

# 什么是虚拟机 (Virtnest)

虚拟机 (Virtnest) 基于 KubeVirt 技术将虚拟机作为云原生应用进行管理，与容器无缝地衔接在一起，使用户能够轻松地部署虚拟机应用，享受与容器应用一致的丝滑体验。

以下是使用 Virtnest 的大致步骤：

1. 在集群内安装 [virtnest-agent 组件](#)。
2. [构建所需的虚拟机镜像](#)，将镜像推送到 Docker Registry 或其他镜像仓库中。
3. [使用镜像创建虚拟机或使用 YAML 创建虚拟机](#)。
4. 通过 VNC/console 方式[访问虚拟机](#)。
5. 查看虚拟机列表以及每个虚拟机的详细信息。
6. 根据需要执行开启/关机、重启、克隆、[快照](#)、恢复快照、[实时迁移](#)等操作。

## Virtnest 的优势

- 提升资源利用率：Virtnest 利用容器镜像作为虚拟机创建的基础，有效提高了资源利用率。容器化的虚拟机可以独立运行应用和配置，同时共享主机内核和硬件资源，减少资源浪费。
- 快速部署和扩展：借助 Virtnest，在 k8s 集群中快速部署和扩展虚拟机。通过镜像和容器的集成，减少了虚拟机启动时间，并实现根据需求动态调整虚拟机数量。这种灵活性提高了资源分配效率和系统性能。
- 简化管理和操作：Virtnest 简化在 k8s 集群中管理和操作虚拟机的流程。管理员可借助集群管理工具轻松管理虚拟机的启动、关机、配置和监控等任务。简化的管理过程降低了复杂性，减轻了管理负担。

- 灵活的应用部署：将虚拟机作为容器镜像部署在 k8s 集群中，为应用部署带来了极大的灵活性。 用户可以方便地将不同的应用和配置打包到容器镜像中，并利用 Virtnest 快速部署到集群中。这种灵活性使应用部署更加敏捷和灵活。
- 与容器生态系统无缝集成：Virtnest 将虚拟机与容器相结合，无缝集成到 k8s 集群中已建立的容器生态系统中。用户可以享受容器网络、服务发现、负载均衡等功能，并与其他容器化应用无缝协作。

借助 Virtnest，用户能够充分发挥 k8s 集群的潜力，高效管理虚拟化基础设施。我们致力于不断优化和完善 Virtnest 的容器化功能，为用户提供更好的虚拟机管理体验。

!!! note

虚拟机 Virtnest 是 DCE 5.0 商业版（标准版、高级版和白金版）专用的特性。

## 虚拟机功能

虚拟机模块提供虚拟机全生命周期的管理能力，虚拟机支持以下操作：

操作	描述	虚拟机状态
<a href="#"><u>创建虚拟机</u></a>	创建单台或多台虚拟机。	/
<a href="#"><u>编辑/更新虚拟机</u></a>	修改虚拟机。	运行中/已关机/处理中/错误
启动虚拟机	将处于停止状态的虚拟机启动。	已关机
关机虚拟机	将虚拟机关机。	运行中/处理中
重启虚拟机	将处于运行状态的虚拟机重启。	运行中/错误

操作	描述	虚拟机状态
	拟机重启。	
<a href="#"><u>访问控制台</u></a>	打开虚拟机控制台， 登录虚拟机系统 (vnc、终端)	运行中
修改计算规格	修改虚拟机的 CPU、 内存。支持在线/关机 修改计算规格。	运行中/已关机
<a href="#"><u>创建快照</u></a>	为虚拟机创建快照。	运行中/已关机
<a href="#"><u>克隆虚拟机</u></a>	克隆当前虚拟机。	运行中/已关机/错误
<a href="#"><u>实时迁移</u></a>	将正在运行的虚拟机 无缝迁移到其他的节 点上。	运行中
<a href="#"><u>冷迁移</u></a>	关机状态下的虚拟机 迁移到其他节点上。	已关机
数据盘扩容	为虚拟机的数据盘扩 容。	运行中/已关机/处理中
加载磁盘	为虚拟机加载可用的 数据磁盘。	运行中/已关机
卸载磁盘	将已加载的磁盘从虚 拟机卸载。	运行中/已关机
<a href="#"><u>切换网络模式</u></a>	更新虚拟机时切换网	运行中/已关机/处理中/错误

操作	描述	虚拟机状态
	络模式。	
加载网卡	为虚拟机加载网卡。	已关机
卸载网卡	将已加载的网卡从虚 拟机卸载。	已关机
<a href="#">虚拟机模板</a>	将当前虚拟机转换为 模板。	运行中/已关机
<a href="#">查看虚拟机详 情</a>	进入虚拟机详情页， 查看虚拟机基本信息、 监控、事件、快 照、配置信息。	运行中/已关机/处理中
删除虚拟机	立即释放虚拟机的 CPU、内存、IP 地址 等资源，并将虚拟机 彻底删除，请谨慎操 作。	运行中/已关机/处理中/错误

## 虚拟机 Release Notes

本页列出虚拟机的 Release Notes，便于您了解各版本的演进路径和特性变化。

**2024-11-30****v0.15.0**

- **新增** 支持半自动化沐曦 GPU
- **新增** 快照管理
- **新增** 虚拟机监控概览
- **修复** 重启虚拟机网卡 down 问题
- **修复** 带 GPU 的虚拟机创建不成功问题

**2024-10-31****v0.14.0**

- **升级** VirtNest Agent 升级为 v0.8.0
- **修复** 创建带 GPU 的虚拟机报错问题
- **修复** 虚拟机无法扩容磁盘问题

**2024-9-30****v0.13.0**

## 功能

- **新增** 支持更新虚拟机网络
- **新增** 支持集群内虚拟机冷迁移

## 修复

- **修复** 通过模板创建虚拟机时，界面系统盘为空的问题
- **修复** 通过模板创建虚拟机镜像来源为 http 时，界面显示错误问题
- **修复** 虚拟机 CPU 监控不显示的问题
- **修复** 修复 kpanda 版本高于 v0.29 时 VNC 界面无法打开的问题

## 2024-8-31

### v0.12.0

- **新增** 支持虚拟机磁盘热扩容
- **新增** 后端支持虚拟机冷迁移
- **修复** 监控 CPU 无数据问题
- **修复** 不指定节点热迁移虚拟机出错问题

## 2024-7-31

### v0.11.0

- **新增** 支持虚拟机热添加磁盘
- **修复** 模板创建虚拟机出错问题
- **修复** 快照恢复失败问题
- **修复** 快照和克隆菜单显示问题

## 2024-6-30

### v0.10.0

- **新增** 支持实时更新虚拟机的内存、CPU
- **新增** 支持指定节点实时迁移虚拟机
- **新增** 支持更新虚拟机的 GPU 信息
- **优化** KubeVirt 组件升级到 1.2.2
- **修复** 通过内置模板创建的虚拟机开机报错问题

## 2024-5-30

### v0.9.0

- **新增** 虚拟机模块支持在 Helm 应用界面中直接更新版本
- **修复** 在通过 YAML 配置创建虚拟机时，由于用户信息解析错误导致创建失败的问题
- **修复** 虚拟机资源监控计算异常问题

## 2024-4-30

### v0.8.1

- **新增** 虚拟机和虚拟机模板的详情内的 GPU 信息
- **优化** 虚拟机列表性能
- **修复** 虚拟机 bridge 网络模式的使用问题

**2024-3-26****v0.7.0**

- **新增** 虚拟机支持 GPU 配置
- **新增** OpenAPI 文档放至文档站
- **新增** 对接审计日志
- **优化** 通过虚拟机模板创建虚拟机时，继承模板内的网络和存储信息
- **修复** 虚拟机监控内存使用率值不准确的问题

**2024-1-26****v0.6.0**

## 功能

- **新增** 支持将 VMware 中的虚拟机导入到 DCE 5.0 的虚拟机模块中使用
- **新增** 虚拟机网络增加 IPv6 池
- **新增** 模板创建虚拟机时支持配置网络信息

## 优化

- **优化** 虚拟机详情内的监控信息
- **优化** 创建虚拟机配置存储的提示信息

## 修复

- **修复** 虚拟机处于关机状态时网络配置显示信息错误的问题

- **修复** 创建虚拟机提示中网络配置创建 Multus CR 跳转位置有误
- **修复** 无法更新关机状态的虚拟机

## 2023-12-26

### v0.5.0

- **新增** 支持虚拟机更丰富的网络配置，例如多网卡能力
- **新增** 虚拟机详情内的监控信息
- **优化** 虚拟机列表响应速度慢的问题
- **优化** 虚拟机模板列表的排序问题
- **修复** kubevirt client 可能获取失败的问题

## 2023-11-30

### v0.4.0

- **升级** KubeVirt 到 1.0.0
- **优化** 虚拟机详情内容，增加用户名、密码、cpu、内存信息
- **修复** 仪表盘偶尔看不到虚拟机监控数据的问题

## 2023-10-31

### v0.3.0

- **新增** 支持实时迁移
- **新增** 支持通过终端访问虚拟机

- **新增** 支持 YAML 编辑虚拟机
- **新增** 支持为虚拟机增加数据盘
- **新增** 支持从镜像仓库中选择镜像创建虚拟机
- **新增** 支持使用模板创建虚拟机
- **新增** 支持虚拟机配置转化为模板
- **新增** 支持删除自定义模板
- **新增** 支持自定义操作系统
- **优化** 虚拟机列表排序
- **修复** VNC 无法访问问题

**2023-8-31**

**v0.2.0**

- **新增** 支持重启虚拟机
- **新增** 支持表单编辑虚拟机
- **新增** 支持 YAML 创建虚拟机
- **新增** 支持克隆虚拟机
- **新增** 支持为虚拟机创建快照
- **新增** 支持从快照恢复虚拟机，并展示恢复记录

**2023-7-31**

**v0.1.0**

- **新增** 支持通过集群展示虚拟机列表

- **新增** 支持通过容器镜像创建虚拟机
- **新增** 支持关机/启动虚拟机
- **新增** 支持删除虚拟机
- **新增** 支持通过控制台访问（VNC）虚拟机
- **新增** 支持查看虚拟机详情

## 安装虚拟机模块

本页说明如何安装虚拟机模块。

!!! info

下述命令或脚本内出现的 `_virtnest_` 字样是虚拟机模块的内部开发代号。

### 配置 `virtnest` helm 仓库

```
helm-charts 仓 库 地 址 : <https://release.daocloud.io/harbor/projects/10/helm-charts/virtnest/versions>
helm repo add virtnest-release https://release.daocloud.io/chartrepo/virtnest
helm repo update virtnest-release
```

如果您想体验最新开发版的 `virtnest`, 那么请添加如下仓库地址 (开发版本的 `virtnest` 极其不稳定)

```
helm repo add virtnest-release-ci https://release-ci.daocloud.io/chartrepo/virtnest
helm repo update virtnest-release-ci
```

### 选择您想安装的 `virtnest` 版本

建议安装最新版本。

```
[root@master ~]# helm search repo virtnest-release/virtnest --versions
NAME          CHART VERSION APP VERSION DESCRIPTION
virtnest-release/virtnest 0.6.0        v0.6.0      A Helm chart for virtnest
```

## 创建 namespace

```
kubectl create namespace virtnest-system
```

## 执行安装步骤

```
helm install virtnest virtnest-release/virtnest -n virtnest-system --version 0.6.0
```

## 升级

## 更新 virtnest helm 仓库

```
helm repo update virtnest-release
```

## 备份 -set 参数

在升级 virtnest 版本之前，我们建议您执行如下命令，备份上一个版本的 -set 参数

```
helm get values virtnest -n virtnest-system -o yaml > bak.yaml
```

## 执行 helm upgrade

```
helm upgrade virtnest virtnest-release/virtnest \
-n virtnest-system \
-f ./bak.yaml \
--version 0.6.0
```

## 卸载

```
helm delete virtnest -n virtnest-system
```

# 安装依赖和前提条件

本页说明安装虚拟机模块的依赖和前提条件。

!!! info

下述命令或脚本内出现的 `_virtnest_` 字样是全局管理模块的内部开发代号。

## 前提条件

### 操作系统内核版本需要在 4.11 以上

目标集群所有节点的操作系统内核版本需要大于 4.11 (详见 [kubevirt issue](#))。运行以下命令查看内核版本：

```
uname -a
```

示例输出：

```
Linux master 6.5.3-1.el7.elrepo.x86_64 #1 SMP PREEMPT_DYNAMIC Wed Sep 13 11:46:28  
EDT 2023 x86_64 x86_64 x86_64 GNU/Linux
```

### CPU 需支持 x86-64-v2 及以上的指令集

使用以下脚本检查当前节点的 CPU 是否支持：

!!! note

若出现与输出信息无关的报错（如下所示），可无需关注，不影响最终结果。

```
```bash title="示例"
$ sh detect-cpu.sh
detect-cpu.sh: line 3: fpu: command not found
```
cat <<EOF > detect-cpu.sh
#!/bin/sh -eu

flags=$(cat /proc/cpuinfo | grep flags | head -n 1 | cut -d: -f2)

supports_v2='awk "/cx16/&/lahf/&/popcnt/&/sse4_1/&/sse4_2/&/ssse3/ {found=1} END
{exit !found}"'
supports_v3='awk "/avx/&/avx2/&/bmi1/&/bmi2/&/f16c/&/fma/&/abm/&/movbe/&/x
save/ {found=1} END {exit !found}"'
supports_v4='awk "/avx512f/&/avx512bw/&/avx512cd/&/avx512dq/&/avx512vl/ {found=1}
END {exit !found}"'

echo "$flags" | eval $supports_v2 || exit 2 && echo "CPU supports x86-64-v2"
echo "$flags" | eval $supports_v3 || exit 3 && echo "CPU supports x86-64-v3"
echo "$flags" | eval $supports_v4 || exit 4 && echo "CPU supports x86-64-v4"
EOF
```

```
chmod +x detect-cpu.sh
sh detect-cpu.sh
```

## 所有节点必须启用硬件虚拟化（嵌套虚拟化）

- 运行以下命令检查：

```
virt-host-validate qemu
# 成功的情况
QEMU: Checking for hardware virtualization : PA
SS
QEMU: Checking if device /dev/kvm exists : P
ASS
QEMU: Checking if device /dev/kvm is accessible : PA
SS
QEMU: Checking if device /dev/vhost-net exists : PAS
S
QEMU: Checking if device /dev/net/tun exists : PA
SS
QEMU: Checking for cgroup 'cpu' controller support : PAS
S
QEMU: Checking for cgroup 'cpacct' controller support : PAS
S
QEMU: Checking for cgroup 'cpuset' controller support : PAS
S
QEMU: Checking for cgroup 'memory' controller support : PA
SS
QEMU: Checking for cgroup 'devices' controller support : PASS
QEMU: Checking for cgroup 'blkio' controller support : PAS
S
QEMU: Checking for device assignment IOMMU support : 
PASS
QEMU: Checking if IOMMU is enabled by kernel : 
PASS
QEMU: Checking for secure guest support : 
WARN (Unknown if this platform has Secure Guest support)
```

# 失败的情况

```
QEMU: Checking for hardware virtualization : FA
IL (Only emulated CPUs are available, performance will be significantly limited)
QEMU: Checking if device /dev/vhost-net exists : PAS
S
QEMU: Checking if device /dev/net/tun exists : PA
SS
```

```

QEMU: Checking for cgroup 'memory' controller support : PA
SS

QEMU: Checking for cgroup 'memory' controller mount-point : PA
SS

QEMU: Checking for cgroup 'cpu' controller support : PAS
S

QEMU: Checking for cgroup 'cpu' controller mount-point : PAS
S

QEMU: Checking for cgroup 'cpuacct' controller support : PAS
S

QEMU: Checking for cgroup 'cpuacct' controller mount-point : PASS
QEMU: Checking for cgroup 'cpuset' controller support : PAS
S

QEMU: Checking for cgroup 'cpuset' controller mount-point : PASS
QEMU: Checking for cgroup 'devices' controller support : PASS
QEMU: Checking for cgroup 'devices' controller mount-point : PASS
QEMU: Checking for cgroup 'blkio' controller support : PAS
S

QEMU: Checking for cgroup 'blkio' controller mount-point : PASS
WARN (Unknown if this platform has IOMMU support)

```

- 安装 virt-host-validate：

==== “在 CentOS 上安装”

```

```bash
yum install -y qemu-kvm libvirt virt-install bridge-utils
```

```

==== “在 Ubuntu 上安装”

```

```bash
apt install qemu-kvm libvirt-daemon-system libvirt-clients bridge-utils
```

```

- 硬件虚拟化启用方法：

不同平台启用硬件虚拟化的方法也不一样，以 vsphere 为例，方法请参照 [vmware 官网文档](#)。

## 如果使用 Docker Engine 作为容器运行时

如果集群使用 Docker Engine 作为容器运行时，则 Docker Engine 版本需要大于 20.10.10

## 建议开启 IOMMU

为了后续功能做准备，建议开启 IOMMU。

# 离线升级

本页说明从[下载中心](#)下载虚拟机模块后，应该如何安装或升级。

!!! info

下述命令或脚本内出现的 \_\_virtnest\_\_ 字样是虚拟机模块的内部开发代号。

## 从安装包中加载镜像

您可以根据下面两种方式之一加载镜像，当环境中存在镜像仓库时，建议选择 chart-syncer 同步镜像到镜像仓库，该方法更加高效便捷。

### chart-syncer 同步镜像到镜像仓库

#### 1. 创建 load-image.yaml

!!! note

该 YAML 文件中的各项参数均为必填项。您需要一个私有的镜像仓库，并修改相关配置。

==== “已安装 chart repo”

若当前环境已安装 chart repo，chart-syncer 也支持将 chart 导出为 tgz 文件。

```
```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: virtnest-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/virtnest # 需更改为你的镜像仓库
repo:
  kind: HARBOR # 也可以是任何其他支持的 Helm Chart 仓库类别
  url: http://10.16.10.111/chartrepo/release.daocloud.io # 需更改为 chart repo url
```

```

auth:
  username: "admin" # 你的镜像仓库用户名
  password: "Harbor12345" # 你的镜像仓库密码
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

**== “未安装 chart repo”**

若当前环境未安装 chart repo, chart-syncer 也支持将 chart 导出为 tgz 文件，并存放在指定路径。

```

```yaml title="load-image.yaml"
source:
  intermediateBundlesPath: virtnest-offline # 到执行 charts-syncer 命令的相对路径,
而不是此 YAML 文件和离线包之间的相对路径
target:
  containerRegistry: 10.16.10.111 # 需更改为你的镜像仓库 url
  containerRepository: release.daocloud.io/virtnest # 需更改为你的镜像仓库
repo:
  kind: LOCAL
  path: ./local-repo # chart 本地路径
containers:
  auth:
    username: "admin" # 你的镜像仓库用户名
    password: "Harbor12345" # 你的镜像仓库密码
```

```

## 2. 执行同步镜像命令。

```
charts-syncer sync --config load-image.yaml
```

## Docker 或 containerd 直接加载

解压并加载镜像文件。

### 1. 解压 tar 压缩包。

```
tar xvf virtnest.bundle.tar
```

解压成功后会得到 3 个文件：

- hints.yaml
- images.tar
- original-chart

## 2. 从本地加载镜像到 Docker 或 containerd。

```
==== "Docker"
```shell
docker load -i images.tar
```
==== "containerd"
```shell
ctr -n k8s.io image import images.tar
```

```

### !!! note

每个 node 都需要做 Docker 或 containerd 加载镜像操作，  
加载完成后需要 tag 镜像，保持 Registry、Repository 与安装时一致。

## 升级

有两种升级方式。您可以根据前置操作，选择对应的升级方案：

### ==== “通过 helm repo 升级”

1. 检查虚拟机 helm 仓库是否存在。

```
```shell
helm repo list | grep virtnest
```

```

若返回结果为空或如下提示，则进行下一步；反之则跳过下一步。

```
```none
Error: no repositories to show
```

```

1. 添加虚拟机的 helm 仓库。

```
```shell
helm repo add virtnest http://{harbor url}/chartrepo/{project}
```

```

1. 更新虚拟机的 helm 仓库。

```
```shell
helm repo update virtnest # (1)
```

```

1. helm 版本过低会导致失败，若失败，请尝试执行 helm update repo
1. 选择您想安装的虚拟机版本（建议安装最新版本）。

```
```shell
helm search repo virtnest/virtnest --versions
```
```
```none
[root@master ~]# helm search repo virtnest/virtnest --versions
NAME          CHART VERSION APP VERSION DESCRIPTION
virtnest/virtnest  0.2.0      v0.2.0       A Helm chart for virtnest
...
```
```

```

1. 备份 `--set` 参数。

在升级虚拟机版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values virtnest -n virtnest-system -o yaml > bak.yaml
```
```

```

1. 更新 virtnest crds

```
```shell
helm pull virtnest/virtnest --version 0.2.0 && tar -zxf virtnest-0.2.0.tgz
kubectl apply -f virtnest/crds
```
```

```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 字段为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
```
```
```shell
helm upgrade virtnest virtnest/virtnest \
-n virtnest-system \
-f ./bak.yaml \
--set global.imageRegistry=$imageRegistry \
```
```

```

```
--version 0.2.0
````
```

==== “通过 chart 包升级”

1. 备份 `--set` 参数。

在升级虚拟机版本之前，建议您执行如下命令，备份老版本的 `--set` 参数。

```
```shell
helm get values virtnest -n virtnest-system -o yaml > bak.yaml
````
```

1. 更新 virtnest crds

```
```shell
kubectl apply -f ./crds
````
```

1. 执行 `helm upgrade`。

升级前建议您覆盖 bak.yaml 中的 `global.imageRegistry` 为当前使用的镜像仓库地址。

