
[ConnectX-4]
    ...
    Capabilities: [180 v1] Single Root I/O Virtualization (SR-IOV)
        IOVCap: Migration-, Interrupt Message Number: 000
        IOVCtl: Enable- Migration- Interrupt- MSE- ARIHierarchy+
        IOVSta: Migration-
        Initial VFs: 3, Total VFs: 3, Number of VFs: 0, Function Dependency
Link: 00
        VF offset: 1, stride: 1, Device ID: 1014
        Supported Page Size: 000007ff, System Page Size: 00000001
        Region 0: Memory at 000005464e000000 (64-bit, prefetchable)
        VF Migration: offset: 00000000, BIR: 0
    Capabilities: [1c0 v1] Secondary PCI Express
        LnkCtl3: LnkEquIntrruptEn- PerformEqu-
        LaneErrStat: 0
    Kernel driver in use: mlx5_core
    Kernel modules: mlx5_core
```

显示如下代码段，说明固件已启动SR-IOV功能

5.在MLNX_OFED驱动上启用SR-IOV

5.1 定位设备（通常为 'mlx5_0'）

```
# ibstat
CA 'mlx5_0'
    CA type: MT4115
    Number of ports: 1
    Firmware version: 12.28.2006
    Hardware version: 0
    Node GUID: 0xe41d2d03006768fa
    System image GUID: 0xe41d2d03006768fa
    Port 1:
```

```
        State: Down
        Physical state: Disabled
        Rate: 10
        Base lid: 65535
        LMC: 0
        SM lid: 0
        Capability mask: 0x2650e848
        Port GUID: 0xe41d2d03006768fa
        Link layer: InfiniBand
```

5.2 获取此设备上的当前vf数

```
# cat /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
0
```

> 如果命令失败，这可能意味着驱动程序没有加载。

5.3 设置所需的vf数量

```
# echo 2 > /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
```

> 更改mlx5_num_vfs不是持久的，并且不能在服务器重新启动后继续存在。

5.4 检查PCI总线

```
# lspci -nnn |grep -i mellanox
61:00.0 Infiniband controller [0207]: Mellanox Technologies MT27700 Family
[ConnectX-4] [15b3:1013]
61:00.1 Infiniband controller [0207]: Mellanox Technologies MT27700 Family
[ConnectX-4 Virtual Function] [15b3:1014]
61:00.2 Infiniband controller [0207]: Mellanox Technologies MT27700 Family
[ConnectX-4 Virtual Function] [15b3:1014]
```

5.5 恢复vf数量为0

```
# echo 0 > /sys/class/infiniband/mlx5_0/device/mlx5_num_vfs
```

1.1.2 使用Yaml配置SR-IOV网络

背景描述

SR-IOV (Single Root I/O Virtualization) 是一种允许物理设备如网络接口卡(NIC)在没有软件的情况下将其资源分割成多个虚拟设备的技术。使用SR-IOV可以在性能和资源隔离方面提供显著的优势，因为它减少了网络数据在主机和虚拟机之间传输的开销。安全容器服务支持SR-IOV网络设备插件，用于发现、公告和分配SR-IOV网络虚拟功能 (VF) 资源。本章节通过使用Yaml创建功能，配置SR-IOV网络。

前置条件

为了使用SR-IOV功能，需要满足以下前提条件：

1. 服务器BIOS中已开启SR-IOV功能。
2. 启用Intel IOMMU功能，并以Pass-Through模式进行配置。
3. 已安装NVIDIA MLNX_OFED驱动。
4. 网卡固件启用SR-IOV功能。
5. 在MLNX_OFED驱动上启用SR-IOV。 详细操作步骤请参见前置条件。这些前提条件的满足将确保系统具备使用SR-IOV功能所需的硬件支持和驱动程序。请确保在使用SR-IOV之前，你的系统已按照要求进行相应的设置和安装。

操作步骤

前置操作完成后，需重启环境中sriov-config-daemon的Pod

因为 `sriov-config-daemon` 是 `daemonSet` 部署，则删除 `namespace` 为 `eks-managed` ，label为 `app=sriov-network-config-daemon` 的Pod即可。

创建SriovNetworkNodePolicy对象：

SriovNetworkNodePolicy是SR-IOV Network Operator的一部分，用于定义如何配置SR-IOV网络。它是一个Kubernetes自定义资源(CR)，用于指定SR-IOV网络配置策略。 **可以通过定义SriovNetworkNodePolicy对象来指定节点的SR-IOV网络设备配置：**

1.查看节点SR-IOV设备信息 节点sriov-config-daemon上报上来的SR-IOV设备情况会更新到节点对应的 `sriovnetworknodestates.status` ，即查看 `sriovnetworknodestates.status` 内容

