虚拟化

概览

介绍

Container-Orchestrated Virtual Machine 解决方案

特性

产品功能

约束与限制

安装

安装

前提条件

操作步骤

资源配额说明

镜像

介绍

优势

操作指南

权限

实用指南

虚拟机

介绍

操作指南

实用指南

故障排除

网络

介绍

优势

操作指南

实用指南

存储

介绍

优势

操作指南

备份与恢复

介绍

应用场景

使用限制

操作指南

概览

介绍

介绍

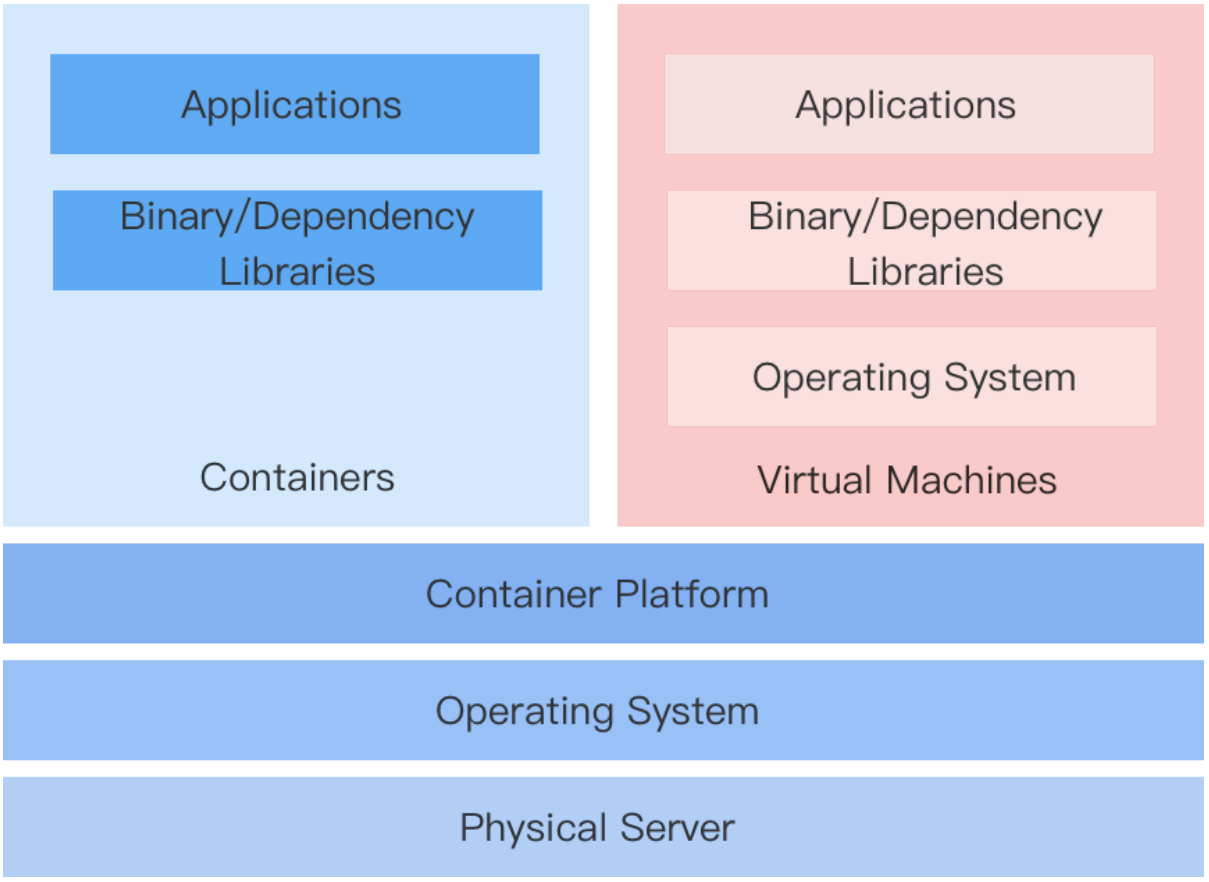
对于采用基于虚拟机架构的企业来说，向 Kubernetes 和基于容器的架构转型不可避免地需要应用现代化改造。然而，由于业务持续在线的需求或开发习惯难以改变等限制，企业通常无法在短时间内完全脱离虚拟化架构。

因此，能够在同一平台上统一配置、管理和控制容器资源与虚拟机资源的解决方案显得尤为重要。

目录

Container-Orchestrated Virtual Machine 解决方案

该平台基于开源组件 KubeVirt 实现了虚拟机（VMI，VirtualMachineInstance）解决方案，使得容器编排虚拟机的创建更加便捷快速，并能够运行虚拟化应用。



特性

快速转型

无需重写应用或修改镜像，只需将现有应用打包成 qcow2 或 raw 格式的虚拟机镜像，在平台上使用该镜像创建虚拟机，即可将应用部署到容器平台。

保持行为习惯

容器化虚拟机可采用类似传统虚拟机的管理方式，无需关注底层容器实现，包括虚拟机生命周期管理、磁盘和网络、快照管理等。

虚拟化与容器化共存

- 统一平台支持管理虚拟化服务，同时支持基于 Kubernetes 的容器调度和管理。
- 在持续使用虚拟机工作负载的基础上，实现容器化应用的渐进式现代化。
- 新开发的容器化应用与虚拟化应用的交互不受影响。

产品功能

- 虚拟机：支持创建管理员分配镜像的虚拟机并进行管理，包括启动和停止虚拟机、管理快照、远程登录虚拟机以及修改虚拟机配置。
- 虚拟磁盘：支持查看和管理当前项目中创建的磁盘信息，包括创建磁盘、查看磁盘名称、存储类、容量及关联虚拟机。
- 虚拟机快照：支持查看虚拟机快照的状态、关联虚拟机及最近回滚时间等详细信息。
- 虚拟机镜像：支持查看当前项目下的虚拟机镜像信息，包括镜像提供方式和操作系统。
- 密钥对：支持查看和管理当前项目中创建的密钥对，包括创建密钥对及查看关联虚拟机列表。

约束与限制

必须基于物理机集群实现，且 KubeVirt 组件需部署在启用虚拟化的集群内。平台提供通过 Operator 部署 KubeVirt 组件及启用虚拟化的接口，所有相关配置由平台管理员完成。

安装

为了让项目人员能够充分利用容器平台内的虚拟化功能，平台管理员必须执行以下操作以准备虚拟化环境。

目录

前提条件

- 下载与您的平台架构对应的**ACP Virtualization with KubeVirt**安装包。
- 通过上架软件包机制上传******ACP Virtualization with KubeVirt**安装包。
- 使用虚拟化功能时，需要提前规划和准备网络及存储环境。
  注意：

- 如果需要通过 IP 直接连接虚拟机，集群必须使用 Kube-OVN Underlay 网络模式。您可以参考最佳实践准备 Kube-OVN Underlay 物理网络。
- 推荐与 Kubevirt 一起使用 TopoLVM，因为它能提供接近硬件级别的性能。如果性能要求不高，也可以使用 Ceph 分布式存储。

存储产品	说明
TopoLVM	优点：相对轻量且性能良好。 缺点：不能跨节点使用，可靠性较低，无法提供冗余。
Ceph 分布式存储	优点：可跨节点使用，高可用且具备冗余能力。 缺点：磁盘副本冗余导致利用率较低，性能较差。

  - 如果使用 TopoLVM 并配置了多块磁盘，请确保启用虚拟化的节点剩余存储容量能够满足多块磁盘的总容量，否则虚拟机创建将失败。
  - 如果使用 Ceph 分布式存储，请确保存储所在网络与虚拟机所在网络能够互通。

操作步骤

1 启用节点虚拟化

当自建集群的节点为物理机时，可以通过启用或禁用节点虚拟化开关来控制 Kubernetes 是否允许在该节点调度虚拟机实例（VMI）。

- 开启开关时，允许在物理机节点上调度新的虚拟机；Windows 物理节点不支持开启虚拟化。
- 关闭开关时，阻止在物理机节点上调度新的虚拟机，但不影响已在该节点运行的虚拟机。

操作步骤

1. 进入管理员。
2. 在左侧导航栏点击集群管理 > 集群。
3. 点击*自建集群名称*。
4. 在节点标签页，点击目标节点右侧的⋮> 启用虚拟化。

5. 点击确认。

2 部署 Operator

1. 登录，进入管理员页面。
2. 点击 **Marketplace** > **OperatorHub**，进入 **OperatorHub** 页面。
3. 找到 **ACP Virtualization with KubeVirt**，点击安装，进入安装 **ACP Virtualization with KubeVirt** 页面。
   配置参数：

参数	推荐配置
Channel	默认通道为 `alpha`。
安装模式	`Cluster`：集群内所有命名空间共享单个 Operator 实例进行创建和管理，资源占用较低。
安装位置	选择 `Recommended`，命名空间仅支持 **kubevirt**。
升级策略	`Manual`：Operator Hub 有新版本时，需要手动确认升级 Operator 到最新版本。

3 创建 HyperConverged 实例

1. 进入管理员。
2. 点击 **Marketplace** > **OperatorHub**。
3. 找到 **ACP Virtualization with KubeVirt**，点击进入 **ACP Virtualization with KubeVirt** 详情页。
4. 点击所有实例
5. 在 **HyperConverged** 实例卡片上点击创建实例。
   注意：每个集群只需创建一个 **HyperConverged** 实例。
6. 切换到 YAML 视图，仅将示例中 `spec.storageImport.insecureRegistries` 字段中指定的 ***placeholder*** 替换为正确的虚拟机镜像仓库地址，例如：`192.168.16.214:60080`，其他参数保持默认值。

```
spec:
  storageImport:
    insecureRegistries:
      - placeholder
```

替换结果：

```
spec:
  storageImport:
    insecureRegistries:
      - '192.168.16.214:60080'
```

7. 点击创建，等待资源列表中自动创建 CDI 和 KubeVirt 类型实例，同时确保 YAML 中显示的 **status.phase** 为 `deployed`，表示 HyperConverged 实例创建成功。

4 配置虚拟机超售比（可选）

- 在集群管理 > 集群中配置虚拟机所在集群的超售比。
- 或在项目管理 > 命名空间中配置虚拟机所在命名空间的超售比。

重要说明

- 虚拟机仅支持 CPU 超售比，推荐配置值为 2 到 4 之间。
- 虚拟机启用超售比后，创建虚拟机时容器的请求值（requests）固定为指定限制值（**limits**）/ 虚拟机超售比，用户通过 YAML 设置的请求值将无效。
  例如：假设虚拟机 CPU 资源超售比设置为 4，用户创建虚拟机时指定 CPU 限制值为 4c，则 CPU 请求值为 4c/4 = 1c。

资源配额说明

虚拟机的内存资源配额受其所在命名空间的内存资源配额限制。由于承载虚拟机的 Pod 内存通常大于虚拟机实际可用内存，建议预留 20% 资源。当命名空间剩余可用资源低于 20% 时，请及时扩容资源。

镜像

介绍

操作指南

实用指南

权限

介绍

Alauda Container Platform Virtualization with KubeVirt 利用 Kubernetes 扩展 API 功能，将虚拟机镜像抽象为自定义资源定义（CRD）。它提供了一个用户界面（UI），方便用户将存储在远程仓库中的虚拟机镜像导入到 ACP 中使用。

目录

优势

- 支持主流操作系统
  支持多种常用的 Linux 发行版和 Windows 操作系统。
- 多架构支持
  兼容 **X86_64** 和 **ARM64** 架构。
- 多源支持
  允许从以下来源导入虚拟机镜像：
  - 镜像仓库
  - 文件服务器
  - 兼容 S3 的对象存储
- 多格式支持

支持 **QCOW2** 和 **RAW** 格式的虚拟机镜像。

操作指南

添加虚拟机镜像

更新/删除虚拟机镜像

更新/删除镜

操作步骤

添加虚拟机镜像

平台支持添加 **X86_64** 和 **ARM64 (Alpha)** 架构的虚拟机镜像，帮助开发者快速创建现有服务的虚拟机，促进业务系统的迁移。

目录

操作步骤

1. 进入 管理员。
2. 在左侧导航栏点击 虚拟化管理 > 虚拟机镜像。
3. 点击 添加虚拟机镜像。
4. 参考以下说明配置相关参数。

参数	说明
配置方式	目前仅支持 公共镜像 方式，即添加的镜像可在分配的项目中使用。
操作系统	支持的操作系统包括：**CentOS/Ubuntu/RedHat/Debian/TLinux/**其他 **Linux/Windows (Alpha)**。 支持的系统架构为：**X86_64** 和 **ARM64 (Alpha)**。

参数	说明
来源	• 镜像仓库：存储在容器镜像仓库中的虚拟机镜像。 • **HTTP**：存储在文件服务器上，使用 HTTP 协议的虚拟机镜像。 • 对象存储（**S3**）：可通过对象存储协议（S3）获取的虚拟机镜像，若不需要认证，请使用 HTTP 作为来源。
CPU 架构	标记 CPU 架构信息。镜像仓库来源支持多选，其他来源仅支持单选。
镜像地址	支持 KVM 虚拟机镜像，包括 qcow2/raw 格式。 • 若来自镜像仓库，填写 `repository_address:image_version`，例如 `registry.example.com/library/ubuntu:latest`。 • 若来自 HTTP 来源，填写镜像文件 URL，必须以 `http://` 或 `https://` 开头，例如 `http://192.168.0.1/vm_image/centos_7.8.qcow2`。 • 若来自对象存储（S3），填写可通过对象存储协议（S3）获取的镜像地址，例如 `https://endpoint/bucket/centos.qcow2`。
认证	根据镜像仓库是否需要认证，可开启或关闭开关。开启时可选择已有镜像凭证或点击 添加凭证，仅支持 用户名/密码 类型凭证。 注意：当来源为对象存储（S3）时，认证不可关闭。
分配项目	为该镜像分配使用权限给项目。 • 所有项目：将镜像使用权限分配给所有项目。 • 指定项目：将镜像使用权限分配给指定项目。 • 不分配：暂不分配给任何项目，镜像创建后可通过 更新镜像 操作进行分配。

5. 点击 添加。

更新/删除虚拟机镜像

1. 进入 管理员。
2. 在左侧边栏，点击 虚拟化管理 > 虚拟机镜像。
3. 点击⋮> 更新/删除。
4. 确认后，点击 更新/删除。

更新/删除镜像凭据

1. 进入 管理员。
2. 在左侧导航栏中，点击 虚拟化管理 > 虚拟机镜像。
3. 在 镜像凭据 标签页，点击⋮> 更新/删除。
4. 确认后，点击 更新/删除。

实用指南

使用 KubeVirt 基于 ISO 创建 Windows 镜像

本文档介绍基于开源组件 KubeVirt 的虚拟机方案，利用 KubeVirt 虚拟化技术通过 ISO 镜像文件创建 Windows 操作系统镜像。ISO 文件通过 CDI DataVolume 上传至集群，无需构建并推送容器镜像到镜像仓库。

目录

前提条件

- 集群中所有组件均正常运行。
- 请提前准备好 Windows 镜像及最新的 virtio-win-tools↗。

- 已安装并配置好 `kubectl` 命令行工具以访问集群。
- 集群中存在支持 `ReadWriteOnce`（RWO）访问模式的 StorageClass。

约束与限制

- 启动 KubeVirt 时，自定义镜像文件系统大小会影响镜像写入 PVC 磁盘的速度，文件系统过大可能导致创建时间延长。
- 建议将 Windows C 盘保持在 100G 以下以减小初始大小，后续扩容需在 Windows 系统内手动完成。

操作步骤

1 上传 **ISO** 文件至 **DataVolumes**

通过 CDI 上传机制将 Windows ISO 和 virtio-win ISO 文件直接上传至集群存储。

设置 **CDI** 上传代理

1. 设置端口转发至 CDI 上传代理，该端口转发会话用于上传两个 ISO 文件。

```
kubectl port-forward -n cdi svc/cdi-uploadproxy 8443:443 &
```

2. 获取认证令牌。

```
TOKEN=$(kubectl create token default -n default)
```

上传 **Windows ISO**

1. 创建上传类型的 DataVolume，保存以下 YAML 至 `dv-win-iso.yaml` 文件。根据实际 ISO 文件大小调整 `storage` 大小（Windows ISO 通常为 4–6 GB）。

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: win-iso-dv
  namespace: default
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
spec:
  source:
    upload: {}
  storage:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 8Gi
    storageClassName: vm-cephrbd  # 替换为实际 StorageClass
    volumeMode: Block
```

2. 执行以下命令创建 DataVolume。

```
kubectl apply -f dv-win-iso.yaml
```

3. 等待 DataVolume 进入 `UploadReady` 阶段。

```
kubectl get dv win-iso-dv -w
# PHASE 列应显示 UploadReady
```

4. 使用 curl 上传 Windows ISO 文件，替换文件路径为实际 ISO 路径。

```
curl -v --insecure \
  -H "Authorization: Bearer ${TOKEN}" \
  --data-binary @/path/to/en_windows_server_2019_x64_dvd_4cb967d8.
iso \
  "https://localhost:8443/v1beta1/upload"
```

5. 确认 DataVolume 状态变为 `Succeeded`。

```
kubectl get dv win-iso-dv
# PHASE 列应显示 Succeeded
```

上传 **virtio-win ISO**

1. 创建上传类型的 DataVolume，保存以下 YAML 至 `dv-virtio-iso.yaml` 文件。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: virtio-iso-dv
  namespace: default
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
spec:
  source:
    upload: {}
  storage:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 1Gi
    storageClassName: vm-cephrbd  # 替换为实际 StorageClass
    volumeMode: Block
```

2. 执行以下命令创建 DataVolume。

```
kubectl apply -f dv-virtio-iso.yaml
```

3. 等待 DataVolume 进入 `UploadReady` 阶段。

```
kubectl get dv virtio-iso-dv -w
# PHASE 列应显示 UploadReady
```

4. 使用 curl 上传 virtio-win ISO 文件。

```
curl -v --insecure \
  -H "Authorization: Bearer ${TOKEN}" \
  --data-binary @/path/to/virtio-win.iso \
  "https://localhost:8443/v1beta1/upload"
```

5. 确认 DataVolume 状态变为 `Succeeded`。

```
kubectl get dv virtio-iso-dv
# PHASE 列应显示 Succeeded
```

注意：`--insecure` 参数用于跳过自签名证书验证，生产环境请配置正确证书。大文件上传可能耗时较长，请保证网络稳定。

2 创建虚拟机

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机。
3. 点击 创建虚拟机。
4. 在表单页面填写必要参数，如 名称、镜像 等。详细参数及配置请参考 Create Virtual Machine。
5. 切换至 YAML 模式。
6. 将 spec.template.spec.domain.devices 字段下的配置替换为以下内容。USB tablet 输入设备是 VNC 控制台鼠标定位正确所必需的（参见 kubevirt#2392↗、kubevirt#3474↗）。否则鼠标指针会错位，因为 VNC 使用绝对坐标，而默认 PS/2 鼠标仅支持相对坐标。

```
domain:
  devices:
    inputs:
      - type: tablet
        bus: usb
        name: tablet
    disks:
      - disk:
          bus: virtio
        name: cloudinitdisk
      - bootOrder: 1
        cdrom:
          bus: sata
        name: win-iso
      - cdrom:
          bus: sata
        name: virtio-iso
      - disk:
          bus: sata
        name: rootfs
        bootOrder: 10
```

7. 将 spec.template.spec.volumes 字段替换为以下内容。两个 ISO 文件均引用 DataVolumes，而非容器镜像。

```
      volumes:
        - cloudInitConfigDrive:
            userData: >-
              #cloud-config
              disable_root: false
              ssh_pwauth: true
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: "<hash>"  # 使用 mkpasswd --method
=SHA-512 --rounds=4096 生成
          name: cloudinitdisk
        - dataVolume:
            name: win-iso-dv
          name: win-iso
        - dataVolume:
            name: virtio-iso-dv
          name: virtio-iso
        - dataVolume:
            name: aa-test-rootfs
          name: rootfs
```