```
```shell
export imageRegistry={你的镜像仓库}
````

```shell
helm upgrade virtnest . \
-n virtnest-system \
-f ./bak.yaml \
--set global.imageRegistry=$imageRegistry
````
```

## 安装 virtnest-agent

本文将介绍如何在指定集群内安装 virtnest-agent。

### 前提条件

安装 virtnest-agent 之前，需要满足以下前提条件：

- 操作系统内核版本需要在 v4.11 以上。

## 安装步骤

初始集群需要在 Helm 中安装 virtnest-agent 组件后方可使用虚拟机的相关能力。

- 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，若未安装 virtnest-agent 组件，则无法正常使用虚拟机能力。将提醒用户在所需集群内进行安装。

安装提示

安装提示

- 选择所需集群，点击左侧导航栏的 **Helm 应用**，然后点击 **Helm 模板**，查看模板列表。

helm 模板

helm 模板

- 搜索 **virtnest-agent** 组件，进入组件详情，选择合适版本，点击 **安装** 按钮，进行安装。

virtnest-agent 组件

virtnest-agent 组件

详情

详情

- 进入安装表单页面，填写基本信息后，点击 **确定**，安装完成。

安装信息

安装信息

- 重新点击 **虚拟机** 导航栏，成功出现虚拟机列表，可以正常使用虚拟机能力。

## 虚拟机列表

### 虚拟机列表

# 创建虚拟机

本文将介绍如何通过镜像和 YAML 文件两种方式创建虚拟机。

虚拟机基于 KubeVirt 技术将虚拟机作为云原生应用进行管理，与容器无缝地衔接在一起，使用户能够轻松地部署虚拟机应用，享受与容器应用一致的丝滑体验。

## 前提条件

创建虚拟机之前，需要满足以下前提条件：

- 向用户机操作系统公开硬件辅助的虚拟化。
- 在指定集群[安装 virtnest-agent](#)，操作系统内核版本需要在 3.15 以上。
- 创建一个[命名空间](#)和[用户](#)。
- 提前准备好镜像，平台内置三种镜像（如下文所示），如需制作镜像，可参考开源项目[制作镜像](#)。

## 镜像创建

参考以下步骤，使用镜像创建一个虚拟机。

1. 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，进入 **虚拟机** 页面。

| 名称             | 状态  | 命名空间    | 操作系统   | 节点                | IP            | CPU    | 内存   |
|----------------|-----|---------|--------|-------------------|---------------|--------|------|
| backup-1       | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| backup         | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| test-mg        | 运行中 | default | Ubuntu | controller-node-2 | 10.233.76.181 | 1 Core | 2 Gi |
| stop-vm        | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| test-migration | 运行中 | default | Ubuntu | controller-node-3 | 10.233.83.234 | 1 Core | 2 Gi |
| migration      | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| test           | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |

## 虚拟机

2. 点击 **创建虚拟机** -> 选择 **通过镜像创建**
3. 依次填写基本信息、镜像配置、存储与网络、登录设置后，在页面右下角点击 **确定** 完成创建。

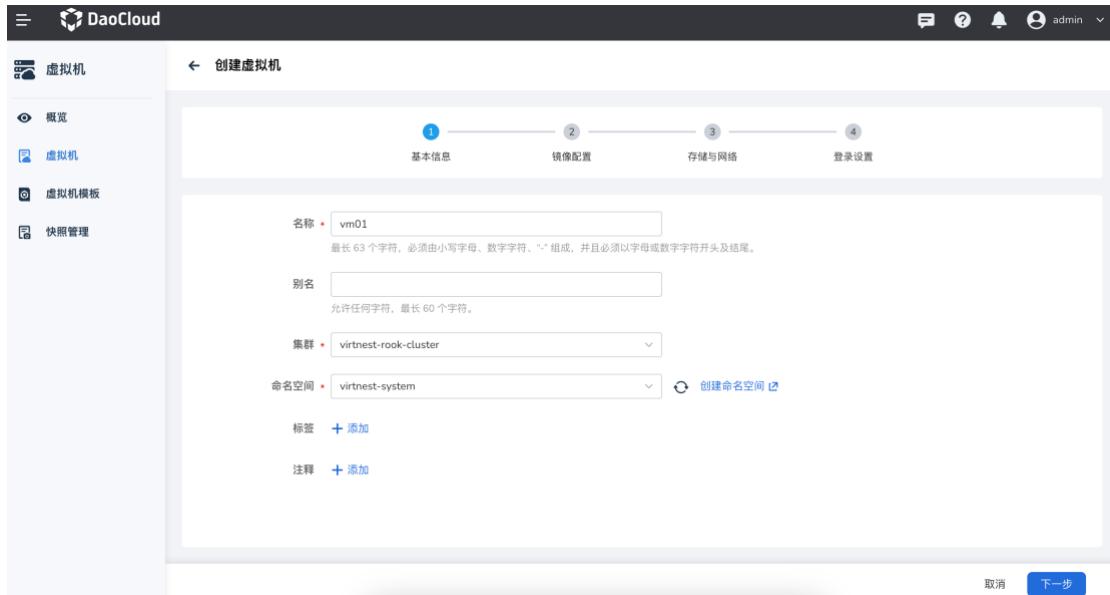
系统将自动返回虚拟机列表。点击列表右侧的 ，可以对虚拟机执行关机/开启、重启、克隆、更新、创建快照、控制台访问 (VNC)、删除等操作。克隆和快照能力依赖于存储池的选择。

| 名称             | 状态  | 命名空间    | 操作系统   | 节点                | IP            | CPU    | 内存   |
|----------------|-----|---------|--------|-------------------|---------------|--------|------|
| backup-1       | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| backup         | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| test-mg        | 运行中 | default | Ubuntu | controller-node-2 | 10.233.76.181 | 1 Core | 2 Gi |
| stop-vm        | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| test-migration | 运行中 | default | Ubuntu | controller-node-3 | 10.233.83.234 | 1 Core | 2 Gi |
| migration      | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |
| test           | 关机  | default | Ubuntu | -                 | -             | 1 Core | 2 Gi |

## 虚拟机操作

## 基本信息

在 **创建虚拟机** 页面中，根据下表输入信息后，点击 **下一步**。



### 基础信息

- 名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾。同一命名空间内名称不得重复，而且名称在虚拟机创建好之后不可更改。
- 别名：允许任何字符，最长 60 个字符。
- 集群：选择将新建的虚拟机部署在哪个集群内，若有使用 GPU 能力的需求，则需要选择有 GPU/vGPU 卡的集群。
- 命名空间：选择将新建的虚拟机部署在哪个命名空间。找不到所需的命名空间时可以根据页面提示去[创建新的命名空间](#)。
- 标签/注解：选择为虚拟机添加所需的标签/注解信息。

## 镜像配置

根据下表填写镜像相关信息后，点击 **下一步**



### 使用镜像仓库

- 镜像来源：支持三种类型的来源。
  - 镜像仓库类型：镜像存储在容器镜像仓库中，支持从镜像仓库中按需选择镜像；
  - HTTP 类型：镜像存储于 HTTP 协议的文件服务器中，支持 HTTPS:// 和 HTTP:// 前缀；
  - 对象存储 (S3)：支持通过对象存储协议 (S3) 获取的虚拟机镜像，若是无需认证的对象存储文件，请使用 HTTP 来源。
- 以下是平台内置的镜像信息，包括操作系统和版本、镜像地址。同时也支持自定义虚拟机镜像。

操作 对应版本 镜像地址

## 系统

|            |                 |                                                                       |
|------------|-----------------|-----------------------------------------------------------------------|
| CentO<br>S | CentOS<br>7.9   | release-ci.daocloud.io/virtnest/system-images/centos-7.9-x86_64: v1   |
| Ubunt<br>u | Ubuntu<br>22.04 | release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_64: v1 |
| Debia<br>n | Debian 12       | release-ci.daocloud.io/virtnest/system-images/debian-12-x86_64: v1    |

- 镜像密钥：仅支持默认（Opaque）类型密钥，具体格式请参考[创建密钥](#)。

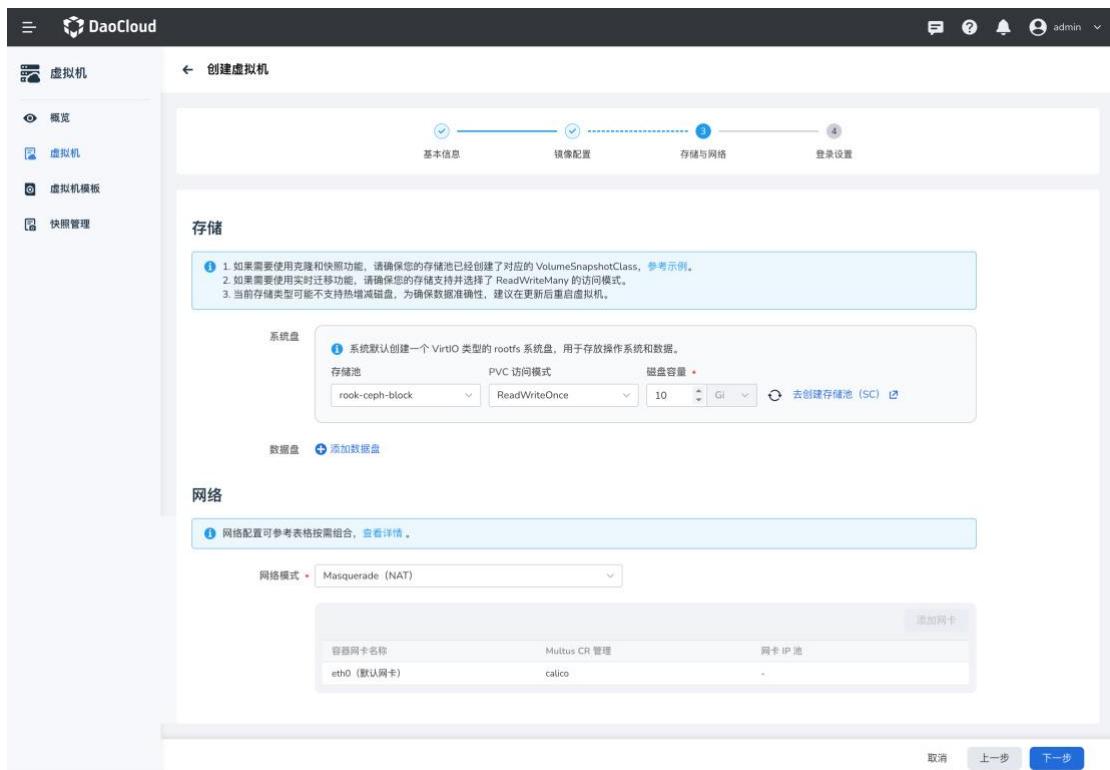
平台内置镜像存储在点火集群中，而点火集群的镜像仓库未加密，因此当选择内置镜像时，无需选择密钥。

### !!! note

CPU 和内存的热加载配置要求：virtnest 的版本不低于 v0.10.0，并且 virtnest-agent 版本不低于 v0.7.0；支持实时迁移（确保 PVC 访问模式为 ReadWriteMany）；带有 GPU 卡的虚拟机暂时无法实现 CPU 和内存的热加载能力。

- 资源配置：CPU 建议使用整数，若填写小数则会向上取整。支持 CPU、内存的热加载。
- GPU 配置：启用 GPU 功能需要满足前提条件，具体可参考[虚拟机配置 GPU \(Nvidia\)](#)。虚拟机支持 Nvidia—GPU 和 Nvidia—vGPU 两种类型，选择所需类型后，需要选择对应的 GPU 型号和卡的数量。

## 存储与网络配置



### 存储与网络配置

- 存储：
  - 存储和虚拟机的功能息息相关，主要是通过使用 Kubernetes 的持久卷和存储类，提供了灵活且可扩展的虚拟机存储能力。比如虚拟机镜像存储在 pvc 里，支持和其他数据一起克隆、快照等。
  - 系统盘：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：数据盘是虚拟机中用于存储用户数据、应用程序数据或其他非操作系统的相关文件的存储设备。与系统盘相比，数据盘是非必选的，可以根据需要动态添加或移除。数据盘的容量也可以根据需求进行灵活配置。
  - 默认使用块存储。如果需要使用克隆和快照功能，请确保您的存储池已经创

建了对应的 VolumeSnapshotClass，可以参考以下示例。如果需要使用实时迁移功能，请确保您的存储支持并选择了 ReadWriteMany 的访问模式。

大多数情况下，存储在安装过程中不会自动创建这样的 VolumeSnapshotClass，因此您需要手动创建 VolumeSnapshotClass。以下是[一个 HwameiStor 创建 VolumeSnapshotClass 的示例](#)：

```
```yaml
kind: VolumeSnapshotClass
apiVersion: snapshot.storage.k8s.io/v1
metadata:
  name: hwameistor-storage-lvm-snapshot
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true"
parameters:
  snapsize: "1073741824"
  driver: lvm.hwameistor.io
  deletionPolicy: Delete
```

```

- 执行以下命令检查 VolumeSnapshotClass 是否创建成功。  
kubectl get VolumeSnapshotClass
- 查看已创建的 Snapshotclass，并且确认 provisioner 属性同存储池中的 Driver 属性一致。

- 网络：

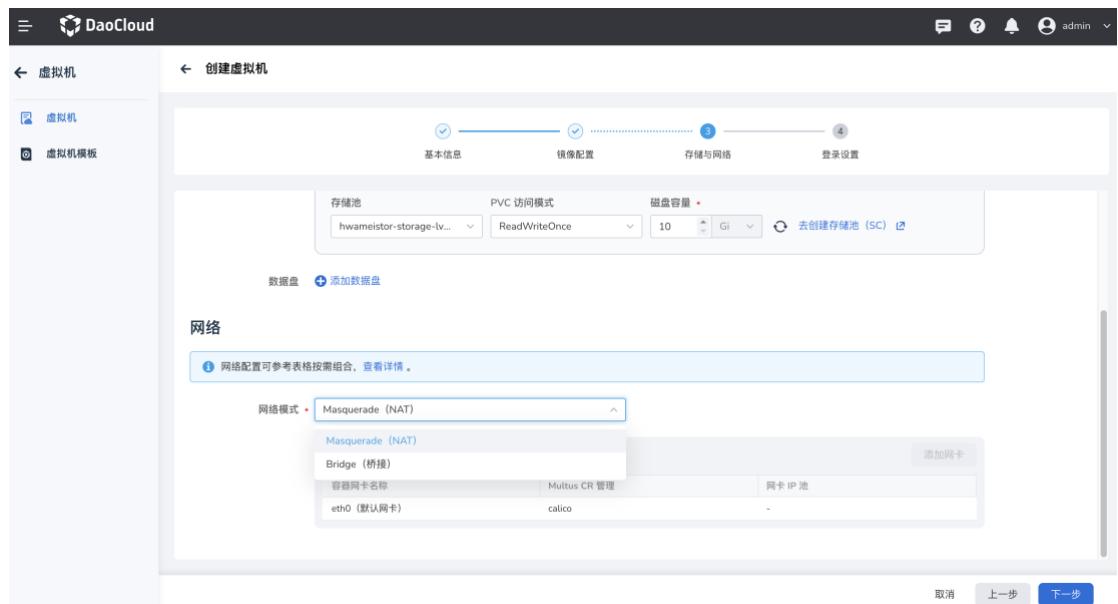
- 网络配置可以根据表格信息按需组合。

| 网络模式                | CNI    | 是否安装       | 网卡 | 固定    | 实时迁 |
|---------------------|--------|------------|----|-------|-----|
|                     |        | Spiderpool |    | IP 模式 | 移   |
| Masquerade<br>(NAT) | Calico | ✗          |    | ✗     | ✓   |
|                     | Cilium | ✗          |    | ✗     | ✓   |

| 网络模式       | CNI | 是否安装 | 网卡  | 固定       | 实时迁移 |
|------------|-----|------|-----|----------|------|
| Spiderpool |     |      |     | IP<br>模式 | 移    |
| Flannel    |     | ×    | 单网卡 | ×        | √    |

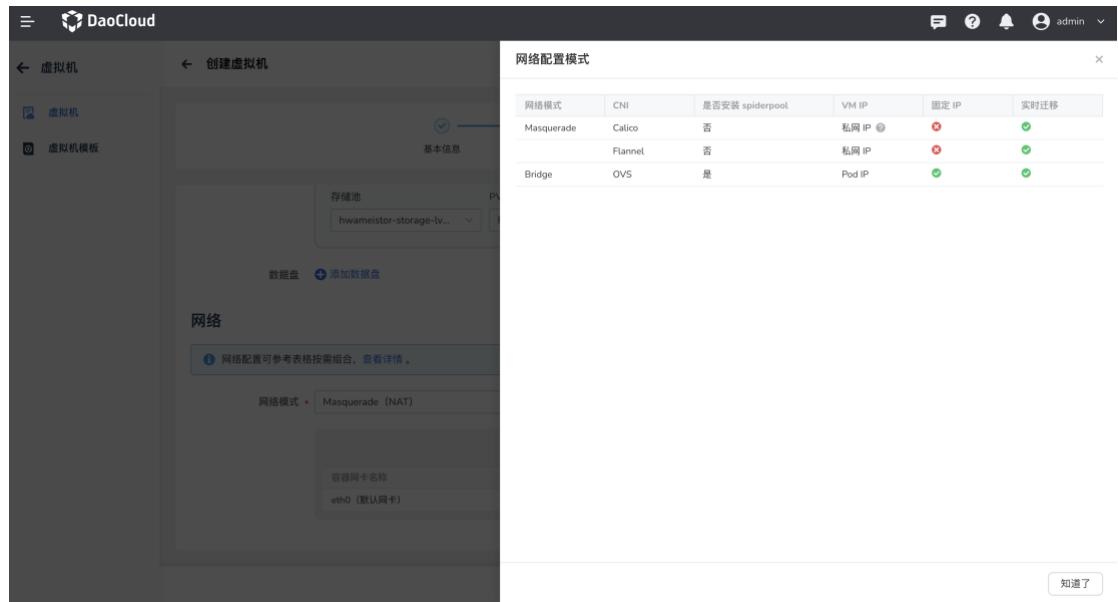
  

| Bridge (桥接) | OVS | √ | 多网卡 | √ | √ |
|-------------|-----|---|-----|---|---|
|             |     |   | 卡   |   |   |



## 网络配置

- 网络模式分为 Masquerade (NAT) 和 Bridge (桥接) , Bridge (桥接) 模式需要安装了 Spiderpool 组件后方可使用。
  - 默认选择 Masquerade (NAT) 的网络模式，使用 eth0 默认网卡。
  - 若集群内安装了 spiderpool 组件，则支持选择 Bridge (桥接) 模式，Bridge (桥接) 模式支持多网卡形式。



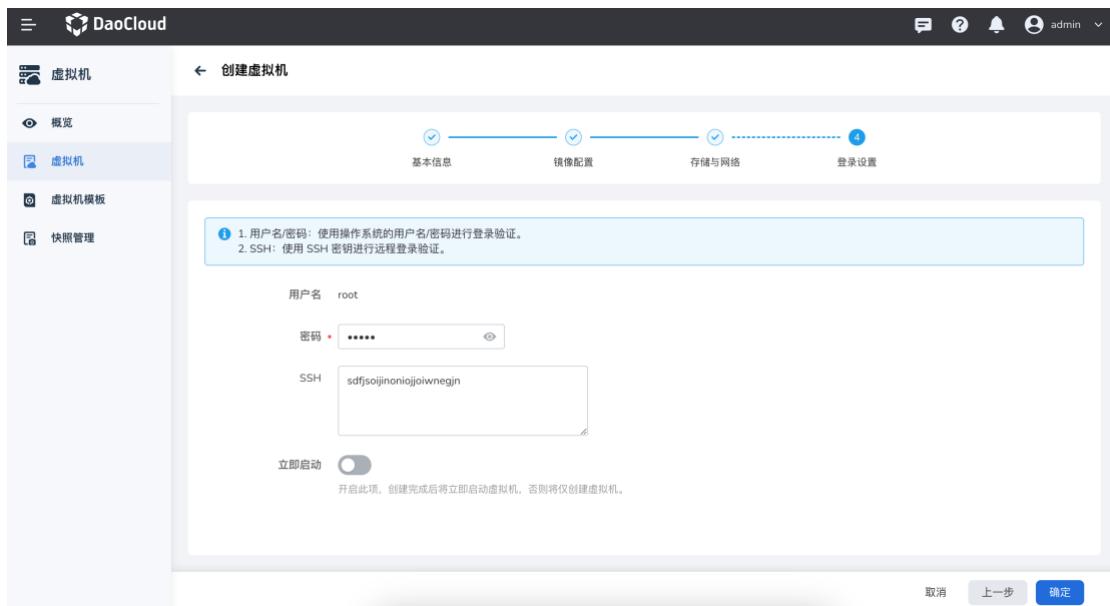
## 网络模式

- 添加网卡

- Bridge (桥接) 模式下支持手动添加网卡。点击 **添加网卡**，进行网卡 IP 池的配置。选择和网络模式匹配的 Multus CR，若没有则需要自行创建。
- 若打开 **使用默认 IP 池** 开关，则使用 multus CR 配置中的默认 IP 池。若关闭开关，则手动选择 IP 池。

## 登录设置

- 用户名/密码：可以通过用户名和密码登录至虚拟机。
- SSH：选择 SSH 登录方式时可为虚拟机绑定 SSH 密钥，用于日后登录虚拟机。



## 登录设置

## YAML 创建

除了通过镜像方式外，还可以通过 YAML 文件更快速地创建虚拟机。

进入虚拟机列表页，点击 **通过 YAML 创建** 按钮。

### yaml 创建

#### yaml 创建

??? note “[点击查看创建虚拟机的 YAML 示例](#)”