```
# kubectl get sriovnetworknodestates.sriovnetwork.openshift.io  -n eks-managed -o yaml
```

```
status:
  interfaces:
  - deviceID: "1013"
    driver: mlx5_core
    linkType: IB
    mac: 00:00:07:ff:fe:80:00:00:00:00:00:00:e4:1d:2d:03:00:67:68:fa
    mtu: 2048
    name: ib0
    pciAddress: 0000:61:00.0
    totalvfs: 3
    vendor: 15b3
 syncStatus: Succeeded
```

2.根据节点SR-IOV设备情况，创建SriovNetworkNodePolicy资源。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name:  1
  namespace: eks-managed
spec:
  resourceName:  2
  nodeSelector:
    : 3
  priority:  3
  mtu:  4
  numVfs:  5
  nicSelector: 6
    vendor: "" 7
    deviceID: "" 8
    pfNames: ["", ...] 9
    rootDevices: ["", "..."] 10
```

```
  deviceType: vfio-pci
  isRdma: false 11
```

参数	说明
name	CR对象的名称。
resourceName	SR-IOV设备插件的资源名称。您可以为一个资源名称创建多个"SriovNetworkNodePolicy"对象。
priority（可选）	"0"到"99"之间的整数。较小的数值具有较高的优先权，优先级"10"高于优先级"99"。默认值为"99"。
mtu（可选）	为虚拟功能（VF）的最大传输单位 (MTU) 指定一个值。MTU的值必须是在"1"到"9000"之间。如果您不需要指定MTU，请指定一个"值。
numVfs	为SR-IOV物理网络设备指定要创建的虚拟功能 (VF) 的数量。对于Intel网络接口卡 (NIC)，VF的数量不能超过该设备支持的VF总数。对于Mellanox NIC，VF的数量不能超过"128"。
nicSelector	nicSelector 映射为Operator选择要配置的设备。您不需要为所有参数指定值。建议您以足够的准确度来识别网络设备，以便尽量减小意外选择其他网络设备的可能性。如果指定了rootDevices，则必须同时为vendor、deviceID或pfNames指定一个值。如果同时指定了"pfNames"和"rootDevices"，请确保它们指向同一个设备。
vendor（可选）	SR-IOV网络设备的厂商十六进制代码。允许的值只能是"8086"和"15b3"。
deviceID（可选）	SR-IOV网络设备的设备十六进制代码。允许的值只能是"158b"、"1015"和"1017"。
pfNames（可选）	该设备的一个或多个物理功能（PF）名称的数组。
rootDevices	用于该设备的PF的一个或多个PCI总线地址的数组。使用以下格式提供地址: "0000:02:00.1"。
isRdma（可选）	是否启用远程直接访问（RDMA）模式。默认值为"false"。

> 说明：
> 写**sriovNetworkNodePolicy**的 `spec` 的定义，要根据 `sriovNetworkNodeState.Status.interface` 去写。 若**sriovNetworkNodePolicy**的 `spec` 定义，不在 `sriovNetworkNodeState.Status.interface` 的设备范围内，则生成只有 `DpConfigVersion` 的 `spec` 。

