Menu

虚拟化

概览

介绍

容器编排虚拟机解决方案

功能特点

产品功能

约束和限制

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

实用指南

权限说明

虚拟机

介绍

操作指南

实用指南

问题处理

网络

介绍

优势

操作指南

实用指南

存储

介绍

优势

操作指南

备份和恢复

介绍

应用场景

限制

操作指南

Menu

概览

介绍

容器编排虚拟机解决方案

功能特点

产品功能

约束和限制

介绍

对于使用基于虚拟机架构的企业而言，向基于 Kubernetes 和容器架构的转型不可避免地需要进行应用现代化。然而，由于需要持续的业务在线或者改变开发习惯的困难等因素，企业往往无法在短时间内完全脱离虚拟化架构。

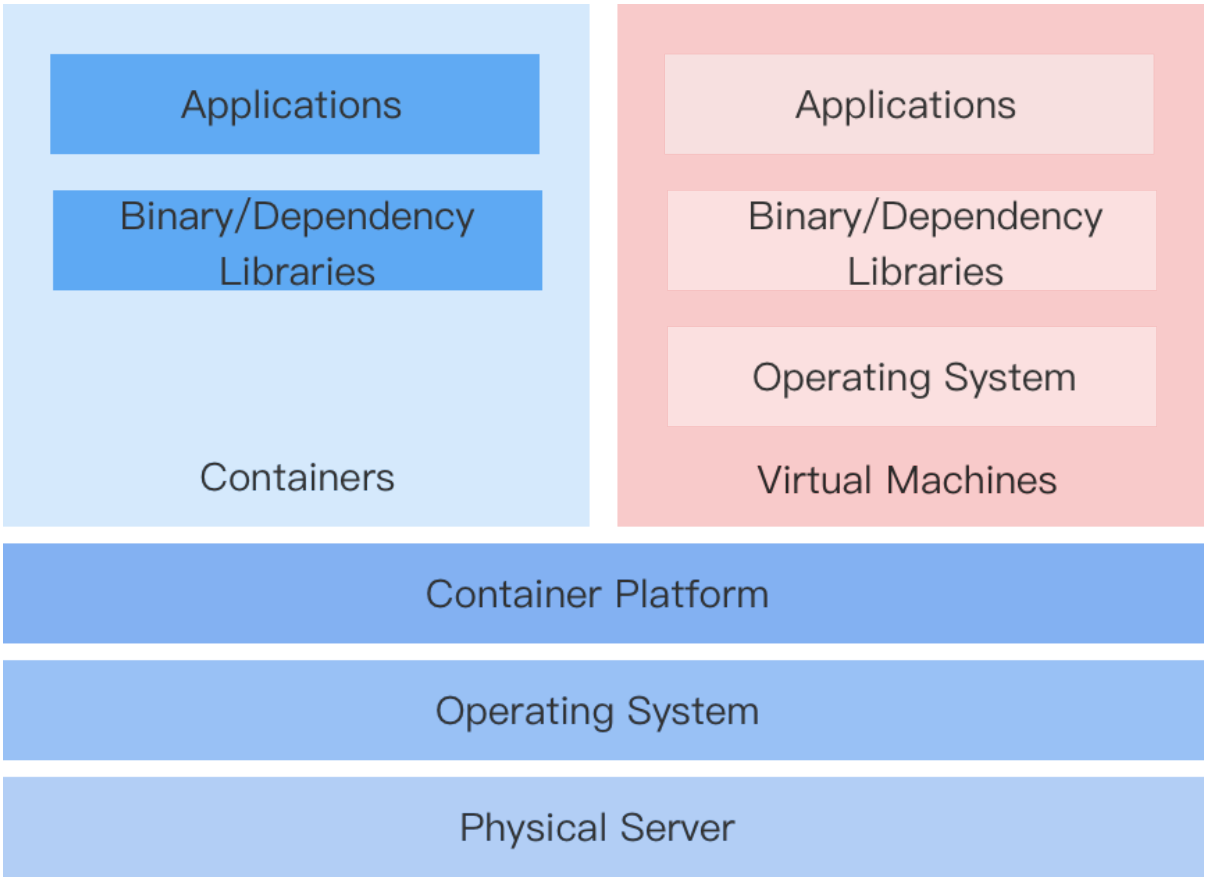
因此，一个能够在同一平台上统一配置、管理和控制容器资源与虚拟机资源的解决方案变得尤为重要。

目录

容器编排虚拟机解决方案
功能特点
产品功能
约束和限制

容器编排虚拟机解决方案

该平台基于开源组件 KubeVirt 实现了一种虚拟机（VMI，VirtualMachineInstance）解决方案，使得创建容器编排的虚拟机和运行虚拟化应用程序变得更加简便和快速。



功能特点

快速转型

无需重写应用程序或修改镜像。只需将现有应用打包为 qcow2 或 raw 格式的虚拟机镜像，然后在平台上使用该镜像创建虚拟机，从而将应用部署到容器平台。

保持行为习惯

容器化的虚拟机可以采用与传统虚拟机类似的管理方法，无需关注底层的容器实现，包括虚拟机生命周期管理、磁盘和网络管理以及快照管理。

虚拟化与容器化的共存

- 统一平台支持管理虚拟化服务，同时能够实现基于 Kubernetes 的容器调度与管理。
- 在继续使用虚拟机工作负载的基础上，允许逐步实现容器化应用的现代化。
- 新开发的需要与虚拟化应用交互的容器化应用不受影响。

产品功能

- 虚拟机：支持使用管理员分配的镜像创建虚拟机并进行管理，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、容量以及关联的虚拟机。
- 虚拟机快照：支持查看虚拟机快照的状态、关联的虚拟机以及最近的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等。
- 密钥对：支持查看和管理当前项目中创建的密钥对，包括创建密钥对和查看关联的虚拟机列表。

约束和限制

必须在物理机集群的基础上实现，并且 KubeVirt 组件必须在集群中部署并启用虚拟化。平台提供通过 Operator 部署 KubeVirt 组件的能力，并具备启用虚拟化的接口，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
操作步骤
启用节点虚拟化
操作步骤
部署 Operator
创建 HyperConverged 实例
配置虚拟机超售比（可选）
重要说明
资源配额说明

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该**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。

- 建议结合使用 TopoLVM 和 Kubevirt，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	描述
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：磁盘冗余副本导致利用率较低，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满足多块磁盘的总容量，否则虚拟机创建会失败。

  - 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是否允许 Kubernetes 在该节点上调度虚拟机实例（VMI）。

- 开启开关时，允许在物理机节点上调度新的虚拟机；Windows 物理节点不支持启用虚拟化。

- 关闭开关时，阻止在物理机节点上调度新的虚拟机，但不影响该节点上已运行的虚拟机。

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`：当 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` 字段内的 ***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。
- 一旦为虚拟机启用超售比，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% 的资源。当命名空间剩余可用资源低于 20% 时，请及时扩容资源。

Menu

镜像

介绍

介绍

优势

操作指南

添加虚拟机镜像

操作步骤

更新/删除虚拟机镜像

更新/删除镜像凭据

实用指南

使用 KubeVirt 基于 ISO 创建 Windows 镜像

前提条件

约束与限制

操作步骤

远程访问

使用 KubeVirt 基于 ISO 创建 Linux 镜像

前提条件

约束与限制

操作步骤

导出虚拟机镜像

操作步骤

权限说明

权限说明

介绍

Alauda Container Platform Virtualization with KubeVirt 利用 Kubernetes 扩展 API 能力将虚拟机镜像抽象为一个 **Custom Resource Definition** (CRD)，并提供用户界面 (UI)，让用户可以便捷地将存放在远端的虚拟机镜像导入到 ACP 中使用。

目录

优势

优势

- 支持主流操作系统

  支持各种常用的 Linux 发行版和 Windows 操作系统。

- 多架构支持

  兼容 **X86_64** 和 **ARM64** 架构。

- 多来源支持

  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。

Menu

操作指南

添加虚拟机镜像

操作步骤

更新/删除虚拟机镜像

更新/删除镜像凭据

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，便于业务系统的迁移。

目录

操作步骤

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，若无需认证，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 `index.docker.io/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据或点击 添加凭据，仅支持 用户名/密码 类型凭据。 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。

Menu

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。

Menu

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 标签页中，点击⋮> 更新/删除。

4. 确认后，点击 更新/删除。

Menu

实用指南

使用 KubeVirt 基于 ISO 创建 Windows 镜像

前提条件

约束与限制

操作步骤

远程访问

使用 KubeVirt 基于 ISO 创建 Linux 镜像

前提条件

约束与限制

操作步骤

导出虚拟机镜像

操作步骤

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。

目录

前提条件

约束与限制

操作步骤

- 创建镜像
- 创建虚拟机
- 安装 Windows 操作系统
- 安装 virtio-win-tools
- 导出自定义 Windows 镜像
- 使用 Windows 镜像
- 添加内部路由

远程访问

前提条件

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

- 请准备好用于存储镜像的仓库，本文以 build-harbor.example.cn 仓库为例，请根据实际环境替换。

约束与限制

- 启动 KubeVirt 时，自定义镜像的文件系统大小会影响将镜像写入 PVC 磁盘的速度，文件系统过大可能导致创建时间延长。
- 建议保持 Linux 根分区或 Windows C 盘小于 100G，以减少初始大小，后续可通过 cloud-init 扩容（Windows 系统需创建后手动扩容）。

操作步骤

1 创建镜像

通过准备好的 Windows 和 virtio-win ISO 镜像创建 Docker 镜像，并推送到仓库。本文以 Windows Server 2019 为例。