```
```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: demo
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        name: systemdisk-demo
        namespace: default
  spec:
    pvc:
      accessModes:
```

```
- ReadWriteOnce
resources:
requests:
storage: 10Gi
storageClassName: hwameistor-storage-lvm-hdd
source:
registry:
url: >-
docker://release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_64:v1
runStrategy: Always
template:
spec:
architecture: amd64
domain:
cpu:
cores: 1
sockets: 1
threads: 1
devices:
disks:
- bootOrder: 1
disk:
bus: virtio
name: systemdisk-demo
- disk:
bus: virtio
name: cloudinitdisk
interfaces:
- masquerade: {}
name: default
machine:
type: q35
resources:
requests:
memory: 2Gi
networks:
- name: default
pod: {}
volumes:
- dataVolume:
name: systemdisk-demo
name: systemdisk-demo
- cloudInitNoCloud:
userDataBase64: >-
```

```
I2Nsb3VkLWNvbmZpZwpzc2hfcHdhdXRoOiB0cnVlCmRpc2FibGVfcm9vdDogZ  
mFsc2UKY2hwYXNzd2Q6IHsibGlzdCI6ICJyb290OjEyMzQ1NiIsIGV4cGlyZTogRmFsc2V9CgoKc  
nVuY21kOgogIC0gc2VkIC1pIC1vI1w/UGVybWl0Um9vdExvZ2luL3MvXi4qJC9QZXJtaXRSb290T  
G9naW4geWVzL2ciIC9ldGMvc3NoL3NzaGRfY29uZmlnCiAgLSBzeXN0ZW1jdGwgcmVzdGFydC  
Bzc2guc2VydmljZQ==  
name: cloudinitdisk  
...  
...
```

## 通过模板创建

自定义模板是由虚拟机配置转化而来的模板。 参见[通过模板创建虚拟机](#)。

## 更新虚拟机

本文将介绍如何通过表单和 YAML 文件两种方式更新虚拟机。

### 前提条件

开机状态下更新虚拟机 CPU、内存、数据盘之前，需要满足以下前提条件：

- 虚拟机支持实时迁移能力。

### 表单更新虚拟机

在虚拟机列表页面，点击 **更新** 进入虚拟机更新页面。

名称	状态	命名空间	操作系统	节点	IP	CPU	内存
lulu1-clone	关机	default	Debian	-	-	1 Core	2 Gi
peter-slurm-3	运行中	default	Ubuntu	virtnest-master-01	10.6.202.158	2 Core	4 Gi
peter-slurm-2	运行中	default	Ubuntu	virtnest-master-01	10.6.202.91	2 Core	
peter-slurm-1	运行中	default	Ubuntu	virtnest-master-01	10.6.202.123	2 Core	
import-v2v	处理中	default	-	-	-	不限制	
export-vmware-peter	运行中	default	-	virtnest-master-01	10.233.95.47	1 Core	
eee	运行中	ghippo-team	Ubuntu	virtnest-master-03	10.6.202.74	4 Core	
abc	运行中	ghippo-team	Ubuntu	virtnest-master-03	10.6.202.111	6 Core	
sfafsd	关机	amamba-team	Debian	-	-	1 Core	
akjnzkhadsfja	关机	amamba-team	Debian	-	-	1 Core	

更新

### == “基本信息”

基本信息页面中，别名与标签注解支持更新，其他信息无法更改。完成更新后点击下一步进入镜像配置的界面。

![更新基本信息](docs/zh/docs/virtnest/images/edit02.png)

### == “镜像配置”

在镜像配置页面中，镜像来源、操作系统、版本等参数一旦选择后无法更改，允许用户更新GPU 配置，

包括启用或禁用 GPU 支持，选择 GPU 的类型，指定所需的型号，以及配置 GPU 卡的数量，更新后需要重启才能生效。

完成更新后点击下一步进入存储与网络的界面。

![更新镜像配置](docs/zh/docs/virtnest/images/edit03.png)

### == “存储与网络”

在存储与网络页面中，系统盘的存储池和 PVC 访问模式一旦选择后无法更改，支持增加磁盘容量，不可减少。

此外，用户可以自由添加或者移除数据盘。不支持更新网络配置。完成更新后点击下一步进入登录设置的界面。

### !!! note

建议在修改存储容量或增加数据盘后重启虚拟机，以确保配置生效。

![存储](docs/zh/docs/virtnest/images/edit04.png)

![网络](docs/zh/docs/virtnest/images/edit05.png)

### ==== “登录设置”

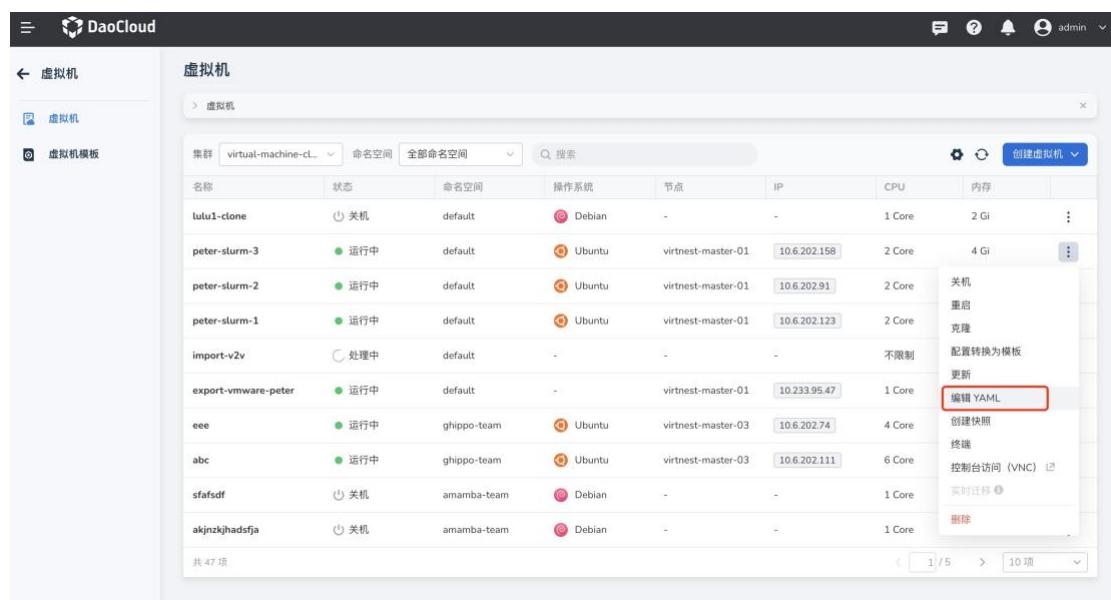
在登录设置页面中，用户名、密码以及 SSH 密钥配置一旦设置，不允许更改。确认您的登录信息无误后，点击确定按钮以完成更新流程。

![登录配置](docs/zh/docs/virtnest/images/edit06.png)

## 编辑 YAML

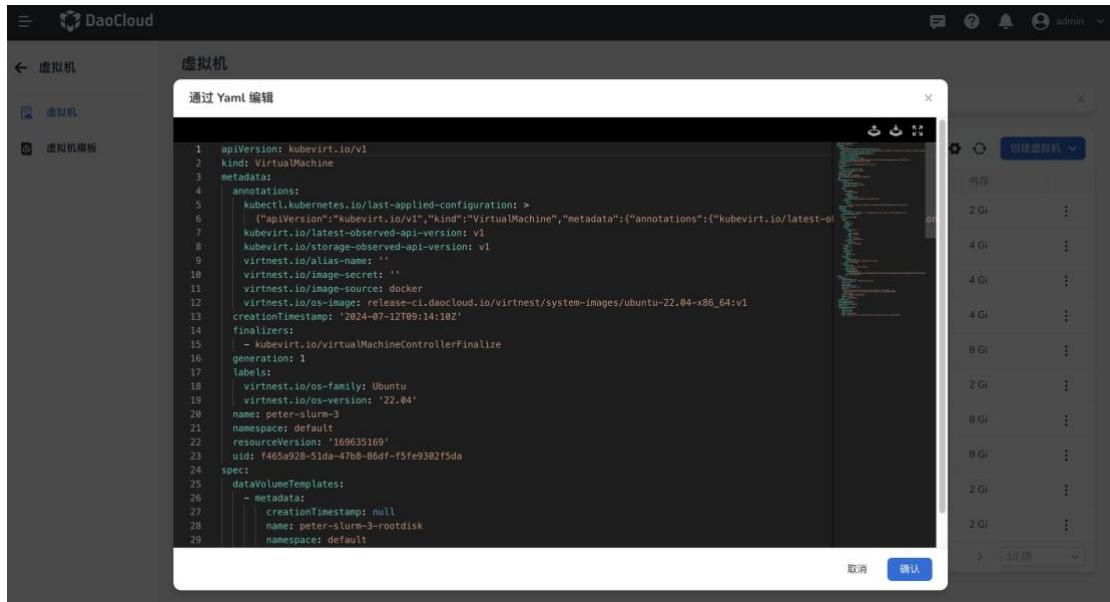
除了通过表单方式更新虚拟机外，还可以通过 YAML 文件更快速地更新虚拟机。

进入虚拟机列表页，点击 编辑 YAML 按钮。



The screenshot shows the DaoCloud Enterprise 5.0 web interface. On the left, there's a sidebar with 'Virtual Machine' selected. The main area is titled 'Virtual Machine' and lists several VMs. A context menu is open over the second row from the top. The menu items are: 关机 (Power Off), 重启 (Restart), 克隆 (Clone), 配置转换为模板 (Convert Configuration to Template), 更新 (Update), 编辑 YAML (Edit YAML, highlighted with a red box), 创建快照 (Create Snapshot), 终端 (Terminal), 控制台访问 (VNC), 实时迁移 (Live Migration), and 删除 (Delete). At the bottom of the list, it says '共 47 项' (Total 47 items).

yaml 编辑



编辑 yaml

## 连接虚拟机

本文将介绍两种连接虚拟机的方式，分别为 控制台访问（VNC）和终端方式。

### 终端

通过终端访问虚拟机的方式更加灵活和轻量，但是无法直接展示图形界面，交互性较差，且无法多终端同时在线。

从左侧导航栏，依次点击 **虚拟机** -> **虚拟机** 进入虚拟机列表页面。如果虚拟机状态为 **运行中**，点击列表右侧的 **⋮**，可以通过 **终端** 访问虚拟机。

名称	状态	命名空间	操作系统	节点	IP	CPU
backup-1	关机	default	Ubuntu	-	-	1 Core
backup	关机	default	Ubuntu	-	-	1 Core
test-mg	运行中	default	Ubuntu	controller-node-2	10.233.76.181	1 Core
stop-vm	关机	default	Ubuntu	-	-	1 Core
test-migration	运行中	default	Ubuntu	controller-node-3	10.233.83.234	1 Core
migration	关机	default	Ubuntu	-	-	1 Core
test	关机	default	Ubuntu	-	-	1 Core

terminal

## 控制台访问（VNC）

通过 VNC 访问虚拟机的方式可以实现对远程计算机的完整图形界面的访问和控制，能够直观地操作远程设备，交互性更加好，但是性能会受到一定影响，且无法多终端同时在线。

!!! tip

Windows 系统选择控制台访问（VNC）。

从左侧导航栏，依次点击 **虚拟机** -> **虚拟机** 进入虚拟机列表页面。如果虚拟机状态为 **运行中**，点击列表右侧的 **⋮**，可以通过 **控制台访问（VNC）** 的方式访问虚拟机。

vnc

vnc

## 通过 NodePort 实现虚拟机的多终端 SSH 访问

本文将介绍如何通过 NodePort 从多个终端以 SSH 访问虚拟机。

## 现有访问方式的缺陷

1. 虚拟机支持通过 VNC 或者 console 访问，但这两种访问方式都有一个弊端，无法多终端同时在线。
2. 通过 NodePort 形式的 Service，可以帮助解决这个问题。

## 创建 service 的方式

1. 通过容器管理页面
  - 选择目标访问的虚拟机所在集群页面创建服务（Service）
  - 选择访问类型为节点访问（NodePort）
  - 选择命名空间（虚拟机所在的 namespace）
  - 标签选择器填写 `vm.kubevirt.io/name: you-vm-name`
  - 端口配置：协议选择 TCP，端口名称自定义，服务端口、容器端口填写 22
2. 创建成功后，就可以通过 `ssh username@nodeip -p port` 来访问虚拟机

## 通过 kubectl 创建 svc

1. 编写 YAML 文件，示例如下：

```
apiVersion: v1
kind: Service
metadata:
  name: test-ssh
spec:
  ports:
    - name: tcp-ssh
      nodePort: 32090
      protocol: TCP
      // 22 端口，不要更改
      port: 22
      targetPort: 22
```

```
selector:  
    // 虚拟机的 name  
    vm.kubevirt.io/name: test-image-s3  
    type: NodePort
```

2. 执行以下命令：

```
kubectl apply -f you-svc.yaml
```

3. 创建成功后，就可以通过 ssh username@nodeip -p 32090 来访问虚拟机

## 虚拟机详情

成功[创建虚拟机](#)后，可进入虚拟机详情页面，支持查看基本信息、配置信息、GPU 信息、概览、存储、网络、快照、事件等。

点击左侧导航栏上的 **容器管理**，然后点击 **集群列表**，进入虚拟机所在集群详情，点击虚拟机名称查看虚拟机详情。

## 基本信息

虚拟机基本信息包含状态、别名、集群、命名空间、IP、标签、节点、用户名、密码、创建时间等。其中，

- 状态：虚拟机当前的运行状态（运行中/处理中/关机/错误）。
- IP：虚拟机的 IP 地址。对于添加多张网卡的虚拟机，会为其分配多个 IP 地址。

## 配置信息 & GPU 配置

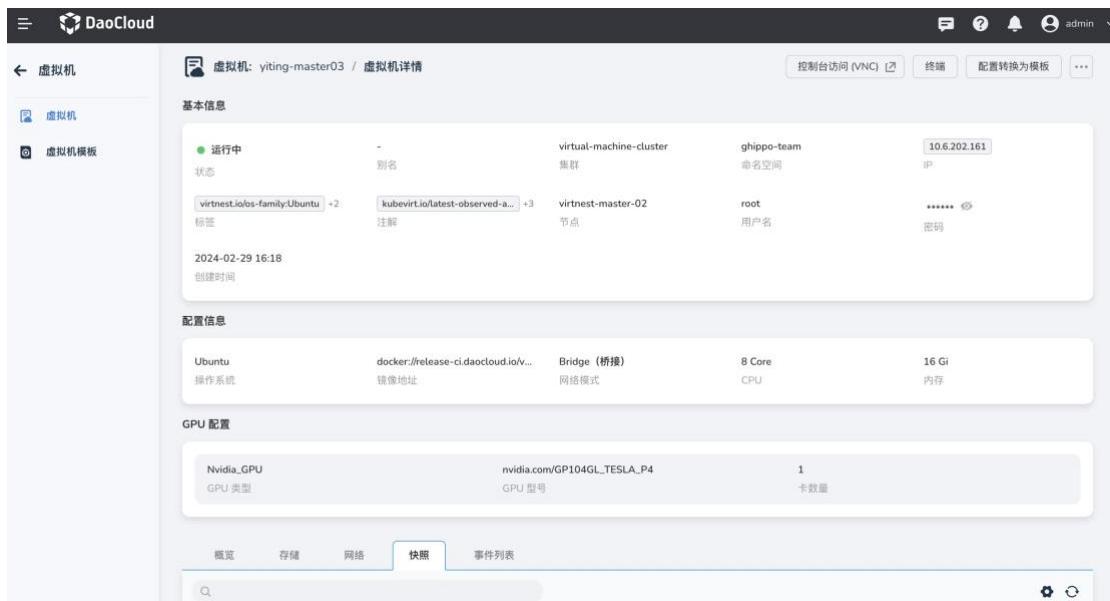
虚拟机配置信息包括：

- 操作系统：安装在虚拟机上用于执行程序的操作系统。
- 镜像地址：向一个虚拟硬盘文件或操作系统安装介质的链接，这个地址用于在虚拟

机软件中加载和安装操作系统。

- 网络模式：虚拟机配置的网络模式，Bridge（桥接）或Masquerade（NAT）。
- CPU、内存：为虚拟机分配的资源。

GPU 配置信息包含 GPU 类型、GPU 型号以及卡数量。



## 虚拟机详情

## 其他信息

==== “概览”

虚拟机概览页可查看虚拟机的监控内容。请注意，若未安装 insight-agent 组件，则无法获取监控信息。

![概览](docs/zh/docs/virtnest/images/monitor02.png)

==== “储存”

展示虚拟机所用的存储，包括系统盘和数据盘的信息。

![存储](docs/zh/docs/virtnest/images/detail-sc.png)

==== “网络”

展示虚拟机的网络配置，包括 Multus CR、网卡名称、IP 地址等信息。

![网络](docs/zh/docs/virtnest/images/detail-network.png)

### ==== “快照”

若已经[创建快照](./vm/snapshot.md)，本页将展示虚拟机的快照信息，支持通过快照恢复虚拟机。

![快照](docs/zh/docs/virtnest/images/detail-snapshot.png)

### ==== “事件列表”

事件列表包含虚拟机的生命周期中发生的各种状态变化、操作记录和系统消息等。

![事件](docs/zh/docs/virtnest/images/detail-event.png)

## 创建密钥

当创建虚拟机使用对象存储 (S3) 作为镜像来源时，有时候需要填写密钥来获取通过 S3 的验证。以下将介绍如何创建符合虚拟机要求的密钥。

1. 点击左侧导航栏上的 **容器管理**，然后点击 **集群列表**，进入虚拟机所在集群详情，  
点击 **配置与密钥**，选择 **密钥**，点击 **创建密钥**。

创建密钥

创建密钥

2. 进入创建页面，填写密钥名称，选择和虚拟机相同的命名空间，注意需要选择 **默认 (Opaque)** 类型。密钥数据需要遵循以下原则

- accessKeyId: 需要以 Base64 编码方式表示的数据
- secretKey: 需要以 Base64 编码方式表示的数据

密钥要求

密钥要求

3. 创建成功后可以在创建虚拟机时使用所需密钥，最后通过验证。

使用密钥

使用密钥

# 克隆虚拟机

本文将介绍如何克隆一台新的虚拟机。

用户可以克隆一台新的虚拟机，克隆后的虚拟机将具有与原始虚拟机相同的操作系统和系统配置，能够实现快速部署和扩展，快速创建相似配置的新虚拟机，而无需从头开始安装。

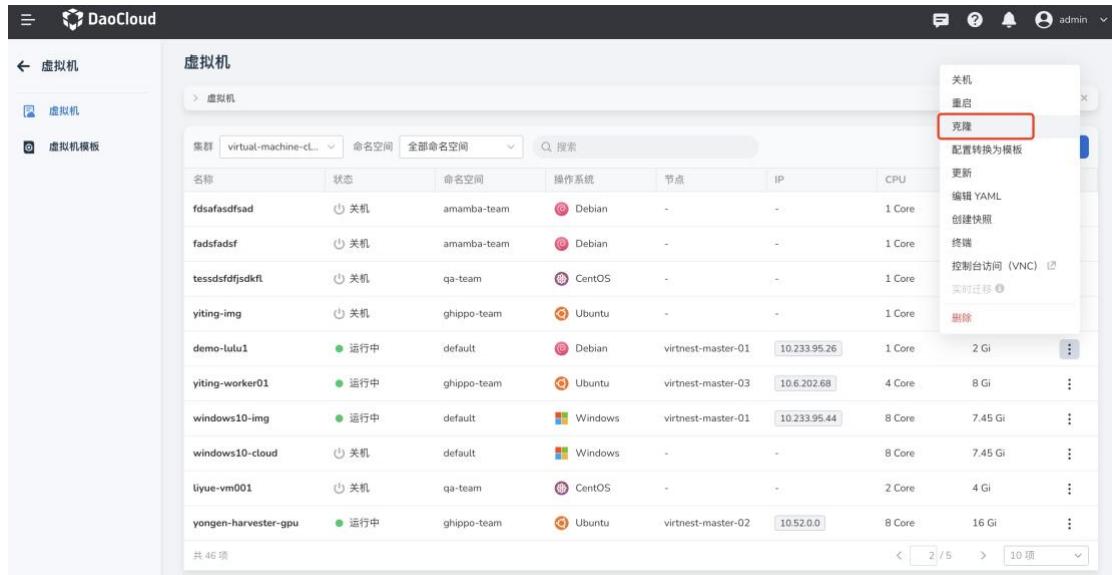
## 前提条件

使用克隆功能之前，需要满足以下前提条件（和快照功能的前提条件一致）：

- 只有非错误状态下的虚拟机才能使用克隆功能。
- 安装 Snapshot CRDs、Snapshot Controller、CSI Driver。具体安装步骤可参考 [CSI Snapshotter](#)。
- 等待 snapshot-controller 组件准备就绪，该组件会监控 VolumeSnapshot 和 VolumeSnapshotContent 相关事件，并触发相关操作。
- 等待 CSI Driver 准备就绪，确保 csi-snapshotter sidecar 跑在 CSI Driver 里，csi-snapshotter sidecar 会监控 VolumeSnapshotContent 相关事件，并触发相关操作。
  - 如存储是 Rook-Ceph，可参考 [ceph-csi-snapshot](#)
  - 如存储是 HwameiStor，可参考 [huameistor-snapshot](#)

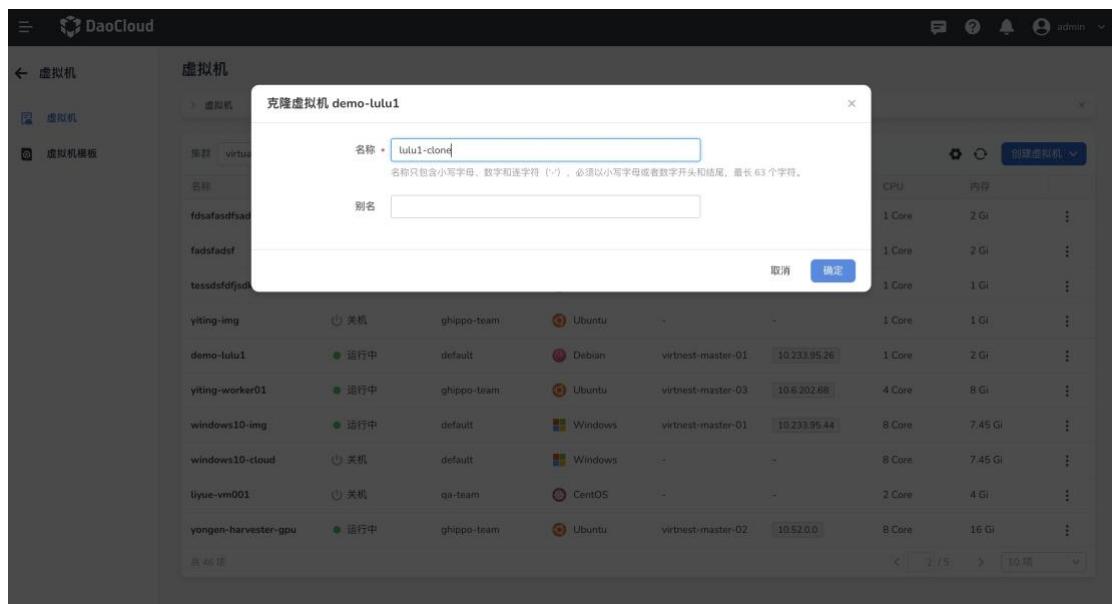
## 克隆虚拟机

1. 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，进入列表页面，点击列表右侧的 **⋮**，可以对非错误状态下的虚拟机执行快照操作。