8. 检查 YAML 文件，完成配置后的完整 YAML 如下。

```yaml
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  annotations:
    cpaas.io/creator: test@example.io
    cpaas.io/display-name: ""
    cpaas.io/updated-at: 2024-09-01T14:57:55Z
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  generation: 16
  labels:
    virtualization.cpaas.io/image-name: debian-2120-x86
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: debian
    virtualization.cpaas.io/image-supply-by: public
    vm.cpaas.io/name: aa-test
  name: aa-test
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        labels:
          vm.cpaas.io/reclaim-policy: Delete
          vm.cpaas.io/used-by: aa-test
        name: aa-test-rootfs
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 100Gi
          storageClassName: vm-cephrbd
          volumeMode: Block
        source:
          blank: {}
  running: true
  template:
    metadata:
      annotations:
        cpaas.io/creator: test@example.io
        cpaas.io/display-name: ""
```

```
    cpaas.io/updated-at: 2024-09-01T14:55:44Z
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  creationTimestamp: null
  labels:
    virtualization.cpaas.io/image-name: debian-2120-x86
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: debian
    virtualization.cpaas.io/image-supply-by: public
    vm.cpaas.io/name: aa-test
spec:
  affinity:
    nodeAffinity: {}
  architecture: amd64
  domain:
    devices:
      inputs:
        - type: tablet
          bus: usb
          name: tablet
      disks:
        - disk:
            bus: virtio
          name: cloudinitdisk
        - bootOrder: 1
          cdrom:
            bus: sata
          name: win-iso
        - cdrom:
            bus: sata
          name: virtio-iso
        - disk:
            bus: sata
          name: rootfs
          bootOrder: 10
      interfaces:
        - bridge: {}
          name: default
    machine:
      type: q35
    resources:
      limits:
        cpu: "4"
        memory: 8Gi
```

```
            requests:
              cpu: "4"
              memory: 8Gi
        networks:
          - name: default
            pod: {}
        nodeSelector:
          kubernetes.io/arch: amd64
          vm.cpaas.io/baremetal: "true"
        volumes:
          - cloudInitConfigDrive:
              userData: >-
                #cloud-config
                disable_root: false
                ssh_pwauth: true
                users:
                  - default
                  - name: root
                    lock_passwd: false
                    hashed_passwd: "<hash>"  # 使用 mkpasswd --method
=SHA-512 --rounds=4096 生成
            name: cloudinitdisk
          - dataVolume:
              name: win-iso-dv
            name: win-iso
          - dataVolume:
              name: virtio-iso-dv
            name: virtio-iso
          - dataVolume:
              name: aa-test-rootfs
            name: rootfs
```

9. 点击 创建。
10. 点击 操作 > **VNC** 登录。
11. 当出现提示 **press any key boot from CD or DVD** 时，按任意键进入 Windows 安装程序；若未看到提示，点击页面左上角的 发送远程命令，从下拉菜单选择 **Ctrl-Alt-Delete** 重启服务器。

    注意：若虚拟机详情页顶部出现提示 当前虚拟机有配置变更需要重启生效，请重启，可忽略该提示，无需重启。

3 安装 **Windows** 操作系统

1. 进入安装页面后，按照安装指引完成系统安装。
   注意：分区选择步骤中，磁盘总线必须为 sata，才能正确识别磁盘。请依次选择每个分区并点击 删除，清除所有分区，由系统自动处理。
2. 配置管理员账户密码后，点击页面左上角的 发送远程命令，选择 **Ctrl-Alt-Delete**。
3. 出现提示 **Ctrl+Alt+Delete** 组合键将重启服务器，确认重启 时，点击 确定。
4. 输入密码进入 Windows 系统桌面，至此 Windows 操作系统安装完成。

4 安装 virtio-win-tools

该工具主要包含必要驱动。

1. 打开文件资源管理器。
2. 双击 **CD** 驱动器**(E:) virtio-win-<version>**，运行 **virtio-win-guest-tools** 目录进入安装页面，按照安装指引完成安装。`<version>` 部分根据实际情况替换。
3. 安装完成后，关闭 Windows 系统电源。

5 导出自定义 Windows 镜像

具体操作请参考 导出虚拟机镜像。

6 使用 Windows 镜像

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机。
3. 点击列表中使用 Windows 镜像创建的虚拟机名称进入详情页。
4. 在表单页面填写必要参数，镜像选择导出的 Windows 镜像。详细参数及配置请参考 Create Virtual Machine。
5. （可选）若使用较新操作系统，如 Windows 11，需启用时钟、UEFI、TPM 等功能。切换至 YAML，替换原 YAML 文件为以下内容。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-windows
  name: vmi-windows
spec:
  domain:
    clock:
      timer:
        hpet:
          present: false
        hyperv: {}
        pit:
          tickPolicy: delay
        rtc:
          tickPolicy: catchup
      utc: {}
    cpu:
      cores: 2
    devices:
      inputs:
        - type: tablet
          bus: usb
          name: tablet
      disks:
      - disk:
          bus: sata
        name: pvcdisk
      interfaces:
      - masquerade: {}
        model: e1000
        name: default
      tpm: {}
    features:
      acpi: {}
      apic: {}
      hyperv:
        relaxed: {}
        spinlocks:
          spinlocks: 8191
        vapic: {}
      smm: {}
```

```
    firmware:
      bootloader:
        efi:
          secureBoot: true
      uuid: 5d307ca9-b3ef-428c-8861-06e72d69f223
    resources:
      requests:
        memory: 4Gi
  networks:
  - name: default
    pod: {}
  terminationGracePeriodSeconds: 0
  volumes:
  - name: pvcdisk
    persistentVolumeClaim:
      claimName: disk-windows
  - name: winiso
    persistentVolumeClaim:
      claimName: win11cd-pvc
```

6. 点击 创建。

7 添加内部路由

通过配置 NodePort 类型的内部路由，暴露远程桌面连接端口。

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机。
3. 点击列表中使用 Windows 镜像创建的虚拟机名称进入详情页。
4. 在 登录信息 区域的 内部路由 旁点击 添加 图标。
5. 按照以下说明配置参数。

参数	说明
类型	选择 **NodePort**。
端口	• 协议：选择 TCP。 • 服务端口：使用 3389。 • 虚拟机端口：使用 3389。

参数	说明
	• 服务端口名称：使用 rdp。

6. 点击 确定 返回详情页。
7. 点击 登录信息 区域的 内部路由 链接。
8. 记录基础信息区的 虚拟 **IP** 和端口区的 主机端口 信息。

远程访问

本文以 Windows 操作系统远程连接为例，其他操作系统可使用支持 RDP 协议的软件进行连接。

1. 打开 远程桌面连接。
2. 输入在 添加内部路由 步骤中保存的虚拟 IP 和主机端口，格式为 虚拟 **IP:** 主机端口 ，例如：192.1.1.1:3389 。
3. 点击 连接。

使用 KubeVirt 基于 ISO 创建 Linux 镜像

本文档介绍了基于开源组件 KubeVirt 实现的虚拟机方案，利用 KubeVirt 虚拟化技术从 ISO 镜像文件创建 Linux 操作系统镜像。通过 CDI DataVolume 将 ISO 上传至集群，无需构建并推送容器镜像到镜像仓库。

目录

前提条件

- 集群中所有组件均正常运行。
- 需提前准备好 Linux 镜像，本文以 Ubuntu 操作系统↗ 为例。
- 已安装并配置好 `kubectl` 命令行工具以访问集群。
- 集群中存在支持 `ReadWriteOnce` (RWO) 访问模式的 StorageClass。

约束与限制

- 启动 KubeVirt 时，自定义镜像的文件系统大小会影响写入镜像到 PVC 磁盘的速度，文件系统过大可能导致创建时间过长。
- 建议 Linux 根分区大小保持在 100G 以下以减小初始大小，配置 cloud-init 后，创建虚拟机时可为根分区分配更大存储，系统会自动扩展。

操作步骤

1 将 Linux ISO 上传至 DataVolume

通过 CDI 上传机制将 ISO 文件直接上传至集群存储，无需构建容器镜像。

1. 创建上传类型的 DataVolume，将以下 YAML 保存为 `dv-iso-upload.yaml` 文件，根据实际 ISO 文件大小调整 `storage` 大小。

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: iso-upload-dv
  namespace: default
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
spec:
  source:
    upload: {}
  storage:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 8Gi
    storageClassName: vm-cephrbd  # 替换为实际的 StorageClass
    volumeMode: Block
```

2. 执行以下命令创建 DataVolume。

```
kubectl apply -f dv-iso-upload.yaml
```

3. 等待 DataVolume 进入 `UploadReady` 阶段。

```
kubectl get dv iso-upload-dv -w
# PHASE 列应显示 UploadReady
```

4. 设置端口转发至 CDI 上传代理。

```
kubectl port-forward -n cdi svc/cdi-uploadproxy 8443:443 &
```

5. 获取认证令牌。

```
TOKEN=$(kubectl create token default -n default)
```

6. 使用 curl 上传 ISO 文件，替换文件路径为实际 ISO 路径。

```
curl -v --insecure \
  -H "Authorization: Bearer ${TOKEN}" \
  --data-binary @/path/to/ubuntu-24.04-live-server-amd64.iso \
  "https://localhost:8443/v1beta1/upload"
```

注意：`--insecure` 参数用于跳过自签名证书验证，生产环境请配置正确证书。大文件上传可能耗时较长，请确保网络稳定。

7. 确认 DataVolume 状态变为 `Succeeded`。

```
kubectl get dv iso-upload-dv
# PHASE 列应显示 Succeeded
```

2 创建虚拟机

1. 进入 容器平台。
2. 左侧导航栏点击 虚拟化 > 虚拟机。
3. 点击 创建虚拟机。
4. 在表单页面填写参数，具体参数及配置请参考 Create Virtual Machine。

参数	说明
选择镜像	选择虚拟机的模板镜像。
IP 地址	保持默认，通过 **DHCP** 获取。
网络模式	使用 **NAT** 模式，此处不要使用 桥接 模式。

5. 切换至 YAML。
6. 替换 spec.template.spec.domain.devices.disks 字段下的配置，ISO DataVolume 以 CDROM 形式挂载，且启动优先级最高，rootfs 磁盘作为安装目标。

```
domain:
  devices:
    disks:
      - bootOrder: 1
        cdrom:
          bus: sata
        name: iso-disk
      - disk:
          bus: virtio
        name: cloudinitdisk
      - disk:
          bus: virtio
        name: rootfs
        bootOrder: 10
```

7. 替换 spec.template.spec.volumes 字段，ISO 以 DataVolume 方式引用，而非容器镜像。

```
      volumes:
        - dataVolume:
            name: iso-upload-dv
          name: iso-disk
        - cloudInitConfigDrive:
            userData: |-
              #cloud-config
              disable_root: false
              ssh_pwauth: false
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: "<hash>"  # 使用 mkpasswd --method
=SHA-512 --rounds=4096 生成
          name: cloudinitdisk
        - dataVolume:
            name: aa-rootfs
          name: rootfs
```

8. 审核 YAML 文件，完成后的完整 YAML 配置如下。

```yaml
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  labels:
    virtualization.cpaas.io/image-name: debian-2120-x86
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: debian
    virtualization.cpaas.io/image-supply-by: public
    vm.cpaas.io/name: aa
  name: aa
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        labels:
          vm.cpaas.io/reclaim-policy: Delete
          vm.cpaas.io/used-by: aa
        name: aa-rootfs
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 100Gi
          storageClassName: vm-cephrbd
          volumeMode: Block
        source:
          blank: {}
  running: true
  template:
    metadata:
      annotations:
        cpaas.io/creator: test@example.io
        cpaas.io/display-name: ""
        cpaas.io/updated-at: 2024-09-09T03:49:08Z
        kubevirt.io/latest-observed-api-version: v1
        kubevirt.io/storage-observed-api-version: v1
      creationTimestamp: null
      labels:
```

```
    virtualization.cpaas.io/image-name: debian-2120-x86
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: debian
    virtualization.cpaas.io/image-supply-by: public
    vm.cpaas.io/name: aa
spec:
  accessCredentials:
    - sshPublicKey:
        propagationMethod:
          qemuGuestAgent:
            users:
              - root
        source:
          secret:
            secretName: test-xeon
  affinity:
    nodeAffinity: {}
  architecture: amd64
  domain:
    devices:
      disks:
        - bootOrder: 1
          cdrom:
            bus: sata
          name: iso-disk
        - disk:
            bus: virtio
          name: cloudinitdisk
        - disk:
            bus: virtio
          name: rootfs
          bootOrder: 10
      interfaces:
        - bridge: {}
          name: default
    machine:
      type: q35
    resources:
      limits:
        cpu: "1"
        memory: 2Gi
      requests:
        cpu: "1"
        memory: 2Gi
```

```
      networks:
        - name: default
          pod: {}
      nodeSelector:
        kubernetes.io/arch: amd64
        vm.cpaas.io/baremetal: "true"
      volumes:
        - dataVolume:
            name: iso-upload-dv
          name: iso-disk
        - cloudInitConfigDrive:
            userData: |-
              #cloud-config
              disable_root: false
              ssh_pwauth: false
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: "<hash>"  # 使用 mkpasswd --method
=SHA-512 --rounds=4096 生成
          name: cloudinitdisk
        - dataVolume:
            name: aa-rootfs
          name: rootfs
```

9. 点击 创建。
10. 点击 操作 > **VNC** 登录。
11. 出现 **press any key boot from CD or DVD** 提示时，按任意键进入安装程序；若未看到提示，点击页面左上角的 发送远程命令，然后从下拉菜单选择 **Ctrl-Alt-Delete** 重启服务器。
    注意：若虚拟机详情页顶部出现提示 当前虚拟机有配置变更需重启生效，请重启。，可忽略该提示，无需重启。

3 安装 Linux 操作系统

1. 进入安装页面后，按照安装向导进行操作。本文以安装 Ubuntu 操作系统为例，不同操作系统安装过程中的配置项大致相同，故不再赘述。部分配置项说明如下。

配置项	说明
安装类型	建议选择最小安装以减小镜像大小。
存储配置	选择自定义存储，将磁盘格式化为 ext4 或 xfs 格式并挂载为根分区（/）。 注意：不要使用 LVM（创建卷组）。
SSH 配置	选择安装 OpenSSH 工具以支持 SSH 访问。

2. 等待安装完成。

4 修改 YAML 文件

1. 进入 容器平台。
2. 左侧导航栏点击 虚拟化 > 虚拟机。
3. 点击列表中的 虚拟机名称 进入详情页。
4. 点击 停止。
5. 点击右上角 操作 > 更新。
6. 切换至 YAML。
7. 确认 spec.template.spec.domain.devices.disks 中名为 **rootfs** 的磁盘的 bootOrder 为 1，若不是，修改为 1。
8. 删除 spec.template.spec.domain.devices.disks 中名为 **iso-disk** 的相关内容，具体如下。

```
- bootOrder: 1
  cdrom:
    bus: sata
  name: iso-disk
```

9. 删除 spec.template.spec.volumes 中名为 **iso-disk** 的相关内容，具体如下。

```
      - dataVolume:
          name: iso-upload-dv
        name: iso-disk
```

10. 点击 更新。
11. 点击 启动。

5 安装所需软件并修改配置

注意：以下命令及配置文件在不同操作系统间可能略有差异，请根据实际环境调整。

1. 输入用户名和密码登录操作系统。
2. 切换至 root 用户权限。
3. 安装软件包。
   - CentOS 系列执行：

     ```
     yum install cloud-utils cloud-init qemu-guest-agent vim
     ```

   - Debian 系列执行：

     ```
     apt install cloud-init cloud-guest-utils qemu-guest-agent vim
     ```

4. 编辑 SSHD 配置文件。
   1. 执行以下命令编辑 sshd_config 文件。

      ```
      vim /etc/ssh/sshd_config
      ```

   2. 添加以下配置。

      ```
      PermitRootLogin yes  # 允许 root 用户密码登录
      PubkeyAuthentication yes  # 允许密钥登录
      ```

   3. 保存修改后的配置。
5. 执行以下命令删除 root 用户默认密码。

```
passwd -d root
```

6. 修改源地址文件。

   1. 执行以下命令修改系统源地址文件，替换为合适的镜像站地址。

      ```
      vim /etc/apt/sources.list.d/ubuntu.sources
      ```

   2. 修改完成后保存配置。

7. 修改 cloud-init 配置，实现根目录自动扩容。

   1. 执行以下命令编辑 cloud.cfg 配置文件。

      ```
      vim /etc/cloud/cloud.cfg
      ```

   2. 添加以下配置内容。

      ```
      runcmd:
        - [growpart, /dev/vda, 1]  # growpart 命令用于扩展磁盘分区，此处扩展 /dev/vda1 分区。
        - [xfs_growfs, /dev/vda1]  # xfs_growfs 命令用于扩展 XFS 文件系统以占满分区空间，/dev/vda1 为需扩展的文件系统所在分区。扩展分区后，使用 xfs_growfs 确保文件系统同步扩展。
      ```

   3. 修改完成后保存配置。

8. 配置完成后，关闭操作系统。

6 导出并使用自定义 Linux 镜像

具体操作请参考 导出虚拟机镜像。

导出虚拟机镜像

该功能用于导出虚拟机的系统镜像并上传至对象存储，允许将对象存储中的文件添加为平台虚拟机镜像的来源。

目录

操作步骤

注意：以下所有操作均需在虚拟机所在集群的控制节点上执行。

1 停止虚拟机

1. 进入 管理员。
2. 在左侧导航栏点击 虚拟化管理 > 虚拟机。
3. 点击需要导出系统镜像的虚拟机名称，跳转至容器平台的虚拟机详情页面。
4. 点击 停止。

2 创建 vmexport 资源

1. 打开 CLI 工具。
2. 执行以下命令设置变量。

```
NAMESPACE=<namespace>
VM_NAME=<vm_name>
TTL_DURATION=2h
```

参数说明：

- **NAMESPACE**：虚拟机所在的命名空间名称；将 *<namespace>* 部分替换为该名称。
- **VM_NAME**：需要导出系统镜像的虚拟机名称；将 *<vm_name>* 部分替换为该名称。
- **TTL_DURATION**：导出任务的存活时间，默认为 2 小时，可根据需要延长。

3. 执行以下命令创建 vmexport 资源。

```
cat <<EOF | kubectl create -f -
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: export-$VM_NAME
  namespace: $NAMESPACE
spec:
  ttlDuration: $TTL_DURATION
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: $VM_NAME
EOF
```

若出现类似回显信息，表示创建成功。

```
virtualmachineexport.export.kubevirt.io/export-k1 created
```

4. 执行以下命令查看 vmexport 资源状态。

```
kubectl -n $NAMESPACE get vmexport export-$VM_NAME -w
```

回显信息：

```
NAME         SOURCEKIND       SOURCENAME   PHASE
export-k1    VirtualMachine   k1           Ready
```

5. 当回显信息中的 PHASE 字段变为 Ready 时，按 ctrl（control）+ c 停止 watch 操作。
6. 执行以下命令获取 TOKEN。

```
TOKEN=$(kubectl -n $NAMESPACE get secret export-token-export-$VM_NAME -o jsonpath={.data.token} | base64 -d)
```

3 下载虚拟机镜像文件

1. 执行以下命令获取指定命名空间中虚拟机导出 Pod 的 IP 地址，并存入 EXPORT_SERVER_IP 环境变量。

```
EXPORT_SERVER_IP=$(kubectl -n $NAMESPACE get po virt-export-export-$VM_NAME -o jsonpath='{.status.podIP}')
```

2. 执行以下命令设置 URL 环境变量，指向虚拟机的磁盘镜像文件。

```
URL=https://$EXPORT_SERVER_IP:8443/volumes/$VM_NAME-rootfs/disk.img.gz
```

3. 执行以下命令下载镜像文件，下载后的文件名为 disk.img.gz。

```
curl -k -O -H "x-kubevirt-export-token: $TOKEN" $URL
```

4 将虚拟机镜像文件上传至对象存储

将下载的镜像文件上传至对象存储。上传可使用任意 S3 工具，本文以 mc（minio-client）工具为例介绍。

1. 执行以下命令配置 mc 工具，连接指定的 S3 存储服务。

```
mc alias set minio <ENDPOINT> <ACCESSKEY> <SECRETKEY>
```