创建 SriovIBNetwork资源：

您可以通过定义对象来配置 InfiniBand (IB) 网络设备"SriovIBNetwork"。

1.下方示例Yaml展示了如何通过 SriovIBNetwork 对象，配置 Infiniband network attachment

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name:  1
  namespace: eks-managed
spec:
  resourceName:  2
  networkNamespace:  3
  vlan:  4
  spoofChk: "" 5
  ipam: |- 6
    {}
  linkState:  7
  maxTxRate:  8
  minTxRate:  9
  vlanQoS:  10
  trust: "" 11
  capabilities:  12
```

参数	说明
name	对象的名称。SR-IOV Network Operator创建一个名称相同的"NetworkAttachment Definition"对象。

参数	说明
resourceName	定义附加网络的SR-IOV硬件"SriovNetworkNodePolicy"对象中的"spec.resourceName"参数的值。
networkNamespace	SriovIBNetwork对象的目标命名空间。只有目标命名空间中的Pod才能连接到网络设备。
vlan（可选）	额外网络的虚拟LAN（VLAN）ID。它需要是一个从"0"到"4095"范围内的一个整数值。默认值为"0"。
spoofChk（可选）	VF的"spoof"检查模式。允许的值是字符串"on"和"off"。 重要：指定的值必须由引号包括，否则SR-IOV Network Operator将拒绝对象。
ipam	为IPAM CNI插件指定一个配置对象做为一个Yaml块scalar。该插件管理网络附加定义的IP地址分配。
linkState（可选）	虚拟功能（VF）的链接状态。允许的值是"enable"、"disable"和"auto"。
maxTxRate（可选）	VF的最大传输率（以Mbps为单位）。
minTxRate（可选）	VF的最低传输率（以Mbps为单位）。这个值必须小于或等于最大传输率。 说明:Intel NIC不支持minTxRate参数。如需更多信息，请参阅 BZ#1772847
vlanQoS（可选）	VF的"IEEE 802.1p"优先级级别。默认值为"0"。
trust（可选）	VF的信任模式。允许的值是字符串"on"和"off"。 说明:您必须在引号中包含指定的值，或者SR-IOV Network Operator拒绝对象。
capabilities（可选）	为这个额外网络配置功能。您可以指定 "{ "ips": true }" 来启用IP地址支持，或指定 "{ "mac": true }" 来启用MAC地址支持。

2.使用Whereabouts进行动态IP地址分配配置

```
{
 "ipam": {
 "type": "whereabouts",
```

```
 "range": "", 1
 "exclude": [", ..."], 2
 }
}
```

参数	说明
range	以CIDR标记指定一个IP地址范围。IP地址是通过这个地址范围来分配的。
exclude（可选）	在CIDR标记中指定IP地址和范围列表。包含在排除地址范围中的IP地址。

创建业务容器（Pod）资源：

指定业务容器的规范，包括容器运行时（如runc、rune）、容器镜像、资源需求（如CPU和内存）、环境变量、命令等。

```
apiVersion: v1
kind: Pod
metadata:
  name: sriov-rune-pod
  annotations:
    io.katacontainers.config.runtime.enable_sriov: "true"
    k8s.v1.cni.cncf.io/networks: eks-managed/nics
    #v1.multus-cni.io/default-network: eks-managed/kube-ovn
spec:
  runtimeClassName: rune
  containers:
  - name: appcntr1
    image: hub.easystack.io/captain/nginx-ingress-controller:v0.49.3
    imagePullPolicy: IfNotPresent
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 300000; done;" ]
    resources:
      requests:
        : '1'
      limits:
        : '1'
```

> 注意：业务Pod的resources.requests、resources.limits的资源名称必须与sriovNetworkNodePolicy.spec.resourceName、sriovNetwork.spec.resourceName相等。

1.1.3 runc容器和InfiniBand卡IB模式场景

基于runc运行时容器和InfiniBand卡（IB模式）的组合场景，主要原理是将容器运行时环境与高性能的InfiniBand网络卡相结合。通过利用runc容器的轻量级和可移植性优势，与InfiniBand卡（IB模式）相结合，适用于安全隔离的需求较低，对网络性能要求较高的应用场景。本文将通过Yaml配置信息和参数，演示如何定义SR-IOV网络节点的策略。

操作步骤

配置SriovNetworkNodePolicy对象

指定切分 `kubernetes.io/hostname=node-10` 节点上， `rootDevices: 0000:71:00.0` 的PF设备

> 警告:
> 创建SR-IOV SriovNetworkNodePolicy对象时，节点应用修改会重启。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: node-policy-10
  namespace: eks-managed
spec:
  resourceName: mlxnics
  nodeSelector:
    kubernetes.io/hostname: node-10
  nicSelector:
    vendor: "15b3"
    deviceID: "1017"
    rootDevices:
      - 0000:71:00.0
 deviceType: netdevice
 numVfs: 3
 priority: 50
 isRdma: true
 linkType: IB
```