从 **Windows ISO** 创建 **Docker** 镜像

1. 进入存放 ISO 镜像的目录，在终端执行以下命令，将 ISO 镜像重命名为 win.iso。

```
mv <ISO image name> win.iso # 将 <ISO image name> 替换为实际镜像名称，例如 mv en_windows_server_2019_x64_dvd_4cb967d8.iso win.iso
```

2. 执行以下命令创建 Dockerfile。

```
touch Dockerfile
```

3. 编辑 Dockerfile，添加以下内容并保存。

```
FROM scratch
ADD --chown=107:107 win.iso /disk/
```

4. 执行以下命令构建 Docker 镜像。

```
docker build -t build-harbor.example.cn/3rdparty/vmdisks/winiso:2019 . # 根据实际环境替换仓库地址
```

5. 执行以下命令将镜像推送到仓库。

```
docker push  build-harbor.example.cn/3rdparty/vmdisks/winiso:2019 # 根据实际环境替换仓库地址
```

从 **virtio-win ISO** 创建 **Docker** 镜像

1. 进入存放 ISO 镜像的目录，执行以下命令创建 Dockerfile。

```
touch Dockerfile
```

2. 编辑 Dockerfile，添加以下内容并保存。

```
FROM scratch
ADD --chown=107:107 virtio-win.iso  /disk/
```

3. 执行以下命令构建 Docker 镜像。

```
docker build -t build-harbor.example.cn/3rdparty/vmdisks/win-virtio:latest . # 根据实际环境替换仓库地址
```

4. 执行以下命令将镜像推送到仓库。

```
docker push  build-harbor.example.cn/3rdparty/vmdisks/win-virtio:latest # 根据实际环境替换仓库地址
```

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 创建虚拟机。

5. 切换到 YAML。

6. 将 spec.template.spec.domain.devices.disks 字段下的配置替换为以下内容。

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

7. 在 spec.template.spec.volumes 字段下添加以下内容。

```
      - containerDisk:
          image: registry.example.cn:60070/3rdparty/vmdisks/winiso:2019 # 根据实际环境替换镜像
        name: containerdisk
      - containerDisk:
          image: registry.example.cn:60070/3rdparty/vmdisks/win-virtio # 根据实际环境替换镜像
        name: virtio
```

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: acp-service-self
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:
          http:
            url: http://192.168.254.12/kube-debian-12.2.0-x86-out.qcow2
  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:
        disks:
          - disk:
              bus: virtio
            name: cloudinitdisk
          - bootOrder: 1
            cdrom:
              bus: sata
            name: containerdisk
          - cdrom:
              bus: sata
            name: virtio
          - 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:
$6$0vlhl57e$0rawYwaeu9jL6hBf3XP9lk6XXaMUS9/W6LPbWRinUoXujo39lP3l98VOcOObtr.LDoAv/ylm
          name: cloudinitdisk
        - containerDisk:
            image: registry.example.cn:60070/3rdparty/vmdisks/winiso:2019 # 根据实际
          name: containerdisk
          - containerDisk
             image: registry.example.cn:60070/3rdparty/vmdisks/win-virtio # 根据实际
           name: virtio
         - 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；如果未看到提示，请点击页面左上角的 发送远程命令，从下拉菜单选择 **Ctrl-Alt-Delete** 重启服务器。

    注意：如果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 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。需依次选择每个分区点击 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，选择下拉菜单中的 **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. 点击 创建虚拟机。

5. 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 创建虚拟机。

6. （可选）若使用较新操作系统，如 Windows 11，需启用时钟、UEFI、TPM 等功能。切换到 YAML，将原 YAML 替换为以下内容。

```yaml
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:
      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
```

7. 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，点击 内部路由 旁的 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。 • 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. 记录基本信息区的 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。

目录

前提条件

约束与限制

操作步骤

- 将 Linux ISO 镜像转换为 Docker 镜像
- 创建虚拟机
- 安装 Linux 操作系统
- 修改 YAML 文件
- 安装所需软件并修改配置
- 导出并使用自定义 Linux 镜像

前提条件

- 集群中的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。
- 需提前准备好用于存储镜像的仓库，本文以 build-harbor.example.cn 仓库为例，请根据实际环境替换。

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。
- 建议保持 Linux 根分区大小低于 100G，以减少初始大小。配置 cloud-init 后，创建虚拟机时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 镜像转换为 Docker 镜像

1. 进入存放 ISO 镜像的目录，在终端执行以下命令，将 ISO 镜像重命名为 ubuntu.iso。

```
mv <ISO image name> ubuntu.iso # 将 <ISO image name> 替换为实际镜像名称，例如 mv ubuntu-24.04-live-server-amd64.iso ubuntu.iso
```

2. 执行以下命令创建 Dockerfile。

```
touch Dockerfile
```

3. 编辑 Dockerfile，添加以下内容并保存。

```
FROM scratch
ADD --chown=107:107 ubuntu.iso /disk/
```

4. 执行以下命令构建 Docker 镜像。

```
docker build -t build-harbor.example.cn/3rdparty/vmdisks/ubuntu-iso:24.04 . # 请根据实际环境替换仓库地址
```

5. 执行以下命令将镜像推送到仓库。

```
docker push build-harbor.example.cn/3rdparty/vmdisks/ubuntu-iso:24.04 # 请根据实际环境替换仓库地址
```

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. 在表单页面填写参数，具体参数及配置请参考 创建虚拟机。

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. 将 spec.template.spec.domain.devices.disks 字段下的配置替换为以下内容。

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

7. 在 spec.template.spec.volumes 字段下添加以下内容。

```
        - containerDisk:
            image: registry.example.cn:60070/3rdparty/vmdisks/ubuntu-iso:24.04 #
请根据实际环境替换镜像地址
          name: containerdisk
```

8. 检查 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:
          http:
            url: http://192.168.254.12/kube-debian-12.2.0-x86-out.qcow2
  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: containerdisk
          - 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:
        - containerDisk:
            image: registry.example.cn:60070/3rdparty/vmdisks/ubuntu-iso:24.04 # 请根据实际环境替换镜像地址
          name: containerdisk
        - cloudInitConfigDrive:
            userData: |-
              #cloud-config
              disable_root: false
              ssh_pwauth: false
              users:
                - default
                - name: root
                  lock_passwd: false
                  hashed_passwd: ""
          name: cloudinitdisk
        - dataVolume:
            name: aa-rootfs
          name: rootfs
```

9. 点击 创建。

10. 点击 操作 > **VNC** 登录。

11. 当出现 **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：如果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，以减少镜像体积。
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 分区（创建卷组 (LVM)）。
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. 在左侧导航栏点击 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 下名为 **rootfs** 的磁盘的 bootOrder 为 1，若不是，修改为 1。
8. 删除 spec.template.spec.domain.devices.disks 下名为 **containerdisk** 的相关内容，具体删除内容如下。

```
        - bootOrder: 1
          cdrom:
            bus: sata
          name: containerdisk
```

9. 删除 spec.template.spec.volumes 下名为 **containerdisk** 的相关内容，具体删除内容如下。

```
      - containerDisk:
          image: registry.example.cn:60070/3rdparty/vmdisks/ubuntu-iso:24.04
        name: containerdisk
```

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. 切换到 root 用户权限。

3. 安装软件包。

   - CentOS 系列执行命令：

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

   - Debian 系列执行命令：

     ```
     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 命令用于扩展磁盘上的分区，这里扩展 /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 是待扩展文件系统所在分区。扩展分区后，使用 xfs_growfs 确保文件系统也扩展到新分区大小。
      ```

   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
停止虚拟机
创建 vmexport 资源
下载虚拟机镜像文件
将虚拟机镜像文件上传至对象存储
创建虚拟机镜像

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进行。

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>* 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>* 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>` 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。

   ```
   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> 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。

Menu

权限说明

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	✓	×	×	×	×

Menu

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 登录虚拟机

操作步骤

管理密钥对

创建密钥对

更新密钥对

删除密钥对

管理虚拟机

重置密码

更新密钥

更新规格

热迁移

更新 NAT 网络配置

更新标签和注释

添加服务

重装操作系统

配置 IP

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 上克隆虚拟机

确保先决条件

快速开始

了解 VirtualMachineClone 对象

物理 GPU 直通环境准备

约束与限制

前提条件

操作步骤

结果验证

相关操作

配置虚拟机的高可用性

概述

术语表

组件概述

fencing 和 remediation 过程中的事件流程

操作步骤

从现有虚拟机创建虚拟机模板

前提条件

操作步骤

问题处理

虚拟机节点正常关机下的 Pod 迁移及异常宕机恢复问题

问题描述

原因分析

解决方法

热迁移错误信息及解决方案

介绍

KubeVirt 提供了诸如 **VirtualMachine** 和 **VirtualMachineInstance** 等 CRD（Custom Resource Definitions，自定义资源定义）来抽象虚拟机（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。

Menu

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 登录虚拟机

操作步骤

管理密钥对

创建密钥对

更新密钥对

删除密钥对

管理虚拟机

重置密码

更新密钥

更新规格

热迁移

更新 NAT 网络配置

更新标签和注释

添加服务

重装操作系统

配置 IP

监控与告警

监控

告警

虚拟机快速定位

前提条件

操作步骤

创建虚拟机/虚拟机组

使用镜像创建虚拟机（VirtualMachineInstance），并将虚拟机调度到安装了 Kubevirt 组件且开启虚拟化的物理机节点上。

您可以通过 创建虚拟机 创建单个虚拟机，您也可以通过 创建虚拟机组（virtualMachinePool），快速创建出多个相同配置的虚拟机（VirtualMachineInstance）。