参数说明：

- ENDPOINT：S3 存储服务地址；将 `<ENDPOINT>` 部分替换为该地址。
- ACCESSKEY、SECRETKEY：用于认证的 S3 存储服务用户 ak 和 sk；相关信息请参考 MinIO Object Storage ↗。

2. 执行以下命令创建用于存储虚拟机镜像文件的桶。

```
mc mb minio/vmdisks
```

3. 执行以下命令将导出的虚拟机镜像文件 disk.img.gz 上传至创建的桶中。

```
mc put disk.img.gz minio/vmdisks
```

5 创建虚拟机镜像

1. 进入 管理员。
2. 在左侧导航栏点击 虚拟化管理 > 虚拟机镜像。
3. 点击 添加虚拟机镜像。
4. 在镜像地址中填写 *<ENDPOINT>*/vmdisks/disk.img.gz，将 <ENDPOINT> 部分替换为 S3 存储服务地址。其他参数说明请参考 添加虚拟机镜像。
5. 点击 添加。

权限

功能	操作	平台管理员	平台审计人员	项目管理员	命名空间管理员	开发人员
虚拟机镜像 `acp-virtualmachineimagetemplates`	查看	✓	✓	✓	✓	✓
	创建	✓	×	×	×	×
	更新	✓	×	×	×	×
	删除	✓	×	×	×	×

虚拟机

介绍

介绍

操作指南

创建虚拟机/虚拟机组

前提条件

注意事项

创建虚拟机

创建虚拟机组

虚拟机批量操作

操作步骤

使用 VNC 登

操作步骤

管理密钥对

创建密钥对

管理虚拟机

重置密码

监控与告警

监控

告警

虚拟机快速定位

前提条件

操作步骤

实用指南

配置 USB 主机直通

功能概述

使用场景

前提条件

操作步骤

操作结果

了解更多

虚拟机热迁移

Overview

约束与限制

前提条件

操作步骤

虚拟机恢复

操作步骤

在 KubeVirt

确保先决条件

物理 GPU 直通环境准备

配置虚拟机高可用性

概述

术语表

组件概述

fencing 和 remediation 事件流程

操作步骤

从现有虚拟机

前提条件

操作步骤

故障排除

Pod 迁移及虚拟机节点异常关闭恢

热迁移错误信息及解决方案

问题描述

原因分析

解决方案

介绍

KubeVirt 提供了诸如 **VirtualMachine** 和 **VirtualMachineInstance** 等 CRD（自定义资源定义），用于抽象虚拟机（VM）资源。基于这些 CRD，用户可以获得全面的虚拟机管理能力。在此基础上，**ACP Virtualization With KubeVirt** 通过提供一个 **Web Console**，进一步提升了易用性，使用户能够更轻松地执行各种操作。

操作指南

创建虚拟机/虚拟机组

前提条件

注意事项

创建虚拟机

创建虚拟机组

监控与告警

监控

告警

虚拟机批量操作

操作步骤

管理密钥对

创建密钥对

虚拟机快速定位

前提条件

操作步骤

使用 VNC 登

操作步骤

管理虚拟机

重置密码

创建虚拟机/虚拟机组

使用镜像创建虚拟机（VirtualMachineInstance），并将虚拟机调度到已安装 Kubevirt 组件且启用虚拟化的物理节点上。

您可以通过创建虚拟机创建单个虚拟机，也可以通过创建虚拟机组（virtualMachinePool）快速创建多个配置相同的虚拟机（VirtualMachineInstance）。

目录

前提条件

- 在使用镜像创建虚拟机前，请与平台管理员确认以下事项：
  - 目标集群为自建集群，且已部署 Kubevirt 组件。
  - 目标节点必须为启用虚拟化的物理节点。
  - 已将虚拟机镜像添加至平台。

- 如果需要使用虚拟机的物理 GPU 直通功能，请联系平台管理员进行以下配置：

  1. 获取 GPU 直通环境准备方案并准备所需环境。
  2. 准备所需的物理 GPU，并为虚拟机启用物理 GPU 直通相关功能。

注意事项

使用 Windows 虚拟机时，仅支持通过虚拟机镜像中设置的用户名/密码登录。请提前联系平台管理员获取相关信息。

创建虚拟机

操作步骤

注意：以下内容以表单方式创建虚拟机为例，您也可以切换为 YAML 格式进行操作。

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机。
3. 点击 创建虚拟机。
4. 在 基本信息 区域，填写虚拟机名称和显示名称，并设置标签或注解。

参数	说明
标签	用于选择对象和查找符合条件的对象集合。必须为键值对，例如：*app.kubernetes.io/name: hello-app*。
注解	用于向开发和运维团队提供任何信息。必须为键值对，例如：*cpaas.io/maintainer: kim*。

5. 设置机器类型并选择虚拟机镜像。

参数	说明
规格	可根据需求选择推荐使用场景或自定义资源限制。
物理 **GPU**（**Alpha**）	选择物理 GPU 型号；每台虚拟机仅能分配一个物理 GPU。 注意：虚拟机物理 GPU 直通是指在虚拟化环境中将实际的图形处理单元（GPU）直接分配给虚拟机，使其能够直接访问和利用物理 GPU，实现图形性能等同于直接运行在物理机上，避免虚拟图形适配器带来的性能瓶颈，提升整体性能。
镜像	选择平台管理员分配给平台项目的公共镜像。 注意：仅支持选择与集群架构相同**CPU** 架构的镜像。

6. 在 存储 区域，参照以下说明配置相关信息。

参数	说明
磁盘名称	存储磁盘名称；系统盘名称不可修改。
类型	• 根磁盘：系统自动创建 VirtIO 类型的 rootfs 系统盘，用于存储操作系统和数据。 • 数据盘：点击添加多个数据盘用于持久化数据存储，默认为 VirtIO 设备。 注意：数据盘名称不得与已有磁盘名称重复。
卷模式	• 文件系统：将磁盘挂载为挂载文件目录。 • 块设备：将磁盘挂载为块设备。
存储类	平台通过创建和管理持久卷声明来维护虚拟机磁盘，需指定用于动态创建持久卷声明的存储类。 不同存储类支持不同的卷模式；若所选卷模式无可用存储类，请联系管理员添加。
容量	虚拟机存储所需容量；系统盘最小为 20 G。

参数	说明
随虚拟机删除	默认为启用，且不可修改，表示删除虚拟机时磁盘数据也会被删除。

7. 在 网络 区域，参照以下说明配置相关信息。

参数	说明
IP 地址	• 默认选择 动态（**DHCP**）；虚拟机启动时动态分配 IP，虚拟机停止时释放 IP。 • 若绑定静态 **IP**，虚拟机重启后仍使用该 IP 地址。若当前项目无可用 IP，请先适当释放 IP。
网络模式	• 桥接：虚拟机与容器组共享同一 IP 地址，通过该 IP 地址对外通信。 • **NAT**：虚拟机分配内部 IP 地址，但对外通信时会转换为容器组 IP 地址。开放端口指虚拟机暴露的端口，如 SSH 服务端口 22；不填写开放端口表示所有端口均开放。
辅助网卡	根据需要添加辅助网卡。 注意： • 若需要辅助网卡功能或无可用辅助网卡网络类型，请联系平台管理员配置。 • SR-IOV 类型仅支持 x86_64 架构的 Linux 操作系统。 • 默认通过 DHCP 获取 IP 地址。 • 多次重启后，SR-IOV 虚拟机可能出现两个不同 VF 但 MAC 地址相同的情况。

8. 在 初始化设置 区域，参照以下说明配置相关信息。

参数	说明
密钥	始终使用 SSH 密钥进行远程登录验证。此方式无需密码验证，推荐使用密钥登录虚拟机。

参数	说明
	• 可使用平台已有密钥或立即创建新密钥；所有相关密钥可在 虚拟化 > 密钥对 页面查看。 • 仅持有私钥的人员可通过 SSH 访问虚拟机。若多人共同维护虚拟机，可关联多个密钥，并分配私钥给不同用户。若密钥泄露，可及时撤销关联密钥以减少损失。 • SSH 密钥的公钥以保密形式存储于平台，平台不存储您的私钥，请自行妥善保管。 • 请参考相关操作系统文档获取 root 用户密码。
密码	使用操作系统用户及密码进行登录验证，后续仍可更新为密钥方式。 • 该用户仅为初始账户；虚拟机创建成功后，您也可在虚拟机内创建其他操作系统用户进行登录。 • 平台加密存储您的 root 用户密码，您将无法再次查看明文密码，请自行妥善保管。
立即启动	默认为启用。启用后虚拟机创建完成后立即启动，否则仅创建虚拟机。

9. （可选）在 高级配置 区域，参照以下说明配置相关信息。

参数	说明
健康检查	• 存活检查：检测虚拟机是否处于健康状态；若检测结果异常，将根据健康检查配置判断是否重启实例。 • 可用性检查：检测虚拟机是否完成启动并处于正常服务状态；若虚拟机实例健康状态异常，将更新虚拟机状态。 相关参数说明请参见文档。
节点亲和性	• 首选：虚拟机尽可能调度到满足亲和性要求的节点。系统将结合亲和性权重及其他调度需求（如计算资源需求）确定可运行虚拟机的节点。

参数	说明
	• 必需：虚拟机仅调度到完全满足亲和性要求的节点。

10. 确认信息无误后，点击 创建。
    等待虚拟机状态从 创建中 变为 运行中。

相关操作

您可以点击列表页右侧的⋮图标或详情页右上角的 操作，根据需要更新或删除虚拟机。有关重置密码或更新密钥等其他相关操作，请参见管理虚拟机。

注意：

- 仅当虚拟机处于 异常、未知 或 已停止 状态时，才支持更新操作。
- 更新操作不支持显示虚拟机创建后单独挂载或创建的磁盘。
- 更新时默认立即启动为禁用状态，您可根据需要启用。

创建虚拟机组

操作步骤

注意：以下内容以表单方式创建虚拟机组为例，您也可以切换为 YAML 格式进行操作。

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机组。
3. 点击 创建虚拟机组。
4. 在 基本信息 区域，参照以下说明配置虚拟机组信息。

参数	说明
实例数量	虚拟机组创建的虚拟机数量。

参数	说明
实例间反亲和性	启用后，在调度多个虚拟机到节点时，会尽量将虚拟机分布到不同节点，提升虚拟机组的高可用性。
标签	可为虚拟机组添加标签。标签用于选择对象和查找符合条件的对象集合。必须为键值对，例如：*app.kubernetes.io/name: hello-app*。

5. 在 虚拟机模板 区域，参照创建虚拟机为组内所有虚拟机统一配置标签、注解、规格、镜像、存储等信息。
6. 确认信息无误后，点击 创建。
   提示：创建成功后，您可以进入 虚拟机 列表页查看通过虚拟机组创建的虚拟机信息。

虚拟机批量操作

执行启动、停止、重启和删除虚拟机等批量操作。

目录

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏点击 **Virtualization** > **Virtual Machines**。
3. 定位目标虚拟机，点击⋮对单个虚拟机进行操作，或参考下图对虚拟机进行批量操作。
   注意：
   - 当虚拟机处于挂起或停止状态时，可执行 启动/批量启动 操作；当虚拟机处于准备中、启动中、运行中、挂起、未知或异常状态时，可执行 停止/批量停止 操作；当虚拟机处于运行状态时，可执行 重启/批量重启 操作。
   - 对虚拟机执行强制 重启/停止 操作相当于切断虚拟机电源，可能导致未写入磁盘的数据丢失。
4. 按照界面提示完成操作。当虚拟机状态变为以下状态时，操作成功。

操作	状态
启动虚拟机	Running

操作	状态
停止虚拟机	Stopped
重启虚拟机	Running

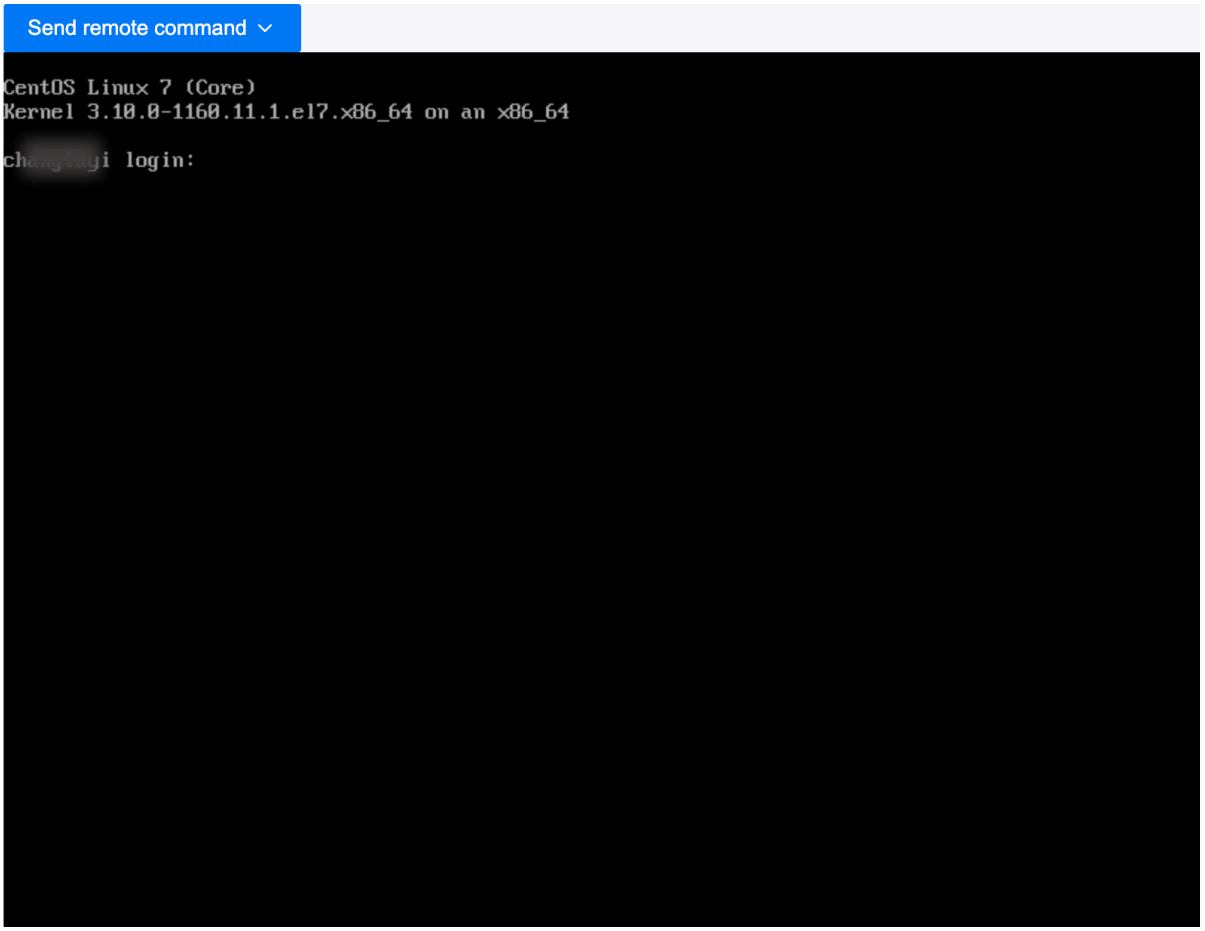
使用 VNC 登录虚拟机

通过 Web 控制台（VNC）登录虚拟机，作为紧急操作方法。

目录

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 点击⋮> **VNC Login**。
4. 控制台窗口将自动打开，您需要输入用户名和密码进行登录。



注意：

- 支持发送常用键盘命令。
- 支持复制和粘贴命令及参数。

管理密钥对

创建、更新或删除密钥对。

目录

创建密钥对

1. 进入 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Key Pairs**。
3. 点击 **Create Key Pair**。
   目前仅支持 SSH 类型的密钥对。您可以手动导入密钥，也可以让系统自动生成密钥对。使用系统生成的密钥对时，平台支持将私钥自动下载到本地，平台不会保存私钥。
4. 点击 **Create**。

更新密钥对

1. 进入 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Key Pairs**。

3. 找到 ***Key Pair Name***，点击⋮> **Update**。

4. 重新导入或让系统生成新的密钥对后，点击 **Update**。

删除密钥对

1. 进入 **Container Platform**。

2. 在左侧导航栏中，点击 **Virtualization** > **Key Pairs**。

3. 找到 ***Key Pair Name***，点击⋮> **Delete**，并确认。

管理虚拟机

目录

重置密码

重置 **root** 用户密码。该密码同时作为使用密码登录虚拟机时的登录密码。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 找到虚拟机并选择⋮> **Reset Password**。
4. 设置密码。
5. 点击 **Reset**。
   注意：请妥善保管您的密码。为保障环境安全，平台会对密码进行加密存储，您将无法再次查看明文密码。

更新密钥

更新 SSH 密钥。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 找到虚拟机并选择⋮> **Update Key**。
4. 选择一个或多个关联密钥，或点击 **Create Key**。
5. 选择是否立即重启；更新密钥需要重启虚拟机后生效。
6. 点击 **Update**。

更新规格

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 找到目标虚拟机并点击⋮> **Update Specifications**。
4. 根据平台推荐场景或自定义需求修改相关资源。
5. 选择是否 **Restart Immediately**；配置将在重启后生效。
6. 点击 **Update**。

热迁移

注意：如需热迁移操作相关文档，请联系管理员协助。

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 找到目标虚拟机并点击⋮> **Live Migration**。
4. 点击 **Confirm**。

更新 **NAT** 网络配置

使用 NAT 网络模式时，平台默认开放端口 22 用于 SSH 服务，您可根据需要开放其他端口。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 点击 *虚拟机名称*。
4. 在 **Basic Information** 区域，点击 **Open Port** 右侧的图标。
5. 输入端口号并按回车键确认。
6. 选择是否 **Restart Immediately**；配置将在重启后生效。
7. 点击 **Update**。

更新标签和注释

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 点击 *虚拟机名称*。
4. 在 **Basic Information** 区域，点击 **Tags** 或 **Annotations** 右侧的图标。
5. 按需配置并点击 **Update**。

添加服务

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 点击 *虚拟机名称*。
4. 在 **Login Information** 区域，点击 Internal Route 右侧的图标。
5. 参考 Create Service 页面快速添加虚拟机内部路由。
6. 点击 **Confirm**。

重装操作系统

强烈建议在重装操作系统前备份数据，以防数据丢失。

注意：此操作将清除虚拟机 系统盘 中的所有数据及所有 快照，且不可恢复，请谨慎操作！

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 找到虚拟机并选择⋮> **Reinstall Operating System**。
4. 在 **Reinstall Operating System** 窗口中，配置以下参数。
   - **Provisioning Method**：当前支持公有镜像。
   - **Select Image**：默认使用当前操作系统镜像进行重装。如需重装新操作系统，先选择虚拟机镜像所属的操作系统，再选择该操作系统对应的虚拟机镜像。
5. 点击 **Reinstall**。

配置 IP

为虚拟机分配动态分配（DHCP）IP 或绑定固定 IP；新 IP 在虚拟机重启后生效。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 **Virtualization** > **Virtual Machines**。
3. 找到目标虚拟机并点击⋮> **Configure IP**。
4. 配置 **IP Address**。
   - 填写可用 IP：绑定固定 IP 意味着即使重启，虚拟机也会一直使用该 IP 地址。
   - 留空：使用动态分配（DHCP）获取 IP，虚拟机启动时分配 IP，停止时释放 IP。
5. 选择是否 **Restart Immediately**；配置将在重启后生效。
6. 点击 **Configure**。