配置SriovIBNetwork对象

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: ibnics
  namespace: eks-managed
spec:
  ipam: |-
    {
    "type": "whereabouts",
    "range": "192.168.100.0/24",
    "gateway": "192.168.100.1",
    "exclude": [
      "192.168.100.0/26"
    ]
    }
  resourceName: mlxnics
  linkState: auto
```

配置runc运行时环境中的业务Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: sriov-runc-pod-demo
  annotations:
    k8s.v1.cni.cncf.io/networks: eks-managed/ibnics
spec:
  containers:
  - name: app-demo
    image: hub.ecns.io/test/nginx:latest
    imagePullPolicy: Always
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 300000; done;" ]
    resources:
      requests:
        ecnf.io/mlxnics: "1"
      limits:
```

```
        ecnf.io/mlxnics: "1"
  nodeName: node-10
```

1.1.4 runc容器和InfiniBand卡ETH模式场景

基于runc运行时容器和InfiniBand卡（ETH模式）的组合场景，主要原理是将容器运行时环境与高性能的InfiniBand网络卡相结合。通过利用runc容器的轻量级和可移植性优势，与InfiniBand卡（ETH模式）相结合，适用于安全隔离的需求较低，需要高性能数据传输和低延迟的应用场景。本文将通过Yaml配置信息和参数，演示如何定义SR-IOV网络节点的策略。

操作步骤

配置SriovNetworkNodePolicy对象

指定切分 `kubernetes.io/hostname=node-10` 节点上， `rootDevices: 0000:71:00.0` 的PF设备.

> 警告:
> 创建SR-IOV SriovNetworkNodePolicy对象时，节点应用修改会重启。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: node-policy-10
  namespace: eks-managed
spec:
  resourceName: mlxnics
  nodeSelector:
    kubernetes.io/hostname: node-10
  nicSelector:
    vendor: "15b3"
    deviceID: "1017"
    rootDevices:
      - 0000:71:00.0
 deviceType: netdevice
 numVfs: 3
 priority: 50
 isRdma: false
 linkType: ETH
```

配置SriovIBNetwork对象

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: nics
  namespace: eks-managed
spec:
  ipam: |-
    {
    "type": "whereabouts",
    "range": "192.168.100.0/24",
    "gateway": "192.168.100.1",
    "exclude": [
      "192.168.100.0/26"
    ]
    }
 resourceName: mlxnics
```

配置runc运行时环境中的业务Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: sriov-runc-pod-demo
  annotations:
    k8s.v1.cni.cncf.io/networks: eks-managed/nics
spec:
  containers:
  - name: app-demo
    image: hub.ecns.io/test/nginx:latest
    imagePullPolicy: Always
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 300000; done;" ]
    resources:
      requests:
        ecnf.io/mlxnics: "1"
      limits:
```

```
        ecnf.io/mlxnics: "1"
  nodeName: node-10
```

1.1.5 rune容器和InfiniBand卡IB模式场景

基于rune（安全运行时）容器和InfiniBand卡（IB模式）的组合场景，主要原理是将容器运行时环境与高性能的InfiniBand网络卡相结合。通过利用rune容器的安全性和隔离性优势，与InfiniBand卡（IB模式）相结合，适用于对安全性和隔离性有一定需求的轻量级传输应用场景。本文将通过Yaml配置信息和参数，演示如何定义SR-IOV网络节点的策略。

操作步骤

配置SriovNetworkNodePolicy对象：

指定切分 `kubernetes.io/hostname=node-10` 节点上， `rootDevices: 0000:71:00.0` 的PF设备

> 警告：
> 创建SR-IOV SriovNetworkNodePolicy对象时，节点应用修改会重启。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: node-policy-5
  namespace: eks-managed
spec:
  resourceName: mlxnics
  nodeSelector:
    kubernetes.io/hostname: node-5
  nicSelector:
    vendor: "15b3"
    deviceID: "1017"
    rootDevices:
      - 0000:71:00.0
  deviceType: vfio-pci
  numVfs: 3
  priority: 50
  isRdma: false
  linkType: IB
```

配置SriovIBNetwork对象：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: ibnics
  namespace: eks-managed
spec:
  ipam: |-
    {
    "type": "whereabouts",
    "range": "192.168.100.0/24",
    "gateway": "192.168.100.1",
    "exclude": [
      "192.168.100.0/26"
    ]
    }
  resourceName: mlxnics
  linkState: auto
```

配置 rune(安全运行时) 环境中的业务 Pod：

如果需要对容器（Pod）进行资源限制（limit）的设置，您可以在Pod的request字段中设置limit值。为了实现这个需求，您需要为Pod添加以下的annotation配置：

`io.katacontainers.config.runtime.sandbox_cgroup_only: "false"`