目录

前提条件

注意事项

创建虚拟机

- 操作步骤
- 相关操作

创建虚拟机组

- 操作步骤

前提条件

- 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。
  - 目标节点需为物理机节点，并开启虚拟化。

- 平台中已添加虚拟机镜像。

- 若需使用虚拟机物理 GPU 直通功能，请联系平台管理员进行如下配置：

  1. 获取 GPU 直通环境准备方案，并准备相关环境。

  2. 准备所需物理 GPU，并开启虚拟机物理 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 账号/密码 登录，请事先联系平台管理员获取。

创建虚拟机

操作步骤

说明：下述内容以表单方式创建虚拟机为例，您也可切换至 YAML 方式完成操作。

1. 进入 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 虚拟机。

3. 单击 创建虚拟机。

4. 在 基本信息 区域填写虚拟机的名称和显示名称，设置标签或注解。

参数	说明
标签	用来选择对象和查找满足某些条件的对象集合。需为键值对，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。需为键值对，例如：*cpaas.io/maintainer: kim*。

5. 设置机型并选择虚拟机镜像。

参数	说明
规格	可根据需求选择平台推荐的使用场景或自定义资源限额。
物理 **GPU**（**Alpha**）	选择物理 GPU 的型号，仅可为每个虚拟机分配一张物理 GPU。 说明：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，从而达到在虚拟机中却可以获得与在物理机上直接运行的同等图形性能，避免虚拟图形适配器引起的性能瓶颈，从而提升整体性能。
镜像	选择平台管理员已分配至平台项目中的公共镜像。 说明：仅支持选择与集群架构相同的 **CPU** 架构。

6. 在 存储 区域，参考以下说明配置相关信息。

参数	说明
磁盘名	存储磁盘的名称，系统盘名不支持修改。
类型	• 根磁盘：系统默认创建一个 VirtIO 类型的 rootfs 系统盘，用于存放操作系统和数据。 • 数据盘：单击添加，添加多块数据盘，可用于持久化存储数据。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：以挂载文件目录的方式挂载磁盘。 • 块设备：以挂载块设备的方式挂载磁盘。
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。此处需要指定动态创建持久卷声明所需的存储类。 各存储类支持的卷模式不同，如果所选卷模式下无可用存储类，请联系管理员进行添加。

参数	说明
容量	虚拟机存储所需的容量，系统盘最小为 20 G。
随虚拟机删除	默认开启，不支持修改，表示删除虚拟机的同时也会删除磁盘数据。

7. 在 网络 区域，参考以下说明配置相关信息。

参数	说明
IP 地址	• 默认采用 动态获取方式（**DHCP**），启动虚拟机时为其动态分配 IP，停止虚拟机后释放 IP。 • 若绑定 固定 **IP**，即使重启，虚拟机也始终使用此 IP 地址。如果当前项目中无可用 IP，请先适当释放 IP。
网络模式	• 桥接：虚拟机与容器组使用相同 IP 地址，并通过此 IP 地址与外部通信。 • **NAT**：虚拟机将被分配内部 IP 地址，但会转换为容器组 IP 地址与外部通信。开放端口表示虚拟机的暴露端口，例如 SSH 服务 22 端口，不填写 开放端口 则表示开放所有端口。
辅助网卡	按需添加辅助网卡。 注意： • 若需使用辅助网卡相关功能或无可用的辅助网卡网络类型，请联系平台管理员配置。 • SR-IOV 类型仅支持 x86_64 架构的 Linux 操作系统使用。 • 默认使用 DHCP 获取 IP 地址。 • SR-IOV 虚拟机多次重启后会出现两个不同的 VF 但是 MAC 地址相同的情况。

8. 在 初始化设置 区域，参考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。此方式无需校验密码，推荐使用密钥方式登录虚拟机。 • 您可使用平台中已有密钥，也可当即创建新密钥，相关密钥均可在 虚拟化 **>** 密钥对 页面查看。 • 仅拥有私钥的人员可通过 SSH 方式访问虚拟机。如需多人共同维护虚拟机，可关联多个密钥，并将私钥分配给不同的使用人员。一旦发生密钥泄露，可及时取消关联该密钥以减少损失。 • SSH 密钥的公钥以保密字典的形式存储于平台中，平台不会存储您的私钥，请自行妥善保管。 • 请查阅操作系统相关文档获取 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。 • 用户仅为初始帐号，虚拟机创建成功后，您也可在虚拟机中创建其他操作系统用户用于登录。 • 平台会加密存储您的 root 用户密码，您将无法再次看见其明文密码，请自行妥善保管。
立即启动	默认开启。开启此项，创建完毕后会立即启动虚拟机，否则将仅创建虚拟机。

9. （可选）在 高级配置 区域，参考以下说明配置相关信息。

参数	说明
健康检查	• 存活性健康检查：检查虚拟机是否处于健康状态，如果检测结果为非正常时，会根据健康检查的配置决定是否重启实例。 • 可用性健康检查：检查虚拟机是否启动完成并处于正常服务状态，如果检测到虚拟机实例的健康状态为非正常时，虚拟机状态会被更新。 。

参数	说明
节点亲和	• **Preferred**：虚拟机将被尽量调度到符合亲和要求的节点上。系统将结合亲和性权重与其它调度需求（例如计算资源需求）确定可运行虚拟机的节点。 • **Required**：虚拟机只会被调度到完全符合亲和要求的节点上。

10. 确认信息无误后，单击 创建。

    等待虚拟机由 创建中 变为 运行中 状态。

相关操作

您可以在列表页面单击右侧的⋮或在详情页面单击右上角的 操作，按需更新或删除虚拟机。重置密码、更新密钥等其它相关操作，请参见 管理虚拟机 。

注意：

- 仅当虚拟机为 异常、未知、已停止 状态时可执行更新操作。
- 更新时不支持展示虚拟机创建完成后单独挂载或创建的磁盘。
- 更新时 立即启动 默认为关闭状态，您可以按需开启。

创建虚拟机组

操作步骤

说明：下述内容以表单方式创建虚拟机组为例，您也可切换至 YAML 方式完成操作。

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 虚拟化 > 虚拟机组。
3. 单击 创建虚拟机组。

4. 在 基本信息 区域，参考以下说明配置虚拟机组的信息。

参数	说明
实例数	通过虚拟机组创建的虚拟机的个数。
实例间反亲和	启用后，调度多个虚拟机至节点时，会尽量让虚拟机分布到不同的节点上，可提升一组虚拟机的高可用性。
标签	可为虚拟机组添加标签。标签可用于选择对象和查找满足某些条件的对象集合。需为键值对，例如：*app.kubernetes.io/name: hello-app*。

5. 在 虚拟机模板 区域，参考 创建虚拟机 为组中的所有虚拟机配置统一的标签、注解、规格、镜像、存储等信息。

6. 确认信息无误后，单击 创建。

   提示：创建成功后，可前往 虚拟机 的列表页面查看通过虚拟机组创建出的虚拟机的信息。

虚拟机批量操作

批量进行启动、停止、重启和删除虚拟机操作。

目录

操作步骤

操作步骤

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 虚拟化 > 虚拟机。
3. 找到目标虚拟机，单击⋮可对单个虚拟机进行操作，或参考下图进行虚拟机的批量操作。

   注意：

   - 当虚拟机处于休眠或已停止状态时，可以执行 启动/批量启动 操作；当虚拟机处于准备中、启动中、运行中、休眠、未知或异常状态时，可以执行 停止/批量停止 操作；当虚拟机处于运行中状态时，可以执行 重启/批量重启 操作。
   - 对虚拟机进行强制 重启/停止 操作，相当于对虚拟机断电，这可能会导致未写入磁盘的数据丢失。
4. 根据界面提示完成操作。当虚拟机变为如下状态时，操作成功。