监控与告警

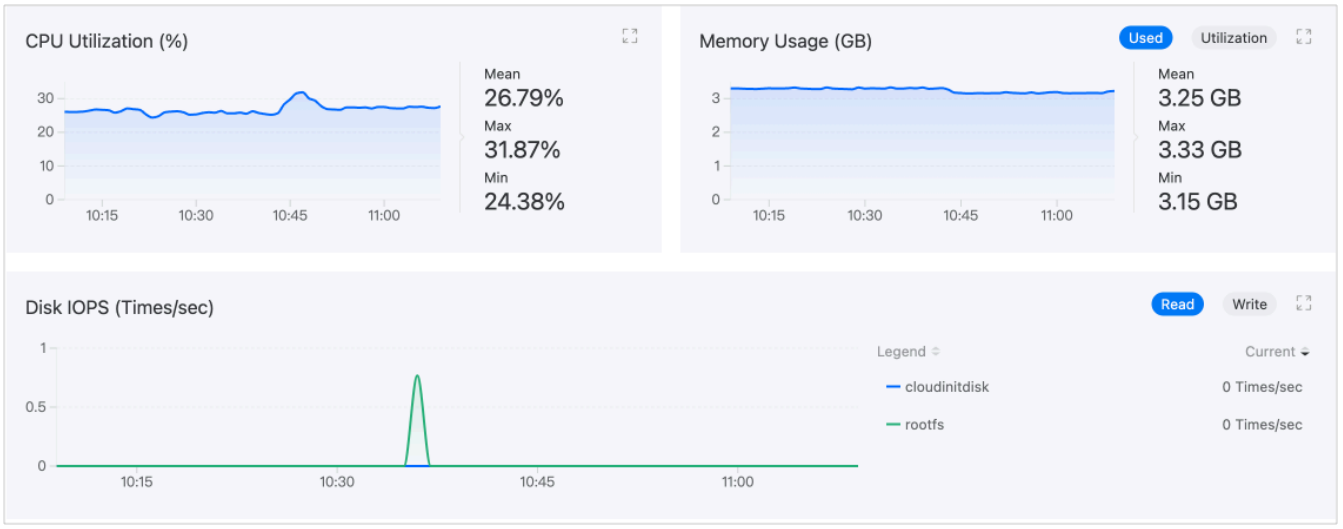
针对虚拟机的 CPU、内存、存储和网络进行监控和告警。为了便于及时告警，还可以配置通知策略。

直观呈现的监控数据可用于为运维巡检或性能调优提供决策支持，而完善的告警和通知机制将有助于保障虚拟机的稳定运行。

目录

监控

平台默认采集虚拟机常用的性能监控指标，包括 CPU、内存、存储和网络。进入 **Virtualization** > **Virtual Machines**，在虚拟机详情的 **Monitoring** 标签页中，可以查看指标的实时监控数据。



告警

配置告警策略

要启用告警，必须先创建告警策略。告警策略描述了您希望监控的对象、希望触发告警的条件，以及如何接收相关告警通知。进入 **Container Platform** > **Virtualization** > **Virtual Machines**，在虚拟机详情的 **Alerts** 标签页中点击 **Create Alert Policy** 完成配置。

参数	描述
Alert Type	- Metric Alert：监控对象为平台预定义的指标，如 *Memory Usage Rate*。 - Event Alert：监控对象为事件的原因，即虚拟机转变到当前状态的原因，例如 BackOff、Pulling、Failed。
Trigger Condition	由比较运算符、告警阈值和持续时间组成。通过将实时监控结果与设置的阈值进行比较，判断是否触发告警。 如果设置了持续时间，平台还会比较监控对象处于告警状态的持续时长。
Alert Level	- Hint：监控对象存在预期问题，暂时不会影响业务运行，但存在潜在风险。例如 CPU 使用率超过 70% 持续 3 分钟。 - Warning：监控对象存在可能影响正常业务运行的风险，若不及时处理可能导致问题。例如 CPU 使用率超过 80% 持续 3 分钟。 - Serious：监控对象存在已知问题，可能导致平台功能故障，影响正常业务运行。 - Disaster：监控对象发生故障，导致平台服务中断、数据丢失，影响严重。

提示：虚拟机告警功能与平台通用告警功能类似。有关更详细的配置指导，请参阅通用的 Alerts 文档。

处理告警

进入 **Alerts** 标签页，如有告警状态策略提示，请及时处理。

绑定通知策略

除了在 **Alerts** 标签页实时告警外，平台还支持通过邮件、短信等方式将告警信息发送给相关人员，通知其采取必要措施解决问题或防止故障。通知策略需联系管理员进行设置。

虚拟机快速定位

平台支持按集群展示虚拟机列表，方便平台管理员快速定位虚拟机所在的命名空间，并进行扩容、故障排查等操作，从而提升运维效率。

目录

前提条件

确保当前集群已启用虚拟化功能。详情请参见 Install。

操作步骤

1. 进入 管理员。
2. 在左侧导航栏点击 虚拟化管理 > 虚拟机。
3. 选择 *集群*，查看该集群下的虚拟机列表。
4. 可通过虚拟机名称、IP 地址或创建者快速定位虚拟机。
5. 点击虚拟机 *名称* 链接，进入该虚拟机详情页，可进行扩容、故障排查等操作。

实用指南

配置 USB 主机直通

功能概述

使用场景

前提条件

操作步骤

操作结果

了解更多

虚拟机热迁移

Overview

约束与限制

前提条件

操作步骤

虚拟机恢复

操作步骤

在 KubeVirt

确保先决条件

物理 GPU 直通环境准备

配置虚拟机高可用性

概述

术语表

组件概述

fencing 和 remediation 事件流程

操作步骤

从现有虚拟机

前提条件

操作步骤

配置 USB 主机直通

目录

功能概述

USB（通用串行总线）直通功能使您能够从虚拟机访问和管理 USB 设备。

使用场景

某些运行在虚拟机（VM）中的应用程序具有加密需求，需要与专用的 USB 设备交互。在这种情况下，需要将 USB 设备从主机直通到虚拟机。

前提条件

- 平台版本必须至少为 v3.18。

操作步骤

1 暴露 USB 设备

要将 USB 设备分配给虚拟机，必须通过 **ResourceName** 暴露该 USB 设备。可以通过编辑 kubevirt 命名空间下的 **HyperConverged CR** 中的 `spec.permittedHostDevices.usbHostDevices` 部分来配置。

以下是一个 USB 设备的示例配置，**ResourceName** 为 kubevirt.io/storage，厂商 ID 为 `0bda`，产品 ID 为 `8812`：

```yaml
spec:
  permittedHostDevices:
    usbHostDevices:
      - resourceName: kubevirt.io/storage
        selectors:
          - vendor: '0bda'
            product: '8812'
```

> 提示
>
> USB 设备的厂商和产品标识符可以通过 `lsusb` 命令获取。 例如：
>
> ```
> lsusb
> Bus 001 Device 007: ID 0bda:8812 Realtek Semiconductor Corp. RT
> L8812AU 802.11a/b/g/n/ac 2T2R DB WLAN Adapter
> ```
>
> 该命令列出所有连接的 USB 设备，其中 ID 显示厂商:产品对 。

2 将 USB 设备分配给虚拟机

现在，在虚拟机配置中，可以添加 `spec.domain.devices.hostDevices.deviceName` 来引用上一步中提供的 `ResourceName`，并为其分配一个本地名称。 例如：

```
spec:
  domain:
    devices:
      hostDevices:
        - deviceName: kubevirt.io/storage
          name: usb-storage
```

> **提示**
>
> 编辑配置前，请确保虚拟机已停止。

操作结果

完成配置后，在虚拟机内执行 `lsusb` 命令。如果输出中列出了主节点的 USB 设备，则表示直通成功。 例如：

```
lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 0bda:8812 Realtek Semiconductor Corp. RTL8812AU 80
2.11a/b/g/n/ac 2T2R DB WLAN Adapter
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

了解更多

您可能希望将多个 USB 设备直通到虚拟机，例如键盘、鼠标或智能卡设备。我们支持在同一 resourceName 下分配多个 USB 设备。配置方法如下：

1 暴露多个 **USB** 设备

```yaml
spec:
  permittedHostDevices:
    usbHostDevices:
      - resourceName: kubevirt.io/peripherals
        selectors:
          - vendor: '0bda'
            product: '8812'
          - vendor: '062a'
            product: '4102'
          - vendor: '072f'
            product: 'b100'
```

> **提示**
>
> 注意：所有 USB 设备必须物理连接并被主机检测到，才能确保成功分配给虚拟机。

2 将 USB 设备分配给虚拟机

```yaml
spec:
  domain:
    devices:
      hostDevices:
        - deviceName: kubevirt.io/peripherals
          name: local-peripherals
```

虚拟机热迁移

目录

Overview

虚拟机热迁移技术允许在不关闭或中断虚拟机的情况下，将虚拟机从一台物理服务器迁移到另一台。平台的虚拟机解决方案基于开源组件 KubeVirt 实现，默认采用一种称为 ProCopy 的模式进行热迁移。

ProCopy

ProCopy（预拷贝内存迁移）是一种常用的虚拟机迁移技术，通过预先拷贝虚拟机的内存数据，确保迁移过程中的服务连续性。具体流程如下：

1. 初始阶段：迁移开始时，源主机在虚拟机继续运行的同时，将虚拟机的内存页拷贝到目标主机。由于虚拟机持续运行，部分内存页可能在拷贝过程中被修改。
2. 迭代拷贝：源主机反复将被修改的内存页拷贝到目标主机，直到被修改的内存页数量降至可接受的水平。每轮拷贝称为一次迭代，经过每次迭代后未修改的内存页数量逐渐减少。
3. 停止并拷贝：当剩余未拷贝的内存页足够少时，虚拟机将短暂暂停（通常仅几秒到十几秒），在此期间将最后的内存页拷贝到目标主机，并将虚拟机的 CPU 和设备状态同步到目标主机。
4. 恢复运行：虚拟机在目标主机上恢复运行。

约束与限制

建议参与热迁移操作的两台物理机使用相同的硬件配置。如果配置不一致（例如 CPU 型号不同），迁移可能失败。

前提条件

请提前启用相关的虚拟机热迁移功能。

操作步骤

部署 **kubevirt-operator**

注意：详细步骤及参数说明请参考 Deploy Operator。

1. 进入 管理员。
2. 在左侧导航栏点击 应用商店管理 > **Operators**。
3. 点击页面顶部的 集群，切换到需要部署 Operator 的集群。
4. 在 OperatorHub 标签页中，点击 **KubeVirt HyperConverged Cluster Operator** 卡片上的部署。
5. 根据需要配置参数，点击 部署。可在 已部署 标签页查看 Operator 部署状态。

创建 HyperConverged 实例

具体创建步骤请参考 Create HyperConverged Instance。

准备虚拟机

注意：建议使用 Kube-OVN Underlay 网络。相关配置请参考 Create Subnet (Kube-OVN Underlay Network)。

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机。
3. 点击 创建虚拟机。
4. 在 基本信息 区域点击 更多 展开更多配置选项，点击 **Annotations** 对应的 添加，根据下表添加注解。如果网络插件是 Kube-OVN，则无需手动填写此注解。
   注意：由于表单限制，请先输入注解的 值，再输入注解的 键。

	注解
值	true
键	kubevirt.io/allow-pod-bridge-network-live-migration

5. 根据需要配置其他虚拟机参数。具体参数说明请参考相关产品文档。

参数	说明
卷模式	必须使用 块模式。
存储类	必须使用 **CephRBD** 块存储类型存储类。
网络模式	推荐使用 桥接。

6. 点击 创建。

启动热迁移

注意：虚拟机状态为 运行中 时才能启动热迁移。

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机。
3. 启动热迁移，有两种方式：
   - 在列表中需要迁移的虚拟机右侧点击⋮> 热迁移。
   - 点击列表中需要迁移的虚拟机名称进入详情页，然后点击 操作 > 热迁移。
4. 点击 确认。可通过 虚拟机状态 或 实时事件 查看迁移进度。当状态由 迁移中 变为 运行中，或实时事件出现类似 **Migrated: The VirtualMachineInstance migrated to node 10.1.1.1.** 的信息时，表示迁移成功。

虚拟机恢复

在某些场景下，例如错误修改 fstab 或文件系统错误需要 fsck，虚拟机可能无法正常启动。此时，可以利用救援模式修复根文件系统（rootfs）或从系统中取回数据。

目录

操作步骤

获取镜像地址

1. 在左侧导航栏点击 虚拟化管理 > 虚拟机镜像。
2. 选择平台提供的 来源 为 镜像仓库，操作系统选择 **CentOS** 或 **Ubuntu**。右侧点击⋮> 更新。
3. 复制并保存 镜像地址。本文以 `192.168.1.1:11443/3rdparty/vmdisks/centos:7.9` 为例。
4. 点击 取消。

修改虚拟机 YAML 文件

1. 进入 容器平台。
2. 在左侧导航栏点击 虚拟化 > 虚拟机。
3. 在需要修复的虚拟机右侧点击⋮> 停止 或 强制停止。
4. 在虚拟机右侧点击⋮> 更新。
5. 切换到 **YAML**，修改以下字段。

- 在 `spec.template.spec.domain.devices.disks` 下添加如下内容。添加 bootOrder 参数可以控制虚拟机启动时磁盘的优先级，bootOrder 值越小优先级越高。
  注意：如果原 `spec.template.spec.domain.devices.disks` 字段中已有 `bootOrder: 1`，请将原值调大，确保新添加的 bootOrder 值小于原值。

  ```
  disks:
    - bootOrder: 1
      disk:
        bus: virtio
      name: containerdisk
  ```

  修改后的 YAML 示例：

  ```
  domain:
    devices:
      disks:
        - bootOrder: 1  # 新增字段
          disk:
            bus: virtio
          name: containerdisk
        - disk:
            bus: virtio
          name: cloudinitdisk
        - disk:      # 将原 bootOrder: 1 值调大
            bus: virtio
          name: rootfs
          bootOrder: 10
        - disk:
            bus: virtio
          name: "1"
  ```

- 在 `spec.template.spec.volumes` 下添加如下内容。
  注意：请将以下 `image` 字段中的镜像地址替换为从获取镜像地址中获得的地址。

```
- containerDisk:
    image: 192.168.1.1:11443/3rdparty/vmdisks/centos:7.9
  name: containerdisk
```

  修改后的 YAML 示例：

```
volumes:
  - containerDisk:  # 新增字段
      image: 192.168.1.1:11443/3rdparty/vmdisks/centos:7.9
    name: containerdisk
  - dataVolume:
      name: k2-rootfs
    name: rootfs
  - dataVolume:
      name: k2-1
    name: "1"
```

6. 点击 更新。
   注意：修改 YAML 文件后，不要切换到 表单，直接点击 更新 即可。
7. 在虚拟机右侧点击⋮> 启动。

挂载原 rootfs 并进行修复

1. 使用原密码或密钥登录虚拟机，执行命令 `df -h /`，会发现 rootfs 文件系统已被替换。此时可以使用挂载相关命令挂载，或使用 fsck 相关命令检查并修复原文件系统。
2. 修复完成后，关闭虚拟机。

恢复虚拟机 YAML 文件

按照修改虚拟机 YAML 文件中的步骤，将虚拟机 YAML 文件恢复至原始状态。此时虚拟机即可正常启动。

在 KubeVirt 上克隆虚拟机

本文档提供了使用 KubeVirt 的 `VirtualMachineClone` API 克隆虚拟机（VM）的分步指导。

目录

确保先决条件

在启动虚拟机克隆操作之前，请确保满足以下要求：

- 支持快照的存储：Clone API 依赖于快照与恢复功能。虚拟机所使用的存储类必须支持卷快照，且该存储后端必须显式启用快照功能。

快速开始

按照以下快速步骤克隆虚拟机：

1. 准备克隆清单：

创建一个名为 `clone.yaml` 的文件，内容结构如下：

```yaml
apiVersion: clone.kubevirt.io/v1beta1
kind: VirtualMachineClone
metadata:
  name: example-vm-clone
  namespace: ns-where-vm-run
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: source-vm
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: target-vm
```

2. 执行克隆操作：

   应用该清单：

   ```
   kubectl create -f clone.yaml
   ```

3. 监控克隆状态：

   等待克隆成功完成：

   ```
   kubectl wait vmclone example-vm-clone --for condition=Ready
   ```

4. 验证克隆的虚拟机：

   查看克隆虚拟机的配置：

   ```
   kubectl get vm target-vm -o yaml
   ```

5. 修正 **DataVolume** 标签（**UI** 元数据）：

   平台 UI 通过标签 `vm.cpaas.io/used-by=<vm-name>` 将虚拟机与其磁盘关联，该标签会自动添加到每个 DataVolume 上。 克隆操作后，新创建的 DataVolume 会继承*源*虚拟机的标签，因此 UI 仍然认为它属于旧虚拟机。 需要更新新创建的 DV 上的标签，以便正确显示关联关系（功能不受影响）。

```
# 列出虚拟机所在命名空间的 DataVolumes；克隆的 DV 名称通常以 "restore-" 开头
kubectl get datavolumes -n <ns-where-vm-run>

# 覆盖标签指向克隆的虚拟机
kubectl label datavolume <new-dv-name> -n <ns-where-vm-run> vm.cpaas.io/used-by=<target-vm> --overwrite
```

了解 **VirtualMachineClone** 对象

查看完整的 **VirtualMachineClone** 示例

下面是一个带有详细内联注释的完整 `VirtualMachineClone` 资源示例：

```yaml
apiVersion: clone.kubevirt.io/v1beta1
kind: VirtualMachineClone
metadata:
  name: detailed-vm-clone
  namespace: ns-where-vm-run
spec:
  # 源虚拟机详情
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: vm-source

  # 目标虚拟机详情
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: vm-target

  # 从源复制的标签和注解过滤器
  labelFilters:
    - "*"
    - "!exclude-key/*"
  annotationFilters:
    - "include-annotations/*"

  # 模板过滤器，用于管理网络注解
  template:
    labelFilters:
      - "*"
    annotationFilters:
      - "!network-info/*"

  # 显式设置新的 MAC 地址
  newMacAddresses:
    eth0: "02-00-00-aa-bb-cc"

  # 显式设置 SMBios 序列号
  newSMBiosSerial: "unique-serial-1234"

  # JSON 补丁，用于进一步自定义克隆的虚拟机
  patches:
    - '{"op": "add", "path": "/metadata/labels/new-label", "value": "new-value"}'
```

```
    - '{"op": "replace", "path": "/spec/template/metadata/annotations/new
-annotation", "value": "updated-value"}'
```

理解各字段含义

- **source** 和 **target**：
  - 定义原始虚拟机（`source`）和克隆虚拟机（`target`）。
  - 如果省略 `target` 名称，则自动生成。
  - 两个虚拟机必须位于同一命名空间中。
- 标签和注解过滤器：
  - 使用通配符（`*`）和否定（`!`）控制是否复制源虚拟机的标签和注解。
- 模板标签和注解过滤器：
  - 主要用于管理网络相关注解，尤其是使用 Kube-OVN 等 CNI 时。
- **newMacAddresses**：
  - 可选，指定网络接口的新 MAC 地址。
  - 如果省略，将自动重新生成。
- **newSMBiosSerial**：
  - 可选，指定新的 SMBios 序列号。
  - 如果未提供，将基于虚拟机名称自动生成。
- **JSON** 补丁：
  - 直接应用于虚拟机规格的高级自定义。

检查克隆操作阶段

`VirtualMachineClone` 对象的 `.status.phase` 会根据克隆进度变化。下表说明了各阶段含义：

阶段	说明
SnapshotInProgress	正在创建源虚拟机的快照，克隆运行中虚拟机时的初始步骤。
CreatingTargetVM	快照完成；正在创建目标虚拟机的元数据和规格。
RestoreInProgress	DataVolume 和 PersistentVolumeClaim 创建中，正在从快照恢复数据。
Succeeded	操作成功完成，目标虚拟机和存储已准备就绪。
Failed	操作失败。请检查 `events` 和 `status.conditions` 以获取详细错误信息。
Unknown	无法确定克隆操作状态，可能表示控制器出现问题。

物理 GPU 直通环境准备

虚拟机中的物理 GPU 直通是指在虚拟化环境中将实际的图形处理单元（GPU）直接分配给虚拟机的过程。这样虚拟机可以直接访问和使用物理 GPU，实现与直接在物理机上运行相当的图形性能，避免了虚拟图形适配器带来的性能瓶颈，从而提升整体性能。

目录

约束与限制

物理 GPU 直通功能需要使用 kubevirt-gpu-device-plugin；但目前 kubevirt-gpu-device-plugin 尚无 ARM64 镜像，意味着该功能无法在 ARM64 CPU 架构的操作系统中使用。

前提条件

Chart 和镜像准备

获取以下 Chart 和镜像并上传至镜像仓库。本文档以 `build-harbor.example.cn` 作为示例仓库地址。具体获取 Chart 和镜像的方法，请联系相关人员。

Chart

- `build-harbor.example.cn/acp/chart-gpu-operator:v23.9.1`

镜像

- `build-harbor.example.cn/3rdparty/nvidia/gpu-operator:v23.9.0`
- `build-harbor.example.cn/3rdparty/nvidia/cloud-native/gpu-operator-validator:v23.9.0`
- `build-harbor.example.cn/3rdparty/nvidia/cuda:12.3.1-base-ubi8`
- `build-harbor.example.cn/3rdparty/nvidia/kubevirt-gpu-device-plugin:v1.2.4`
- `build-harbor.example.cn/3rdparty/nvidia/nfd/node-feature-discovery:v0.14.2`
- `build-harbor.example.cn/3rdparty/nvidia/cloud-native/k8s-driver-manager:v0.6.2`

启用 IOMMU

启用 IOMMU 的操作步骤因操作系统不同而异，请参考对应操作系统的文档。本文档以 CentOS 为例，所有命令均在终端执行。

1. 编辑 `/etc/default/grub` 文件，在 `GRUB_CMDLINE_LINUX` 配置项中添加 `intel_iommu=on iommu=pt`。