```
apiVersion: v1
kind: Pod
metadata:
  name: sriov-rune-pod-demo
  annotations:
    k8s.v1.cni.cncf.io/networks: eks-managed/ibnics
    io.katacontainers.config.runtime.enable_sriov: "true"
spec:
  runtimeClassName: rune
  containers:
  - name: app-demo
    image: hub.ecns.io/test/nginx:latest
    imagePullPolicy: Always
```

```
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 300000; done;" ]
    resources:
      requests:
        ecnf.io/mlxnics: "1"
      limits:
        ecnf.io/mlxnics: "1"
  nodeName: node-10
```

1.1.6 rune容器和InfiniBand卡ETH模式场景

基于rune（安全运行时）容器和InfiniBand卡（ETH模式）的组合场景，主要原理是将容器运行时环境与高性能的InfiniBand网络卡相结合。通过利用rune容器的安全性和隔离性优势，与InfiniBand卡（ETH模式）相结合，适用于对安全性和隔离性有一定需求的，需要快速数据传输和低延迟的应用场景。本文将通过Yaml配置信息和参数，演示如何定义SR-IOV网络节点的策略。

操作步骤

配置SriovNetworkNodePolicy对象：

指定切分 `kubernetes.io/hostname=node-10` 节点上， `rootDevices: 0000:71:00.0` 的PF设备

> 警告：
> 创建SR-IOV SriovNetworkNodePolicy对象时，节点应用修改会重启。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: node-policy-10
  namespace: eks-managed
spec:
  resourceName: mlxnics
  nodeSelector:
    kubernetes.io/hostname: node-10
  nicSelector:
    vendor: "15b3"
    deviceID: "1017"
    rootDevices:
      - 0000:71:00.0
  deviceType: vfio-pci
  numVfs: 3
  priority: 50
  isRdma: false
  linkType: ETH
```

配置SriovIBNetwork对象：

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: nics
  namespace: eks-managed
spec:
  ipam: |-
    {
    "type": "whereabouts",
    "range": "192.168.100.0/24",
    "gateway": "192.168.100.1",
    "exclude": [
    "192.168.100.0/26"
    ]
    }
  resourceName: mlxnics
```

配置 rune(安全运行时) 环境中的业务 Pod：

如果需要对容器（Pod）进行资源限制（limit）的设置，您可以在Pod的request字段中设置limit值。为了实现这个需求，您需要为Pod添加以下的annotation配置：

`io.katacontainers.config.runtime.sandbox_cgroup_only: "false"`

```
apiVersion: v1
kind: Pod
metadata:
  name: sriov-rune-pod-demo
  annotations:
  k8s.v1.cni.cncf.io/networks: eks-managed/nics
  io.katacontainers.config.runtime.enable_sriov: "true"
spec:
  runtimeClassName: rune
  containers:
  - name: app-demo
    image: hub.ecns.io/test/nginx:latest
    imagePullPolicy: Always
    command: [ "/bin/bash", "-c", "--" ]
```