操作	状态
启动虚拟机	运行中
停止虚拟机	已停止
重启虚拟机	运行中

使用 **VNC** 登录虚拟机

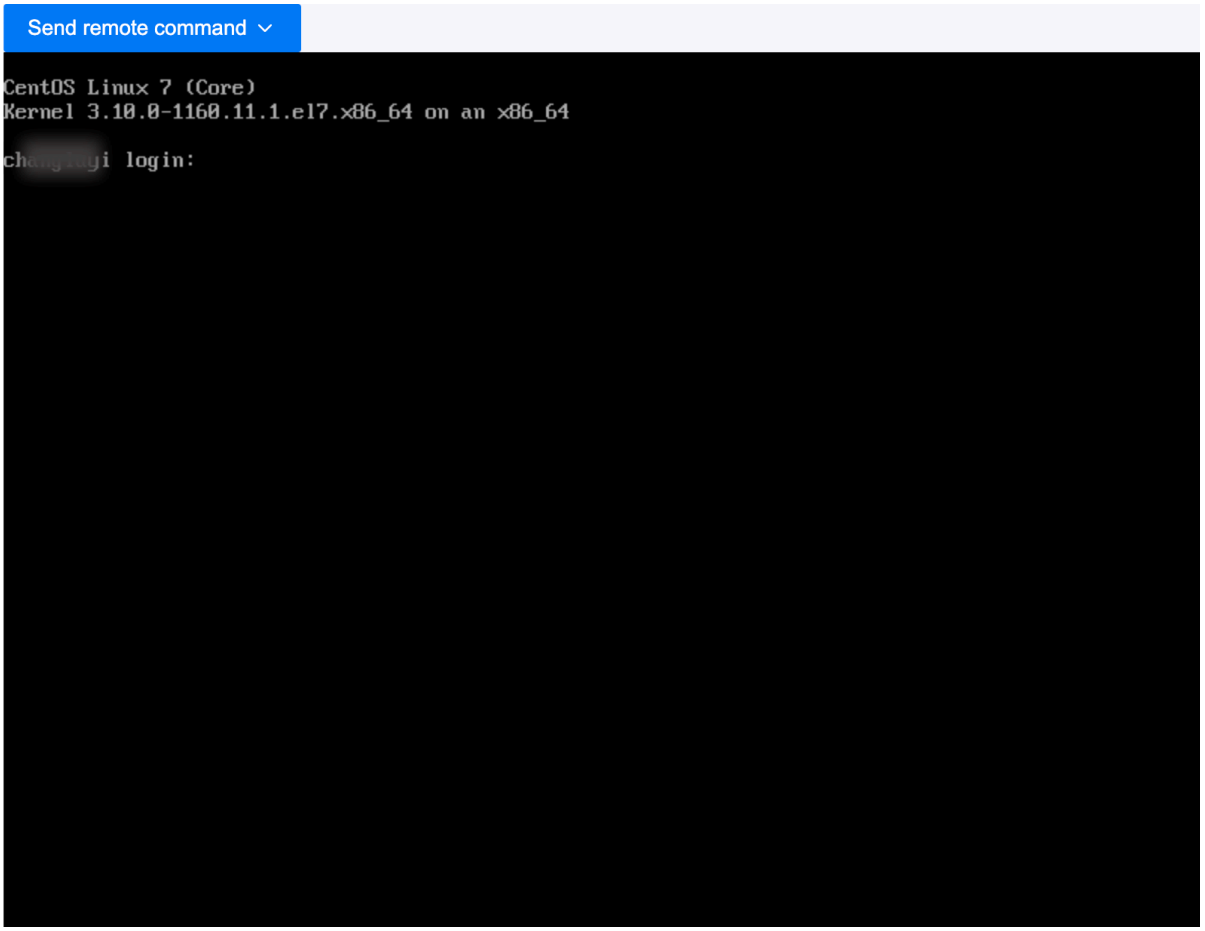
使用 Web 控制台（VNC）登录虚拟机，作为应急操作手段。

目录

操作步骤

操作步骤

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 虚拟化 > 虚拟机。
3. 单击⋮> **VNC** 登录。
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。



说明：

- 支持发送常用键盘命令。
- 支持复制和粘贴命令和参数。

管理密钥对

创建、更新或删除密钥对。

目录

创建密钥对
更新密钥对
删除密钥对

创建密钥对

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 虚拟化 > 密钥对。
3. 单击 创建密钥对。

   当前仅支持 SSH 类型的密钥对。您可以手动导入密钥或让系统自动生成密钥对。使用系统生成的密钥对时，平台支持自动将私钥下载到您的本地机器。平台不会保存私钥。
4. 单击 创建。

更新密钥对

1. 进入 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 密钥对。

3. 找到 *密钥对名称*，单击⋮> 更新。

4. 重新导入或让系统生成新的密钥对后，单击 更新。

删除密钥对

1. 进入 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 密钥对。

3. 找到 *密钥对名称*，单击⋮> 删除，并确认。

管理虚拟机

目录

重置密码
操作步骤
更新密钥
操作步骤
更新规格
热迁移
更新 NAT 网络配置
操作步骤
更新标签和注释
添加服务
重装操作系统
操作步骤
配置 IP
操作步骤

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，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，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** 创建密钥。
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，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，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，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，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，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，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，先选择虚拟机镜像的操作系统，再选择该操作系统下的虚拟机镜像。
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，虚拟机重启后新 IP 生效。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏，点击 **Virtualization** > **Virtual Machines**。
3. 找到目标虚拟机并点击⋮> **Configure IP**。
4. 配置 **IP Address**。
   - 填写可用 IP：绑定固定 IP 表示即使重启，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，停止时释放。
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

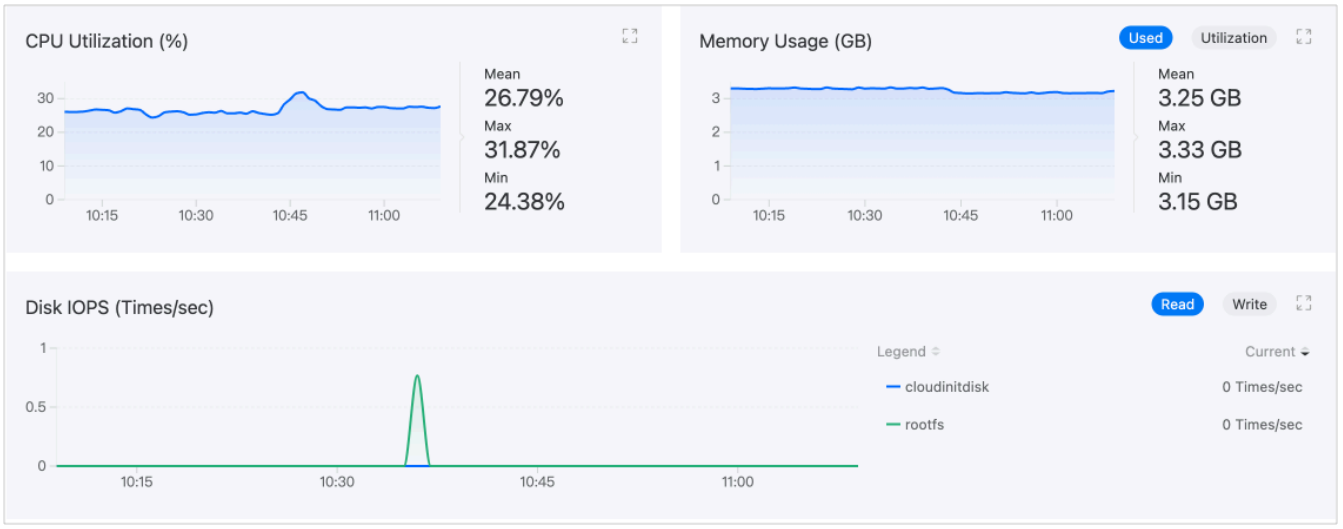
监控

告警

- 配置告警策略
- 处理告警
- 绑定通知策略

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间。
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：监控对象存在运行风险，若不及时处理可能影响正常业务运行。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，可能导致平台功能异常，影响正常业务运行。

参数	描述
	- 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。更多详细配置指导，请参见通用 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

操作步骤

前提条件

确保当前集群已启用虚拟化功能，具体请参见安装。

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。

Menu

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 上克隆虚拟机

确保先决条件

快速开始

了解 VirtualMachineClone 对象

物理 GPU 直通环境准备

约束与限制

前提条件

操作步骤

结果验证

相关操作

配置虚拟机的高可用性

概述

术语表

组件概述

fencing 和 remediation 过程中的事件流程

操作步骤

从现有虚拟机创建虚拟机模板

前提条件

操作步骤

配置 **USB** 主机直通

目录

功能概述

使用场景

前提条件

操作步骤

- 暴露 USB 设备
- 将 USB 设备分配给虚拟机

操作结果

了解更多

- 暴露多个 USB 设备
- 将 USB 设备分配给虚拟机

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` 部分进行配置。

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. RTL8812AU
> 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机节点的 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 802.11a/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。我们支持在同一个 resourceName 下分配多个 USB 设备。配置方法如下：

1 暴露多个 USB 设备

```
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

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

虚拟机热迁移

目录

Overview
ProCopy
约束与限制
前提条件
操作步骤
部署 kubevirt-operator
创建 HyperConverged 实例
准备虚拟机
启动热迁移

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物理服务器。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来保证迁移过程中的服务连续性。具体过程如下：

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，直到修改的内存页数量降至可接受的水平。每一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. 停止拷贝：当剩余未拷贝的内存页足够少时，虚拟机会短暂暂停（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，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，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

在某些场景中，例如错误修改 fstab 或文件系统错误需要执行 fsck，虚拟机可能无法正常启动。在这种情况下，您可以利用救援模式来修复根文件系统（rootfs）或从系统中检索数据。

目录

操作步骤
- 获取镜像地址
- 修改虚拟机 YAML 文件
- 挂载原有 rootfs 并进行修复
- 还原虚拟机 YAML 文件

操作步骤

获取镜像地址

1. 在左侧导航栏中，单击 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，并将 操作系统 选择为 **CentOS** 或 **Ubuntu**。然后单击右侧的⋮> 更新。
3. 复制并保存 镜像地址。本文档以 `192.168.1.1:11443/3rdparty/vmdisks/centos:7.9` 为例。
4. 单击 取消。

修改虚拟机 **YAML** 文件

1. 访问 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 虚拟机。

3. 单击需要修复的虚拟机右侧的⋮> 停止，将虚拟机 停止 或 强制停止。

4. 单击虚拟机右侧的⋮> 更新。

5. 切换至 **YAML** 并修改以下字段。

   - 在 `spec.template.spec.domain.devices.disks` 字段下增加以下内容。增加 bootOrder 参数可以控制虚拟机启动时的磁盘优先级；bootOrder 数值越小表示优先级越高。

     注意：如果原 `spec.template.spec.domain.devices.disks` 字段中已有 `bootOrder: 1`，请增大原数值，以确保新增的 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  # 增加的字段
        disk:
          bus: virtio
        name: containerdisk
      - disk:
          bus: virtio
        name: cloudinitdisk
      - disk:      # 增大原 bootOrder: 1 的数值
          bus: virtio
        name: rootfs
        bootOrder: 10
      - disk:
          bus: virtio
        name: "1"
```