```
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=centos/root rhgb quiet i
ntel_iommu=on iommu=pt"
```

2. 执行以下命令生成 `grub.cfg` 文件。

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

3. 重启服务器。
4. 运行以下命令确认 IOMMU 是否成功启用。如果输出包含 `IOMMU enabled`，则表示启用成功。

```
dmesg | grep -i iommu
```

卸载 **Nvidia** 驱动

GPU 直通要求 GPU 使用 vfio-pci。 如果已安装 GPU 的 NVIDIA 驱动，请先卸载。

操作步骤

注意：以下所有命令均应在对应集群 Master 节点的 CLI 工具中执行，非特别说明除外。

创建命名空间

执行以下命令创建名为 `gpu-system` 的命名空间。若输出显示 `namespace/gpu-system created`，表示创建成功。

```
kubectl create ns gpu-system
```

部署 **gpu-operator**

1. 执行以下命令部署 gpu-operator。

```
# 将 <registry> 替换为 gpu-operator 镜像所在的仓库地址
# 例如：export REGISTRY=build-harbor.example.cn
export REGISTRY=<registry>

cat <<EOF | kubectl create -f -
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: gpu-operator
  namespace: gpu-system
spec:
  destination:
    cluster: ""
    namespace: "gpu-operator"
  source:
    charts:
    - name: acp/chart-gpu-operator
      releaseName: gpu-operator
      targetRevision: v23.9.1
    repoURL: $REGISTRY
  timeout: 120
  values:
    global:
      registry:
        address: $REGISTRY
    nfd:
      enabled: true
    sandboxWorkloads:
      enabled: true
      defaultWorkload: "vm-passthrough"
EOF
```

2. 执行以下命令检查 gpu-operator 是否同步完成。若 `SYNC` 显示为 `Synced`，表示同步成功。

```
kubectl -n gpu-system get apprelease gpu-operator
```

输出信息：

```
NAME           SYNC          HEALTH        MESSAGE         UPDATE   AGE
gpu-operator   Synced        Ready         chart synced    28s      32s
```

3. 执行以下命令获取所有节点名称，找到 GPU 节点名称。

```
kubectl get nodes -o wide
```

4. 执行以下命令检查 GPU 节点是否存在支持直通的 GPU。如果输出包含类似 `nvidia.com/GK210GL_TESLA_K80` 的 GPU 信息，表示存在支持直通的 GPU。

```
# 将 <gpu-node-name> 替换为步骤 3 中获取的 GPU 节点名称
kubectl get node <gpu-node-name> -o json | jq '.status.allocatable | wi
th_entries(select(.key | startswith("nvidia.com/"))) | with_entries(sel
ect(.value != "0"))'
```

输出信息：

```
{
    "nvidia.com/GK210GL_TESLA_K80": "8"
}
```

5. 至此，gpu-operator 已成功部署。

配置 Kubevirt

1. 执行以下命令启用 DisableMDEVConfiguration 功能。若返回类似 `hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched` 的信息，表示启用成功。

```
kubectl patch hco kubevirt-hyperconverged -n kubevirt --type='json' -p
='[{"op": "add", "path": "/spec/featureGates/disableMDevConfiguration",
"value": true }]'
```

2. 在 GPU 节点终端执行以下命令获取 pciDeviceSelector。输出中的 `10de:102d` 即为 pciDeviceSelector 的值。

```
lspci -nn | grep -i nvidia
```

输出信息：

```
04:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
05:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
08:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
09:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
85:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
86:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
89:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
8a:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [1
0de:102d] (rev a1)
```

3. 执行以下命令获取所有节点名称，找到 GPU 节点名称。

```
kubectl get nodes -o wide
```

4. 执行以下命令获取 resourceName。输出中的 `nvidia.com/GK210GL_TESLA_K80` 即为 resourceName 的值。

```
# 将 <gpu-node-name> 替换为步骤 3 中获取的 GPU 节点名称
kubectl get node <gpu-node-name> -o json | jq '.status.allocatable | wi
th_entries(select(.key | startswith("nvidia.com/"))) | with_entries(sel
ect(.value != "0"))'
```

输出信息：

```
{
    "nvidia.com/GK210GL_TESLA_K80": "8"
}
```

5. 执行以下命令添加直通 GPU。

   注意：将下述命令中 <pci-devices-id> 替换为步骤 2 中获取的 pciDeviceSelector 值时，**pciDeviceSelector** 中的所有字母必须转换为大写。例如，若获取的 pciDeviceSelector 值为 `10de:102d`，则应替换为 `export DEVICE=10DE:102D`。

   - 添加单张 GPU 卡

     ```
     # 将 <pci-devices-id> 替换为步骤 2 中获取的 pciDeviceSelector, 例如：expo
     rt DEVICE=10DE:102D
     export DEVICE=<pci-devices-id>
     # 将 <resource-name> 替换为步骤 4 中获取的 resourceName, 例如：export RES
     OURCE=nvidia.com/GK210GL_TESLA_K80
     export RESOURCE=<resource-name>

     kubectl patch hco kubevirt-hyperconverged -n kubevirt --type='json' -
     p='
     [
       {
         "op": "add",
         "path": "/spec/permittedHostDevices",
         "value": {
           "pciHostDevices": [
             {
               "externalResourceProvider": true,
               "pciDeviceSelector": "'"$DEVICE"'",
               "resourceName": "'"$RESOURCE"'"
             }
           ]
         }
       }
     ]'
     ```

   - 添加多张 GPU 卡

     注意：添加多张 GPU 卡时，用于替换 <pci-devices-id> 的每个 pciDeviceSelector 值必须唯一。

```
# 将 <pci-devices-id1> 替换为步骤 2 中获取的第一个 pciDeviceSelector
export DEVICE1=<pci-devices-id1>
# 将 <resource-name1> 替换为步骤 4 中获取的第一个 resourceName
export RESOURCE1=<resource-name1>
# 将 <pci-devices-id2> 替换为步骤 2 中获取的第二个 pciDeviceSelector
export DEVICE2=<pci-devices-id2>
# 将 <resource-name2> 替换为步骤 4 中获取的第二个 resourceName
export RESOURCE2=<resource-name2>

kubectl patch hco kubevirt-hyperconverged -n kubevirt --type='json' -
p='
[
  {
    "op": "add",
    "path": "/spec/permittedHostDevices",
    "value": {
      "pciHostDevices": [
        {
          "externalResourceProvider": true,
          "pciDeviceSelector": "'"$DEVICE1"'",
          "resourceName": "'"$RESOURCE1"'"
        },
        {
          "externalResourceProvider": true,
          "pciDeviceSelector": "'"$DEVICE2"'",
          "resourceName": "'"$RESOURCE2"'"
        }
      ]
    }
  }
]'
```

- 已添加 GPU 卡后新增 GPU 卡

```
# 将 <pci-devices-id> 替换为步骤 2 中获取的 pciDeviceSelector
export DEVICE=<pci-devices-id>
# 将 <resource-name> 替换为步骤 4 中获取的 resourceName
export RESOURCE=<resource-name>
# index 是从零开始的数组索引，用数字替换 <index>
# 例如：已添加一张 GPU 卡，现要添加另一张，index 应为 1，即 export INDEX=1
export INDEX=<index>

kubectl patch hco kubevirt-hyperconverged -n kubevirt --type='json' -
p='
[
  {
    "op": "add",
    "path": "/spec/permittedHostDevices/pciHostDevices/'"${INDEX}"'",
    "value": {
      "externalResourceProvider": true,
      "pciDeviceSelector": "'"$DEVICE"'",
      "resourceName": "'"$RESOURCE"'"
    }
  }
]'
```

结果验证

完成上述配置步骤后，创建虚拟机时能选择对应的物理 GPU，表示物理 GPU 直通环境已成功准备。

注意：如需配置物理 GPU 直通，请提前启用相关功能。

1. 进入 **Container Platform**。
2. 在左侧导航栏点击 **Virtualization** > **Virtual Machines**。
3. 点击 **Create Virtual Machine**。
4. 配置虚拟机的 **Physical GPU (Alpha)** 参数。

参数	说明
Physical GPU (Alpha)	选择已配置的物理 GPU 型号。每台虚拟机只能分配一块物理 GPU。

5. 至此，物理 GPU 直通环境已成功准备。

相关操作

删除带直通 GPU 的虚拟机

1. 进入 **Container Platform**。
2. 在左侧导航栏点击 **Virtualization** > **Virtual Machines**。
3. 在列表页，点击待删除虚拟机右侧的⋮> **Delete**，或点击待删除虚拟机名称进入详情页，点击 **Actions** > **Delete**。
4. 输入确认信息删除带直通 GPU 的虚拟机。

从 KubeVirt 移除 GPU 相关配置

1. 在 GPU 所在的对应集群 Master 节点，使用 CLI 工具执行以下命令，从 KubeVirt 移除 GPU 相关配置。

```
kubectl patch hco kubevirt-hyperconverged -n kubevirt --type='json' -p
='[{"op": "remove", "path": "/spec/permittedHostDevices"}]'
```

2. 删除后，通过 **Container Platform** 创建虚拟机时无法选择对应的物理 GPU 型号，表示删除成功。具体创建虚拟机步骤请参考 Select Physical GPU Model。

卸载 gpu-operator

1. 在 GPU 所在的对应集群 Master 节点，使用 CLI 工具执行以下命令卸载 gpu-operator。

```
kubectl -n gpu-system delete apprelease gpu-operator
```

输出信息：

```
apprelease.operator.alauda.io "gpu-operator" deleted
```

2. 执行命令后，若收到如下响应，表示 gpu-operator 已成功卸载。

```
kubectl -n gpu-system get apprelease gpu-operator
```

输出信息：

```
Error from server (NotFound): appreleases.operator.alauda.io "gpu-opera
tor" not found
```

配置虚拟机高可用性

目录

概述

硬件并不完美，软件也存在缺陷。当发生节点级别的故障，例如内核挂起或网络接口控制器（NIC）故障时，集群的工作量不会减少，受影响节点上的工作负载需要在其他地方重新启动。然而，某些工作负载，如 ReadWriteOnce (RWO) 卷和 StatefulSets，可能要求最多只能有一个实例运行。

影响这些工作负载的故障可能导致数据丢失、损坏或两者兼有。因此，确保节点达到一个安全状态（称为 fencing）后再启动工作负载的恢复（称为 remediation），理想情况下还包括节点本身的恢复，这一点非常重要。

依赖管理员干预来确认节点和工作负载的真实状态并不总是实际可行的。为促进这种干预，Alauda Container Platform 提供了多个组件，用于自动化故障检测、fencing 和 remediation。

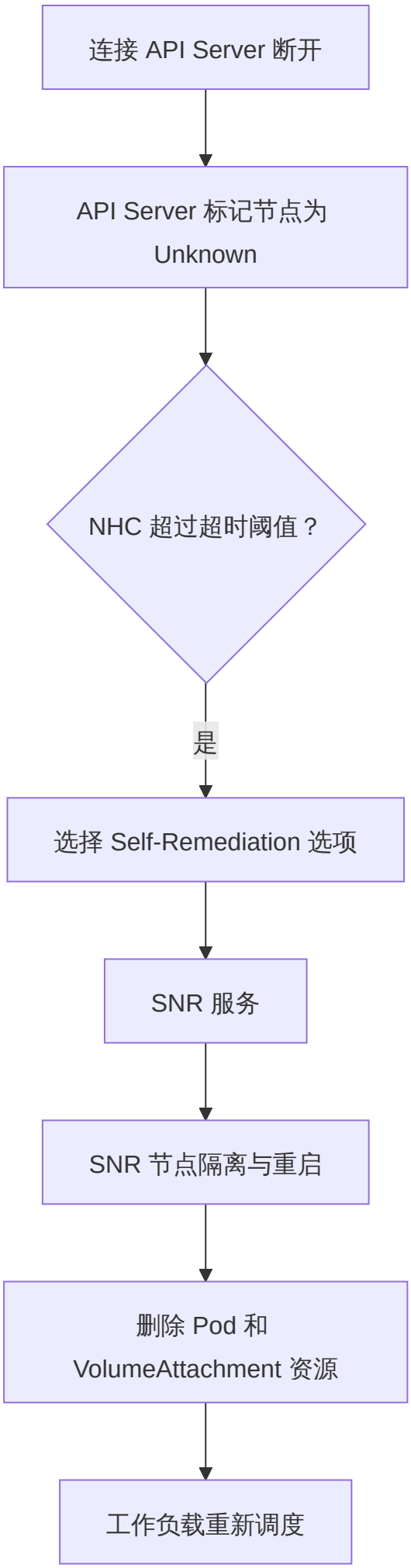
术语表

缩写	术语
SNR	Self Node Remediation
NHC	Node Health Check

组件概述

- **Self Node Remediation Operator**
  Self Node Remediation Operator 是 Alauda Container Platform 的一个附加 Operator，实现了一个外部的 fencing 和 remediation 系统，该系统会重启不健康的节点并删除诸如 Pod 和 VolumeAttachment 等资源。重启确保工作负载被 fencing，资源删除加速受影响工作负载的重新调度。与其他外部系统不同，Self Node Remediation 不需要任何管理接口，例如智能平台管理接口（IPMI）或节点配置的 API。
  Self Node Remediation 可被故障检测系统使用，如 Machine Health Check 或 Node Health Check。
- **Node Health Check Operator**
  Node Health Check Operator 是 Alauda Container Platform 的一个附加 Operator，实现了一个监控节点状态的故障检测系统。它没有内置的 fencing 或 remediation 系统，因此必须配置一个提供这些功能的外部系统。默认情况下，它配置为使用 Self Node Remediation 系统。

fencing 和 **remediation** 事件流程


连接 API Server 断开
API Server 标记节点为 Unknown
NHC 超过超时阈值？
是
选择 Self-Remediation 选项
SNR 服务
SNR 节点隔离与重启
删除 Pod 和 VolumeAttachment 资源
工作负载重新调度

操作步骤

1 Operator 列表

- 下载 与您的平台架构对应的 **Alauda Build of SelfNodeRemediation** 安装包。
- 使用 Upload Packages 机制上传 **Alauda Build of SelfNodeRemediation** 安装包。
- 下载 与您的平台架构对应的 **Alauda Build of NodeHealthCheck** 安装包。
- 使用 Upload Packages 机制上传 **Alauda Build of NodeHealthCheck** 安装包。

2 部署 Self Node Remediation Operator

1. 登录，进入 管理员 页面。
2. 点击 **Marketplace** > **OperatorHub** 进入 **OperatorHub** 页面。
3. 找到 **Alauda Build of SelfNodeRemediation**，点击 **Install**，进入 **Install Alauda Build of SelfNodeRemediation** 页面。
   配置参数：

参数	推荐配置
Channel	默认通道为 `stable`。
Installation Mode	`Cluster`：集群中所有命名空间共享单个 Operator 实例进行创建和管理，资源使用更低。
Installation Place	选择 `Recommended`，命名空间仅支持 **workload-availability**。
Upgrade Strategy	`Manual`：当 Operator Hub 有新版本时，需要手动确认升级 Operator 到最新版本。

3 配置 Self Node Remediation Operator（可选）

Self Node Remediation Operator 会创建名为 `self-node-remediation-config` 的 `SelfNodeRemediationConfig` CR。该 CR 创建在 Self Node Remediation Operator 所在的命名空间中。

> **注意**
>
> 修改 `SelfNodeRemediationConfig` CR 会重新创建 Self Node Remediation daemon set。

`SelfNodeRemediationConfig` CR 示例 YAML 文件如下：

```yaml
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationConfig
metadata:
  name: self-node-remediation-config
  namespace: workload-availability
spec:
  safeTimeToAssumeNodeRebootedSeconds: 180
  watchdogFilePath: /dev/watchdog
  isSoftwareRebootEnabled: true
  apiServerTimeout: 15s
  apiCheckInterval: 5s
  maxApiErrorThreshold: 3
  peerApiServerTimeout: 5s
  peerDialTimeout: 5s
  peerRequestTimeout: 5s
  peerUpdateInterval: 15m
  hostPort: 30001
  customDsTolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io.infra
    operator: Equal
    value: "value1"
    tolerationSeconds: 3600
```

参数说明：

参数	描述
safeTimeToAssumeNodeRebootedSeconds	指定 Operator 在恢复运行于不健康节点上的受影响工作负载前等待的可选时间。若在故障节点上工作负载仍在运行时启动替换 Pod，可能导致数据损坏并违反最多运行一次的语义。Operator 会根据 ApiServerTimeout、

参数	描述
	ApiCheckInterval、MaxApiErrorThreshold、PeerDialTimeout、PeerRequestTimeout 字段的值，以及 watchdog 超时和恢复时的集群规模，计算出最小等待时间。
watchdogFilePath	指定节点中 watchdog 设备的文件路径。如果输入了错误的 watchdog 设备路径，Self Node Remediation Operator 会自动检测 softdog 设备路径。 如果没有 watchdog 设备，`SelfNodeRemediationConfig` CR 会使用软件重启。
isSoftwareRebootEnabled	指定是否启用不健康节点的软件重启。默认值为 `true`。若要禁用软件重启，将该参数设置为 `false`。
apiServerTimeout	指定检查与每个 API server 连接的超时时间。超时后，Operator 开始 remediation。超时时间必须大于等于 10 毫秒。
apiCheckInterval	指定检查与每个 API server 连接的频率。超时时间必须大于等于 1 秒。
maxApiErrorThreshold	指定阈值，达到该阈值后节点开始联系其 peer。阈值必须大于等于 1 秒。
peerApiServerTimeout	指定 peer 连接 API server 的超时时间。超时时间必须大于等于 10 毫秒。
peerDialTimeout	指定与 peer 建立连接的超时时间。超时时间必须大于等于 10 毫秒。
peerRequestTimeout	指定从 peer 获取响应的超时时间。超时时间必须大于等于 10 毫秒。

参数	描述
peerUpdateInterval	指定更新 peer 信息（如 IP 地址）的频率。超时时间必须大于等于 10 秒。
hostPort	指定 Self Node Remediation 代理用于内部通信的端口，可选。值必须大于 0，默认端口为 30001。
customDsTolerations	指定运行在 DaemonSet 上的 Self Node Remediation 代理的自定义容忍度，以支持不同类型节点的 remediation。

> 注意
>
> - Self Node Remediation Operator 默认在部署命名空间创建该 CR。
> - CR 名称必须为 `self-node-remediation-config`。
> - 只能存在一个 `SelfNodeRemediationConfig` CR。
> - 删除 `SelfNodeRemediationConfig` CR 会禁用 Self Node Remediation。

4 配置 Self Node Remediation 模板（可选）

Self Node Remediation Operator 还会创建 SelfNodeRemediationTemplate Custom Resource Definition (CRD)。该 CRD 定义了针对节点的 remediation 策略，旨在更快地恢复工作负载。可用的 remediation 策略包括：

- **Automatic**
  该策略简化 remediation 过程，由 Self Node Remediation Operator 决定集群最合适的 remediation 策略。该策略会检查集群中是否支持 `OutOfServiceTaint` 策略。如果支持，Operator 选择 `OutOfServiceTaint` 策略；如果不支持，则选择 `ResourceDeletion` 策略。`Automatic` 是默认的 remediation 策略。
- **ResourceDeletion**
  该策略通过删除节点上的 Pod，而不是删除节点对象来进行 remediation。
- **OutOfServiceTaint**

该策略通过在节点上设置 `OutOfServiceTaint`，隐式导致节点上的 Pod 和相关的 volume attachments 被移除，而不是删除节点对象。

Self Node Remediation Operator 会为策略 `self-node-remediation-automatic-strategy-template` 创建 SelfNodeRemediationTemplate CR，该策略使用 Automatic remediation 策略。

SelfNodeRemediationTemplate CR 示例 YAML 文件如下：