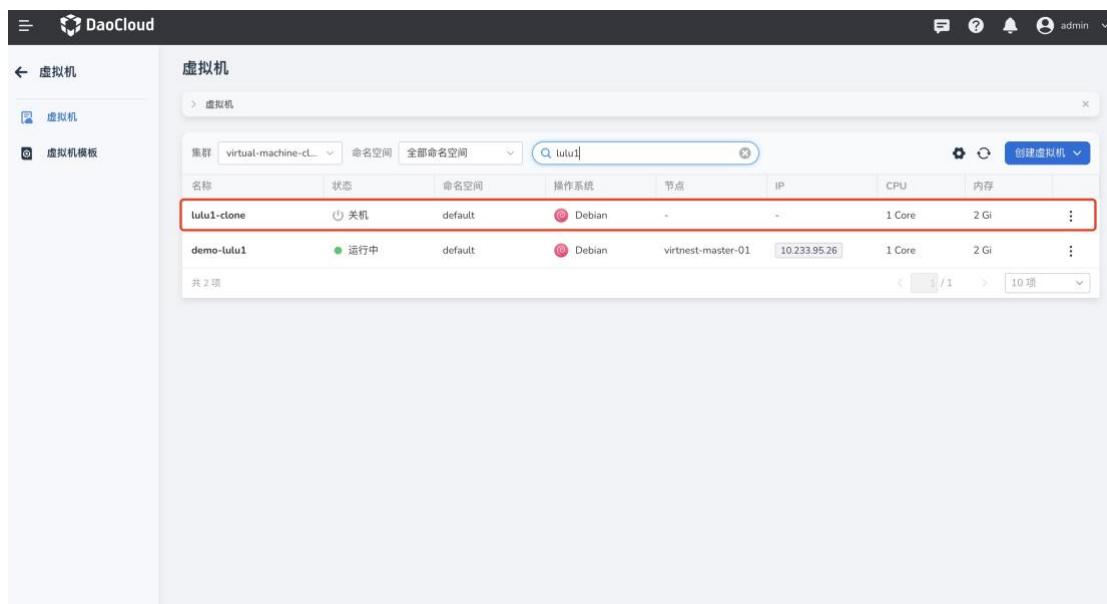
## 克隆虚拟机

2. 弹出弹框，需要填写克隆新的虚拟机的名称和描述，克隆操作可能需要一些时间，具体取决于虚拟机的大小和存储性能。



## 克隆过程

3. 克隆成功后可以在虚拟机列表内查看到新的虚拟机，新创建出来的虚拟机处于关机状态，若需要开机需要手动操作。



新的虚拟机

- 克隆前建议对原有虚拟机进行快照，如果克隆过程中遇到问题，请检查先决条件是否满足，并尝试重新执行克隆操作。

## 快照管理

本文将介绍如何为虚拟机创建快照，并从快照中恢复的。

用户可以为虚拟机创建快照，保存虚拟机当下的状态，一个快照可以支持多次恢复，每次恢复时，虚拟机将被还原到快照创建时的状态。通常可以用于备份、恢复、回滚等场景。

## 前提条件

使用快照功能之前，需要满足以下前提条件：

- 只有非错误状态下的虚拟机才能使用快照功能。
- 安装 Snapshot CRDs、Snapshot Controller、CSI Driver。具体安装步骤可参考 [CSI Snapshotters](#)。

- 等待 snapshot-controller 组件准备就绪，该组件会监控 VolumeSnapshot 和 VolumeSnapshotContent 相关事件，并触发相关操作。
- 等待 CSI Driver 准备就绪，确保 csi-snapshotter sidecar 跑在 CSI Driver 里，csi-snapshotter sidecar 会监控 VolumeSnapshotContent 相关事件，并触发相关操作。如 POC 使用的存储是 rook-ceph，可参考 [ceph-csi-snapshot](#)
  - 如存储是 Rook-Ceph，可参考 [ceph-csi-snapshot](#)
  - 如存储是 HwameiStor，可参考 [huameistor-snapshot](#)

## 创建快照

1. 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，进入列表页面，点击列表右侧的 ，可以对非错误状态下的虚拟机执行快照操作。

创建快照

创建快照

2. 弹出弹框，需要填写快照的名称和描述，创建快照大概需要几分钟的时间，在此期间无法对虚拟机做任何操作。

快照名称

快照名称

3. 创建成功后可以在虚拟机详情内查看快照信息，支持编辑描述、从快照中恢复、删除等操作。

虚拟机详情

虚拟机详情

## 从快照中恢复

1. 点击 **从快照恢复**，需要填写虚拟机恢复记录的名称，同时恢复操作可能需要一些时间来完成，具体取决于快照的大小和其他因素。恢复成功后，虚拟机将回到快照创建时的状态。

快照恢复

快照恢复

2. 一段时间后，下拉快照信息，可以查看当前快照的所有恢复记录，并且支持展示定位恢复的位置。

恢复记录

恢复记录

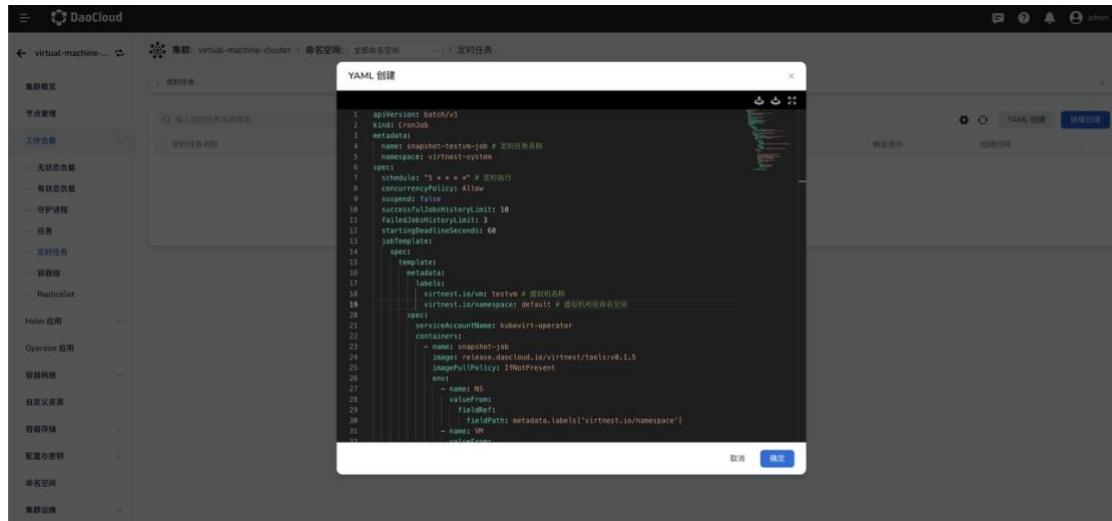
## 定时快照

本文将介绍如何为虚拟机定时创建快照。

用户可以为虚拟机定时创建快照，能够为数据提供持续的保护，确保在发生数据丢失、损坏或删除的情况下可以进行有效的数据恢复。

## 定时快照步骤

1. 点击左侧导航栏上的 **容器管理** -> **集群列表**，在列表页面，选择目标虚拟机所在的集群。进入集群后，点击 **工作负载** -> **定时任务**，选择 **YAML 创建** 定时任务，参考以下 YAML 示例可为指定虚拟机定时创建快照。



## yaml 创建定时任务

??? note “点击查看创建定时任务的 YAML 示例”

```

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: xxxxx-xxxxx-cronjob # 定时任务名称, 可自定义
  namespace: virtnest-system # 请勿修改此 namespace
spec:
  schedule: "5 * * * *" # 按需修改定时任务执行间隔
  concurrencyPolicy: Allow
  suspend: false
  successfulJobsHistoryLimit: 10
  failedJobsHistoryLimit: 3
  startingDeadlineSeconds: 60
  jobTemplate:
    spec:
      template:
        metadata:
          labels:
            virtnest.io/vm: xxxx # 修改为需要快照的虚拟机名称
            virtnest.io/namespace: xxxx # 修改为虚拟机所在的命名空间
        spec:
          serviceAccountName: kubevirt-operator
          containers:
            - name: snapshot-job
              image: release.daocloud.io/virtnest/tools:v0.1.5 # 离线环境下,仓库地址
              imagePullPolicy: IfNotPresent

```

```
env:  
  - name: NS  
    valueFrom:  
      fieldRef:  
        fieldPath: metadata.labels['virtnest.io/namespace']  
  - name: VM  
    valueFrom:  
      fieldRef:  
        fieldPath: metadata.labels['virtnest.io/vm']  
command:  
  - /bin/sh  
  - -c  
  - |  
    export SUFFIX=$(date +"%Y%m%d-%H%M%S")  
    cat <<EOF | kubectl apply -f -  
    apiVersion: snapshot.kubevirt.io/v1alpha1  
    kind: VirtualMachineSnapshot  
    metadata:  
      name: $(VM)-snapshot-$SUFFIX  
      namespace: $(NS)  
    spec:  
      source:  
        apiGroup: kubevirt.io  
        kind: VirtualMachine  
        name: $(VM)  
    EOF  
  restartPolicy: OnFailure  
  ...
```

2. 创建定时任务并成功运行后，可点击 **虚拟机** 在列表页面选择目标虚拟机，进入详情后可查看快照列表。

The screenshot shows the DaoCloud Enterprise 5.0 interface for managing virtual machines. On the left, there's a navigation bar with '虚拟机' (Virtual Machines) selected. The main area is divided into sections: '基本信息' (Basic Information), '配置信息' (Configuration Information), and a '快照' (Snapshot) tab. In the '基本信息' section, it shows the VM is '运行中' (Running), has 1 CPU core, and 2 GB of memory. The '配置信息' section shows the operating system is 'Ubuntu'. The '快照' tab displays a list of snapshots with their names and statuses:

名称	描述	状态	最近恢复时间
testvm-snapshot-20240926-030501	-	成功	-
testvm-snapshot-20240926-020501	-	成功	-
testvm-snapshot-20240926-010501	-	成功	-
testvm-snapshot-20240926-000501	-	成功	-
testvm-snapshot-20240925-230501	-	成功	-
testvm-snapshot-20240925-220501	-	成功	-
testvm-snapshot-20240925-210501	-	成功	-
testvm-snapshot-20240925-200501	-	成功	-
testvm-snapshot-20240925-190501	-	成功	-

定时快照

## 实时迁移

本文将介绍如何将虚拟机从一个节点移动到另一个节点。

当节点维护或者升级时，用户可以将正在运行的虚拟机无缝迁移到其他的节点上，同时可以保证业务的连续性和数据的安全性。

### 前提条件

使用实时迁移之前，需要满足以下前提条件：

- 虚拟机必须处于运行状态才能进行实时迁移。
- 确保您的 PVC 访问模式为 ReadWriteMany，以便使用实时迁移功能。
- 确保集群内至少有两个节点可供使用。

### 实时迁移

1. 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，进入列表页面，点击列表右侧的 **⋮**，

可以对运行状态下的虚拟机进行迁移动作。目前虚拟机所在节点为 controller-node-3。

3.

名称	状态	命名空间	操作系统	节点	IP	CPU	内存
export-ubuntu-to-other	处理中	default	Ubuntu	-	-	1 Core	2 Gi
demo	运行中	default	Debian	controller-node-3	10.233.83.196	1 Core	2 Gi
export	关机	default	Debian	-	-	1 Core	关机
expand2	运行中	default	Debian	controller-node-1	10.233.74.84	1 Core	克隆
hotplug	关机	default	Ubuntu	-	-	4 Core	配置转换为模板 更新 编辑 YAML 创建快照 终端 控制台访问 (VNC)

### 实时迁移

2. 弹出弹框，提示在实时迁移期间，正在运行的虚拟机实例会移动到另一个节点，可以选择指定节点迁移，也可以随机迁移，请确保其他节点资源充足。

### 迁移提示

3. 迁移需要一段时间，请耐心等待，成功后可以在虚拟机列表内查看节点信息，此时

节点迁移到 controller-node-1 。

名称	状态	命名空间	操作系统	节点	IP	CPU	内存
export-ubuntu-to-other	关机	default	Ubuntu	-	-	1 Core	2 Gi
demo	运行中	default	Debian	controller-node-1	10.233.74.82	1 Core	2 Gi
export	关机	default	Debian	-	-	1 Core	2 Gi
expand2	运行中	default	Debian	controller-node-1	10.233.74.84	1 Core	2 Gi
hotplug	关机	default	Ubuntu	-	-	4 Core	8 Gi

迁移结果

## 集群内冷迁移

本文将介绍在关机状态下如何将虚拟机在同一集群内从一个节点移动到另一个节点。

冷迁移的主要特点是，虚拟机在迁移过程中会处于离线状态，这可能会对业务连续性产生影响。因此，在实施冷迁移时需要仔细规划迁移时间窗口，并考虑业务需求和系统可用性。

通常，冷迁移适用于对停机时间要求不是非常严格的场景。

## 前提条件

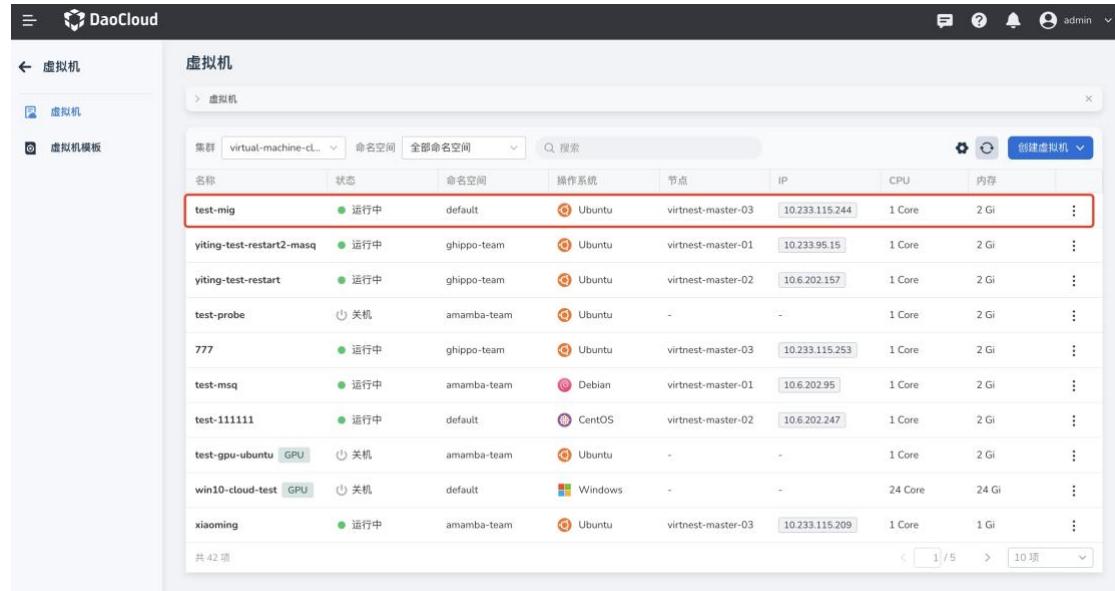
使用冷迁移之前，需要满足以下前提条件：

- 虚拟机必须处于关机状态才能进行冷迁移。

# 冷迁移

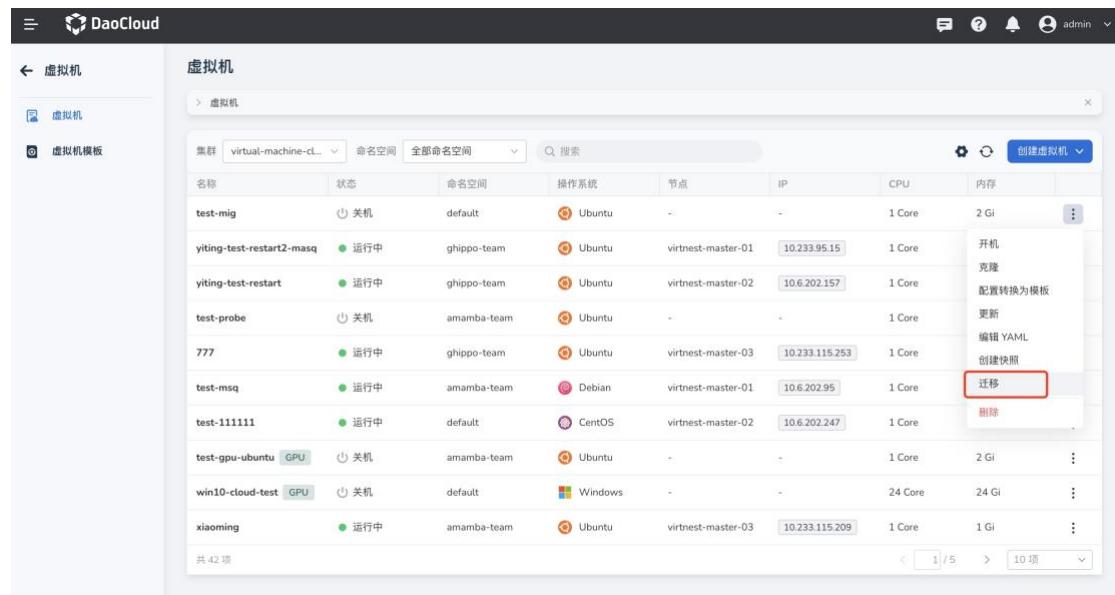
1. 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，进入列表页面，点击列表右侧的 ，

可以对关机状态下的虚拟机进行迁移动作。虚拟机在关机状态下时无法查看所在节点，需要提前规划或者开机查询。



名称	状态	命名空间	操作系统	节点	IP	CPU	内存	操作
test-mig	运行中	default	Ubuntu	virtnest-master-03	10.233.115.244	1 Core	2 Gi	
yiting-test-restart2-masq	运行中	ghippo-team	Ubuntu	virtnest-master-01	10.233.95.15	1 Core	2 Gi	
yiting-test-restart	运行中	ghippo-team	Ubuntu	virtnest-master-02	10.6.202.157	1 Core	2 Gi	
test-probe	关机	amamba-team	Ubuntu	-	-	1 Core	2 Gi	
777	运行中	ghippo-team	Ubuntu	virtnest-master-03	10.233.115.253	1 Core	2 Gi	
test-msq	运行中	amamba-team	Debian	virtnest-master-01	10.6.202.95	1 Core	2 Gi	
test-111111	运行中	default	CentOS	virtnest-master-02	10.6.202.247	1 Core	2 Gi	
test-gpu-ubuntu	关机	amamba-team	Ubuntu	-	-	1 Core	2 Gi	
win10-cloud-test	关机	default	Windows	-	-	24 Core	24 Gi	
xiaoming	运行中	amamba-team	Ubuntu	virtnest-master-03	10.233.115.209	1 Core	1 Gi	

## 迁移前



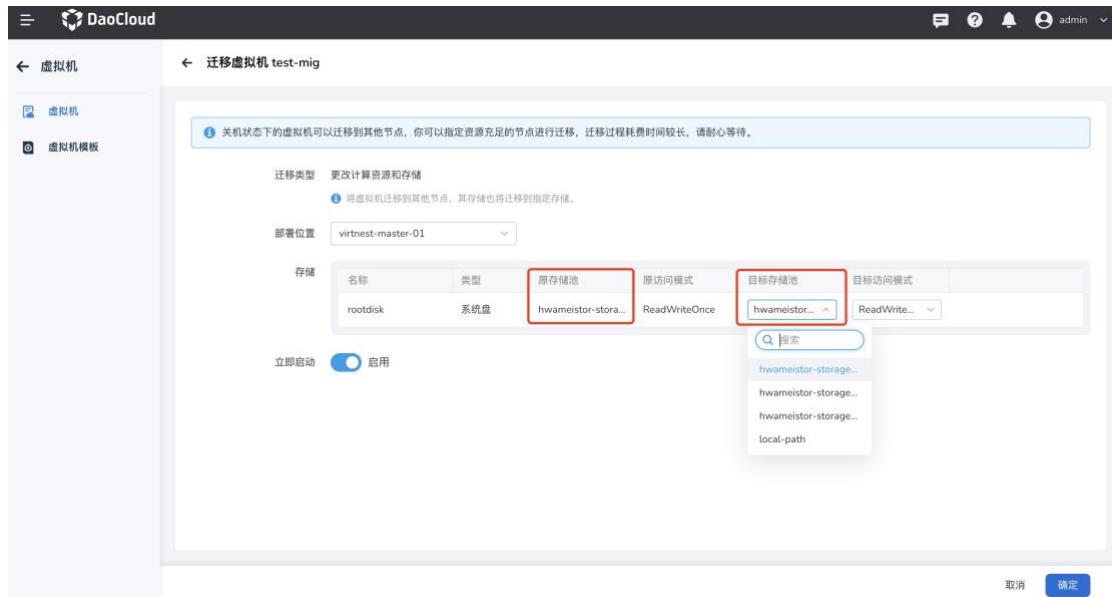
名称	状态	命名空间	操作系统	节点	IP	CPU	内存	操作
test-mig	关机	default	Ubuntu	-	-	1 Core	2 Gi	
yiting-test-restart2-masq	运行中	ghippo-team	Ubuntu	virtnest-master-01	10.233.95.15	1 Core	2 Gi	开机
yiting-test-restart	运行中	ghippo-team	Ubuntu	virtnest-master-02	10.6.202.157	1 Core	2 Gi	克隆
test-probe	关机	amamba-team	Ubuntu	-	-	1 Core	2 Gi	配置转换为模板
777	运行中	ghippo-team	Ubuntu	virtnest-master-03	10.233.115.253	1 Core	2 Gi	更新
test-msq	运行中	amamba-team	Debian	virtnest-master-01	10.6.202.95	1 Core	2 Gi	编辑 YAML
test-111111	运行中	default	CentOS	virtnest-master-02	10.6.202.247	1 Core	2 Gi	创建快照
test-gpu-ubuntu	关机	amamba-team	Ubuntu	-	-	1 Core	2 Gi	
win10-cloud-test	关机	default	Windows	-	-	24 Core	24 Gi	
xiaoming	运行中	amamba-team	Ubuntu	virtnest-master-03	10.233.115.209	1 Core	1 Gi	