- 在 `spec.template.spec.volumes` 字段下增加以下内容。

  注意：请用在 获取镜像地址 中获取的镜像地址替换下述 `image` 字段的镜像地址。

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

  修改后的 YAML 示例：

```
volumes:
  - containerDisk:  # 增加的字段
      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. 单击 更新。

注意：修改 YAML 文件后，请勿切换至 表单，直接单击 更新 即可。

7. 单击虚拟机右侧的⋮> 启动。

挂载原有 **rootfs** 并进行修复

1. 使用原密码或密钥登录虚拟机，输入 `df -h /` 命令，发现 rootfs 文件系统已被替换。您可以使用与挂载相关的命令进行挂载，或使用 fsck 相关命令检查和修复原文件系统。

2. 完成后关闭虚拟机。

还原虚拟机 **YAML** 文件

按照 修改虚拟机 YAML 文件 的步骤，将虚拟机 YAML 文件恢复至原始状态，此时虚拟机应能正常启动。

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

快速开始

了解 VirtualMachineClone 对象

查看完整的 VirtualMachineClone 示例

理解各字段含义

检查克隆操作阶段

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，并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。 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，使关系正确显示（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

以下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

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

  # 模板过滤器，用于管理网络相关注解
  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"

  # 用于进一步自定义克隆虚拟机的 JSON 补丁
  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内。
- 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，则自动重新生成。
- **newSMBiosSerial**：
  - 可选，指定新的 SMBios 序列号。
  - 如果未提供，则基于虚拟机名称自动生成。
- **JSON** 补丁：
  - 用于对虚拟机规格进行高级自定义。

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。

阶段	说明
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，目标虚拟机及存储已准备就绪。
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，从而达到在虚拟机中获得与在物理机上直接运行的同等图形性能，避免虚拟图形适配器引起的性能瓶颈，从而提升整体性能。

目录

- 约束与限制
- 前提条件
  - Chart 及镜像准备
  - 开启 IOMMU
- 操作步骤
  - 创建命名空间
  - 部署 gpu-operator
  - 配置 Kubevirt
- 结果验证
- 相关操作
  - 删除直通 GPU 的虚拟机
  - 将 GPU 相关配置从 KubeVirt 配置中删除
  - 卸载 gpu-operator

约束与限制

物理 GPU 直通功能需使用 kubevirt-gpu-device-plugin，但目前没有适用于 ARM64 CPU 架构的 kubevirt-gpu-device-plugin 镜像，因此无法在 ARM64 架构的操作系统中使用此功能。

前提条件

Chart 及镜像准备

获取下述 Chart 及镜像并上传至镜像仓库中，本文档以 `build-harbor.example.cn` 仓库地址为例进行介绍，具体 Chart 及镜像的获取方式请联系相关人员。

Chart

- build-harbor.example.cn/example/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

开启 IOMMU

在不同操作系统开启 IOMMU 的操作会有所区别，请参考对应操作系统文档，本文以 CentOS 为例进行介绍，所有命令均在终端中执行。

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

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

2. 执行下述命令生成 `grub.cfg` 文件。

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

3. 重新启动服务器。

4. 执行下述命令确认 IOMMU 是否开启成功。若回显信息中显示 `IOMMU enabled`，则表示开启成功。

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

操作步骤

注意：下述所有命令如无特殊说明均需在对应集群 Master 节点上的 CLI 工具中执行。

创建命名空间

执行下述命令创建名称为 `gpu-system` 的命名空间，若出现 `namespace/gpu-system created` 回显信息则表示已创建成功。

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

部署 **gpu-operator**

1. 执行下述命令部署 gpu-operator。

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

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: <chartName> # 需使用实际的 chart 路径替换 <chartName> 部分，例如：name = example/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. 执行下述命令检查 gpu-operator 是否已同步，若 `SYNC` 显示为 `Synced` 表示已同步。

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

回显信息：

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

3. 执行下述命令获取所有节点名称，并找到 GPU 节点名称。

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

4. 执行下述命令查看 GPU 节点是否已有可直通的 GPU，若回显信息中出现类似于 `nvidia.com/GK210GL_TESLA_K80` 的 GPU 信息，则表示已有可直通的 GPU。

```
kubectl get node <gpu-node-name> -o jsonpath='{.status.allocatable}' # 使用步骤 3 中获取到的 GPU 节点名称替换 <gpu-node-name> 部分
```

回显信息：

```
{"cpu":"39","devices.kubevirt.io/kvm":"1k","devices.kubevirt.io/tun":"1k","devices.kubevirt.io/vhost-net":"1k","ephemeral-storage":"426562784165","hugepages-1Gi":"0","hugepages-2Mi":"0","memory":"122915848Ki","nvidia.com/GK210GL_TESLA_K80":"8","pods":"256"}
```

5. 至此 gpu-operator 已经成功部署。

配置 **Kubevirt**

1. 执行下述命令开启 DisableMDEVConfiguration 特性，若返回类似 `hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched` 的回显信息，则表示开启成功。

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

2. 在 GPU 节点的终端中执行下述命令，获取 pciDeviceSelector。回显信息中的 `10de:102d` 部分即为 pciDeviceSelector 的值。 {#pciDeviceSelector}

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

回显信息：

```
04:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
05:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
08:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
09:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
85:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
86:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
89:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
8a:00.0 3D controller [0302]: NVIDIA Corporation GK210GL [Tesla K80] [10de:102d] (rev
a1)
```

3. 执行下述命令获取所有节点名称，并找到 GPU 节点名称。

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

4. 执行下述命令获取 resourceName。回显信息中的 `nvidia.com/GK210GL_TESLA_K80` 部分即为 resourceName 的值。

```
kubectl get node <gpu-node-name> -o jsonpath='{.status.allocatable}' # 使用步骤 3 中获
取到的 GPU 节点名称替换 <gpu-node-name> 部分
```

回显信息：

```
{"cpu":"39","devices.kubevirt.io/kvm":"1k","devices.kubevirt.io/tun":"1k","devices.kubevir
net":"1k","ephemeral-storage":"426562784165","hugepages-1Gi":"0","hugepages-
2Mi":"0","memory":"122915848Ki","nvidia.com/GK210GL_TESLA_K80":"8","pods":"256"}
```

5. 执行下述命令添加直通 GPU。

注意：使用 步骤 2 中获取的 pciDeviceSelector 值替换下述命令中的 <pci-devices-id> 部分时，pciDeviceSelector 中的 所有英文字母全部需要转换为大写。例如：获取到的 pciDeviceSelector 的值为 `10de:102d` ，则应替换为 `export DEVICE=10DE:102D` 。

- 添加单块 GPU 卡

```
export DEVICE=<pci-devices-id> # 使用步骤 2 中获取的 pciDeviceSelector 替换 <pci-devices-id> 部分。例如：export DEVICE=10DE:102D
export RESOURCE=<resource-name> # 使用步骤 4 中获取的 resourceName 替换 <resource-name> 部分。例如：export RESOURCE=nvidia.com/GK210GL_TESLA_K80

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

- 添加多块 GPU 卡

  注意：添加多块 GPU 卡时，每个用来替换 <pci-devices-id> 的 pciDeviceSelector 值必须不相同。

```
export DEVICE1=<pci-devices-id1> # 使用步骤 2 中获取的 pciDeviceSelector 替换 <pci-devices-id1> 部分
export RESOURCE1=<resource-name1> # 使用步骤 4 中获取的 resourceName 替换 <resource-name1> 部分
export DEVICE2=<pci-devices-id2> # 使用步骤 2 中获取的 pciDeviceSelector 替换 <pci-devices-id2> 部分
export RESOURCE2=<resource-name2> # 使用步骤 4 中获取的 resourceName 替换 <resource-name2> 部分

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

- 已经添加过 GPU 卡，再次添加新的 GPU 卡

```
export DEVICE=<pci-devices-id> # 使用步骤 2 中获取的 pciDeviceSelector 替换 <pci-devices-id> 部分
export RESOURCE=<resource-name> # 使用步骤 4 中获取的 resourceName 替换 <resource-name> 部分
export INDEX=<index> # index 是从 0 开始的数组编号，使用编号替换 <index> 部分。例如：已经添加过一块 GPU 卡，现在要新增一块 GPU 卡，那么 index 应该为 1，即 export INDEX=1

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

上述步骤配置完成后，若在创建虚拟机时能够选择到对应物理 GPU，则表示物理 GPU 直通环境已经准备完成。

注意：若需配置物理 GPU 直通，请提前开启相关功能。

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 虚拟化 > 虚拟机。
3. 单击 创建虚拟机。
4. 配置虚拟机 物理 **GPU**（**Alpha**） 参数。

参数	说明
物理 **GPU**（**Alpha**）	选择配置的物理 GPU 的型号，仅可为每个虚拟机分配一张物理 GPU。

5. 至此，物理 GPU 直通环境已经准备完成。

相关操作

删除直通 **GPU** 的虚拟机

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 虚拟化 > 虚拟机。
3. 在列表页中单击需要删除的虚拟机右侧的⋮> 删除，或单击需要删除的虚拟机名称进入详情信息页面，单击 操作 > 删除。
4. 输入确认信息，删除直通 GPU 的虚拟机。