```yaml
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationTemplate
metadata:
  creationTimestamp: "2022-03-02T08:02:40Z"
  name: self-node-remediation-<remediation_object>-deletion-template
  namespace: workload-availability
spec:
  template:
    spec:
      remediationStrategy: <remediation_strategy>
```

参数说明：

参数	描述
remediation_strategy	可选值：Automatic、ResourceDeletion、OutOfServiceTaint

5 部署 Node Health Check Operator

1. 登录，进入 管理员 页面。
2. 点击 **Marketplace** > **OperatorHub** 进入 **OperatorHub** 页面。
3. 找到 **Alauda Build of NodeHealthCheck**，点击 **Install**，进入 **Install Alauda Build of NodeHealthCheck** 页面。
   配置参数：

参数	推荐配置
Channel	默认通道为 `stable`。

参数	推荐配置
Installation Mode	`Cluster`：集群中所有命名空间共享单个 Operator 实例进行创建和管理，资源使用更低。
Installation Place	选择 `Recommended`，命名空间仅支持 **workload-availability**。
Upgrade Strategy	`Manual`：当 Operator Hub 有新版本时，需要手动确认升级 Operator 到最新版本。

6 创建 NodeHealthCheck 实例

在集群控制节点执行以下命令：

命令

```
cat << EOF | kubectl apply -f -
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nodehealthcheck-<name>
spec:
  minHealthy: <minHealthy>
  remediationTemplate:
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    kind: SelfNodeRemediationTemplate
    name: self-node-remediation-automatic-strategy-template
    namespace: workload-availability
  selector: <selector>
  unhealthyConditions:
    - duration: 300s
      status: 'False'
      type: Ready
    - duration: 300s
      status: Unknown
      type: Ready
EOF
```

示例

```
cat << EOF | kubectl apply -f -
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nodehealthcheck-worker
spec:
  minHealthy: 51%
  remediationTemplate:
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    kind: SelfNodeRemediationTemplate
    name: self-node-remediation-automatic-strategy-template
    namespace: workload-availability
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/control-plane
        operator: DoesNotExist
      - key: node-role.kubernetes.io/master
        operator: DoesNotExist
  unhealthyConditions:
    - duration: 300s
      status: 'False'
      type: Ready
    - duration: 300s
      status: Unknown
      type: Ready
EOF
```

参数说明：

参数	描述
name	资源名称
minHealthy	指定健康节点的最小比例。只有当健康节点比例大于或等于该值时，才会修复故障节点。默认值为 51%。
selector	指定 LabelSelector 以匹配需要检测和自我修复的节点。请避免在同一个实例中同时指定 control-plane 和 worker 节点。

7 验证（可选）

模拟虚拟机运行节点的故障，确认虚拟机能自动调度到其他节点运行。

从现有虚拟机创建虚拟机模板

本文档介绍如何从现有虚拟机创建可复用的虚拟机（VM）模板，以便快速部署新的虚拟机。

目录

前提条件

- 已正确部署和配置的 KubeVirt 环境。
- 可访问 Web Console 和 kubectl 工具。
- 已配置好并安装必要软件的虚拟机。

操作步骤

步骤 1：在虚拟机上进行基本配置

在虚拟机内部，执行以下操作：

- 安装 cloud-init↗。
- 安装 `qemu-guest-agent`。
- 安装任何所需的软件。

安装完成后，运行以下命令清理 cloud-init 数据并关闭虚拟机：

```
cloud-init clean
shutdown -h now
```

步骤 2：创建虚拟机快照

通过 KubeVirt Web Console：

1. 进入 **Virtualization** > **Virtual Machines**。
2. 选择用于作为模板的虚拟机。
3. 点击 **Actions**，选择 **Create Snapshot**，为快照命名并确认。

步骤 3：获取磁盘快照资源名称

通过以下任一方式获取完整的快照资源名称：

- 通过 **Web Console**：
  - 进入 **Storage** > **Volume Snapshots**。
  - 查找并记录“Data Source”下的完整快照资源名称。
- 使用 **kubectl**：

  ```
  kubectl get volumesnapshots -n <NAMESPACE>
  ```

  从输出中记录完整的快照资源名称。

步骤 4：创建 DataSource 资源

在 `kube-public` 命名空间中创建如下 DataSource 资源，确保将占位符替换为实际的快照名称和命名空间：

```yaml
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataSource
metadata:
  annotations:
    cpaas.io/display-name: MicroOS-Clone
  labels:
    virtualization.cpaas.io/image-os-arch: amd64
    virtualization.cpaas.io/image-os-type: linux
    virtualization.cpaas.io/storage-class: cephrbd
    virtualization.cpaas.io/access-mode: ReadWriteMany
    virtualization.cpaas.io/size: 30Gi
    virtualization.cpaas.io/volume-mode: Block
  name: microos-clone
  namespace: kube-public
spec:
  source:
    snapshot:
      name: <Your Snapshot Resource Name>
      namespace: <Your Snapshot Namespace>
```

标签参数说明：

键	可选值	说明
virtualization.cpaas.io/image-os-arch	amd64, arm64	虚拟机操作系统架构
virtualization.cpaas.io/image-os-type	linux, windows	虚拟机操作系统类型
virtualization.cpaas.io/storage-class	存储类名称	默认存储类，可在创建虚拟机时调整
virtualization.cpaas.io/access-mode	ReadWriteOnce, ReadWriteMany	磁盘访问模式；虚拟机热迁移时需使用 ReadWriteMany

键	可选值	说明
virtualization.cpaas.io/size	容量（Gi、Ti 等）	默认磁盘大小，请根据实际需求填写
virtualization.cpaas.io/volume-mode	Block, Filesystem	磁盘卷模式；推荐使用 Block 模式以获得更好性能

重要提示：

- 命名空间必须为 `kube-public`。
- 这些与磁盘相关的参数可在创建虚拟机时修改，但提供默认值可以简化操作流程。

步骤 5：使用模板创建新的虚拟机

1. 访问 KubeVirt Web Console，进入 **Container Platform** > **Virtualization** > **Virtual Machines**。
2. 点击 **Create Virtual Machine**。
3. 在 **Image** 部分，选择 **Image Instance** 作为提供方式。
4. 从下拉列表中选择刚创建的 DataSource。
5. 根据需要配置其他参数，完成虚拟机创建。

至此，您已成功使用虚拟机模板创建并部署了新的虚拟机。

故障排除

Pod 迁移及虚拟机节点异常关闭恢复

目录

问题描述

无论节点是正常关闭还是发生异常宕机，该节点上运行的虚拟机 Pod 都不会自动迁移到其他健康节点。

原因分析

平台基于开源组件 KubeVirt 实现虚拟机方案，但从 KubeVirt 角度无法区分虚拟机真实宕机和网络等原因导致的连接失败。如果盲目将虚拟机迁移到其他节点，可能导致同一虚拟机存在多个实例。

解决方案

维护虚拟机节点时，需按照本文档进行手动操作。无论是正常关闭还是异常宕机，虚拟机 Pod 都必须手动驱逐或强制删除。

注意：以下命令需在对应集群的主节点上执行。

正常关闭时虚拟机 Pod 迁移

1. 在 CLI 工具中执行以下命令获取节点信息，返回信息中的 `NAME` 字段即为 `Node-Name`。

   ```
   kubectl get nodes
   ```

   输出：

   ```
   NAME             STATUS   ROLES                  AGE   VERSION
   1.1.1.211   Ready    control-plane,master   99d   v1.28.8
   ```

2. （可选）执行以下命令查看该节点下的虚拟机实例。

   ```
   kubectl get vmis --all-namespaces -o wide | grep <Node-Name>  # 将命令中的 <Node-Name> 替换为步骤 1 中获取的 Node-Name
   ```

   输出：

   ```
   test-test         vm-t-export-clone   13d     Running      1.1.1.1   1.1.1.211   True    False
   ```

3. 在正常关闭前，执行以下命令驱逐即将关闭节点上的所有虚拟机 Pod，若输出如下则表示驱逐成功。

   ```
   kubectl drain <Node-Name> --delete-local-data --ignore-daemonsets=true --force --pod-selector=kubevirt.io=virt-launcher   # 将命令中的 <Node-Name> 替换为待关闭节点的 Node-Name
   ```

   输出：

```
Flag --delete-local-data has been deprecated, This option is deprecated
and will be deleted. Use --delete-emptydir-data.
node/1.1.1.211 cordoned
evicting pod test-test/virt-launcher-vm-t-export-clone-hmnkk
pod/virt-launcher-vm-t-export-clone-hmnkk evicted
node/1.1.1.211 drained
```

4. 在所有虚拟机启动到其他节点后，关闭该节点。
5. 节点关闭并重启后，执行以下命令将节点标记为可调度。

```
kubectl uncordon <Node-Name> # 将命令中的 <Node-Name> 替换为已关闭并重启的节点 Node-Name
```

输出：

```
node/1.1.1.211 uncordoned
```

6. 此时，该节点上原有的虚拟机实例已迁移至其他健康节点，节点重启后可用于新的 Pod 调度。

异常关闭恢复

1. 在 CLI 工具中执行以下命令获取节点信息，返回信息中的 `NAME` 字段即为 `Node-Name`。

```
kubectl get nodes
```

输出：

```
NAME            STATUS   ROLES                  AGE   VERSION
1.1.1.211   Ready    control-plane,master   99d   v1.28.8
```

2. 执行以下命令强制删除该节点上的所有虚拟机 Pod。

```
kubectl get po -A -l kubevirt.io=virt-launcher -o wide | grep <Node-Name> | awk '{print "kubectl delete pod --force -n " $1, $2}'  | bash  # 将命令中的 <Node-Name> 替换为异常宕机节点的 Node-Name。
```

3. 执行以下命令删除该节点上的 volume attachments。

```
kubectl get volumeattachments.storage.k8s.io | grep <Node-Name> | awk '{print $1}' | xargs kubectl delete volumeattachments.storage.k8s.io  # 将命令中的 <Node-Name> 替换为异常宕机节点的 Node-Name。
```

4. 执行以下命令查询异常宕机节点上是否存在带有标签 kubevirt.io=virt-api 的 Pod。

```
kubectl -n kubevirt get po -l kubevirt.io=virt-api -o wide | grep <Node-Name> # 将命令中的 <Node-Name> 替换为异常宕机节点的 Node-Name。
```

若存在，执行以下命令删除这些 Pod。

```
kubectl -n kubevirt get po -l kubevirt.io=virt-api -o name | xargs kubectl -n kubevirt delete --force --grace-period=0
```

5. 执行以下命令查询异常宕机节点上是否存在带有标签 kubevirt.io=virt-controller 的 Pod。

```
kubectl -n kubevirt get po -l kubevirt.io=virt-controller -o wide | grep <Node-Name> # 将命令中的 <Node-Name> 替换为异常宕机节点的 Node-Name。
```

若存在，执行以下命令删除这些 Pod。

```
kubectl -n kubevirt get po -l kubevirt.io=virt-controller -o name | xargs kubectl -n kubevirt delete --force --grace-period=0
```

6. 此时，虚拟机实例将在节点异常关闭后迁移至其他健康节点。

热迁移错误信息及解决方案

错误信息	原因	解决方案
cannot migrate VMI which does not use masquerade, bridge with <annotation> VM annotation or a migratable plugin to connect to the pod network	虚拟机的网络配置不支持热迁移。	请检查以下配置： • 检查当前集群使用的 CNI 网络插件，推荐使用 Kube-OVN。 • 检查虚拟机对应 YAML 文件的 `metadata.annotations` 和 `spec.template.metadata.annotations` 字段中是否存在 "kubevirt.io/allow-pod-bridge-network-live-migration": "true" 注解；如果没有，请手动添加。
• cannot migrate VMI: Unable to determine if PVC <pvc name> is shared, live migration requires that all PVCs must be shared (using ReadWriteMany access mode) • cannot migrate VMI: PVC <pvc name> is not shared, live migration requires that all PVCs must be shared (using	虚拟机的存储类型不支持多节点读写（RWX）访问模式。	虚拟机相关参数创建后无法修改，因此请重新创建虚拟机，并选择支持多节点读写（RWX）的存储类型；推荐使用 CephRBD 块存储。如果重新创建后仍有问题，请联系相关人员协助处理。

错误信息	原因	解决方案
ReadWriteMany access mode) • cannot migrate VMI: Backend storage PVC is not RWX • cannot migrate VMI with non-shared HostDisk		
其他错误信息	虚拟机不支持热迁移。	请联系相关人员协助处理。

网络

介绍

介绍

优势

操作指南

配置网络

配置 IP

通过 IP 直接连接虚拟机

添加服务

实用指南

通过网络策略控制虚拟机网络请求

操作步骤

结果验证

配置 SR-IOV

术语

约束与限制

前提条件

配置虚拟机

前提条件

操作步骤

配置多网卡虚拟机

介绍

ACP Virtualization With KubeVirt 与 Kube-OVN 深度集成，扩展了对传统虚拟机（VM）网络需求的支持，并针对特定场景优化了性能。

目录

优势

- IPv6 支持
  完整支持 IPv6。
- 静态 IP 保留
  确保虚拟机重启后保持相同的 IP 地址，符合传统虚拟机的使用习惯。
- 多网络模式支持
  支持容器网络、SR-IOV 等多种网络模式，满足多样化的用户场景。

操作指南

配置网络

配置 IP

通过 IP 直接连接虚拟机

添加服务

配置网络

目录

配置 IP

参考 Configure IP

通过 IP 直接连接虚拟机

参考 Preparing Kube-OVN Underlay Physical Network

添加服务

参考 Add Service

实用指南

通过网络策略控制虚拟机网络请求

操作步骤

结果验证

配置 SR-IOV

术语

约束与限制

前提条件

操作步骤

配置虚拟机

前提条件

操作步骤

配置多网卡虚拟机

前提条件

操作步骤

通过网络策略控制虚拟机网络请求

平台的虚拟机方案基于开源组件 KubeVirt 实现，实际上运行在 Pod 中。通过利用网络策略的功能，可以控制虚拟机的入站和出站请求。

目录

操作步骤

1. 进入 容器平台。
2. 在左侧导航栏点击 网络 > 网络策略。
3. 点击 创建网络策略。
4. 根据需要配置以下参数。

参数	说明
关联方式	• 计算组件：根据需要选择目标计算组件，建议选择 全部 作为目标计算组件。 • 标签选择器：根据 Pod 的标签进行匹配。

参数	说明
方向	• 入站：从外部发送到 Pod 的请求。 • 出站：从 Pod 发送到外部的请求；如果禁止虚拟机请求某个外部地址，选择此项。
协议	选择 TCP 或 UDP。 注意： • 虚拟机中使用域名请求外部服务时，需要添加 UDP 协议白名单，因为 DNS 协议使用 UDP。 • 表单不支持配置 ICMP 协议；一旦启用白名单规则，ICMP 协议将被禁用，导致无法进行 Ping 操作。
访问端口	指定允许入站或出站的端口流量。如果此项为空，默认允许所有端口的流量。 注意：此处需允许 1053 和 53 端口的 UDP 和 TCP 协议，以允许 DNS 流量出站，否则域名解析将失败。
远程类型	指定允许访问的远程类型，选项包括：计算组件、命名空间和 IP 段。
排除远程	当远程类型为 **IP** 段 时，从白名单中移除指定 IP（即禁止访问）。单个 IP 可输入为 `IP/32`。 注意：此项仅支持输入 IP；若域名对应的 IP 不明确，可使用命令 `curl -vvv <domain>` 请求该域名，从返回信息中获取对应 IP 地址。

5. 点击 创建。

结果验证

本文以虚拟机访问 www.example.com ↗ 进行验证。

第一步：创建虚拟机及允许所有流量通过的网络策略

1. 创建虚拟机，详细步骤请参考 创建虚拟机。
2. 在虚拟机所在命令空间配置网络策略，添加 TCP 和 UDP 协议的白名单规则，参数如下：

   - TCP 协议白名单：

参数	说明
关联方式	选择 计算组件。
目标计算组件	选择 全部。
方向	选择 出站。
协议	选择 **TCP**。
远程类型	选择 **IP** 段。
远程	输入 **0.0.0.0/0**，表示允许所有流量出站。

   - UDP 协议白名单规则：

参数	说明
方向	选择 出站。
协议	选择 **UDP**。
远程类型	选择 **IP** 段。
远程	输入 **0.0.0.0/0**，表示允许所有流量出站。

3. 网络策略创建完成后，登录虚拟机，执行以下命令请求 www.example.com↗。

```
curl www.example.com
```

4. 请求成功。

第二步：更新网络策略，将 www.example.com↗ 从白名单中移除

1. 执行以下命令获取 www.example.com↗ 的 IP 地址，得到 IP 为 93.184.215.14。

```
curl -vvv www.example.com
```

2. 更新第一步创建的网络策略，更新参数如下：

参数	说明
排除远程	在 TCP 协议白名单规则中，排除远程参数填写 93.184.215.14/32，表示将 IP 地址 93.184.215.14 从白名单中移除。

3. 更新网络策略后，登录虚拟机，执行以下命令请求 www.example.com↗。

```
curl www.example.com
```

4. 请求超时，表示排除远程功能生效。

配置 SR-IOV

通过配置物理服务器节点支持创建带有 SR-IOV（Single Root I/O Virtualization）网卡的虚拟机，实现虚拟机的更低延迟，同时支持独立 IPv6 以及双栈 IPv4/IPv6 功能。

目录

相关说明

CentOS 虚拟机内核参数配置

术语

术语	定义
Multus CNI	作为其他 CNI 插件的中间件，使 Kubernetes 支持 Pod 的多网卡功能。
SR-IOV	允许对节点上的物理 NIC 进行虚拟化，将其拆分为多个 VF 供 Pod 或虚拟机使用，提供更优的网络性能。
VF	从物理 PCI 设备创建的虚拟设备；VF 可以直接分配给虚拟机或容器，类似独立的物理 PCI 设备，显著提升 I/O 性能。

约束与限制

SR-IOV 功能依赖于 glibc，仅支持 glibc 版本 2.34 及以上。但 Kylin V10 和 CentOS 7.x 操作系统均不支持该版本，因此这两个操作系统无法使用 SR-IOV 功能。

前提条件

获取以下 charts 和镜像，并上传至镜像仓库。本文档以仓库地址 `build-harbor.example.cn` 为例。具体获取 charts 和镜像的方法，请联系相关人员。

Chart

- `build-harbor.example.cn/example/chart-sriov-network-operator:v3.15.0`

镜像

- `build-harbor.example.cn/3rdparty/sriov/sriov-network-operator:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-network-operator-config-daemon:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-cni:4.13`
- `build-harbor.example.cn/3rdparty/sriov/ib-sriov-cni:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-network-device-plugin:4.13`
- `build-harbor.example.cn/3rdparty/sriov/network-resources-injector:4.13`
- `build-harbor.example.cn/3rdparty/sriov/sriov-network-operator-webhook:4.13`
- `build-harbor.example.cn/3rdparty/kubectl:v3.15.1`

操作步骤

注意：以下所有命令均在终端执行。

1 在物理机 **BIOS** 中启用 **SR-IOV**

配置前，使用以下命令查看主板信息。

```
dmidecode -t 1
# dmidecode 3.3
Getting SMBIOS data from sysfs.
SMBIOS 2.7 present.