```
    args: [ "while true; do sleep 300000; done;" ]
    resources:
      requests:
        ecnf.io/mlxnics: "1"
      limits:
        ecnf.io/mlxnics: "1"
  nodeName: node-10
```

1.2 创建GPU资源的容器实例

背景描述

GPU在工作负载中的主要作用是利用其出色的并行计算能力。在安全容器服务中，将GPU卡用于运行计算密集型工作负载对于某些特定的业务场景非常有价值，包括高性能计算、数据处理和分析加速、图形渲染和可视化优化等。通过将GPU与容器技术结合，可以实现灵活、高效且可扩展的计算环境，提供更好的用户体验和性能表现。本章节将详细介绍如何在部署工作负载时使用GPU卡能力。

前提条件

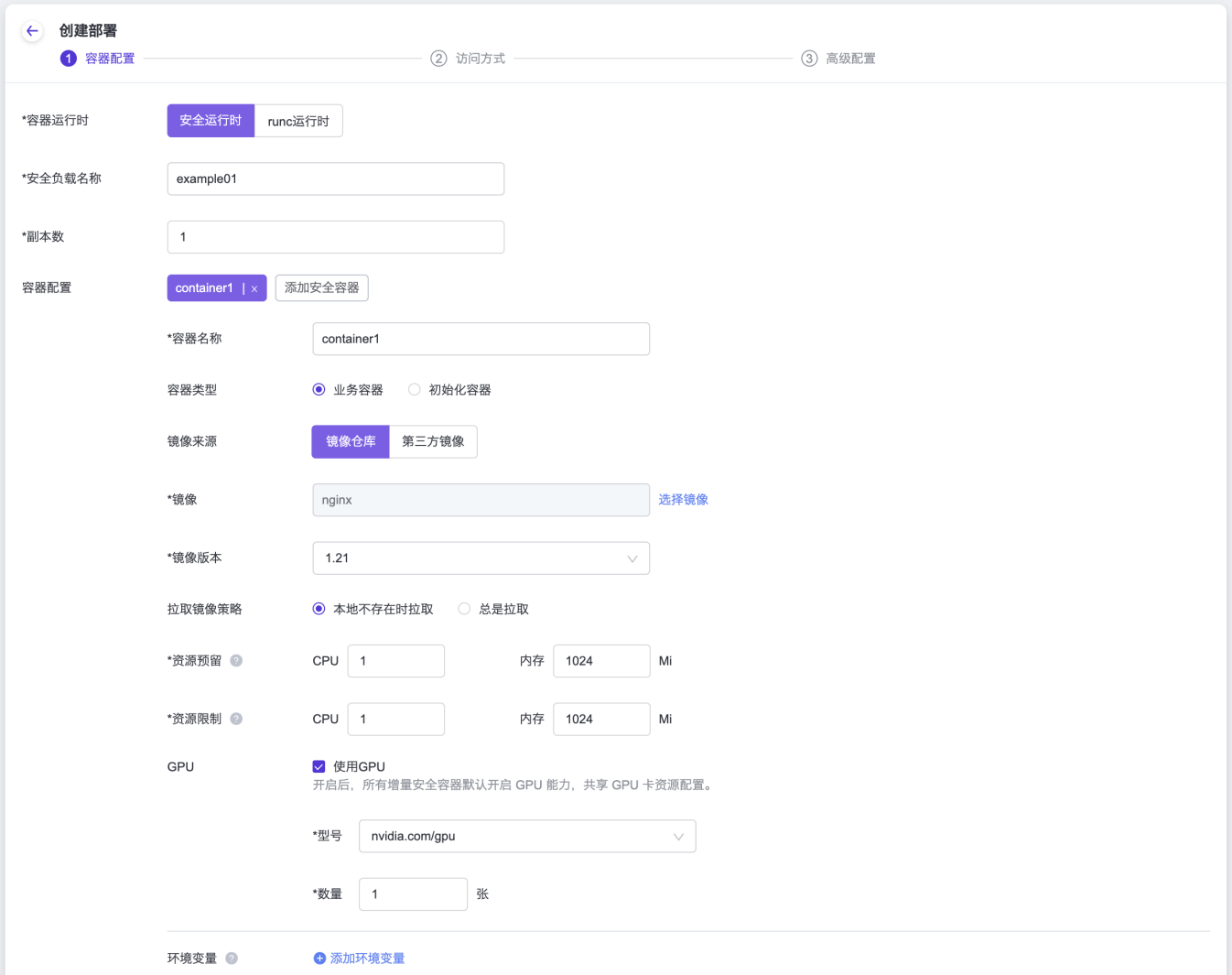
- **上传对应的GPU解决方案对接包：**通过上传解决方案对接包，系统能够正确识别和使用GPU资源，以便在相应的工作负载中进行分配和调度。请确保在需要使用GPU资源时，已上传并配置了对应的GPU解决方案对接包。
- **容器运行时：**当容器运行时为"安全运行时（rune）"时，支持使用GPU资源。（注意："守护进程集（DaemonSet）"和"定时任务（CronJob）"类型的工作负载不支持使用GPU。）
- **资源预留：**当需要勾选"使用GPU"时，建议设置CPU的参数值大于等于1，内存的参数值大于等于1024MiB。这样可以确保在使用GPU时，为工作负载分配足够的计算资源和内存资源，以获得良好的性能和稳定性。

操作步骤

目前平台中支持在安全容器中使用英伟达GPU设备和百度昆仑XPU设备。具体支持型号参见使用限制。下文将分别列出不同品牌GPU的操作步骤和示例，以帮助用户正确配置和使用GPU资源。

英伟达（NVIDIA）GPU设备

在支持使用GPU能力的工作负载类型创建界面中，勾选"使用GPU"选项，并选择"nvidia.com/gpu"作为GPU资源的类型。这样配置后，系统将为工作负载分配相应的GPU资源，以提供所需的计算能力和性能。创建容器组配置时，配置好容器（Pod） GPU参数，所有增量容器（Pod）默认共享GPU卡资源。



如果需要对容器（Pod）进行资源限制（limit）的设置，您可以在Pod的request字段中设置limit值。为了实现这个需求，您需要为Pod添加以下的annotation配置：