将 **GPU** 相关配置从 **KubeVirt** 配置中删除

1. 在 GPU 对应的集群 Master 节点上，使用 CLI 工具执行下述命令，将 GPU 相关配置从 KubeVirt 配置中删除。

```
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，则表示删除成功，具体创建虚拟机的步骤请参考 选择物理 GPU 型号。

卸载 **gpu-operator**

1. 在 GPU 对应的集群 Master 节点上，使用 CLI 工具执行下述命令，卸载 gpu-operator。

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

回显信息：

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

2. 执行命令，若出现类似下述回显信息，则表示 gpu-operator 已卸载成功。

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

回显信息：

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

配置虚拟机的高可用性

目录

概述

术语表

组件概述

fencing 和 remediation 过程中的事件流程

操作步骤

- Operator 列表
- 部署 Self Node Remediation Operator
- 配置 Self Node Remediation Operator（可选）
- 配置 Self Node Remediation 模板（可选）
- 部署 Node Health Check Operator
- 创建 NodeHealthCheck 实例
- 验证（可选）

概述

硬件存在缺陷，软件包含漏洞。当发生节点级故障，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。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。为便于此类干预，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**

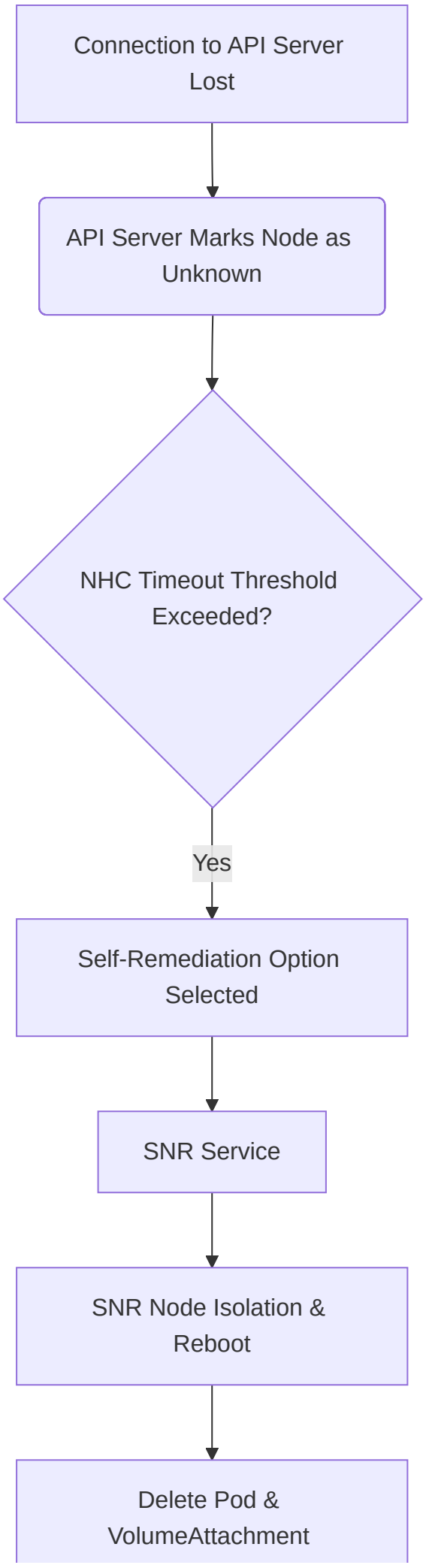
  Self Node Remediation Operator 是 Alauda Container Platform 的一个附加 Operator，实现了一个外部的 fencing 和 remediation 系统，该系统会重启不健康的节点并删除资源，如 Pod 和 VolumeAttachment。重启确保工作负载被 fencing，资源删除加快了受影响工作负载的重新调度。与其他外部系统不同，Self Node Remediation 不需要任何管理接口，例如智能平台管理接口（IPMI）或节点配置的 API。

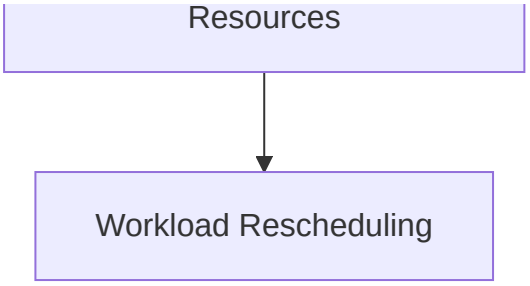
  Self Node Remediation 可被故障检测系统使用，如 Machine Health Check 或 Node Health Check。

- **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 过程中的事件流程


Connection to API Server Lost
API Server Marks Node as Unknown
NHC Timeout Threshold Exceeded?
Yes
Self-Remediation Option Selected
SNR Service
SNR Node Isolation & Reboot
Delete Pod & VolumeAttachment



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。
- 使用上传软件包机制 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。
- 使用上传软件包机制 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 在 Self Node Remediation Operator 所在的命名空间中创建。

> **注意**
>
> 修改 `SelfNodeRemediationConfig` CR 会重新创建 Self Node Remediation 守护进程集。

`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	说明
safeTimeToAssumeNodeRebootedSeconds	指定 Operator 在恢复运行于不健康节点上的受影响工作负载之前等待的可选时间。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、ApiCheckInterval、MaxApiErrorThreshold、PeerDialTimeout、PeerRequestTimeout 字段值，以及 watchdog 超时和恢复时的集群规模计算最小等待时间。
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。 如果没有可用的 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 服务器连接的超时时间。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 服务器连接的频率。超时时间必须大于等于

参数	说明
	1 秒。
maxApiErrorThreshold	指定阈值。达到该阈值后，节点开始联系其对等节点。阈值必须大于等于 1 秒。
peerApiServerTimeout	指定对等节点连接 API 服务器的超时时间。超时时间必须大于等于 10 毫秒。
peerDialTimeout	指定与对等节点建立连接的超时时间。超时时间必须大于等于 10 毫秒。
peerRequestTimeout	指定从对等节点获取响应的超时时间。超时时间必须大于等于 10 毫秒。
peerUpdateInterval	指定更新对等节点信息（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	指定运行在守护进程集上的 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 自定义资源定义（CRD）。该 CRD 定义了节点的 remediation 策略，旨在更快恢复工作负载。可用的 remediation 策略如下：

- **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存在 `OutOfServiceTaint` 策略。如果存在，Operator 选择 `OutOfServiceTaint` 策略；如果不存在，Operator 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，而非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 和相关卷附件被移除，而非删除节点对象。

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	说明
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`。
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**。

参数	推荐配置
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	说明
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，确认虚拟机是否会自动调度到其他节点上运行。

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

操作步骤

步骤 1：虚拟机上的基础配置

步骤 2：创建虚拟机快照

步骤 3：获取磁盘快照资源名称

步骤 4：创建 DataSource 资源

标签参数说明：

步骤 5：使用模板创建新虚拟机

前提条件

- 已正确部署和配置的 KubeVirt 环境。
- 可访问 Web 控制台和 kubectl 工具。
- 已配置并安装必要软件的虚拟机。

操作步骤

步骤 1：虚拟机上的基础配置

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

使用 KubeVirt Web 控制台：

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** 控制台：
  - 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	可能取值	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；虚拟机热迁移需使用

键	可能取值	说明
		ReadWriteMany
virtualization.cpaas.io/size	容量（Gi、Ti 等）	默认磁盘大小，请根据需求指定
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`。
- 这些磁盘相关参数在创建虚拟机时可以修改，但提供默认值可简化操作。

步骤 5：使用模板创建新虚拟机

1. 访问 KubeVirt Web 控制台，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。

Menu

问题处理

虚拟机节点正常关机下的 **Pod** 迁移及异常宕机恢复问题

问题描述

原因分析

解决方法

热迁移错误信息及解决方案

虚拟机节点正常关机下的 **Pod** 迁移及异常宕机恢复问题

目录

问题描述

原因分析

解决方法

正常关机下的虚拟机 Pods 迁移

异常宕机恢复

问题描述

无论节点是 正常关机 还是 异常宕机，运行在该节点上的虚拟机 Pods 都不会自动迁移至其他健康节点。

原因分析

平台基于开源组件 KubeVirt 实现了虚拟机解决方案。然而，从 KubeVirt 的角度来看，无法区分虚拟机是实际宕机还是由于网络或其他原因导致的连接失败。若不加区分地将虚拟机迁移至其他节点，可能导致同一虚拟机的多个实例同时存在。

解决方法

在维护虚拟机节点时，需要根据此文档进行手动操作。针对 正常关机 和 异常宕机 两种情况，需手动驱逐或强制删除虚拟机 Pods。

注意：下述命令均需在对应集群的 Master 节点上执行。

正常关机下的虚拟机 **Pods** 迁移

1. 在 CLI 工具中，执行下述命令获取节点信息。其中，回显信息中的 `NAME` 字段即为 `Node-Name`。

   ```
   kubectl get nodes
   ```

   回显信息：

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

2. （可选）执行下述命令查看节点下的虚拟机实例。

   ```
   kubectl get vmis --all-namespaces -o wide | grep <Node-Name>  # 使用步骤 1 中获取的 Node-Name 替换命令中的 <Node-Name> 部分
   ```

   回显信息：

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

3. 正常关机前，执行下述命令驱逐需关机节点上的所有虚拟机 Pods，若出现如下回显信息，则表示已成功驱逐。

```
kubectl drain <Node-Name> --delete-local-data --ignore-daemonsets=true --force --pod-selector=kubevirt.io=virt-launcher   # 使用需关机节点的 Node-Name 替换命令中的 <Node-Name> 部分
```

回显信息：

```
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. 等所有虚拟机在其他节点上启动后，将节点关闭。

5. 节点关机并重新开启后，执行下述命令将该节点标记为可调度。

```
kubectl uncordon <Node-Name> # 使用关机并重启节点的 Node-Name 替换命令中的 <Node-Name> 部分
```

回显信息：

```
node/1.1.1.211 uncordoned
```

6. 至此，该节点上的原虚拟机实例已迁移至其他健康节点，该节点重启后已允许新的 Pods 调度。

异常宕机恢复

1. 在 CLI 工具中，执行下述命令获取节点信息。其中，回显信息中的 `NAME` 字段即为 `Node-Name`。

```
kubectl get nodes
```