## 关机后迁移

!!! note

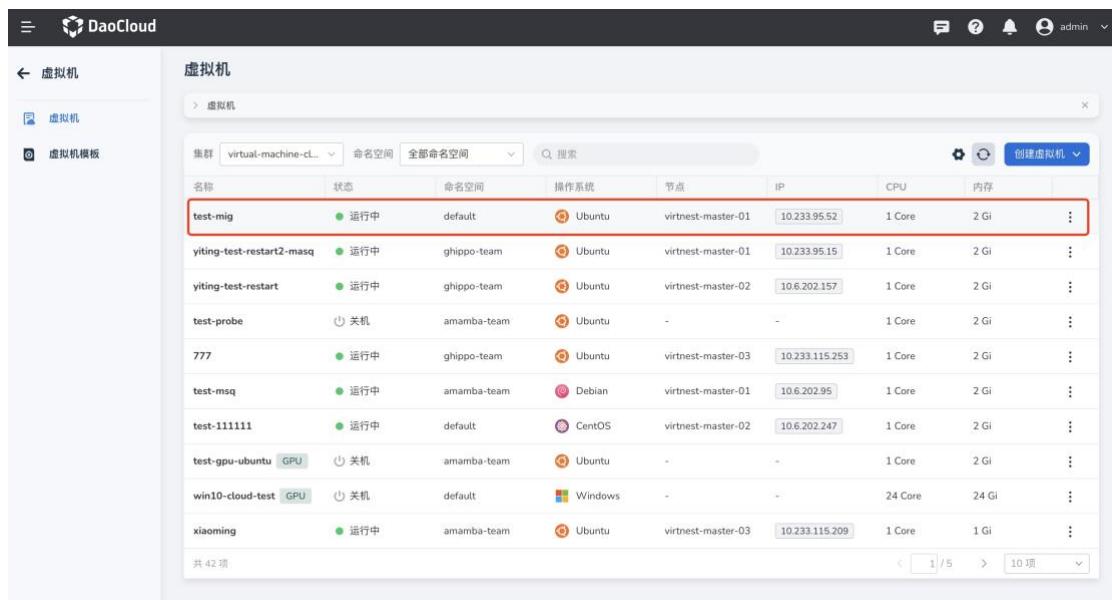
如果您在原始节点的存储池中使用了 local-path，跨节点迁移时可能出现问题，请谨慎选择。

2. 点击迁移后，提示在迁移期间，可以选择指定节点迁移，也可以随机迁移，若需要修改存储池，需要确保目标节点内有可用存储池。同时需要目标节点资源充足，迁移过程耗费时间较长，请耐心等待。



### 迁移提示

3. 迁移需要一段时间，请耐心等待，成功后需要重启查看是否迁移成功。本示例已经开机查看迁移效果。



迁移结果

# 虚拟机跨集群迁移

本功能暂未做 UI 界面能力，请参考以下操作步骤。

## 使用场景

- 当原集群发生故障或性能下降导致该集群上的虚拟机无法访问时，将虚拟机迁移到其他的集群上。
- 需要对集群进行计划内的维护或升级时，将虚拟机迁移到其他的集群上。
- 当特定应用的性能需求变化，需要调整资源分配时，迁移虚拟机到其他的集群上以匹配更合适的资源配置。

## 前提条件

实现虚拟机跨集群迁移之前，需要满足以下前提条件：

- 集群网络互通：确保原有集群与目标迁移集群之间的网络是互通的
- 相同存储类型：目标迁移集群需支持与原有集群相同的存储类型（例如，如果导出集群使用 rook-ceph-block 类型的 StorageClass，则导入集群也必须支持此类型）。
- 在原有集群的 KubeVirt 中开启 VMExport Feature Gate。

## 开启 VMExport Feature Gate

激活 VMExport Feature Gate，在原有集群内执行如下命令，可参考[如何激活特性门控](#)。

```
kubectl edit kubevirt kubevirt -n virtnest-system
```

这条命令将修改 featureGates，增加 VMExport。

```

apiVersion: kubevirt.io/v1
kind: KubeVirt
metadata:
  name: kubevirt
  namespace: virtnest-system
spec:
  configuration:
    developerConfiguration:
      featureGates:
        - DataVolumes
        - LiveMigration
        - VMExport

```

## 配置原有集群的 Ingress

安装 LB 类型的 ingress-controller。

在 virtnest-system 命名空间下创建 tls secret:

```

export KEY_FILE=key.pem
export CERT_FILE=cert.ca
export HOST=upgrade-test.com
openssl req -x509 -nodes -days 365 -newkey rsa: 2048 -keyout ${KEY_FILE} -out ${CERT_F
ILE} -subj "/CN=${HOST}/O=${HOST}" -addext "subjectAltName = DNS:${HOST}"

export CERT_NAME=nginx-tls
kubectl -n virtnest-system create secret tls ${CERT_NAME} --key ${KEY_FILE} --cert ${CER
T_FILE}

```

在 virtnest-system 命名空间下创建 Ingress，配置 Ingress 以指向 virt-exportproxy Service：

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-vm-export
  namespace: virtnest-system
spec:
  tls:
    - hosts:
        - upgrade-test.com
      secretName: nginx-tls
  rules:
    - host: upgrade-test.com
      http:
        paths:

```

```

  - path: /
    pathType: Prefix
    backend:
      service:
        name: virt-exportproxy
        port:
          number: 8443
  ingressClassName: nginx

```

## 配置目标集群的 CoreDNS ConfigMap

kubectl edit cm coredns -n kube-system

找到 Corefile 配置部分，并添加 hosts 配置。这里假设 Ingress 的 LB 配置为 192.168.1.10：

**Corefile:** |

```

.: 53 {
  errors
  health
  ready
  kubernetes cluster.local in-addr.arpa ip6.arpa {
    pods insecure
    fallthrough in-addr.arpa ip6.arpa
  }
  hosts {
    192.168.1.10 upgrade-test.com
    fallthrough
  }
  prometheus :9153
  forward . /etc/resolv.conf
  cache 30
  loop
  reload
  loadbalance
}

```

## 迁移步骤

### 1. 创建 VirtualMachineExport CR

==== “虚拟机关机状态下的冷迁移”

```

```yaml
apiVersion: v1

```

```
kind: Secret
metadata:
  name: example-token # 导出虚拟机所用 token
  namespace: default # 虚拟机所在命名空间
stringData:
  token: 1234567890ab # 导出时所用的 token

---
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export # 导出名称, 可自行修改
  namespace: default # 虚拟机所在命名空间
spec:
  tokenSecretRef: example-token # 和上面创建的 token 名称保持一致
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: testvm # 虚拟机名称
````
```

==== “虚拟机不关机状态下的热迁移”

```
```yaml
apiVersion: v1
kind: Secret
metadata:
  name: example-token # 导出虚拟机所用 token
  namespace: default # 虚拟机所在命名空间
stringData:
  token: 1234567890ab # 导出时所用的 token

---
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: export-snapshot # 导出名称, 可自行修改
  namespace: default # 虚拟机所在命名空间
spec:
  tokenSecretRef: export-token # 和上面创建的 token 名称保持一致
  source:
    apiGroup: "snapshot.kubevirt.io"
    kind: VirtualMachineSnapshot
    name: export-snap-202407191524 # 对应的虚拟机快照名称
````
```

2. 检查 VirtualMachineExport 是否准备就绪：

```
# 这里的 example-export 需要替换为创建的 VirtualMachineExport 名称
kubectl get VirtualMachineExport example-export -n default
```

| NAME           | SOURCEKIND     | SOURCENAME | PHASE |
|----------------|----------------|------------|-------|
| example-export | VirtualMachine | testvm     | Ready |

3. 当 VirtualMachineExport 准备就绪后，导出虚拟机 YAML。

== “如果已安装 virtctl”

使用以下命令导出虚拟机的 YAML：

```
```sh
# 自行将 example-export 替换为创建的 VirtualMachineExport 名称
# 自行通过 -n 指定命名空间
virtctl vmexport download example-export --manifest --include-secret --output=manifest.yaml
```

```

== “如果没有安装 virtctl”

使用以下命令导出虚拟机 YAML：

```
```sh
# 自行替换 example-export 替换为创建的 VirtualMachineExport 名称 和命名空间
manifesturl=$(kubectl get VirtualMachineExport example-export -n default -o=jsonpath='{.status.links.internal.manifests[0].url}')
secreturl=$(kubectl get VirtualMachineExport example-export -n default -o=jsonpath='{.status.links.internal.manifests[1].url}')
# 自行替换 secret 名称和命名空间
token=$(kubectl get secret example-token -n default -o=jsonpath='{.data.token}' | base64 -d)

curl -H "Accept: application/yaml" -H "x-kubevirt-export-token: $token" --insecure $secreturl > manifest.yaml
curl -H "Accept: application/yaml" -H "x-kubevirt-export-token: $token" --insecure $manifesturl >> manifest.yaml
```

```

4. 导入虚拟机

将导出的 manifest.yaml 复制到目标迁移集群并执行以下命令（如果命名空间不存在则需要提前创建）：

```
kubectl apply -f manifest.yaml
```

创建成功后，重启虚拟机，虚拟机成功运行后，在原有集群内删除原虚拟机（虚拟机未启动成功时，请勿删除原虚拟机）。

## 虚拟机监控

虚拟机基于 Kubevirt 开源的 Grafana Dashboard，为了每一个虚拟机生成了监控看板。虚拟机的监控信息可以更好的了解虚拟机的资源消耗情况，比如 CPU、内存、存储和网络等资源的使用情况，从而进行资源的优化和规划，提升整体的资源利用效率。

### 前提条件

查看虚拟机监控的相关信息之前，需要满足以下前提条件：

- 在虚拟机所在的同一集群内安装 Insight-agent 组件，并且保证 Insight-agent 组件正常可用。

## 虚拟机监控

进入虚拟机的详细信息并点击 **概览**，即可查看虚拟机的监控内容。请注意，若未安装 Insight-agent 组件，则无法获取监控信息。以下是详细信息：

- CPU 总量、CPU 使用量、内存总量、内存使用量。

虚拟机: test-pye / 虚拟机详情

**基本信息**

运行中  
状态

CentOS  
操作系统

virtnetx.io/ios-family:CentOS +1  
标签

Masquerade (NAT)  
网络模式

控制台访问 (VNC) | 终端 | 配置转换为模板 | ...

概览 存储 网络 快照

CPU 总量: 1 Core | CPU 使用量: 0 Core | 内存总量: 1 Gi | 内存使用量: 0.300 Gi

CPU 使用率: 0.20% (0.15% - 0.25%)

内存使用率: 0.20% (0.00% - 0.25%)

## 监控

- CPU 使用率: 指当前虚拟机正在使用的 CPU 资源的百分比;
- 内存使用率: 指当前虚拟机正在使用的内存资源占总可用内存的百分比。



## CPU、内存使用率

- 网络流量: 指虚拟机在特定时间段内发送和接收的网络数据量;
- 网络丢包率: 指在数据传输过程中丢失的数据包占总发送数据包数量的比例。



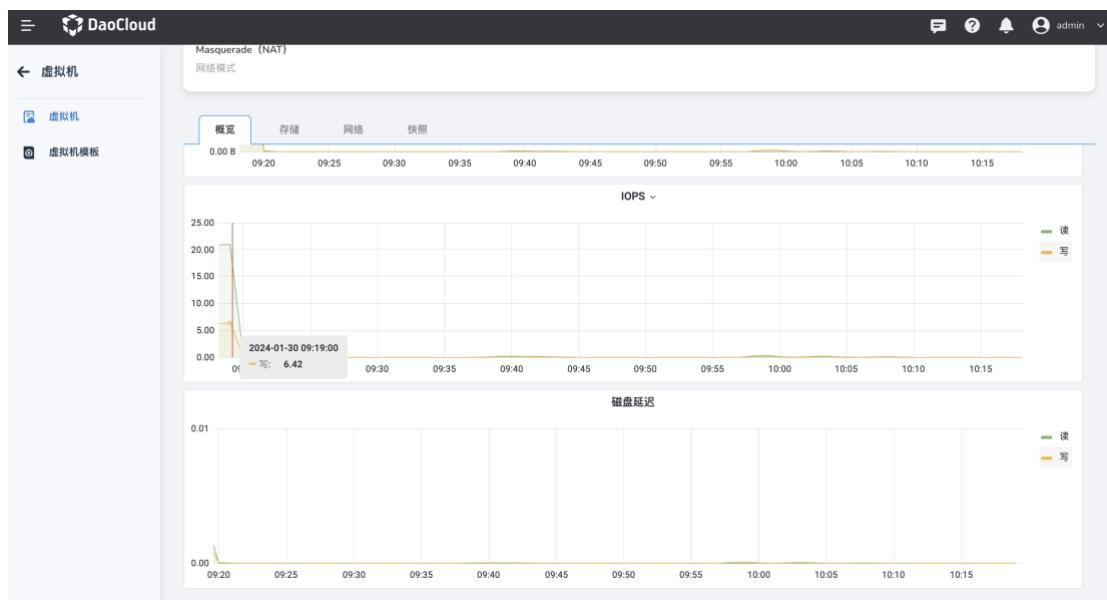
### 网络流量、丢包率

- 网络错误率：指在网络传输过程中发生的错误的比率；
- 磁盘吞吐：指虚拟机系统在一定时间内读取和写入磁盘的速度和能力。



### 网络错误率、磁盘吞吐

- IOPS：指的是在一秒钟内虚拟机系统进行的输入/输出操作的次数。磁盘延迟：指虚拟机系统在进行磁盘读写操作时所经历的时间延迟。



IOPS、磁盘延迟

## 虚拟机网络

本文将介绍如何在创建虚拟机时，配置网络信息。

在虚拟机中，网络管理是一个关键的部分，它使得我们能够在 Kubernetes 环境中管理和配置虚拟机的网络连接，可以根据不同的需求和场景来进行配置，实现更灵活和多样化的网络架构。

1. **单网卡场景：**对于一些简单的只需要基本网络连接的应用，或者存在资源限制的时候，使用单网卡可以节约网络资源，并避免资源的浪费。
2. **多网卡场景：**当需要实现不同网络环境之间的安全隔离时，可以使用多网卡来划分不同的网络区域。同时也可以对控制和流量进行管理。

## 网络配置前提

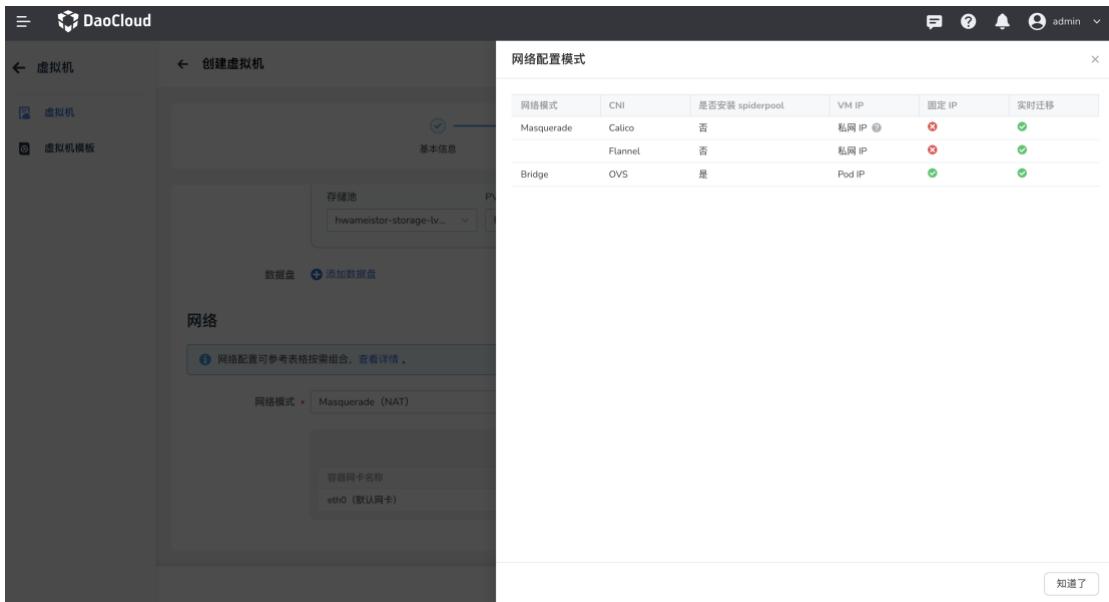
在使用虚拟机网络功能之前，需要根据网络模式的不同配置不同的信息：

- 选择 Bridge 网络模式时需要提前配置一些信息：
  - 在主机节点上安装并运行 Open vSwitch, 可参考[这里](#)
  - 在主机节点上配置 Open vSwitch 网桥, 可参考[这里](#)
  - 安装 Spiderpool, 可参考[安装 Spiderpool](#), Spiderpool 默认会把 Multus CNI 和 Ovs CNI 都装上
  - 创建 ovs 类型的 Multus CR, 可参考[界面创建 Multus CR](#) 或 [YAML 创建 Multus CR](#)
  - 创建子网及 IP 池, 参考[创建子网和 IP 池](#)

## 网络配置

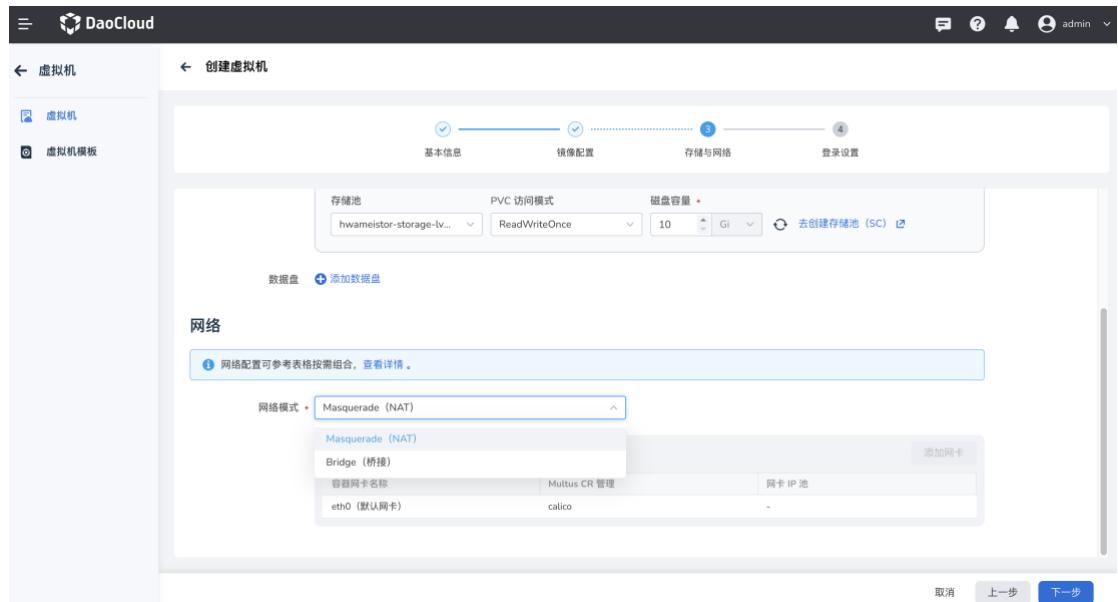
- 配置虚拟机的网络配置, 可以根据表格信息按需组合。

| 网络模式                | CNI        | 是否安装 | 网卡模式 | 固定 IP | 实时迁移 |
|---------------------|------------|------|------|-------|------|
| Masquerade<br>(NAT) | Spiderpool |      |      |       |      |
|                     | Calico     | ✗    | 单网卡  | ✗     | ✓    |
|                     | Cilium     | ✗    |      | ✗     | ✓    |
|                     | Flannel    | ✗    |      | ✗     | ✓    |
| Bridge (桥接)         | OVS        | ✓    | 多网卡  | ✓     | ✓    |



## 网络配置

2. 网络模式：分为 Masquerade (NAT)、Bridge (桥接) 两种，Bridge (桥接) 模式需要安装了 spiderpool 组件后方可使用。
1. 默认选择 Masquerade (NAT) 的网络模式，使用 eth0 默认网卡。
2. 若集群内安装了 spiderpool 组件，则支持选择 Bridge (桥接) 模式，支持多网卡形式。

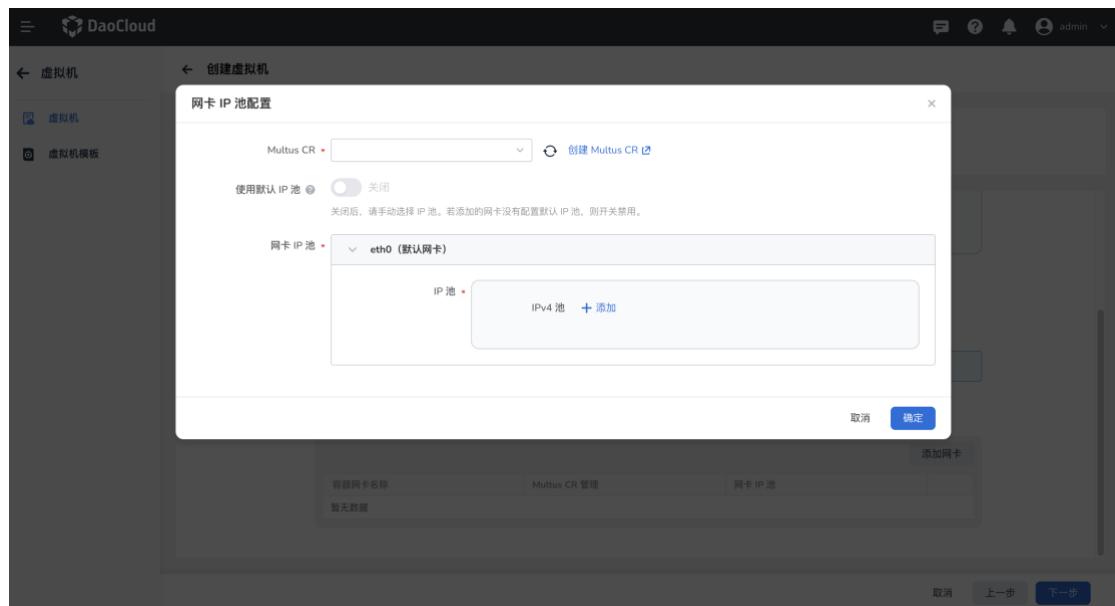


## 网络模式

- 选择 Bridge 模式时，需要有一些[前提条件](#)

### 3. 添加网卡

1. Bridge (桥接) 模式下支持手动添加网卡。点击 **添加网卡**，进行网卡 IP 池的配置。选择和网络模式匹配的 Multus CR，若没有则需要自行创建。
2. 若打开 **使用默认 IP 池** 开关，则使用 multus CR 配置中的默认 IP 池。若关闭开关，则手动选择 IP 池。



添加网卡

## 虚拟机存储

本文将介绍如何在创建虚拟机时，配置存储信息。

存储和虚拟机的功能息息相关，主要是通过使用 Kubernetes 的持久卷和存储类，提供了灵活且可扩展的虚拟机存储能力。比如虚拟机镜像存储在 PVC 里，支持和其他数据一起克隆、快照等。

## 部署不同的存储

在使用虚拟机存储功能之前，需要根据需要部署不同的存储：

1. 参考[部署 hwameistor](#)， 或者在容器管理模块的 Helm 模板中安装 hwameistor-operator。
2. 参考[部署 rook-ceph](#)
3. 部署 localpath， 使用命令 kubectl apply -f 创建以下 YAML：

??? note “[点击查看完整 YAML](#)”