Handle 0x0100, DMI type 1, 27 bytes
System Information
    Product Name: PowerEdge R620
    Version: Not Specified
    Serial Number: 7SJNF62
    UUID: 4c4c4544-0053-4a10-804e-b7c04f463632
    Wake-up Type: Power Switch
    SKU Number: SKU=NotProvided;ModelName=PowerEdge R620
    Family: Not Specified
```

不同服务器厂商启用 BIOS 中 SR-IOV 的操作不同，请参考对应厂商文档。一般步骤如下：

1. 重启服务器。
2. BIOS POST 期间显示品牌 Logo 时，按 F2 键进入系统设置。
3. 点击 **Processor Settings** > **Virtualization Technology**，将 **Virtualization Technology** 设置为 `Enabled`。
4. 点击 **Settings** > **Integrated devices**，将 **SR-IOV Global Enable** 设置为 `Enabled`。
5. 保存配置并重启服务器。

2 启用 IOMMU

不同操作系统启用 IOMMU 的操作不同，请参考对应操作系统文档。本文档以 CentOS 为例。

1. 编辑 `/etc/default/grub` 文件，在 `GRUB_CMDLINE_LINUX` 配置项中添加 `intel_iommu=on iommu=pt`。

```
GRUB_CMDLINE_LINUX="crashkernel=auto rd.lvm.lv=centos/root rhgb qu
iet intel_iommu=on iommu=pt"
```

2. 执行以下命令生成 `grub.cfg` 文件。

```
grub2-mkconfig -o /boot/grub2/grub.cfg
```

3. 重启服务器。
4. 执行以下命令，若输出包含 `IOMMU enabled`，表示启用成功。

```
dmesg | grep -i iommu
```

3 加载系统内核的 VFIO 模块

1. 执行以下命令加载 vfio-pci 模块。

```
modprobe vfio-pci
```

2. 加载后，执行以下命令，若能正常显示配置信息，则 VFIO 内核模块加载成功。

```
# CentOS 下检查 VFIO 加载状态
lsmod | grep vfio
vfio_pci               41993  0
vfio_iommu_type1       22440  0
vfio                   32657  2 vfio_iommu_type1, vfio_pci
irqbypass              13503  2 kvm, vfio_pc


# Ubuntu 下检查 VFIO 加载状态
cat /lib/modules/$(uname -r)/modules.builtin | grep vfio
kernel/drivers/vfio/vfio.ko
kernel/drivers/vfio/vfio_virqfd.ko
kernel/drivers/vfio/vfio_iommu_type1.ko
kernel/drivers/vfio/pci/vfio-pci-core.ko
kernel/drivers/vfio/pci/vfio-pci.ko
```

4 创建 VF 设备

1. 执行以下命令查看当前支持的 VF 设备。

```
find /sys -name *vfs*

/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_totalvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_numvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.0/sriov_totalvfs
/sys/devices/pci0000:00/0000:00:03.0/0000:05:00.0/sriov_numvfs
```

   输出信息说明：

   - **0000:05:00 .1**：SR-IOV 物理网卡 enp5s0f1 的 PCI 地址。
   - **0000:05:00 .0**：SR-IOV 物理网卡 enp5s0f0 的 PCI 地址。
   - **sriov_totalvfs**：支持的 VF 数量。
   - **sriov_numvfs**：当前 VF 数量。

2. 执行以下命令获取物理机的网卡信息。

```
ifconfig

enp5s0f0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.66.213  netmask 255.255.255.0  broadcast 192.
168.66.255
        inet6 1066::192:168:66:213  prefixlen 112  scopeid 0x0<glo
bal>
        inet6 fe80::a236:9fff:fe29:6c00  prefixlen 64  scopeid 0x2
0<link>
        ether a0:36:9f:29:6c:00  txqueuelen 1000  (Ethernet)
        RX packets 13889  bytes 1075801 (1.0 MB)
        RX errors 0  dropped 1603  overruns 0  frame 0
        TX packets 5057  bytes 440807 (440.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp5s0f1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::a236:9fff:fe29:6c02  prefixlen 64  scopeid 0x2
0<link>
        ether a0:36:9f:29:6c:02  txqueuelen 1000  (Ethernet)
        RX packets 1714  bytes 227506 (227.5 KB)
        RX errors 0  dropped 1604  overruns 0  frame 0
        TX packets 70  bytes 19241 (19.2 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

3. 执行命令 `ethtool -i <NIC name>` 获取对应物理网卡的 PCI 地址，如下所示。

```
ethtool -i enp5s0f0
driver: ixgbe
version: 5.15.0-76-generic
firmware-version: 0x8000030d, 14.5.8
expansion-rom-version:
bus-info: 0000:05:00.0     ## enp5s0f0 网卡的 PCI 地址
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: yes


ethtool -i enp5s0f1
driver: ixgbe
version: 5.15.0-76-generic
firmware-version: 0x8000030d, 14.5.8
expansion-rom-version:
bus-info: 0000:05:00.1    ## enp5s0f1 网卡的 PCI 地址
supports-statistics: yes
supports-test: yes
supports-eeprom-access: yes
supports-register-dump: yes
supports-priv-flags: yes
```

4. 执行以下命令创建 VF。本文档以配置 enp5s0f1 网卡为例，若需虚拟化多个网卡，则全部配置。

```
cat /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_totalv
fs   ## 查看支持的 VF 数量
63

echo 8 > /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_n
umvfs  ## 设置当前 VF 数量

cat /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_numvfs
## 查看当前 VF 数量
8
```

5. 执行以下命令检查 VF 是否创建成功。

注意：可见配置的 8 个 VF 地址，如 `05:10.1`。这些 VF 地址需补充 域标识符，最终格式为：`0000:05:10.1`。

```
lspci | grep Virtual
00:11.0 PCI bridge: Intel Corporation C600/X79 series chipset PCI
Express Virtual Root Port (rev 05)
05:10.1 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:10.5 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.1 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.3 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.5 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
05:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Cont
roller Virtual Function (rev 01)
```

5 绑定 VFIO 驱动

1. 下载 绑定脚本，执行 `python3 dpdk-devbind.py -b vfio-pci <带域标识符的 VF 地址>` 命令，将 enp5s0f1 网卡的 8 个 VF 绑定到 vfio-pci 驱动，示例如下。

```
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.1
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.3
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.5
python3 dpdk-devbind.py -b vfio-pci 0000:05:10.7
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.1
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.3
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.5
python3 dpdk-devbind.py -b vfio-pci 0000:05:11.7
```

2. 绑定成功后，执行以下命令检查绑定结果。在输出结果的 **Network devices using DPDK-compatible driver** 区域查找已绑定的 VF，其中 VF 设备 ID 为 `10ed`。

```
python3 dpdk-devbind.py --status

Network devices using DPDK-compatible driver
============================================
0000:05:10.1 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:10.3 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:10.5 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:10.7 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.1 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.3 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.5 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf
0000:05:11.7 '82599 Ethernet Controller Virtual Function 10ed' drv
=vfio-pci unused=ixgbevf

Network devices using kernel driver
===================================
0000:01:00.0 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
1 drv=tg3 unused=vfio-pci
0000:01:00.1 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
2 drv=tg3 unused=vfio-pci
0000:02:00.0 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
3 drv=tg3 unused=vfio-pci
0000:02:00.1 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno
4 drv=tg3 unused=vfio-pci
0000:05:00.0 'Ethernet 10G 2P X520 Adapter 154d' if=enp5s0f0 drv=i
xgbe unused=vfio-pci *Active*
0000:05:00.1 'Ethernet 10G 2P X520 Adapter 154d' if=enp5s0f1 drv=i
xgbe unused=vfio-pci

No 'Baseband' devices detected
==============================

No 'Crypto' devices detected
============================

No 'DMA' devices detected
```

```
==========================

No 'Eventdev' devices detected
==============================

No 'Mempool' devices detected
=============================

No 'Compress' devices detected
==============================

No 'Misc (rawdev)' devices detected
===================================

No 'Regex' devices detected
===========================
```

6 部署 **Multus CNI** 插件

1. 进入 **Administrator**。
2. 在左侧导航栏点击 **Cluster Management** > **Clusters**。
3. 点击虚拟机集群名称，切换到 **Plugins** 标签页。
   - 部署 **Multus CNI** 插件。

7 部署 **sriov-network-operator**

执行以下命令部署 sriov-network-operator。

```
REGISTRY=<$registry>  # 将 <$registry> 替换为 sriov-network-operator 镜像所在的仓库地址，例如：REGISTRY=build-harbor.example.cn
NICSELECTOR=["<nics>"] # 将 <nics> 替换为网卡名称，例如：NICSELECTOR=["ens802f1","ens802f2"]，多个用逗号分隔
NUMVFS=<numVfs> # 将 <numVfs> 替换为 VF 数量，例如：NUMVFS=8

cat <<EOF | kubectl create -f -
apiVersion: operator.alauda.io/v1alpha1
kind: AppRelease
metadata:
  annotations:
    auto-recycle: "true"
    interval-sync: "true"
  name: sriov-network-operator
  namespace: cpaas-system
spec:
  destination:
    cluster: ""
    namespace: "kube-system"
  source:
    charts:
    - name: <chartName> # 将 <chartName> 替换为实际 chart 路径，例如：name = example/chart-sriov-network-operator
      releaseName: sriov-network-operator
      targetRevision: v3.15.0
    repoURL: $REGISTRY
  timeout: 120
  values:
    global:
      registry:
        address: $REGISTRY
    networkNodePolicy:
      nicSelector: $NICSELECTOR
      numVfs: $NUMVFS
EOF
```

8 为物理节点设置 **Node** 角色标识标签

注意：执行此操作前，确保 `sriov-network-operator` 的 Pod 正常运行。

1. 进入 **Administrator**。

2. 在左侧导航栏点击 **Cluster Management** > **Clusters**。
3. 点击集群名称，切换到 **Nodes** 标签页。
4. 点击支持 SR-IOV 的物理节点⋮> **Update Node Labels**。
5. 设置节点标签如下：
   - `node-role.kubernetes.io/worker: ""`
6. 点击 **Update**。

9 检查资源是否创建成功

在 CLI 工具中执行命令 `kubectl -n cpaas-system get sriovnetworknodestates`，检查 `sriovnetworknodestates` 资源是否创建成功。若看到如下类似输出，表示创建成功。若资源创建失败，请检查 Multus CNI 插件和 sriov-network-operator 是否部署成功。

```
kubectl -n cpaas-system get sriovnetworknodestates
NAME                     SYNC STATUS           AGE
192.168.254.88           Succeeded             5d22h
```

10 为物理节点设置 SR-IOV 节点特性标签

注意：执行此操作前，确保 `sriovnetworknodestates` 资源已成功创建。

1. 进入 **Administrator**。
2. 在左侧导航栏点击 **Cluster Management** > **Clusters**。
3. 点击集群名称，切换到 **Nodes** 标签页。
4. 点击支持 SR-IOV 的物理节点⋮> **Update Node Labels**。
5. 设置节点标签如下：
   - `feature.node.kubernetes.io/network-sriov.capable: "true"`

11 检查网卡设备支持情况

1. 执行命令 `lspci -n -s <带域标识符的 VF 地址>`，获取当前网卡设备的厂商 ID 和设备 ID，如下所示。

```
lspci -n -s 0000:05:00.1
05:00.1 0200: 8086:154d (rev 01)
```

输出说明：

- **8086**：厂商 ID。
- **154d**：设备 ID。

2. 执行命令 `lspci -s <带域标识符的 VF 地址> -vvv | grep Ethernet`，获取当前网卡名称，如下所示。

```
lspci -s 0000:05:00.1 -vvv | grep Ethernet
05:00.1 Ethernet controller: Intel Corporation Ethernet 10G 2P X52
0 Adapter (rev 01)
```

3. 在 cpaas-system 命名空间下，找到名为 `supported-nic-ids` 的 ConfigMap 配置文件，检查其 data 部分是否包含当前网卡的配置信息。
   注意：若当前网卡不在支持列表中，需要参考步骤 4 将网卡添加到配置文件中。若已在支持列表中，则跳过步骤 4。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: supported-nic-ids
  namespace: cpaas-system
data:
  Broadcom_bnxt_BCM57414_2x25G: 14e4 16d7 16dc
  Broadcom_bnxt_BCM75508_2x100G: 14e4 1750 1806
  Intel_i40e_10G_X710_SFP: 8086 1572 154c
  Intel_i40e_25G_SFP28: 8086 158b 154c
  Intel_i40e_40G_XL710_QSFP: 8086 1583 154c
  Intel_i40e_X710_X557_AT_10G: 8086 1589 154c
  Intel_i40e_XXV710: 8086 158a 154c
  Intel_i40e_XXV710_N3000: 8086 0d58 154c
  Intel_ice_Columbiaville_E810: 8086 1591 1889
  Intel_ice_Columbiaville_E810-CQDA2_2CQDA2: 8086 1592 1889
  Intel_ice_Columbiaville_E810-XXVDA2: 8086 159b 1889
  Intel_ice_Columbiaville_E810-XXVDA4: 8086 1593 1889
```

4. 以 `<NIC 名称>: <厂商 ID> <设备 ID> <VF 设备 ID>` 格式，将当前网卡添加到支持列表的 data 部分，如下所示。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: supported-nic-ids
  namespace: cpaas-system
data:
  Broadcom_bnxt_BCM57414_2x25G: 14e4 16d7 16dc
  Broadcom_bnxt_BCM75508_2x100G: 14e4 1750 1806

  Intel_Corporation_X520: 8086 154d 10ed              ## 新增网卡信息

  Intel_i40e_10G_X710_SFP: 8086 1572 154c
  Intel_i40e_25G_SFP28: 8086 158b 154c
  Intel_i40e_40G_XL710_QSFP: 8086 1583 154c
  Intel_i40e_X710_X557_AT_10G: 8086 1589 154c
  Intel_i40e_XXV710: 8086 158a 154c
  Intel_i40e_XXV710_N3000: 8086 0d58 154c
  Intel_ice_Columbiaville_E810: 8086 1591 1889
  Intel_ice_Columbiaville_E810-CQDA2_2CQDA2: 8086 1592 1889
  Intel_ice_Columbiaville_E810-XXVDA2: 8086 159b 1889
  Intel_ice_Columbiaville_E810-XXVDA4: 8086 1593 1889
```

参数配置说明：

- **Intel_Corporation_X520**：网卡名称，可自定义。
- **8086**：厂商 ID。
- **154d**：设备 ID。
- **10ed**：VF 设备 ID，可在绑定结果中查看。

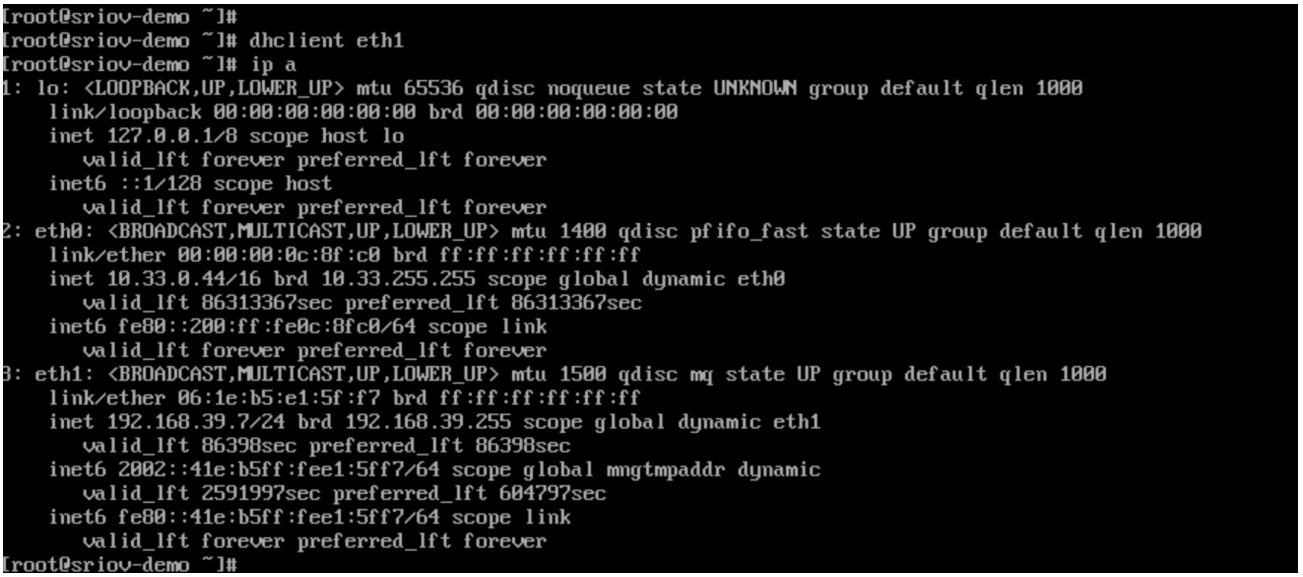
12 配置 IP 地址

登录交换机配置 DHCP（动态主机配置协议）。

注意：若无法使用 DHCP，请在虚拟机中手动配置 IP 地址。

结果验证

1. 进入 **Container Platform**。
2. 在左侧导航栏点击 **Virtualization** > **Virtual Machines**。
3. 点击 **Create Virtual Machine**，添加辅助网卡时，选择 **SR-IOV** 作为 **Network Type**。
4. 完成虚拟机创建。
5. 通过 VNC 访问虚拟机，可看到 eth1 成功获取 IP 地址，表示配置成功。



相关说明

CentOS 虚拟机内核参数配置

CentOS 虚拟机使用 SR-IOV 网卡后，需要修改对应网卡的内核参数，具体步骤如下。

1. 打开终端，执行以下命令修改对应网卡的内核参数。将命令中的 `<NIC Name>` 替换为实际网卡名称。

   ```
   sysctl -w net.ipv4.conf.<NIC Name>.rp_filter=2
   echo "net.ipv4.conf.<NIC Name>.rp_filter=2" >> /etc/sysctl.conf
   ```

2. 执行以下命令加载并应用 /etc/sysctl.conf 文件中的所有内核参数命令，使内核配置生效。输出信息中值为 2 表示修改成功。

```
sysctl -p
```

输出信息：

```
net.ipv4.conf.<NIC Name>.rp_filter = 2
```

配置虚拟机使用网络绑定模式以支持 IPv6

网络绑定模式是虚拟机网络的插件扩展机制。平台默认使用名为 ManagedTap 的插件来实现虚拟机的 IPv6 支持。该插件允许虚拟机通过 CNI 的 DHCP Server 获取 IP 地址。因此，只要 CNI 的 DHCP Server 支持 IPv6，虚拟机也将具备 IPv6 功能。

目前，我们使用 Kube-OVN 作为 CNI。由于 Kube-OVN 的 DHCP Server 完全支持 IPv6，虚拟机可以通过 ManagedTap 和 Kube-OVN 的结合，实现完善的 IPv6 功能。

目录

前提条件

- ACP 版本必须为 v4.0.0 或更高。
- 使用 Kube-OVN 作为 CNI，且虚拟机子网配置为 Underlay。

操作步骤

1 在虚拟机子网中添加 IPv6 配置

```
kubectl edit subnet <subnet-name>
```

在 `spec` 下添加以下参数：

```
spec:
  enableDHCP: true
  enableIPv6RA: true
  u2oInterconnection: true
```

2 在 Web 控制台中创建使用网络绑定模式的虚拟机

创建虚拟机时，选择 **Network Binding** 作为网络模式。

3 通过 VNC 访问虚拟机并配置网络接口

对于 CentOS 系统，编辑 `/etc/sysconfig/network-scripts/ifcfg-enp1s0` 文件，添加以下配置：

```
IPV6INIT=yes
DHCPV6C=yes
IPV6_AUTOCONF=yes
```

重启网络

```
systemctl restart network
```

4 配置 IPv6 默认路由

如果交换机配置了发送 Router Advertisement (RA) 消息，则无需手动配置路由。默认路由可以通过交换机发送的 RA 消息自动学习。

```
ip r r default via <subnet-v6-gateway>
```