```
io.katacontainers.config.runtime.sandbox_cgroup_only: "false"
```

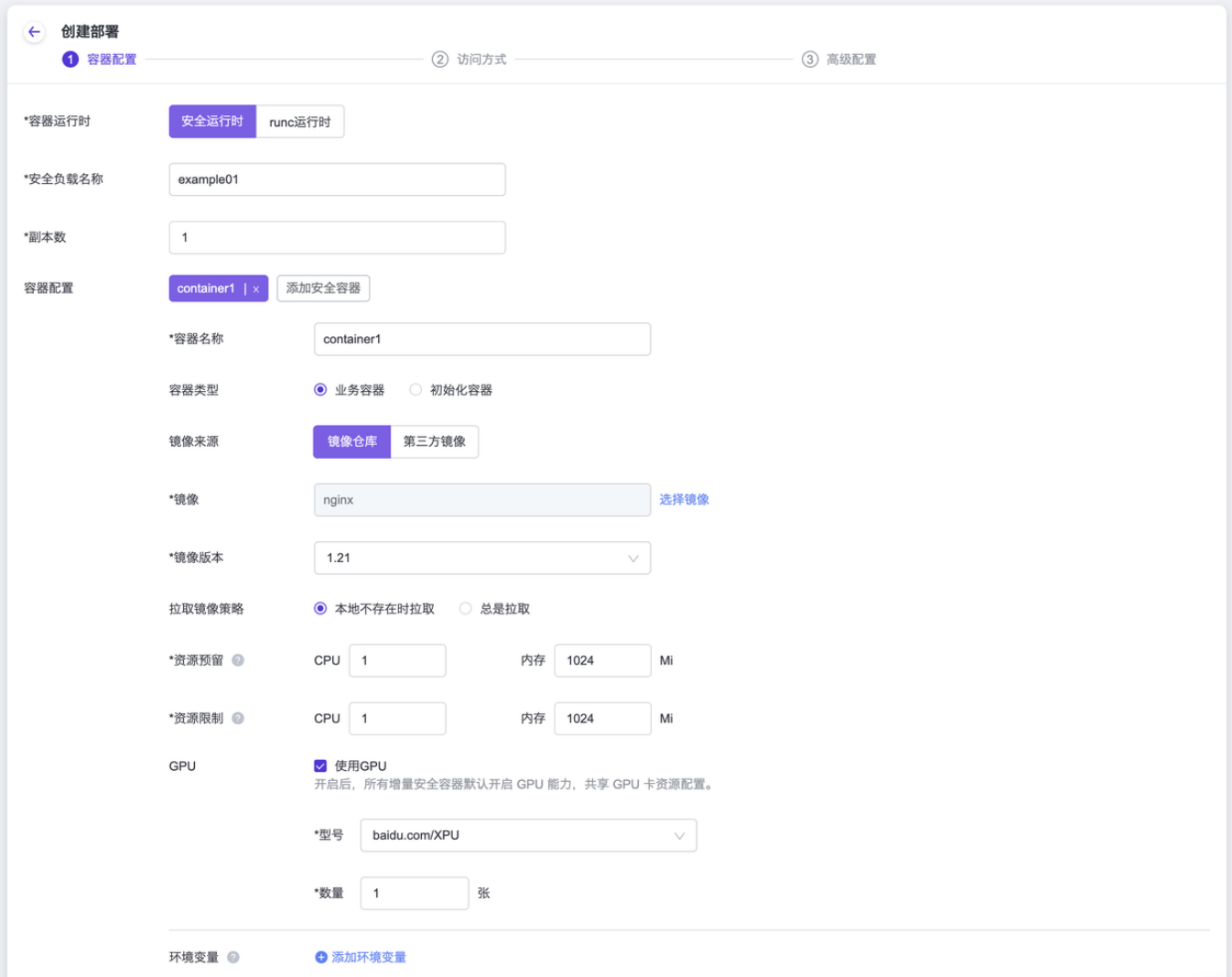
Yaml示例：