```
```yaml
---
apiVersion: v1
kind: Namespace
metadata:
  name: local-path-storage

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: local-path-provisioner-service-account
  namespace: local-path-storage

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: local-path-provisioner-role
rules:
- apiGroups: [""]
  resources: ["nodes", "persistentvolumeclaims", "configmaps"]
  verbs: ["get", "list", "watch"]
- apiGroups: [""]
  resources: ["endpoints", "persistentvolumes", "pods"]
  verbs: ["*"]
- apiGroups: [""]
  resources: ["events"]
  verbs: ["create", "patch"]
- apiGroups: ["storage.k8s.io"]
```

```
resources: ["storageclasses"]
verbs: ["get", "list", "watch"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: local-path-provisioner-bind
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: local-path-provisioner-role
subjects:
- kind: ServiceAccount
  name: local-path-provisioner-service-account
  namespace: local-path-storage

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: local-path-provisioner
  namespace: local-path-storage
spec:
  replicas: 1
  selector:
    matchLabels:
      app: local-path-provisioner
  template:
    metadata:
      labels:
        app: local-path-provisioner
    spec:
      serviceAccountName: local-path-provisioner-service-account
      containers:
        - name: local-path-provisioner
          image: rancher/local-path-provisioner:v0.0.22
          imagePullPolicy: IfNotPresent
          command:
            - local-path-provisioner
            - --debug
            - start
            - --config
            - /etc/config/config.json
```

```
volumeMounts:
- name: config-volume
  mountPath: /etc/config/
env:
- name: POD_NAMESPACE
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
volumes:
- name: config-volume
  configMap:
    name: local-path-config

---
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-path
provisioner: rancher.io/local-path
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Delete

---
kind: ConfigMap
apiVersion: v1
metadata:
  name: local-path-config
  namespace: local-path-storage
data:
  config.json: |-  
  {  
    "nodePathMap": [  
      {  
        "node": "DEFAULT_PATH_FOR_NON_LISTED_NODES",  
        "paths": ["/opt/local-path-provisioner"]  
      }  
    ]  
  }
setup: |-  
#!/bin/sh  
set -eu  
mkdir -m 0777 -p "$VOL_DIR"
teardown: |-  
#!/bin/sh
```

```

set -eu
rm -rf "$VOL_DIR"
helperPod.yaml: |-
  apiVersion: v1
  kind: Pod
  metadata:
    name: helper-pod
  spec:
    containers:
      - name: helper-pod
        image: busybox
        imagePullPolicy: IfNotPresent
```

```

## 虚拟机存储

- 系统盘：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：数据盘是虚拟机中用于存储用户数据、应用程序数据或其他非操作系统相关文件的存储设备。与系统盘相比，数据盘是非必选的，可以根据需要动态添加或移除。  
数据盘的容量可以根据需求进行灵活配置，并且在不中断虚拟机的情况下支持增加磁盘容量。目前，我们依据 StorageClass 中的 AllowVolumeExpansion 参数是否为 true 来判断存储是否支持热扩容。

| NAME                                   | PROVISIONER                   | RECLAIMPOLICY | VOLUMEBINDINGMODE    | ALLOWVOLUMEEXPANSION | AGE |
|----------------------------------------|-------------------------------|---------------|----------------------|----------------------|-----|
| hwameistor-storage-lvm-hdd             | lvm.hwameistor.io             | Delete        | WaitForFirstConsumer | true                 | 94d |
| hwameistor-storage-lvm-hdd-convertible | lvm.hwameistor.io             | Delete        | WaitForFirstConsumer | true                 | 94d |
| rook-ceph-block                        | rook-ceph.rbd.csi.ceph.com    | Delete        | Immediate            | true                 | 94d |
| rook-cephfs                            | rook-ceph.cephfs.csi.ceph.com | Delete        | Immediate            | true                 | 94d |

是否支持热扩容

默认使用块存储。如果需要使用克隆和快照功能，请确保您的存储池已经创建了对应的 VolumeSnapshotClass，可以参考以下示例。如果需要使用实时迁移功能，请确保您的存储支持并选择了 ReadWriteMany 的访问模式。

大多数情况下，存储在安装过程中不会自动创建这样的 VolumeSnapshotClass，因此您需要手动创建 VolumeSnapshotClass。以下是一个 HwameiStor 创建 VolumeSnapshotClass 的示例：

```
kind: VolumeSnapshotClass
apiVersion: snapshot.storage.k8s.io/v1
metadata:
  name: hwameistor-storage-lvm-snapshot
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: "true"
parameters:
  snapsize: "1073741824"
driver: lvm.hwameistor.io
deletionPolicy: Delete
- 执行以下命令检查 VolumeSnapshotClass 是否创建成功。
  kubectl get VolumeSnapshotClass
- 查看已创建的 Snapshotclass，并且确认 Provisioner 属性同存储池中的
  Driver 属性一致。
```

## 虚拟机自动漂移

本文将介绍当集群内某个节点因为断电或网络故障，导致该节点上的虚拟机无法访问时，如何将正在运行的虚拟机无缝迁移到其他的节点上，同时保证业务的连续性和数据的安全性。

与实时迁移相比，自动漂移不需要您在界面中主动操作，而是系统自动触发迁移过程。

### 前提条件

实现自动漂移之前，需要满足以下前提条件：

- 虚拟机未进行磁盘落盘操作，或使用 Rook-Ceph、HwameiStor HA 模式作为存储系统

- 节点失联时间超过五分钟
- 确保集群内至少有两个节点可供使用，并且虚拟机没有指定调度节点
- 虚拟机的 launcher pod 已被删除

## 操作步骤

1. 检查虚拟机 launcher pod 状态：

```
kubectl get pod
```

查看 launcher pod 是否处于 Terminating 状态。

2. 强制删除 launcher pod：

如果 launcher pod 状态为 Terminating，可以执行以下命令进行强制删除：

```
kubectl delete <launcher pod> --force
```

替换 <launcher pod> 为你的 launcher pod 名称。

3. 等待重新创建并检查状态：

删除后，系统将自动重新创建 launcher pod。等待其状态变为 running，然后刷新虚拟机列表，观察虚拟机是否成功迁移到新节点。

## 注意事项

如果使用 rook-ceph 作为存储，需要配置为 ReadWriteOnce 模式：

1. 强制删除 pod 后，需要等待大约六分钟以让 launcher pod 启动，或者可以通过以下命令立即启动 pod：

```
kubectl get pv | grep <vm name>  
kubectl get VolumeAttachment | grep <pv name>
```

替换 <vm name> 和 <pv name> 为你的虚拟机名称和持久卷名称。

2. 然后执行以下命令删除对应的 VolumeAttachment：

```
kubectl delete VolumeAttachment <vm>
```

替换 `<vm>` 为你的虚拟机名称。

# 健康检查

当配置虚拟机的存活 (Liveness) 和就绪 (Readiness) 探针时，与 Kubernetes 的配置过程相似。本文将介绍如何通过 YAML 为虚拟机配置健康检查参数。

但是需要注意：需要在虚拟机创建成功并且处于关机状态下，修改 YAML 进行配置。

## 配置 HTTP Liveness Probe

1. 在 `spec.template.spec` 中配置 `livenessProbe.httpGet`。
2. 修改 `cloudInitNoCloud` 以启动一个 HTTP 服务器。

??? note “点击查看 YAML 示例配置”

```
```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
    virtnest.io/alias-name: "
    virtnest.io/image-secret: "
    virtnest.io/image-source: docker
    virtnest.io/os-image: release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_
64:v1
  creationTimestamp: '2024-10-15T02:39:45Z'
  finalizers:
    - kubevirt.io/virtualMachineControllerFinalize
  generation: 1
  labels:
    virtnest.io/os-family: Ubuntu
    virtnest.io/os-version: '22.04'
  name: test-probe
  namespace: amamba-team
  resourceVersion: '254032135'
```

```
uid: 6d92779d-7415-4721-8c7b-a2dde163d758
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        name: test-probe-rootdisk
        namespace: amamba-team
    spec:
      pvc:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
        storageClassName: hwameistor-storage-lvm-hdd
      source:
        registry:
          url: >-
            docker://release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_64:v1
      runStrategy: Halted
      template:
        metadata:
          creationTimestamp: null
        spec:
          architecture: amd64
          domain:
            cpu:
              cores: 1
              sockets: 1
              threads: 1
            devices:
              disks:
                - bootOrder: 1
                  disk:
                    bus: virtio
                    name: rootdisk
                - disk:
                    bus: virtio
                    name: cloudinitdisk
            interfaces:
              - masquerade: {}
                name: default
          machine:
            type: q35
```

```

memory:
  guest: 2Gi
resources:
  requests:
    memory: 2Gi
networks:
  - name: default
    pod: {}
livenessProbe:
  initialDelaySeconds: 120
  periodSeconds: 20
  httpGet:
    port: 1500
    timeoutSeconds: 10
volumes:
  - dataVolume:
      name: test-probe-rootdisk
    name: rootdisk
  - cloudInitNoCloud:
      userData: |
        #cloud-config
        ssh_pwauth: true
        disable_root: false
        chpasswd: { "list": "root:dangerous", "expire": False}
      runcmd:
        - sed -i "/#\!PermitRootLogin/s/^.*$/PermitRootLogin yes/g" /etc/ssh/
sshd_config
  - systemctl restart ssh.service
  - dhclient -r && dhclient
  - apt-get update && apt-get install -y ncat
    - ["systemd-run", "--unit=httpserver", "ncat", "-klp", "1500", "-e", '/usr/bin/echo -e HTTP/1.1 200 OK\nContent-Length: 12\n\nHello World!']
      name: cloudinitdisk
```

```

3. 根据操作系统（如 Ubuntu/Debian 或 CentOS），userData 的配置可能有所不同。主要区别：

- 包管理器：

Ubuntu/Debian 使用 apt-get 作为包管理器。 CentOS 使用 yum 作为包管理器。

- SSH 服务重启命令：

Ubuntu/Debian 使用 systemctl restart ssh.service。 CentOS 使用 systemctl restart sshd.service (注意 CentOS 7 及之前版本使用 service sshd restart)。

- 安装的软件包：

Ubuntu/Debian 安装 ncat。 CentOS 安装 nmap-ncat (因为 ncat 在 CentOS 的默认仓库中可能不可用)。

## 配置 TCP Liveness Probe

在 spec.template.spec 中配置 livenessProbe.tcpSocket。

??? note “[点击查看 YAML 示例配置](#)”

```
```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
    virtnest.io/alias-name: ""
    virtnest.io/image-secret: ""
    virtnest.io/image-source: docker
    virtnest.io/os-image: release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_64:v1
  creationTimestamp: '2024-10-15T02:39:45Z'
  finalizers:
    - kubevirt.io/virtualMachineControllerFinalize
  generation: 1
  labels:
    virtnest.io/os-family: Ubuntu
    virtnest.io/os-version: '22.04'
  name: test-probe
  namespace: amamba-team
  resourceVersion: '254032135'
  uid: 6d92779d-7415-4721-8c7b-a2dde163d758
spec:
  dataVolumeTemplates:
    - metadata:
```

```
creationTimestamp: null
name: test-probe-rootdisk
namespace: amamba-team
spec:
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 10Gi
    storageClassName: hwameistor-storage-lvm-hdd
  source:
    registry:
      url: >-
        docker://release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_64:v1
  runStrategy: Halted
  template:
    metadata:
      creationTimestamp: null
    spec:
      architecture: amd64
      domain:
      cpu:
        cores: 1
        sockets: 1
        threads: 1
      devices:
      disks:
        - bootOrder: 1
          disk:
            bus: virtio
            name: rootdisk
        - disk:
            bus: virtio
            name: cloudinitdisk
      interfaces:
        - masquerade: {}
          name: default
      machine:
        type: q35
      memory:
        guest: 2Gi
      resources:
        requests:
```

```

        memory: 2Gi
networks:
  - name: default
    pod: {}
livenessProbe:
  initialDelaySeconds: 120
  periodSeconds: 20
  tcpSocket:
    port: 1500
  timeoutSeconds: 10
volumes:
  - dataVolume:
      name: test-probe-rootdisk
      name: rootdisk
  - cloudInitNoCloud:
      userData: |
        #cloud-config
        ssh_pwauth: true
        disable_root: false
        chpasswd: {"list": "root:dangerous", expire: False}
      runcmd:
        - sed -i "/#\!PermitRootLogin/s/^.*$/PermitRootLogin yes/g" /etc/ssh/sshd_conf
ig
        - systemctl restart ssh.service
        - dhclient -r && dhclient
        - apt-get update && apt-get install -y ncat
        - ["systemd-run", "--unit=httpserver", "ncat", "-klp", "1500", "-e", '/usr/bin/ech
o -e HTTP/1.1 200 OK\nContent-Length: 12\n\nHello World!']
      name: cloudinitdisk
```

```

## 配置 Readiness Probes

在 spec.template.spec 中配置 readiness。

??? note “[点击查看 YAML 示例配置](#)”

```

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1

```

```
virtnest.io/alias-name: "
virtnest.io/image-secret: "
virtnest.io/image-source: docker
virtnest.io/os-image: release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_64:v1
creationTimestamp: '2024-10-15T02:39:45Z'
finalizers:
- kubevirt.io/virtualMachineControllerFinalize
generation: 1
labels:
virtnest.io/os-family: Ubuntu
virtnest.io/os-version: '22.04'
name: test-probe
namespace: amamba-team
resourceVersion: '254032135'
uid: 6d92779d-7415-4721-8c7b-a2dde163d758
spec:
dataVolumeTemplates:
- metadata:
  creationTimestamp: null
  name: test-probe-rootdisk
  namespace: amamba-team
spec:
pvc:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 10Gi
storageClassName: hwameistor-storage-lvm-hdd
source:
registry:
url: >-
docker://release-ci.daocloud.io/virtnest/system-images/ubuntu-22.04-x86_64:v1
runStrategy: Halted
template:
metadata:
creationTimestamp: null
spec:
architecture: amd64
domain:
cpu:
cores: 1
sockets: 1
threads: 1
```

```
devices:
disks:
  - bootOrder: 1
    disk:
      bus: virtio
      name: rootdisk
  - disk:
      bus: virtio
      name: cloudinitdisk
interfaces:
  - masquerade: {}
    name: default
machine:
  type: q35
memory:
  guest: 2Gi
resources:
  requests:
    memory: 2Gi
networks:
  - name: default
    pod: {}
readiness:
  initialDelaySeconds: 120
  periodSeconds: 20
  httpGet:
    port: 1500
    timeoutSeconds: 10
volumes:
  - dataVolume:
      name: test-probe-rootdisk
      name: rootdisk
  - cloudInitNoCloud:
      userData: |
        #cloud-config
        ssh_pwauth: true
        disable_root: false
        chpasswd: {"list": "root:dangerous", expire: False}
      runcmd:
        - sed -i "/#\!PermitRootLogin/s/^.*$/PermitRootLogin yes/g" /etc/ssh/sshd_conf
ig
        - systemctl restart ssh.service
        - dhclient -r && dhclient
        - apt-get update && apt-get install -y ncat
```

```
- ["systemd-run", "--unit=httpserver", "ncat", "-klp", "1500", "-e", '/usr/bin/echo -e HTTP/1.1 200 OK\nContent-Length: 12\n\nHello World!']  
name: cloudinitdisk  
--
```

# 虚拟机配置 GPU (直通模式)

本文将介绍如何在创建虚拟机时，配置 GPU 的前提条件。

配置虚拟机的 GPU 的重点是对 GPU Operator 进行配置，以便在工作节点上部署不同的软件组件，具体取决于这些节点上配置运行的 GPU 工作负载。以下三个节点为例：

- controller-node-1 节点配置为运行容器。
- work-node-1 节点配置为运行具有直通 GPU 的虚拟机。
- work-node-2 节点配置为运行具有虚拟 vGPU 的虚拟机。

## 假设、限制和依赖性

工作节点可以运行 GPU 加速容器，也可以运行具有 GPU 直通的 GPU 加速 VM，或者具有 vGPU 的 GPU 加速 VM，但不能运行其中任何一个的组合。

1. 集群管理员或开发人员需要提前了解集群情况，并正确标记节点以指示它们将运行的 GPU 工作负载类型。
2. 运行具有 GPU 直通或 vGPU 的 GPU 加速 VM 的工作节点被假定为裸机，如果工作节点是虚拟机，则需要在虚拟机平台上启用 GPU 直通功能，请向虚拟机平台提供商咨询。
3. 不支持 Nvidia MIG 的 vGPU。
4. GPU Operator 不会自动在 VM 中安装 GPU 驱动程序。

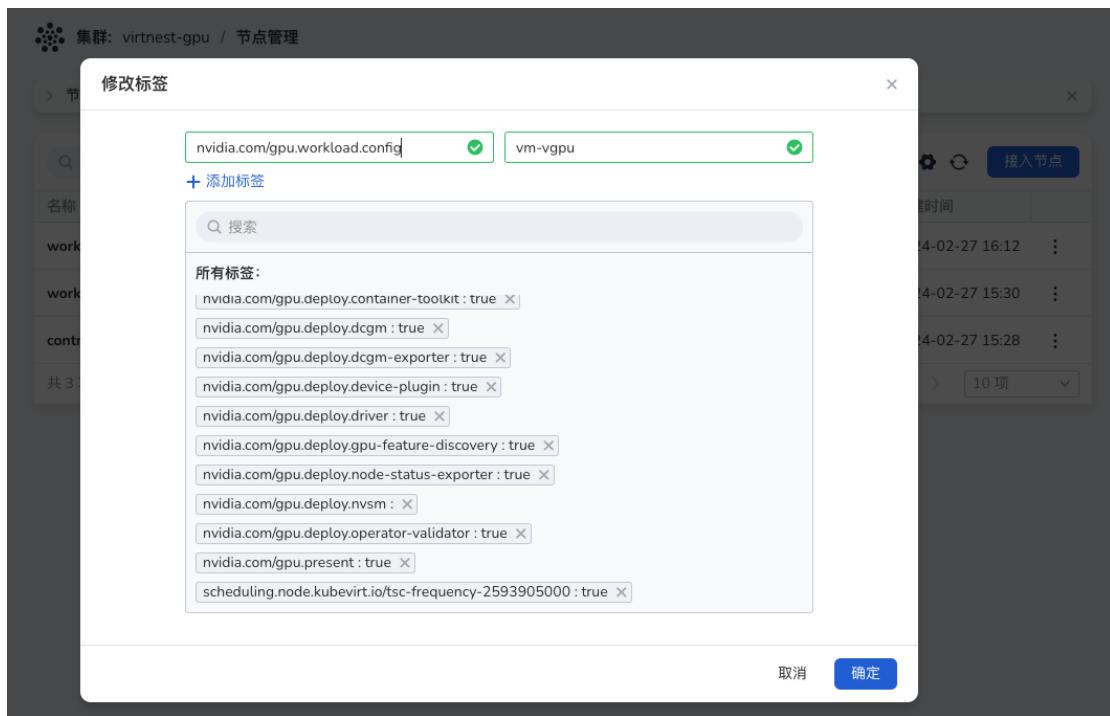
## 启用 IOMMU

为了启用 GPU 直通功能，集群节点需要开启 IOMMU。请参考[如何开启 IOMMU](#)。如果您的集群是在虚拟机上运行，请咨询您的虚拟机平台提供商。

## 标记集群节点

进入 **容器管理**，选取您的工作集群，点击 **节点管理** 的操作栏 **修改标签**，为节点添加标签，每个节点只能有一种标签。

您可以为标签分配以下值：container、vm-passthrough 和 vm-vgpu。



标记标签

## 安装 Nvidia Operator

1. 进入 **容器管理**，选取您的工作集群，点击 **Helm 应用** -> **Helm 模板**，选择并安装 `gpu-operator`。需要修改一些 `yaml` 中的相关字段。

```
gpu-operator.sandboxWorkloads.enabled=true
gpu-operator.vfioManager.enabled=true
gpu-operator.sandboxDevicePlugin.enabled=true
gpu-operator.sandboxDevicePlugin.version=v1.2.4    # (1)#
gpu-operator.toolkit.version=v1.14.3-ubuntu20.04
```

1. version 需要 >= v1.2.4
2. 等待安装成功，如下图所示：

NAME	READY	STATUS	RESTARTS	AGE
gpu-feature-discovery-x7s5t	1/1	Running	0	4m28s
gpu-operator-1706873480-node-feature-discovery-gc-85558fcfg52c	1/1	Running	0	5m
gpu-operator-1706873480-node-feature-discovery-master-5cb62tb4t	1/1	Running	0	5m
gpu-operator-1706873480-node-feature-discovery-worker-st8sz	1/1	Running	0	4m59s
gpu-operator-1706873480-node-feature-discovery-worker-vdjpvt	1/1	Running	0	5m
gpu-operator-1706873480-node-feature-discovery-worker-xbbxr	1/1	Running	0	4m59s
gpu-operator-64c6587b77-s74nf	1/1	Running	0	4m59s
nvidia-container-toolkit-daemonset-pssz4	1/1	Running	0	4m28s
nvidia-cuda-validator-lc566	0/1	Completed	0	2m37s
nvidia-dcgm-exporter-4cdv7	1/1	Running	0	4m28s
nvidia-device-plugin-daemonset-k2cdr	1/1	Running	0	4m28s
nvidia-driver-daemonset-5p6sp	1/1	Running	0	4m38s
nvidia-operator-validator-9fqhs	1/1	Running	0	4m28s
nvidia-sandbox-device-plugin-daemonset-75dfn	1/1	Running	0	4m2s
nvidia-sandbox-device-plugin-daemonset-lhk9	1/1	Running	0	84s
nvidia-sandbox-validator-prqhp	1/1	Running	0	4m2s
nvidia-sandbox-validator-xzrx9	1/1	Running	0	84s
nvidia-vfio-manager-2tkvg	1/1	Running	0	4m38s
nvidia-vgpu-device-manager-95hvc	1/1	Running	0	4m3s
nvidia-vgpu-manager-daemonset-zqpgg	1/1	Running	0	4m38s

安装成功

## 安装 virtnest-agent 并配置 CR

1. 安装 virtnest-agent，参考[安装 virtnest-agent](#)。
2. 将 vGPU 和 GPU 直通加入 Virtnest Kubevirt CR，以下示例是添加 vGPU 和 GPU 直通后的部分关键 yaml：

```
spec:
  configuration:
    developerConfiguration:
      featureGates:
        - GPU
        - DisableMDEVConfiguration
    permittedHostDevices: # (1)#
      mediatedDevices:          # (2)#
        - mdevNameSelector: GRID P4-1Q
      resourceName: nvidia.com /GRID_P4-1Q
```

```

pciHostDevices:          # (3)!