回显信息：

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

2. 执行下述命令，强制删除该节点上的所有虚拟机 Pods。

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

3. 执行下述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  # 需使用异常宕机节点的 Node-
Name 替换命令中的 <Node-Name> 部分
```

4. 执行下述命令查询异常宕机节点上是否存在具有标签 kubevirt.io=virt-api 的 Pods。

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

若存在则执行下述命令删除 Pods。

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

5. 执行下述命令查询异常宕机节点上是否存在具有标签 kubevirt.io=virt-controller 的 Pods。

```
kubectl -n kubevirt get po -l kubevirt.io=virt-controller -o wide | grep <Node-Name> #
需使用异常宕机节点的 Node-Name 替换命令中的 <Node-Name> 部分
```

若存在则执行下述命令删除 Pods。

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

6. 至此，节点异常宕机后，虚拟机实例将迁移至其他健康节点。

Menu

热迁移错误信息及解决方案

错误信息	错误原因	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。	请依次检查如下配置： • 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 文件中的 `metadata.annotations` 和 `spec.template.metadata.annotations` 字段中是否存在 "kubevirt.io/allow-pod-bridge-network-live-migration": "true" 注解信息，若不存在需手动添加。
• 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 ReadWriteMany access mode)	虚拟机的存储类型不支持多节点读写（RWX）访问模式。	虚拟机在创建后不支持修改相关参数，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；若重新创建后依然存在问题，请联系相关人员处理。

错误信息	错误原因	解决方案
• 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。	请联系相关人员处理。

Menu

网络

介绍

介绍
优势

操作指南

配置网络
配置 IP
通过 IP 直接连接到虚拟机
添加内部路由

实用指南

通过网络策略实现虚拟机网络请求控制
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

结果验证

相关说明

配置虚拟机使用网络绑定模式以支持 IPv6

前提条件

操作步骤

介绍

ACP Virtualization With KubeVirt 与 **Kube-OVN** 进行了深度集成，同时扩展了对传统虚拟机 (VM) 网络需求的支持，并优化了特定场景的性能。

目录

优势

优势

- **IPv6** 支持

  完整的 IPv6 支持。

- 静态 **IP** 保留

  确保虚拟机在重启后保留相同的 IP 地址，符合传统虚拟机的使用模式。

- 多网络模式支持

  支持多种网络模式，如容器网络和 SR-IOV，以满足不同用户场景的需求。

Menu

操作指南

配置网络

配置 IP

通过 IP 直接连接到虚拟机

添加内部路由

配置网络

目录

配置 IP

通过 IP 直接连接到虚拟机

添加内部路由

配置 IP

参考 配置 IP

通过 IP 直接连接到虚拟机

参考 准备 Kube-OVN Underlay 物理网络

添加内部路由

参考 添加内部路由

Menu

实用指南

通过网络策略实现虚拟机网络请求控制

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

结果验证

相关说明

配置虚拟机使用网络绑定模式以支持 IPv6

前提条件

操作步骤

通过网络策略实现虚拟机网络请求控制

平台基于开源组件 KubeVirt 实现的虚拟机方案，而 KubeVirt 实际上运行在 Pods 中，使用网络策略（Network Policy）的功能，可以实现对虚拟机进出请求的控制。

目录

操作步骤

结果验证

步骤一：创建虚拟机和允许所有流量通过的网络策略

步骤二：更新网络策略，将 www.example.com 从白名单去除

操作步骤

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 网络 > 网络策略。
3. 单击 创建网络策略。
4. 按需配置如下参数。

参数	说明
关联方式	• 计算组件：按需选择目标计算组件，建议目标计算组件选择 全部。 • 标签选择器：根据标签匹配 Pods。
方向	• 入站：外部发送到 Pod 的请求。 • 出站：由 Pod 发送到外部的请求，若需禁止虚拟机请求某个外部地址可以选择此项。
协议	选择 TCP 或 UDP 协议。 注意： • 在虚拟机中使用域名请求外部服务时，由于 DNS 协议使用的是 UDP 协议，因此需要添加 UDP 协议白名单。 • 表单页面不支持配置 ICMP 协议，当开启白名单规则后，会禁用 ICMP 协议，导致无法进行 Ping 操作。
访问端口	指定哪些端口的流量可以入站或出站。若不填写此项，则默认允许所有端口的流量通过。 注意：此处需要放通 UDP 和 TCP 协议的 1053、53 两个端口，以允许 DNS 流量出站；否则，域名解析将失败。
远端类型	指定允许访问的远端类型。可选项包括：计算组件、命名空间、IP 段。
排除远端	当远端类型为 **IP** 段 时，将指定的 IP 从白名单中移除（即禁止访问）。当输入形式如 `IP/32` 的 IP 时可以移除单个 IP。 注意：此处仅支持输入 IP，若不清楚域名对应的 IP，可以使用命令 `curl -vvv <域名>` 来获取。

5. 单击 创建。

结果验证

本文档以使用虚拟机访问 www.example.com↗ 为例进行验证。

步骤一：创建虚拟机和允许所有流量通过的网络策略

1. 创建虚拟机，具体创建步骤请参考 创建虚拟机。

2. 在虚拟机所在的命名空间配置网络策略，添加 TCP 及 UDP 协议的白名单规则，配置参数如下：

   - TCP 协议的白名单：

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**。
远端类型	选择 **IP** 段。
远端	输入 **0.0.0.0/0**，表示允许所有流量出站。

   - UDP 协议的白名单规则：

参数	说明
方向	选择 出站。
协议	选择 **UDP**。
远端类型	选择 **IP** 段。
远端	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，在虚拟机中执行下述命令请求 www.example.com↗。

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

4. 请求成功。

步骤二：更新网络策略，将 www.example.com↗ 从白名单去除

1. 执行下述命令获取 www.example.com↗ 的 IP 地址，可以得到 IP 地址为 93.184.215.14。

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

2. 更新 步骤一 中创建的网络策略，更新的参数如下：

参数	说明
排除远端	在 TCP 协议的白名单规则中，排除远端参数中填写 93.184.215.14/32，表示将 IP 地址 93.184.215.14 从白名单中移除。

3. 网络策略更新完成后，登录虚拟机，在虚拟机中执行下述命令请求 www.example.com↗。

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

4. 请求超时，排除远端功能生效。

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

术语

约束与限制

前提条件

- Chart
- 镜像

操作步骤

- 在物理机 BIOS 中启用 SR-IOV
- 启用 IOMMU
- 在系统内核加载 VFIO 模块
- 创建 VF 设备
- 绑定 VFIO 驱动
- 部署 Multus CNI 插件
- 部署 sriov-network-operator
- 为物理节点设置 Node Role 标识标签
- 检查资源是否创建成功
- 为物理节点设置 SR-IOV 节点特性标签
- 检查网卡设备支持情况
- 配置 IP 地址

结果验证

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 的多网络接口。
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，提供更优异的网络性能。
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，因此这两种操作系统无法使用 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
```

BIOS 中启用 SR-IOV 的操作因服务器厂商不同而异，请参考对应厂商文档。一般步骤如下：

1. 重启服务器。

2. BIOS POST 期间屏幕显示品牌 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

启用 IOMMU 的操作因操作系统不同而异，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 quiet
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`，则表示启用成功。

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

3 在系统内核加载 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，则表示 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. 执行以下命令查看物理机的网卡信息。

```
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<global>
        inet6 fe80::a236:9fff:fe29:6c00  prefixlen 64  scopeid 0x20<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 0x20<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，则需全部配置。

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

echo 8 > /sys/devices/pci0000:00/0000:00:03.0/0000:05:00.1/sriov_numvfs  ## 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：可看到配置的 8 个 VF 地址，如 `05:10.1`，这些 VF 地址需补充 **Domain Identifier**，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 Controller Virtual
Function (rev 01)
05:10.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
05:10.5 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
05:10.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
05:11.1 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
05:11.3 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
05:11.5 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual
Function (rev 01)
05:11.7 Ethernet controller: Intel Corporation 82599 Ethernet Controller 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=eno1 drv=tg3
unused=vfio-pci
0000:01:00.1 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno2 drv=tg3
unused=vfio-pci
0000:02:00.0 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno3 drv=tg3
unused=vfio-pci
0000:02:00.1 'NetXtreme BCM5720 Gigabit Ethernet PCIe 165f' if=eno4 drv=tg3
unused=vfio-pci
0000:05:00.0 'Ethernet 10G 2P X520 Adapter 154d' if=enp5s0f0 drv=ixgbe
unused=vfio-pci *Active*
0000:05:00.1 'Ethernet 10G 2P X520 Adapter 154d' if=enp5s0f1 drv=ixgbe
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 Role** 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`，检查是否成功创建了 `sriovnetworknodestates` 资源。如果看到如下类似输出，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，确保已成功创建 `sriovnetworknodestates` 资源。

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 X520 Adapter (rev
   01)
   ```

3. 在 cpaas-system 命名空间中，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，可在绑定结果中找到。