配置多网卡虚拟机

结合 Kube-OVN 和 Multus，为虚拟机提供多网卡支持

目录

前提条件

- Alauda Container Platform 版本必须为 v4.1.0 或更高。
- 使用 Kube-OVN 作为 CNI。
- 已安装 Alauda Container Platform Networking for Multus。

操作步骤

1 创建二级网络

1. 创建 NetworkAttachmentDefinition

   在集群控制节点执行以下命令：

   Command

```
cat << EOF | kubectl create -f -
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: <name>
  namespace: <namespace>
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "kube-ovn",
    "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
    "provider": "<provider>"
    }'
EOF
```

   Example

```
cat << EOF | kubectl create -f -
apiVersion: 'k8s.cni.cncf.io/v1'
kind: NetworkAttachmentDefinition
metadata:
  name: attachnet
  namespace: default
spec:
  config: '{
    "cniVersion": "0.3.0",
    "type": "kube-ovn",
    "server_socket": "/run/openvswitch/kube-ovn-daemon.sock",
    "provider": "attachnet.default.ovn"
    }'
EOF
```

   参数说明：

- name：NetworkAttachmentDefinition 的名称。
- namespace：NetworkAttachmentDefinition 所在的命名空间，必须与虚拟机使用相同的命名空间。
- provider：当前 NetworkAttachmentDefinition 的 `<name>.<namespace>.ovn`，Kube-OVN 会根据此信息查找对应的 Subnet 资源。注意后缀必须设置为 ovn。

2. 创建 Kube-OVN 子网
   如果将 Kube-OVN 用作二级网卡，provider 应设置为对应 NetworkAttachmentDefinition 的 `<name>.<namespace>.ovn`，且必须以 ovn 结尾。
   在集群控制节点执行以下命令：

Command

```
cat << EOF | kubectl create -f -
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: <name>
spec:
  protocol: IPv4
  enableDHCP: true
  provider: <provider>
  cidrBlock: <cidrBlock>
  gateway: <gateway>
  excludeIps:
  - <excludeIps>
EOF
```

Example

```
cat << EOF | kubectl create -f -
apiVersion: kubeovn.io/v1
kind: Subnet
metadata:
  name: attachnet
spec:
  protocol: IPv4
  enableDHCP: true
  provider: attachnet.default.ovn
  cidrBlock: 172.17.0.0/16
  gateway: 172.17.0.1
  excludeIps:
  - 172.17.0.0..172.17.0.10
EOF
```

参数说明：

- name：子网名称。
- provider：NetworkAttachmentDefinition 的 provider。
- cidrBlock：子网 CIDR。
- gateway：网关地址。
- excludeIps：保留的 IP 集合，不会被自动分配。例如，可用作计算组件的固定 IP 地址。

2 创建多网卡虚拟机

1. 通过 UI 创建虚拟机
2. 切换到 **YAML** 视图，为虚拟机添加另一张网卡
   在 `spec.template.spec.domain.devices.interfaces` 下添加新接口
   在 `spec.template.spec.networks` 下添加新网络

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - bridge: {}
            name: default
          # 新接口
          - bridge: {}
            name: dyniface1
      networks:
      - name: default
        pod: {}
      # 新网络
      - multus:
          networkName: <networkName>
        name: dyniface1
```

其中 networkName 是 NetworkAttachmentDefinition 的名称。

3 配置新网卡的网络

虚拟机启动后，需要进入虚拟机手动配置新添加的网卡网络。

4 热插拔网络接口

支持在运行中的虚拟机中热插拔网络接口。

热插拔支持使用 virtio 模型且通过桥接绑定或 SR-IOV 绑定连接的接口。

热拔插仅支持通过桥接绑定连接的接口。

1. 向运行中的虚拟机添加接口
   使用 kubectl edit 修改虚拟机的 YAML 配置

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - bridge: {}
            name: default
          # 新接口
          - bridge: {}
            name: dyniface1
      networks:
      - name: default
        pod: {}
      # 新网络
      - multus:
          networkName: <networkName>
        name: dyniface1
```

2. 从运行中的虚拟机移除接口

   使用 kubectl edit 修改虚拟机的 YAML 配置

```
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
          - bridge: {}
            name: default
          # 将接口状态设置为 absent
          - bridge: {}
            name: dyniface1
            state: absent
      networks:
      - name: default
        pod: {}
      # 新网络
      - multus:
          networkName: <networkName>
        name: dyniface1
```

存储

介绍

介绍

优势

操作指南

管理虚拟磁盘

创建虚拟磁盘

挂载虚拟磁盘

扩容虚拟磁盘

卸载虚拟磁盘

删除虚拟磁盘

介绍

ACP Virtualization 结合 KubeVirt Storage 通过无缝集成 Kubernetes 原生存储机制，为虚拟机（VM）提供持久存储能力。它利用 **PersistentVolumeClaim** (PVC) 来存储虚拟机磁盘数据，并通过 **Container Storage Interface** (CSI) 实现与多种存储系统的集成。此外，平台还采用 **Containerized Data Importer** (CDI) 来初始化虚拟机磁盘数据。在此基础上，平台扩展了虚拟机磁盘管理的高级功能，实现了全面的生命周期控制。

目录

优势

- 用户友好的操作
  大多数虚拟机磁盘操作可以通过 **Web UI** 轻松完成，减少对 CLI 专业知识的依赖。
- 虚拟机磁盘生命周期管理
  配置虚拟机磁盘在关联虚拟机终止时是否自动删除。

操作指南

管理虚拟磁盘

管理虚拟磁盘

数据盘可用于满足业务的数据持久化需求。

目录

创建虚拟磁盘

为虚拟机创建数据盘。每次只能添加一个虚拟磁盘；如需多个磁盘，请重复此操作。

注意：虚拟磁盘可在虚拟机处于运行状态时在线挂载。

操作步骤

1. 访问容器平台。
2. 在左侧导航栏点击虚拟化 > 虚拟磁盘。
3. 点击创建虚拟磁盘。
4. 根据以下说明配置信息。

参数	说明
Volume Mode	- **File System**：以挂载文件目录的方式挂载磁盘。 - **Block Device**：以块设备的方式挂载磁盘。
Storage Class	平台通过自动创建和管理持久卷声明来维护虚拟机磁盘。您需要指定用于动态创建持久卷声明的存储类。 不同存储类支持不同的卷模式。如果所选卷模式没有可用的存储类，请联系管理员添加。
Delete with VM	启用后，删除虚拟机时磁盘数据也会被删除。
Mount	- 不挂载：仅创建虚拟磁盘，后续需要时可挂载。 - 挂载到虚拟机：选择需要挂载虚拟磁盘的目标虚拟机。

5. 点击创建。

挂载虚拟磁盘

将数据盘挂载到虚拟机，将已创建的虚拟磁盘附加到目标虚拟机。

注意：虚拟磁盘可在虚拟机处于运行状态时在线挂载。

操作步骤

1. 访问容器平台。
2. 在左侧导航栏点击虚拟化 > 虚拟磁盘。
3. 在要挂载的虚拟磁盘旁点击⋮> 挂载。

4. 选择目标虚拟机，点击挂载。

扩容虚拟磁盘

扩容已挂载到虚拟机的系统盘和数据盘。

操作步骤

1. 访问容器平台。
2. 在左侧导航栏点击虚拟化 > 虚拟机。
3. 点击虚拟机名称进入详情页面。
4. 在虚拟磁盘区域，找到要扩容的磁盘，点击⋮> 扩容。
5. 输入新的容量，点击扩容。

卸载虚拟磁盘

从虚拟机卸载数据盘；仅处于停止状态的虚拟机可卸载磁盘。

操作步骤

1. 访问容器平台。
2. 在左侧导航栏点击虚拟化 > 虚拟磁盘。
3. 在要卸载的虚拟磁盘旁点击⋮> 卸载并确认。

删除虚拟磁盘

仅支持删除处于未挂载状态的虚拟磁盘。

注意：系统盘不可删除。

操作步骤

1. 访问容器平台。

2. 在左侧导航栏点击虚拟化 > 虚拟磁盘。

3. 在要删除的虚拟磁盘旁点击⋮> 删除并确认。

备份与恢复

介绍

操作指南

介绍

ACP Virtualization With Kubevirt 提供虚拟机快照功能，允许用户通过快照备份和恢复虚拟机。

目录

应用场景

- 灾难恢复与故障回滚
  当虚拟机因硬件故障、人为错误（如误删文件）或恶意攻击（如勒索软件）导致数据丢失时，快照作为最后一道防线，用于恢复运行。

使用限制

- 创建快照需要先停止虚拟机。
- 虚拟机磁盘所使用的 PVC（Persistent Volume Claim）必须配置为多节点共享访问模式。

操作指南

使用快照

虚拟机快照保存虚拟机的当前状态，可用于在发生意外故障时将虚拟机恢复到该状态。

目录

前提条件

- 平台管理员已部署了 **Volume Snapshot**。
- 虚拟机快照基于卷快照。请确保至少有一个磁盘绑定到支持卷快照的存储类，例如 CephFS 内置存储。
- 仅支持虚拟机的离线快照。请先停止虚拟机，然后再创建或回滚快照。

注意事项

如果集群中存在多种相同类型的存储，例如挂载了多个不同来源的 Ceph RBD 存储，当虚拟机使用此类存储时，磁盘快照功能可能无法正常工作。

创建快照

虚拟机快照包含的内容：虚拟机设置和支持卷快照的磁盘状态。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏点击 **Virtualization** > **Virtual Machines**。
3. 找到目标虚拟机，点击⋮> **Create Snapshot**。
4. 填写快照描述。描述有助于记录虚拟机当前状态，例如 `Initial Installation`、`Before Application Upgrade`。
5. 点击 **Create**。快照创建所需时间取决于网络状况和工作负载，请耐心等待。
6. 检查快照状态。
   - 当快照状态变为 `Ready`，表示创建成功。
   - 如果快照长时间处于 `Not Ready` 状态，点击 > 查看原因并排查，然后重新创建快照。

回滚快照

将虚拟机设置和支持卷快照的磁盘回滚到快照创建时的状态。例如，快照创建后新增的磁盘将被移除；修改过的磁盘数据将被恢复。

注意事项

如果有磁盘绑定到支持 LVM 机制的存储类（例如 TopoLVM），请确认管理员已将该存储类的回收策略设置为 **Retain**（`reclaimPolicy: Retain`），以正确使用快照回滚功能。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏点击 **Virtualization** > **Virtual Machines**。
3. 点击 *虚拟机名称*。
4. 在 **Snapshots** 标签页，找到目标快照，点击⋮> **Rollback**。
5. 阅读界面提示信息，确认无误后点击 **Rollback**。
   注意：回滚操作不可中止或撤销，请谨慎操作。
6. 点击快照名称，在“Snapshot Rollback Records”中查看回滚是否完成。回滚所需时间取决于网络状况和工作负载，请耐心等待。

说明

- 若回滚失败，虚拟机状态保持不变。您可以正常启动虚拟机或尝试再次回滚快照。
- 回滚过程中若启动虚拟机，虚拟机将回到停止前的状态；再次停止虚拟机后，将继续回滚到快照创建时的状态。
- 为避免操作冲突，请确保最近一次回滚记录已完成后，再对该虚拟机执行其他操作。

删除快照

删除不必要的虚拟机快照以释放磁盘资源。

注意事项

删除已回滚的虚拟机快照时，如果虚拟机磁盘需要基于快照复制数据（例如 TopoLVM），必须等待基于回滚版本启动的虚拟机启动后再删除，否则虚拟机将无法启动。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏点击 **Virtualization** > **Virtual Machines**。
3. 点击 *虚拟机名称*。
4. 在 **Snapshots** 标签页，找到目标快照，点击⋮> **Delete**。

5. 阅读提示信息，确认无误后点击 **Delete**。

使用 Velero

Velero 是 VMware 提供的开源 Kubernetes 集群备份和迁移工具。KubeVirt 提供了 Velero 插件以支持虚拟机的备份与恢复。

目录

前提条件

- **Kubernetes** 版本：1.20 或更高（Velero 要求）
- **S3** 存储：Velero BackupStorageLocation 使用平台提供的 Ceph 或 MinIO 对象存储
- 块存储：平台提供的 Ceph RBD

操作步骤

准备工作

1. 通过 平台管理 → 集群管理 → 备份与恢复 → 备份仓库 在 ACP 平台部署 Velero。使用对象存储信息创建备份仓库。
2. 启用 CSI 功能并添加 KubeVirt 插件：

   ```
   kubectl edit deploy -n cpaas-system velero
   ```

   在 Velero 部署中添加以下内容：

   ```
   containers:
   - args:
     - server
     - --uploader-type=restic
     - --namespace=cpaas-system
     - --features=EnableCSI
     command:
     - /velero
   ```

   在 initContainers 部分添加：

   ```
   initContainers:
   - image: registry.example.org/3rdparty/kubevirt/kubevirt-velero-plugin:
   v0.7.0
     imagePullPolicy: IfNotPresent
     name: velero-plugin-kubevirt
     resources: {}
     terminationMessagePath: /dev/termination-log
     terminationMessagePolicy: File
     volumeMounts:
     - mountPath: /target
       name: plugins
   ```

3. 验证 KubeVirt 插件安装：

```
POD=$(kubectl -n cpaas-system get pod -l app.kubernetes.io/name=velero
-o jsonpath='{.items[0].metadata.name}')
kubectl -n cpaas-system exec -ti "$POD" -- /velero get plugins | grep k
ubevirt
```

示例输出：

```
kubevirt-velero-plugin/backup-datavolume-action              BackupIte
mAction
kubevirt-velero-plugin/backup-datavolume-action              BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachine-action          BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachine-action          BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachineinstance-action  BackupIte
mAction
kubevirt-velero-plugin/backup-virtualmachineinstance-action  BackupIte
mAction
kubevirt-velero-plugin/restore-pod-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-pod-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-pvc-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-pvc-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-vm-action                     RestoreIt
emAction
kubevirt-velero-plugin/restore-vm-action                     RestoreIt
emAction
kubevirt-velero-plugin/restore-vmi-action                    RestoreIt
emAction
kubevirt-velero-plugin/restore-vmi-action                    RestoreIt
emAction
```

4. 调整 node-agent 挂载宿主机 kubelet 目录并开启特权模式（数据通过 `/var/lib/kubelet` 迁移时必需）：

```
kubectl edit ds -n cpaas-system node-agent
```

更新为：

```
securityContext:
  privileged: true
volumeMounts:
- mountPath: /host_pods
  mountPropagation: HostToContainer
  name: host-pods
- mountPath: /var/lib/kubelet/plugins
  mountPropagation: HostToContainer
  name: host-plugins
- mountPath: /scratch
  name: scratch
securityContext:
  runAsUser: 0
volumes:
- hostPath:
    path: /var/lib/kubelet/pods
    type: ""
  name: host-pods
- hostPath:
    path: /var/lib/kubelet/plugins
    type: ""
  name: host-plugins
- emptyDir: {}
  name: scratch
```

备份

1. 按需创建带有多个磁盘的虚拟机。
2. 创建备份资源：

```yaml
apiVersion: velero.io/v1
kind: Backup
metadata:
  annotations:
    velero.io/resource-timeout: 10m0s
    velero.io/source-cluster-k8s-gitversion: v1.30.4
    velero.io/source-cluster-k8s-major-version: "1"
    velero.io/source-cluster-k8s-minor-version: "30"
  labels:
    velero.io/storage-location: default
  name: example-backup
  namespace: cpaas-system
spec:
  csiSnapshotTimeout: 10m0s
  defaultVolumesToFsBackup: false
  hooks: {}
  includedNamespaces:
  - example-namespace
  itemOperationTimeout: 4h0m0s
  metadata: {}
  snapshotMoveData: true
  storageLocation: default
  ttl: 720h0m0s
```

3. 等待备份完成并验证：

```
POD=$(kubectl -n cpaas-system get pod -l app.kubernetes.io/name=velero
-o jsonpath='{.items[0].metadata.name}')
kubectl -n cpaas-system exec -ti "$POD" -- /velero backup describe exam
ple-backup --details
kubectl -n cpaas-system exec -ti "$POD" -- /velero backup get example-b
ackup
```

示例输出：

```
NAME             STATUS                          ERRORS   WARNINGS   CREAT
ED                          EXPIRES   STORAGE LOCATION   SELECTOR
example-backup   WaitingForPluginOperations   0        0          2025-
01-25 14:14:08 +0000 UTC   29d       default            <none>
```

4. 查看数据迁移状态：

```
kubectl get dataupload -n cpaas-system
```

示例输出：

```
NAME                    STATUS      STARTED   BYTES DONE    TOTAL BYTES
STORAGE LOCATION   AGE   NODE
example-backup-fhc6b   Completed   11h       21474836480   21474836480
default            11h   192.168.254.66
example-backup-qwwzh   Completed   11h       21474836480   21474836480
default            11h   192.168.254.15
```

5. 验证备份完成：

```
POD=$(kubectl -n cpaas-system get pod -l app.kubernetes.io/name=velero
-o jsonpath='{.items[0].metadata.name}')
kubectl -n cpaas-system exec -ti "$POD" -- /velero backup get example-b
ackup
```

示例输出：

```
NAME             STATUS      ERRORS   WARNINGS   CREATED
EXPIRES   STORAGE LOCATION   SELECTOR
example-backup   Completed   0        0          2025-01-25 14:14:08 +0
000 UTC   29d       default            <none>
```

6. 检查备份仓库（如 MinIO）中的目录：

```
mc ls <backup-repository>
```

示例输出：

```
[2025-01-26 09:23:08 CST]     0B backups/
[2025-01-26 09:23:08 CST]     0B kopia/
[2025-01-26 09:23:08 CST]     0B restic/
```

恢复

1. 删除原虚拟机。
2. 通过 平台管理 → 集群管理 → 备份与恢复 → 恢复管理 创建恢复资源，或手动创建：

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  annotations:
    cpaas.io/description: ""
  finalizers:
  - restores.velero.io/external-resources-finalizer
  name: example-restore
  namespace: cpaas-system
spec:
  backupName: example-backup
  excludedResources:
  - nodes
  - events
  - events.events.k8s.io
  - backups.velero.io
  - restores.velero.io
  - resticrepositories.velero.io
  - csinodes.storage.k8s.io
  - volumeattachments.storage.k8s.io
  - backuprepositories.velero.io
  hooks: {}
  includedNamespaces:
  - example-namespace
  itemOperationTimeout: 4h0m0s
  namespaceMapping: {}
```

3. 验证恢复：

```
kubectl exec -ti -n cpaas-system velero-5df7bb7598-ljjrn -- /velero res
tore get
```

查看 datadownload 状态：

```
kubectl get datadownload -n cpaas-system
```

示例输出：

```
NAME       STATUS      STARTED   BYTES DONE    TOTAL BYTES   STORAGE LO
CATION   AGE   NODE
example-restore-7lfcm   Completed   11m       21474836480   21474836480
default            15m   192.168.254.66
example-restore-cjcd8   Completed   15m       21474836480   21474836480
default            15m   192.168.254.66
```

恢复状态：

```
NAME   BACKUP           STATUS      STARTED                          COM
PLETED                       ERRORS   WARNINGS   CREATED
SELECTOR
aa     example-backup   Completed   2025-01-26 01:53:14 +0000 UTC   202
5-01-26 02:01:24 +0000 UTC   0          2          2025-01-26 01:53:14 +0
000 UTC   <none>
```

4. 确认虚拟机已恢复且正常运行。

跨集群恢复

1. 在新集群部署 Velero 并完成准备工作。
2. 配置与原集群相同的备份仓库（相同桶和目录），备份资源将自动出现。
3. 按上述恢复步骤操作。

恢复到不同命名空间

在恢复 spec 中添加 `namespaceMapping` 字段：

```yaml
spec:
  namespaceMapping:
    example-namespace: test-namespace
```

恢复到不同存储类

1. 在恢复前，在 `cpaas-system` 命名空间创建 ConfigMap：

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: change-storage-class-config
  namespace: cpaas-system
  labels:
    velero.io/plugin-config: ""
    velero.io/change-storage-class: RestoreItemAction
data:
  vm-cephrbd: vm-topolvm-a
```

2. 确保存储能力（如跨节点访问、RWX）一致。

3. 注意：当前不支持 RWX 模式的修改。