- externalResourceProvider: true
  pciVendorSelector: 10DE:1BB3
  resourceName: nvidia.com/GP104GL_TESLA_P4

```

1. 下面是需要填写的信息
  2. vGPU
  3. GPU 直通
3. 在 kubevirt CR yaml 中, permittedHostDevices 用于导入 VM 设备, vGPU 需在其中添加 mediatedDevices, 具体结构如下:

```

mediatedDevices:
- mdevNameSelector: GRID_P4-1Q      # (1)!

resourceName: nvidia.com/GRID_P4-1Q # (2)!
```

1. 设备名称
  2. GPU Operator 注册到节点的 vGPU 信息
4. GPU 直通需要在 permittedHostDevices 下添加 pciHostDevices, 具体结构如下:

```

pciHostDevices:
- externalResourceProvider: true      # (1)!

  pciVendorSelector: 10DE:1BB3        # (2)!

  resourceName: nvidia.com/GP104GL_TESLA_P4 # (3)!
```

1. 默认不要更改
  2. 当前 pci 设备的 vendor id
  3. GPU Operator 注册到节点的 GPU 信息
5. 获取 vGPU 信息示例 (仅适用于 vGPU) : 在标记为 nvidia.com/gpu.workload.config=vm-gpu 的节点 (例如 work-node-2) 上查看节点信息,

Capacity 中的 nvidia.com/GRID\_P4-1Q: 8 表示可用 vGPU:

```
kubectl describe node work-node-2
```

Capacity:

cpu:	64
devices.kubevirt.io/kvm:	1k
devices.kubevirt.io/tun:	1k
devices.kubevirt.io/vhost-net:	1k

```

ephemeral-storage:           102626232Ki
hugepages-1Gi:              0
hugepages-2Mi:              0
memory:                     264010840Ki
nvidia.com/GRID_P4-1Q :     8
pods:                       110

Allocatable:
cpu:                         64
devices.kubevirt.io/kvm:      1k
devices.kubevirt.io/tun:       1k
devices.kubevirt.io/vhost-net: 1k
ephemeral-storage:            94580335255
hugepages-1Gi:                0
hugepages-2Mi:                0
memory:                      263908440Ki
nvidia.com/GRID_P4-1Q:        8
pods:                        110

```

那么 mdevNameSelector 应该是 “GRID P4-1Q”, resourceName 应该是 “GRID\_P4-1Q”

6. 获取 GPU 直通信息：在标记 nvidia.com/gpu.workload.config=vm-passthrough 的 node 上（本文档示例 node 为 work-node-1），查看 node 信息，Capacity 中 nvidia.com/GP104GL\_TESLA\_P4: 2 就是可用 vGPU：

```

kubectl describe node work-node-1
Capacity:
cpu:                         64
devices.kubevirt.io/kvm:       1k
devices.kubevirt.io/tun:        1k
devices.kubevirt.io/vhost-net:  1k
ephemeral-storage:             102626232Ki
hugepages-1Gi:                 0
hugepages-2Mi:                 0
memory:                      264010840Ki
nvidia.com/GP104GL_TESLA_P4:   2
pods:                        110

Allocatable:
cpu:                         64
devices.kubevirt.io/kvm:       1k
devices.kubevirt.io/tun:        1k
devices.kubevirt.io/vhost-net:  1k
ephemeral-storage:            94580335255
hugepages-1Gi:                0

```

```

hugepages-2Mi: 0
memory: 263908440Ki
nvidia.com/GP104GL_TESLA_P4: 2
pods: 110

```

那么 resourceName 应该是 “GRID\_P4-1Q”, 如何获取 pciVendorSelector 呢? 通过 ssh 登录到 work-node-1 目标节点, 通过 lspci -nnk -d 10de: 命令获取 Nvidia GPU PCI 信息, 如下所示: 红框所示即是 pciVendorSelector 信息。

```

root@virtnest-master-02:~# lspci -nnk -d 10de:
04:00.0 3D controller [0302]: NVIDIA Corporation GP104GL [Tesla P4] [10de:1bb3] (rev a1)
    Subsystem: NVIDIA Corporation GP104GL [Tesla P4] [10de:11d8]
    Kernel driver in use: vfio-pci
    Kernel modules: nvidiafb, nouveau
82:00.0 3D controller [0302]: NVIDIA Corporation GP104GL [Tesla P4] [10de:1bb3] (rev a1)
    Subsystem: NVIDIA Corporation GP104GL [Tesla P4] [10de:11d8]
    Kernel driver in use: vfio-pci
    Kernel modules: nvidiafb, nouveau

```

获取 pciVendorSelector

7. 编辑 kubevirt CR 提示: 如果同一型号 GPU 有多个, 只需在 CR 中写入一个即可, 无需列出每个 GPU。

```

kubectl -n virtnest-system edit kubevirt kubevirt
spec:
  configuration:
    developerConfiguration:
      featureGates:
        - GPU
        - DisableMDEVConfiguration
  permittedHostDevices: # (1)!
    mediatedDevices: # (2)!
      - mdevNameSelector: GRID_P4-1Q
        resourceName: nvidia.com/GRID_P4-1Q
    pciHostDevices: # (3)!
      - externalResourceProvider: true
        pciVendorSelector: 10DE:1BB3
        resourceName: nvidia.com/GP104GL_TESLA_P4

```

1. 下面是需要填写的信息
2. vGPU
3. GPU 直通, 上面的示例中 TESLA P4 有两个 GPU, 这里只需要注册一个即可

## 通过 YAML 创建 VM 并使用 GPU 加速

与普通虚拟机唯一的区别是在 devices 中添加 GPU 相关信息。

??? note “[点击查看完整 YAML](#)”

```
```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm-gpu1
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        name: systemdisk-testvm-gpu1
        namespace: default
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 10Gi
    storageClassName: www
  source:
    registry:
      url: docker://release-ci.daocloud.io/virtnest/system-images/debian-12-x86_64:v1
  runStrategy: Manual
  template:
    metadata:
      creationTimestamp: null
    spec:
      domain:
        cpu:
          cores: 1
          sockets: 1
          threads: 1
        devices:
          disks:
            - bootOrder: 1
              disk:
                bus: virtio
```

```

    name: systemdisk-testvm-gpu1
    - disk:
        bus: virtio
        name: cloudinitdisk
    gpus:
    - deviceName: nvidia.com/GP104GL_TESLA_P4
        name: gpu-0-0
    - deviceName: nvidia.com/GP104GL_TESLA_P4
        name: gpu-0-1
    interfaces:
    - masquerade: {}
        name: default
    machine:
        type: q35
    resources:
        requests:
            memory: 2Gi
    networks:
    - name: default
        pod: {}
    volumes:
    - dataVolume:
        name: systemdisk-testvm-gpu1
    name: systemdisk-testvm-gpu1
    - cloudInitNoCloud:
        userDataBase64: I2Nsb3VkLWNvbmZpZwpzc2hfcHdhdXR0OjB0cnVlCmRpC2FibGVfcm
        9vdDogZmFsc2UKY2hwYXNzd2Q6IHsibGlzdCI6ICJyb290OmRhbmldcm91cyIsIGV4cGlyZTogRm
        Fsc2V9CgoKcnVuY21kOgogIC0gc2VkJC1pICIVI1w/UGVybWl0Um9vdExvZ2luL3MvXi4qJC9QZX
        JtaXRSb290TG9naW4geWVzL2ciIC9ldGMvc3NoL3NzaGRfY29uZmlnCiAgLSBzeXN0ZW1jdGwg
        cmVzdGFydCBzc2guc2VydmljZQ==
        name: cloudinitdisk
    ...

```

## 虚拟机配置 GPU (vGPU)

本文将介绍如何在创建虚拟机时，配置 GPU 的前提条件。

配置虚拟机的 GPU 的重点是对 GPU Operator 进行配置，以便在工作节点上部署不同的软件组件，具体取决于这些节点上配置运行的 GPU 工作负载。以下三个节点为例：

- controller-node-1 节点配置为运行容器。

- work-node-1 节点配置为运行具有直通 GPU 的虚拟机。
- work-node-2 节点配置为运行具有虚拟 vGPU 的虚拟机。

## 假设、限制和依赖性

工作节点可以运行 GPU 加速容器，也可以运行具有 GPU 直通的 GPU 加速 VM，或者具有 vGPU 的 GPU 加速 VM，但不能运行其中任何一个的组合。

1. 工作节点可以单独运行 GPU 加速容器、具有 GPU 直通的 GPU 加速 VM，或者具有 vGPU 的 GPU 加速 VM，不支持任何组合形式。
2. 集群管理员或开发人员需要提前了解集群情况，并正确标记节点以指示它们将运行的 GPU 工作负载类型。
3. 运行具有 GPU 直通或 vGPU 的 GPU 加速 VM 的工作节点被假定为裸机，如果工作节点是虚拟机，则需要在虚拟机平台上启用 GPU 直通功能，请向虚拟机平台提供商咨询。
4. 不支持 Nvidia MIG 的 vGPU。
5. GPU Operator 不会自动在 VM 中安装 GPU 驱动程序。

## 启用 IOMMU

为了启用 GPU 直通功能，集群节点需要开启 IOMMU。请参考[如何开启 IOMMU](#)。如果您的集群是在虚拟机上运行，请咨询您的虚拟机平台提供商。

## 构建 vGPU Manager 镜像

注意：仅当使用 NVIDIA vGPU 时才需要构建 vGPU Manager 镜像。如果您计划仅使用 GPU 直通，请跳过此部分。

以下是构建 vGPU Manager 镜像并将其推送到镜像仓库中的步骤：

1. 从 NVIDIA Licensing Portal 下载 vGPU 软件。
  - 登录 NVIDIA Licensing Portal，转到 Software Downloads 页面。
  - NVIDIA vGPU 软件位于 Software Downloads 页面的 Driver downloads 选项卡中。
  - 在筛选条件中选择 VGPU + Linux，点击 下载 以获取 Linux KVM 的软件包。

请解压下载的文件 (NVIDIA-Linux-x86\_64-<version>-vgpu-kvm.run)。

The screenshot shows the 'Software Downloads' section of the DaoCloud Enterprise 5.0 interface. The left sidebar has 'SOFTWARE DOWNLOADS' selected. The main area shows a table of vGPU drivers for Linux KVM. The columns are: PLATFORM, PLATFORM VERSION, PRODUCT VERSION, DESCRIPTION, RELEASE DATE, and DOWNLOAD. The table lists various versions from 7.5 down to 5.0, each with a 'Download' link.

PLATFORM	PLATFORM VERSION	PRODUCT VERSION	DESCRIPTION	RELEASE DATE	DOWNLOAD
Linux KVM	7.5	6.0	NVIDIA vGPU for Linux KVM 7.5	Mar 27, 2018	Download
Linux KVM	All Supported	6.1	NVIDIA vGPU for Linux KVM ALL	May 15, 2018	Download
Linux KVM	All Supported	6.2	NVIDIA vGPU for Linux KVM ALL	Jul 6, 2018	Download
Linux KVM	All Supported	6.3	NVIDIA vGPU for Linux KVM ALL	Oct 5, 2018	Download
Linux KVM	All Supported	6.4	NVIDIA vGPU for Linux KVM ALL	Feb 22, 2019	Download
Linux KVM	All Supported	5.0	NVIDIA vGPU for Linux KVM ALL	Sep 1, 2017	Download
Linux KVM	All Supported	5.1	NVIDIA vGPU for Linux KVM ALL	Nov 10, 2017	Download
Linux KVM	All Supported	5.2	NVIDIA vGPU for Linux KVM ALL	Jan 17, 2018	Download
Linux KVM	All Supported	5.3	NVIDIA vGPU for Linux KVM ALL	May 9, 2018	Download
Linux KVM	All Supported	5.4	NVIDIA vGPU for Linux KVM ALL	Aug 3, 2018	Download

### 下载 vGPU 软件

2. 打开终端克隆 container-images/driver 仓库

```
git clone https://gitlab.com/nvidia/container-images/driver cd driver
```

3. 切换到您的操作系统对应 vgpu-manager 目录

```
cd vgpu-manager/<your-os>
```

4. 将步骤 1 中提取的 .run 文件 copy 到当前目录

```
cp <local-driver-download-directory>/*-vgpu-kvm.run ./
```

5. 设置环境变量

- PRIVATE\_REGISTRY：专用注册表的名称，用于存储驱动程序映像。

- VERSION: NVIDIA vGPU 管理器的版本，从 NVIDIA 软件门户下载。
- OS\_TAG: 必须与集群节点操作系统版本匹配。
- CUDA\_VERSION: 用于构建驱动程序映像的 CUDA 基本映像版本。

```
export PRIVATE_REGISTRY=my/private/registry VERSION=510.73.06 OS_TAG=ubuntu22.04 CUDA_VERSION=12.2.0
```

#### 6. 构建 NVIDIA vGPU Manager Image

```
docker build \
--build-arg DRIVER_VERSION=${VERSION} \
--build-arg CUDA_VERSION=${CUDA_VERSION} \
-t ${PRIVATE_REGISTRY}/vgpu-manager:${VERSION}-${OS_TAG} .
```

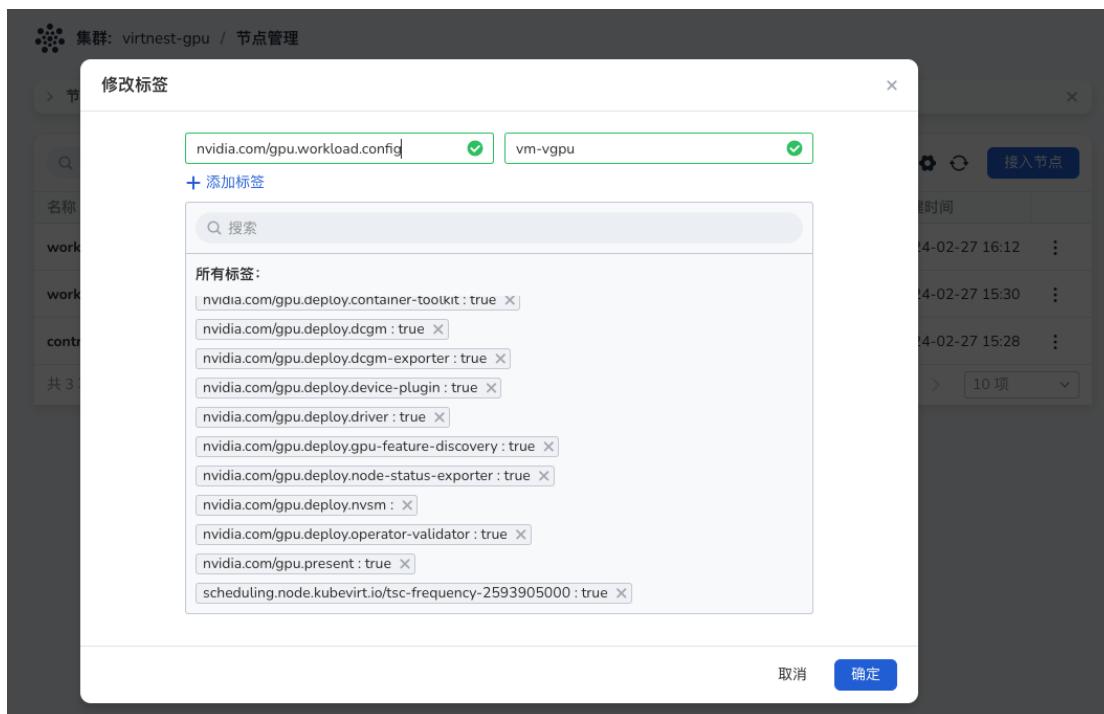
#### 7. 将 NVIDIA vGPU Manager 映像推送到您的镜像仓库

```
docker push ${PRIVATE_REGISTRY}/vgpu-manager:${VERSION}-${OS_TAG}
```

## 标记集群节点

进入 **容器管理**，选取您的工作集群，然后点击 **节点管理**，进入列表页面。点击列表右侧的 **！**，选择 **修改标签**，为节点添加标签，每个节点只能有一种标签。

您可以为标签分配以下值：container、vm-passthrough 和 vm-vgpu。



## 标记标签

# 安装 Nvidia Operator

- 进入 容器管理，选取您的工作集群，点击 Helm 应用 -> Helm 模板，选择并安装 gpu-operator。需要修改一些 yaml 中的相关字段。

```
gpu-operator.sandboxWorkloads.enabled=true
gpu-operator.vgpuManager.enabled=true
gpu-operator.vgpuManager.repository=<your-register-url>      # (1)!
gpu-operator.vgpuManager.image=vgpu-manager
gpu-operator.vgpuManager.version=<your-vgpu-manager-version> # (2)!
gpu-operator.vgpuDeviceManager.enabled=true
```

- “构建 vGPU Manager 镜像”步骤中的镜像仓库地址
- “构建 vGPU Manager 镜像”步骤中的 VERSION

- 等待安装成功，如下图所示：

NAME	READY	STATUS	RESTARTS	AGE
gpu-feature-discovery-x7s5t	1/1	Running	0	4m28s
gpu-operator-1706873480-node-feature-discovery-gc-85558fcfpg52c	1/1	Running	0	5m
gpu-operator-1706873480-node-feature-discovery-master-5cb62tb4t	1/1	Running	0	5m
gpu-operator-1706873480-node-feature-discovery-worker-st8sz	1/1	Running	0	4m59s
gpu-operator-1706873480-node-feature-discovery-worker-vdjpv	1/1	Running	0	5m
gpu-operator-1706873480-node-feature-discovery-worker-xbbxr	1/1	Running	0	4m59s
gpu-operator-64c6587b77-s74nf	1/1	Running	0	4m59s
nvidia-container-toolkit-daemonset-pssz4	1/1	Running	0	4m28s
nvidia-cuda-validator-lc566	0/1	Completed	0	2m37s
nvidia-dcgm-exporter-4cdv7	1/1	Running	0	4m28s
nvidia-device-plugin-daemonset-k2cdr	1/1	Running	0	4m28s
nvidia-driver-daemonset-5p6sp	1/1	Running	0	4m38s
nvidia-operator-validator-9fqhs	1/1	Running	0	4m28s
nvidia-sandbox-device-plugin-daemonset-75dfn	1/1	Running	0	4m2s
nvidia-sandbox-device-plugin-daemonset-lhkdk9	1/1	Running	0	84s
nvidia-sandbox-validator-prqhp	1/1	Running	0	4m2s
nvidia-sandbox-validator-xzrx9	1/1	Running	0	84s
nvidia-vfio-manager-2tkvg	1/1	Running	0	4m38s
nvidia-vgpu-device-manager-95hvc	1/1	Running	0	4m3s
nvidia-vgpu-manager-daemonset-zqpfgg	1/1	Running	0	4m38s

安装成功

# 安装 virtnest-agent 并配置 CR

- 安装 virtnest-agent，参考[安装 virtnest-agent](#)。

2. 将 vGPU 和 GPU 直通加入 Virtnest Kubevirt CR, 以下示例是添加 vGPU 和 GPU 直通后的部分关键 yaml:

```
spec:
  configuration:
    developerConfiguration:
      featureGates:
        - GPU
        - DisableMDEVConfiguration
  permittedHostDevices: # (1)!
    mediatedDevices: # (2)!
      - mdevNameSelector: GRID_P4-1Q
        resourceName: nvidia.com /GRID_P4-1Q
    pciHostDevices: # (3)!
      - externalResourceProvider: true
        pciVendorSelector: 10DE:1BB3
        resourceName: nvidia.com /GP104GL_TESLA_P4
```