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

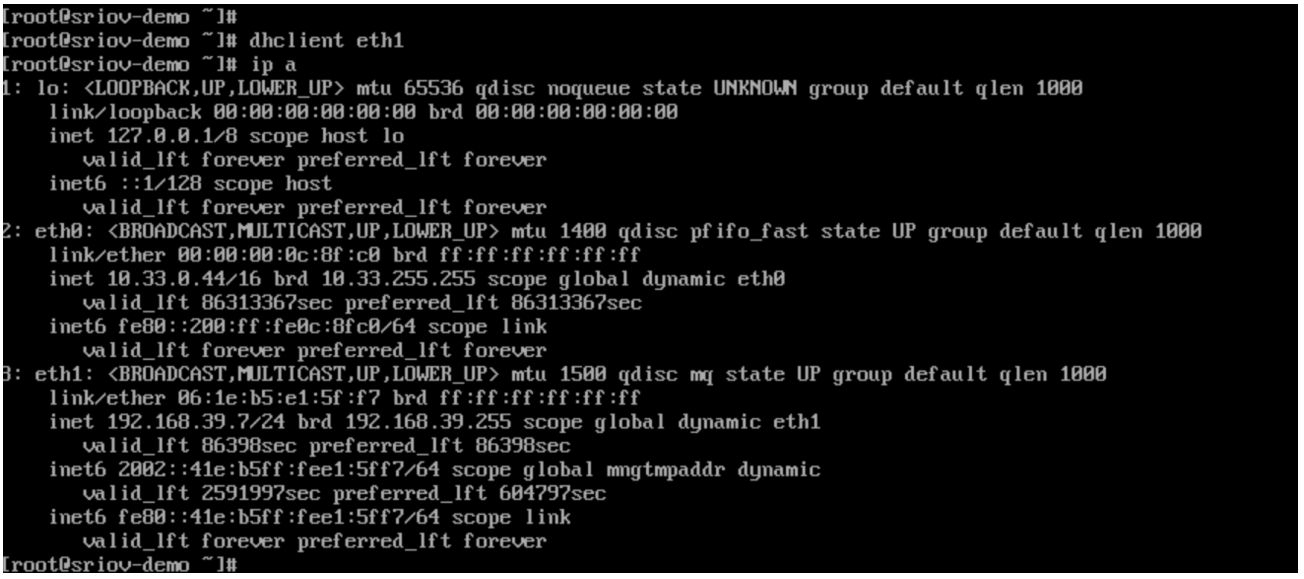
1. 进入 **Container Platform**。

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，应该能看到 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。输出信息中的值为 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一种插件扩展机制。平台默认使用一个名为 ManagedTap 的插件来启用虚拟机的 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，虚拟机也将获得 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 的组合实现强大的 IPv6 功能。

目录

前提条件

操作步骤

为虚拟机子网添加 IPv6 配置

在网页控制台使用网络绑定模式创建虚拟机

通过 VNC 访问虚拟机并配置网络接口

配置 IPv6 默认路由

前提条件

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

操作步骤

1 为虚拟机子网添加 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 在网页控制台使用网络绑定模式创建虚拟机

创建虚拟机时，选择 网络绑定 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，并添加以下配置：

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

重启网络

```
systemctl restart network
```

4 配置 IPv6 默认路由

如果交换机配置为发送路由公告（RA）消息，则不需要手动配置路由。可以通过交换机的 RA 消息自动学习到默认路由。

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

Menu

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 虚拟化与 KubeVirt 存储提供了虚拟机 (VM) 的持久化存储能力，通过无缝集成 Kubernetes 原生的存储机制。它利用 **PersistentVolumeClaim**（PVC）来存储虚拟机磁盘数据，并使用 **Container Storage Interface**（CSI）与各种存储系统集成。此外，**Containerized Data Importer**（CDI）被用于初始化虚拟机磁盘数据。在这些基础上，该平台扩展了虚拟机磁盘管理的高级功能，实现全面的生命周期控制。

目录

优势

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，减少了对命令行界面的依赖。

- 虚拟机磁盘生命周期管理

  可以配置虚拟机磁盘是否在相关虚拟机终止时被自动删除。

Menu

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

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
操作步骤
挂载虚拟磁盘
操作步骤
扩容虚拟磁盘
操作步骤
卸载虚拟磁盘
操作步骤
删除虚拟磁盘
操作步骤

创建虚拟磁盘

为虚拟机创建 数据盘，仅支持添加 一个 虚拟磁盘，若需多个，请重复此操作。

注意：虚拟机处于 运行中 状态时，可以在线挂载虚拟磁盘。

操作步骤

1. 进入 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 虚拟磁盘。

3. 单击 创建虚拟磁盘。

4. 根据以下说明配置相关信息。

参数	说明
卷模式	- 文件系统：以挂载文件目录的形式挂载磁盘。 - 块设备：以块设备的形式挂载磁盘。
存储类	平台通过自动创建和管理持久卷声明来维护虚拟机磁盘。您需要指定动态创建持久卷声明所需的存储类。 不同存储类支持不同的卷模式。如果所选卷模式下没有可用的存储类，请联系管理员进行添加。
随虚拟机删除	若启用，则在删除虚拟机时也会删除该磁盘的数据。
是否挂载	- 暂不挂载：仅创建虚拟磁盘；需要时可后续挂载。 - 挂载到虚拟机：选择需要挂载虚拟盘的目标虚拟机。

5. 单击 创建。

挂载虚拟磁盘

将 数据盘 挂载到虚拟机，连接已创建的虚拟磁盘到目标虚拟机。

注意：虚拟机处于 运行中 状态时，可以在线挂载虚拟磁盘。

操作步骤

1. 进入 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 虚拟磁盘。

3. 在待挂载的虚拟磁盘旁单击⋮> 挂载。

4. 选择目标虚拟机并单击 挂载。

扩容虚拟磁盘

扩展已挂载到虚拟机的 系统盘 和 数据盘。

操作步骤

1. 进入 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 虚拟机。

3. 单击虚拟机名称以进入 详细信息 页面。

4. 在 虚拟磁盘 区域，找到需要扩展的磁盘，并单击⋮> 扩容。

5. 输入新容量并单击 扩容。

卸载虚拟磁盘

从虚拟机卸载 数据盘，仅支持 已停止 状态的虚拟机进行卸载。

操作步骤

1. 进入 **Container Platform**。

2. 在左侧导航栏中，单击 虚拟化 > 虚拟磁盘。

3. 在待卸载的虚拟磁盘旁单击⋮> 卸载 并确认。

删除虚拟磁盘

仅在虚拟磁盘处于卸载状态时支持删除。

注意：系统盘无法删除。

操作步骤

1. 进入 **Container Platform**。
2. 在左侧导航栏中，单击 虚拟化 > 虚拟磁盘。
3. 在待删除的虚拟磁盘旁单击⋮> 删除 并确认。

Menu

备份和恢复

介绍

介绍
应用场景
限制

操作指南

使用快照
前提条件
注意事项
创建快照
回滚快照
删除快照

介绍

ACP Virtualization With Kubevirt 提供虚拟机快照能力，通过快照可以对虚拟机进行备份和恢复。

目录

应用场景

限制

应用场景

- 灾难恢复与故障回退

  虚拟机因硬件故障、人为误操作（如误删文件）或恶意攻击（如勒索软件）导致数据丢失时，快照可作为最后一道防线恢复业务。

限制

- 创建快照需要先停止虚拟机。
- 虚拟机磁盘使用的 PVC 需要配置为多节点共享访问模式。

Menu

操作指南

使用快照

前提条件

注意事项

创建快照

回滚快照

删除快照

使用快照

虚拟机快照保存了虚拟机的当前状态，可在发生意外故障时将虚拟机恢复到该状态。

目录

前提条件
注意事项
创建快照
操作步骤
回滚快照
注意事项
操作步骤
删除快照
注意事项
操作步骤

前提条件

- 管理员已在平台管理中部署了 卷快照。
- 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。请在创建或回滚快照前，先 停止虚拟机。

注意事项

如果集群中存在多个相同类型的存储，例如多个不同来源的 Ceph RBD 存储，在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

虚拟机快照包含以下内容：虚拟机设置以及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 虚拟化 > 虚拟机。
3. 找到虚拟机，点击⋮> 创建快照。
4. 填写快照描述。描述可以帮助您记录虚拟机当前的状态，例如 `初始安装`、`应用升级前`。
5. 点击 创建。快照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。
   - 当快照变为 `就绪`，表示创建成功。
   - 如果快照长时间处于 `未就绪` 状态，请点击 > 查看原因并进行故障排除，然后重新创建快照。

回滚快照

将虚拟机的设置和支持卷快照的磁盘回滚到创建快照时的状态。例如，在创建快照后添加的磁盘将被移除；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），请与管理员确认该存储类的回收策略已设置为 保留（`reclaimPolicy: Retain`），以便正确使用快照回滚功能。

操作步骤

1. 访问 **Container Platform**。
2. 在左侧导航栏中，点击 虚拟化 > 虚拟机。
3. 点击 *虚拟机名称*。
4. 在 快照 标签页中，找到快照并点击⋮> 回滚。
5. 阅读界面上的提示信息，确认无误后点击 回滚。

   注意：回滚操作无法中止或撤销，请谨慎进行。
6. 点击快照名称，在“快照回滚记录”中查看回滚是否已完成。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

- 如果回滚失败，虚拟机状态保持不变。您可以正常启动虚拟机，或再次尝试回滚快照。
- 如果在回滚过程中启动虚拟机，虚拟机将恢复至停止前的状态，在再次停止虚拟机后将继续回滚至快照创建时的状态。
- 为避免操作冲突，请确保最近的回滚记录已完成后，再对该虚拟机执行其他操作。

删除快照

删除不再需要的虚拟机快照，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进行数据复制（例如 TopoLVM），必须等待基于回滚版本的虚拟机启动后再进行删除，否则虚拟机将无法启动。

操作步骤

1. 访问 **Container Platform**。

2. 在左侧导航栏中，点击 虚拟化 > 虚拟机。

3. 点击 *虚拟机名称*。

4. 在 快照 标签页中，找到目标快照并点击⋮> 删除。

5. 阅读提示信息，确认无误后点击 删除。