```
template:
    metadata:
      labels:
        my-app: perf-server0
      annotations:
        # 在limit中配置了memory时，需要添加如下注解
```

```
        io.katacontainers.config.runtime.sandbox_cgroup_only: "false"
    spec:
      runtimeClassName: rune
      containers:
      - name: perf-server0
        image: docker.io/test/perf:0.0.1
        command:
        - iperf3
        args:
        - -s
        resources:
          limits:
            cpu: "2"
            memory: 2Gi
            # 配置GPU资源数量
            nvidia.com/gpu: "1"
          requests:
            cpu: "2"
            memory: 2Gi
            nvidia.com/gpu: "1"
```

百度昆仑XPU

在支持使用GPU能力的工作负载类型创建界面中，勾选"使用GPU"选项，并选择"baidu.com/XPU"作为GPU资源的类型。这样配置后，系统将为工作负载分配相应的GPU资源，以提供所需的计算能力和性能。创建容器组配置时，配置好容器（Pod） GPU参数后，所有增量容器（Pod）默认共享GPU卡资源。



如果需要对容器（Pod）进行资源限制（limit）的设置，您可以在容器（Pod）的“request”字段中设置“limit”值。为了实现这个需求，您需要为容器（Pod）添加以下的“annotation”配置：

```
io.katacontainers.config.runtime.sandbox_cgroup_only: "false"
```

Yaml示例：

```
template:
   metadata:
     labels:
       my-app: perf-server0
     annotations:
       # 在limit中配置了memory时，需要添加如下注解
```

```
        io.katacontainers.config.runtime.sandbox_cgroup_only: "false"
    spec:
      runtimeClassName: rune
      containers:
      - name: perf-server0
        image: docker.io/test/perf:0.0.1
        command:
        - iperf3
        args:
        - -s
        resources:
          limits:
            cpu: "2"
            memory: 2Gi
            # 配置XPU资源数量
            baidu.com/XPU: "1"
          requests:
            cpu: "2"
            memory: 2Gi
            baidu.com/XPU: "1"
```

此外，百度昆仑XPU支持指定某个容器对象独享某个具体的GPU卡（其他容器不可见），可通过环境变量：`CXPU_VISIBLE_DEVICES: 1` 设置GPU卡隔离。当环境可用GPU卡未被指定完，配置后且未被隔离的GPU卡为所有容器共享资源。当环境可用GPU卡被指定完，剩余的容器无可用的GPU卡资源。

百度昆仑XPU在各个容器（Pod）中的可见性环境变量： `CXPU_VISIBLE_DEVICES` 可以设置值为 `all或者是卡序号（1, 2, 3）`

Yaml示例：

```
template:
    metadata:
      labels:
        my-app: perf-server0
      annotations:
        # 在limit中配置了memory时，需要添加如下注解
        io.katacontainers.config.runtime.sandbox_cgroup_only: "false"
    spec:
      runtimeClassName: rune
      containers:
```

```
    - name: perf-server0
      image: docker.io/test/perf:0.0.1
      command:
      - iperf3
      args:
      - -s
      env:
        - name: CXPU_VISIBLE_DEVICES
          value: ALL
      resources:
        limits:
          cpu: "2"
          memory: 2Gi
          # 配置GPU资源数量
          nvidia.com/gpu: "1"
        requests:
          cpu: "2"
          memory: 2Gi
          nvidia.com/gpu: "1"
```

附录

为了正确使用GPU资源，请确保准确指定GPU卡的型号对应的资源名称：

GPU卡型号	资源名称
英伟达（NVIDIA）GPU设备	nvidia.com/gpu
百度昆仑XPU设备	baidu.com/XPU

咨询热线：400-100-3070

北京易捷思达科技发展有限公司：
北京市海淀区西北旺东路10号院东区1号楼1层107-2号
南京易捷思达软件科技有限公司：
江苏省南京市雨花台区软件大道168号润和创智中心4栋109-110

邮箱：
contact@easystack.cn (业务咨询)
partners@easystack.cn(合作伙伴咨询)
marketing@easystack.cn (市场合作)