1. 下面是需要填写的信息
  2. vGPU
  3. GPU 直通
3. 在 kubevirt CR yaml 中, permittedHostDevices 用于导入 VM 设备, vGPU 需在其中添加 mediatedDevices, 具体结构如下:

```
mediatedDevices:
  - mdevNameSelector: GRID_P4-1Q # (1)!
    resourceName: nvidia.com/GRID_P4-1Q # (2)!
```

1. 设备名称
  2. GPU Operator 注册到节点的 vGPU 信息
4. GPU 直通需要在 permittedHostDevices 下添加 pciHostDevices, 具体结构如下:

```
pciHostDevices:
  - externalResourceProvider: true # (1)
    pciVendorSelector: 10DE:1BB3 # (2)
    resourceName: nvidia.com/GP104GL_TESLA_P4 # (3)!
```

1. 默认不要更改
2. 当前 pci 设备的 vendor id

### 3. GPU Operator 注册到节点的 GPU 信息

5. 获取 vGPU 信息示例（仅适用于 vGPU）：在标记为 nvidia.com/gpu.workload.config=vm-gpu 的节点（例如 work-node-2）上查看节点信息，Capacity 中的 nvidia.com/GRID\_P4-1Q: 8 表示可用 vGPU：

```
kubectl describe node work-node-2
Capacity:
cpu: 64
devices.kubevirt.io/kvm: 1k
devices.kubevirt.io/tun: 1k
devices.kubevirt.io/vhost-net: 1k
ephemeral-storage: 102626232Ki
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 264010840Ki
nvidia.com/GRID_P4-1Q : 8
pods: 110

Allocatable:
cpu: 64
devices.kubevirt.io/kvm: 1k
devices.kubevirt.io/tun: 1k
devices.kubevirt.io/vhost-net: 1k
ephemeral-storage: 94580335255
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 263908440Ki
nvidia.com/GRID_P4-1Q: 8
pods: 110
```

那么 mdevNameSelector 应该是 “GRID P4-1Q”，resourceName 应该是 “GRID\_P4-1Q”

6. 获取 GPU 直通信息：在标记 nvidia.com/gpu.workload.config=vm-passthrough 的 node 上（本文档示例 node 为 work-node-1），查看 node 信息，Capacity 中 nvidia.com/GP104GL\_TESLA\_P4: 2 就是可用 vGPU：

```
kubectl describe node work-node-1
Capacity:
cpu: 64
devices.kubevirt.io/kvm: 1k
devices.kubevirt.io/tun: 1k
```

```

devices.kubevirt.io/vhost-net: 1k
ephemeral-storage: 102626232Ki
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 264010840Ki
nvidia.com/GP104GL_TESLA_P4: 2
pods: 110

Allocatable:
cpu: 64
devices.kubevirt.io/kvm: 1k
devices.kubevirt.io/tun: 1k
devices.kubevirt.io/vhost-net: 1k
ephemeral-storage: 94580335255
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 263908440Ki
nvidia.com/GP104GL_TESLA_P4: 2
pods: 110

```

那么 resourceName 应该是 “GRID\_P4-1Q”，如何获取 pciVendorSelector 呢？通过 ssh 登录到 work-node-1 目标节点，通过 lspci -nnk -d 10de: 命令获取 Nvidia GPU PCI 信息，如下所示：红框所示即是 pciVendorSelector 信息。

```

root@virtnest-master-02:~# lspci -nnk -d 10de:
04:00.0 3D controller [0302]: NVIDIA Corporation GP104GL [Tesla P4] [10de:1bb3] (rev a1)
    Subsystem: NVIDIA Corporation GP104GL [Tesla P4] [10de:11d8]
    Kernel driver in use: vfio-pci
    Kernel modules: nvidiafb, nouveau
82:00.0 3D controller [0302]: NVIDIA Corporation GP104GL [Tesla P4] [10de:1bb3] (rev a1)
    Subsystem: NVIDIA Corporation GP104GL [Tesla P4] [10de:11d8]
    Kernel driver in use: vfio-pci
    Kernel modules: nvidiafb, nouveau

```

### 获取 pciVendorSelector

- 编辑 kubevirt CR 提示：如果同一型号 GPU 有多个，只需在 CR 中写入一个即可，无需列出每个 GPU。

```

kubectl -n virtnest-system edit kubevirt kubevirt
spec:
  configuration:
    developerConfiguration:
      featureGates:
        - GPU
        - DisableMDEVConfiguration
  permittedHostDevices: # (1)!
```

```

mediatedDevices: # (2)!

- mdevNameSelector: GRID_P4-1Q
  resourceName: nvidia.com/GRID_P4-1Q

pciHostDevices: # (3)!

- externalResourceProvider: true
  pciVendorSelector: 10DE:1BB3
  resourceName: nvidia.com/GP104GL_TESLA_P4

```

1. 下面是需要填写的信息
2. vGPU
3. GPU 直通, 上面的示例中 TESLA P4 有两个 GPU, 这里只需要注册一个即可

## 通过 YAML 创建 VM 并使用 GPU 加速

与普通虚拟机唯一的区别是在 devices 中添加 gpu 相关信息。

??? note “[点击查看完整 YAML](#)”

```

```yaml
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm-gpu1
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        name: systemdisk-testvm-gpu1
        namespace: default
  spec:
    pvc:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 10Gi
      storageClassName: www
    source:
      registry:
        url: docker://release-ci.daocloud.io/virtnest/system-images/debian-12-x86_64:v1

```

```
runStrategy: Manual
template:
  metadata:
    creationTimestamp: null
  spec:
    domain:
      cpu:
        cores: 1
        sockets: 1
        threads: 1
      devices:
        disks:
          - bootOrder: 1
            disk:
              bus: virtio
              name: systemdisk-testvm-gpu1
          - disk:
              bus: virtio
              name: cloudinitdisk
      gpus:
        - deviceName: nvidia.com/GP104GL_TESLA_P4
          name: gpu-0-0
        - deviceName: nvidia.com/GP104GL_TESLA_P4
          name: gpu-0-1
      interfaces:
        - masquerade: {}
          name: default
    machine:
      type: q35
    resources:
      requests:
        memory: 2Gi
    networks:
      - name: default
        pod: {}
    volumes:
      - dataVolume:
          name: systemdisk-testvm-gpu1
        name: systemdisk-testvm-gpu1
      - cloudInitNoCloud:
          userDataBase64: I2Nsb3VkLWNvbmZpZwpzc2hfcHdhdXR0OjB0cnVlCmRpC2FibGVfcm9vdDogZmFsc2UKY2hwYXNzd2Q6IHsibGlzdCl6ICJyb290OmRhbndlcm91cyIsIGV4cGlyZTogRmFsc2V9CgoKcnVuY21kOgogIC0gc2VkJC1pICIVI1w/UGVybWI0Um9vdExvZ2luL3MvXi4qJC9QZXJtaXRSb290TG9naW4geWVzL2ciIC9ldGMvc3NoL3NzaGRfY29uZmlnCiAgLSBzeXN0ZW1jdGwg
```

```
cmVzdGFydCBzc2guc2VydmljZQ==  
  name: cloudinitdisk  
  ...
```

# 通过模板创建虚拟机

本文将介绍如何通过模板创建虚拟机。

通过内置模板和自定义模板，用户可以轻松创建新的虚拟机。此外，我们还提供将现有虚拟机转换为虚拟机模板的功能，让用户能够更加灵活地管理和使用资源。

## 模板创建

参考以下步骤，使用模板创建一个虚拟机。

1. 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，进入 **虚拟机管理** 页面。在虚拟机列表页面，点击创建虚拟机-选择模板创建虚拟机。

[虚拟机模板创建](#)

[虚拟机模板创建](#)

2. 进入镜像创建页面，依次填写基本信息、模板配置、存储与网络、登录设置后，在页面右下角点击 **确定** 完成创建。

系统将自动返回虚拟机列表。点击列表右侧的 ，可以对虚拟机执行关机/开启、

重启、克隆、更新、创建快照、配置转换为模板、控制台访问 (VNC)、删除等操作。

克隆和快照能力依赖于存储池的选择。

[虚拟机操作](#)

[虚拟机操作](#)

## 基本信息

在创建虚拟机页面中，根据下表输入信息后，点击 **下一步**。

### 虚拟机基础信息

#### 虚拟机基础信息

- 名称：最多包含 63 个字符，只能包含小写字母、数字及分隔符（“-”），且必须以小写字母或数字开头及结尾。同一命名空间内名称不得重复，而且名称在虚拟机创建好之后不可更改。
- 别名：允许任何字符，最长 60 个字符。
- 集群：选择将新建的虚拟机部署在哪个集群内。
- 命名空间：选择将新建的虚拟机部署在哪个命名空间。找不到所需的命名空间时可以根据页面提示去[创建新的命名空间](#)。

## 模板配置

出现模板列表，按需选择内置模板/自定义模板。

- 选择内置模板：平台内置了 2 个标准模板，不允许编辑和删除。选择内置模板后，镜像来源、操作系统、镜像地址等将使用模板内的信息，无法修改；资源配额也将使用模板内的信息，允许修改。

### 内置模板

#### 内置模板

### 内置模板

#### 内置模板

- 选择自定义模板：由虚拟机配置转化而来的模板，支持编辑和删除。使用自定义模

板则根据具体情况支持修改镜像来源等信息。

### 使用镜像仓库

使用镜像仓库

### 使用镜像仓库

使用镜像仓库

## 存储与网络配置

### 存储与网络配置

#### 存储与网络配置

- 存储：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。

默认使用块存储。如果需要使用克隆和快照功能，请确保您的存储池支持 VolumeSnapshots 功能，并在存储池（SC）中进行创建。请注意，存储池（SC）还有其他一些先决条件需要满足。

- 先决条件：

- KubeVirt 利用 Kubernetes CSI 驱动程序的 VolumeSnapshot 功能来捕获持久化虚拟机状态。因此，您需要确保您的虚拟机使用由支持 VolumeSnapshots 的 StorageClass 并配置了正确的 VolumeSnapshotClass。
- 查看已创建的 Snapshotclass，并且确认 provisioner 属性同存储池中的 Driver 属性一致。
- 支持添加一块系统盘和多块数据盘。
- 网络：若您不做任何配置，系统将默认创建一个 VirtIO 类型的网络。

## 登录设置

- 用户名/密码：可以通过用户名和密码登录至虚拟机。
- SSH：选择 SSH 登录方式时可为虚拟机绑定 SSH 密钥，用于日后登录虚拟机。

### 登录设置

#### 登录设置

## 虚拟机模板

本文将介绍内置虚拟机模板和自定义虚拟机模板。

通过内置模板和自定义模板，用户可以轻松创建新的虚拟机。此外，我们还提供将现有虚拟机转换为虚拟机模板的功能，让用户能够更加灵活地管理和使用资源。

## 虚拟机模板

1. 点击左侧导航栏上的 **容器管理**，然后点击 **虚拟机模板**，进入 **虚拟机模板** 页面，若该模板是由配置了 GPU 的虚拟机转换而来，模板也会带有 GPU 的信息，将在模板列表中展示。

The screenshot shows the 'Virtual Machine Template' section of the DaoCloud interface. On the left, there's a sidebar with 'Virtual Machine' and 'Virtual Machine Template' options. The main area displays a table of templates with columns for Name, Operation System, Type, CPU, Memory, and Creation Time. The 'ubuntu' template is selected, indicated by a blue border around its row. A modal window titled 'Virtual Machine Template' is open over the table, containing a search bar and a table with the same columns as the main list.

名称	操作系统	类型	CPU	内存	创建时间
test-status	Debian	自定义模板	1 Core	2Gi	2024-05-31 10:01
vgpu2	Debian	自定义模板	1 Core	2Gi	2024-04-26 15:06
test-bridge-multi	Debian	自定义模板	1 Core	2Gi	2024-04-16 17:33
vm-test-yiting	-	自定义模板	不限制	61.04Mi	2024-04-11 11:49
list-disks	Debian	自定义模板	1 Core	2Gi	2024-04-01 15:45
test-disks	ubuntu	自定义模板	2 Core	3.73Gi	2024-04-01 15:44
vgpu	Debian	自定义模板	1 Core	2Gi	2024-03-28 16:54
ubuntu	Ubuntu	内置模板	1 Core	1Gi	2024-03-27 18:07
centos7	CentOS	内置模板	1 Core	1Gi	2024-03-27 18:07
test-1111	Debian	自定义模板	1 Core	2Gi	2024-03-19 15:55

## 虚拟机模板

- 点击列表右侧的 ，可以对内置模板执行创建虚拟机和查看 YAML 操作；对自定义模板支持创建虚拟机、编辑 YAML 和删除操作。

This screenshot shows the same 'Virtual Machine Template' list as the previous one, but with a context menu open over the 'ubuntu' template. The menu items 'Create VM' and 'View YAML' are highlighted with a red box. The rest of the interface and data table remain the same.

## 内置模板操作

名称	操作系统	类型	CPU	内存	创建时间	操作
test-status	Debian	自定义模板	1 Core	2Gi	2024-05-31 10:01	⋮
vgpu2	vGPU	自定义模板	1 Core	2Gi	2024-04-26 15:06	⋮
test-bridge-multi	Debian	自定义模板	1 Core	2Gi	2024-04-1	⋮
vm-test-yiting	-	自定义模板	不限制	61.04Mi	2024-04-1	⋮
list-disks	Debian	自定义模板	1 Core	2Gi	2024-04-01 15:45	⋮
test-disks	ubuntu	自定义模板	2 Core	3.73Gi	2024-04-01 15:44	⋮
vgpu	vGPU	自定义模板	1 Core	2Gi	2024-03-28 16:54	⋮
ubuntu	Ubuntu	内置模板	1 Core	1Gi	2024-03-27 18:07	⋮
centos7	CentOS	内置模板	1 Core	1Gi	2024-03-27 18:07	⋮
test-1111	Debian	自定义模板	1 Core	2Gi	2024-03-19 15:55	⋮

## 自定义模板操作

```

1 kind: VMTemplate
2 apiVersion: virtnest.io/v1alpha1
3 metadata:
4   name: vgpu2
5   namespace: virtnest-system
6   uid: 2c9a1d01-4204-4f9f-85fe-aabbe87ae16
7   resourceVersion: '1868674492'
8   generation: 2
9   creationTimestamp: '2024-04-26T07:06:38Z'
10 annotations:
11   | aaa: aaa
12   |   virtnest.io/description: ''
13 spec:
14   type: custom
15   vm:
16     kind: VirtualMachine
17     apiVersion: kubevirt.io/v1
18     metadata:
19       name: test-vgpu-1
20       namespace: default
21       creationTimestamp: null
22     labels:
23       fasdfadsf: fasdfasdf
24       virtnest.io/os-family: Debian
25       virtnest.io/os-version: '12'
26       virtnest.io/vmtemplate: vgpu2
27     annotations:
28       kubevirt.io/latest-observed-api-version: v1
29       kubevirt.io/storage-observed-api-version: v1

```

## 编辑自定义模板

## 内置模板

- 平台内内置两种模板，分别是 CentOS 和 Ubuntu。

名称	操作系统	类型	CPU	内存	创建时间
centos7	CentOS	内置模板	1 Core	1Gi	2023-11-08 14:45
ubuntu	Ubuntu	内置模板	1 Core	1Gi	2023-11-08 14:45
test-moban	CentOS	自定义模板	1 Core	1Gi	2024-01-30 16:25
zhengguang-centos7	CentOS	自定义模板	4 Core	8Gi	2024-01-23 17:30
windows-2012r2	Windows	自定义模板	4 Core	1.86Gi	2023-12-07 17:03
gpu-template	Ubuntu	自定义模板	1 Core	1Gi	2023-11-28 13:28
test-qianyi-temp2	Debian	自定义模板	不限制	2Gi	2023-10-23 17:53
processing-template	-	自定义模板	2 Core	1Gi	2023-10-23 16:37
test	Debian	自定义模板	不限制	2Gi	2023-10-23 14:41
123123	Debian	自定义模板	不限制	2Gi	2023-10-23 11:07

内置模板

## 自定义模板

自定义模板是由虚拟机配置转化而来的模板。以下介绍如何从虚拟机配置转换为模板。

1. 从左侧导航栏，依次点击 **虚拟机** -> **虚拟机**，进入列表页面，点击列表右侧的

支持将配置转换为模板。只有运行中/关闭状态下的虚拟机支持转化。

**点击转化为模板**

**点击转化为模板**

2. 填写新模板的名称，提示原始虚拟机将会保留并且可用。转换成功后，将会在模板列表新增一条数据。

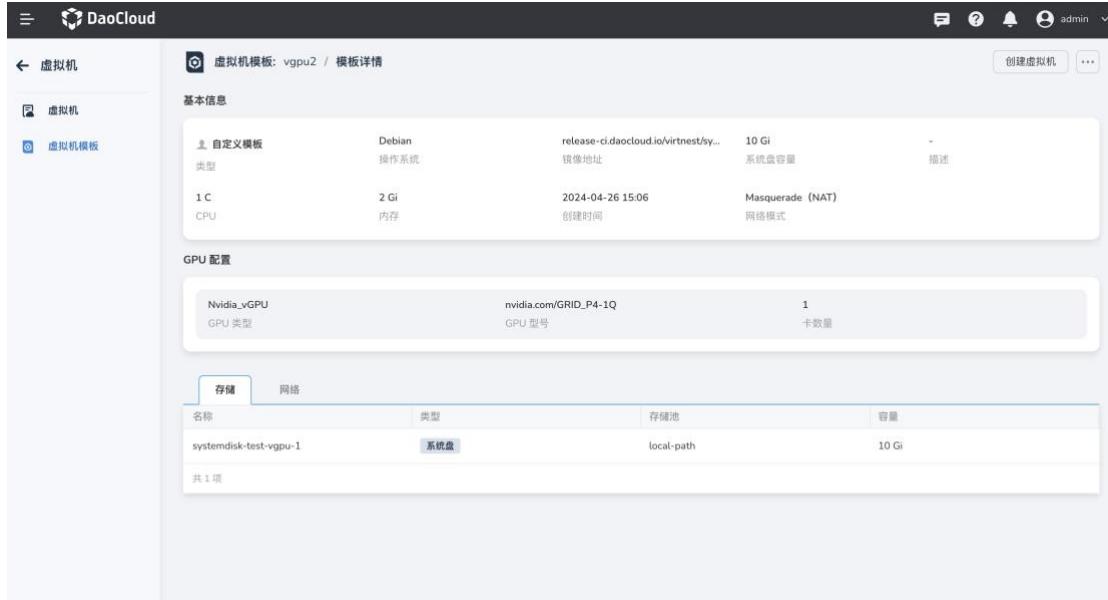
**转化模板**

**转化模板**

## 模板详情

成功创建出来一个模板后，点击模板名称，可以查看虚拟机详情，包括基本信息、GPU 信

息、存储、网络等。如果需要快速基于该模板部署新的虚拟机，只需点击页面右上角的 **创建虚拟机** 按钮即可便捷操作。



## 模板详情

# 构建虚拟机镜像

本文将介绍如何构建需要的虚拟机镜像。

虚拟机镜像其实就是一个副本文件，是安装有操作系统的一个磁盘分区。常见的镜像文件格式包括 raw、qcow2、vmdk 等。

## 构建镜像

下面是构建虚拟机镜像的一些详细步骤：

1. 下载系统镜像

在构建虚拟机镜像之前，您需要下载所需的系统镜像。我们推荐使用 qcow2、raw 或 vmdk 格式的镜像。可以访问以下链接获取 CentOS 和 Fedora 的镜像：

- [CentOS Cloud Images](#): 支持从官方 CentOS 项目或其他资源中获取 CentOS 镜像。请确保选择与您的虚拟化平台兼容的版本。
  - [Fedora Cloud Images](#): 支持从官方 Fedora 项目获取镜像。根据您的需求选择合适的版本。
2. 构建 Docker 镜像并推送到容器镜像仓库

在此步骤中，我们将使用 Docker 构建一个镜像，并将其推送到容器镜像仓库，以便在需要时能够方便地部署和使用。

- 创建 Dockerfile 文件

**FROM** scratch

**ADD** --chown=107:107 CentOS-7-x86\_64-GenericCloud.qcow2 /disk/

向基于空白镜像构建的镜像中添加名为 CentOS-7-x86\_64-GenericCloud.qcow2 的文件，并将其放置在镜像中的 **/disk/** 目录下。通过这个操作，镜像就包含了这个文件，可以在创建虚拟机时使用它来提供 CentOS 7 x86\_64 的操作系统环境。

- 构建镜像

`docker build -t release-ci.daocloud.io/hippo/kubevirt-demo/centos7:v1 .`

上述命令将使用 Dockerfile 中的指令构建一个名为 release-ci.daocloud.io/hippo/kubevirt-demo/centos7:v1 的镜像。您可以根据项目需求修改镜像名称。

- 推送镜像至容器镜像仓库

执行以下命令将构建好的镜像推送到名为 release-ci.daocloud.io 的镜像仓库，您还可以根据需要修改镜像仓库的名称和地址。

`docker push release-ci.daocloud.io/hippo/kubevirt-demo/centos7:v1`

以上是构建虚拟机镜像的详细步骤和说明。通过按照这些步骤操作，您将能够成功构建并推

送用于虚拟机的镜像，以满足您的使用需求。

# 如何从 VMWare 导入传统 Linux 虚拟机到云原生虚拟机平台

本文将详细介绍如何通过命令行将外部平台 VMware 上的 Linux 虚拟机导入到 DCE 5.0 的虚拟机中。

## !!! info

本文档外部虚拟平台是 VMware vSphere Client，后续简写为 vSphere。

技术上是依靠 kubevirt cdi 来实现的。操作前，vSphere 上被导入的虚拟机需要关机。  
以 Ubuntu 操作系统的虚拟机为例。

## 获取 vSphere 的虚拟机基础信息

- vSphere URL：目标平台的 URL 地址信息
- vSphere SSL 证书指纹 thumbprint：需要通过 openssl 获取  

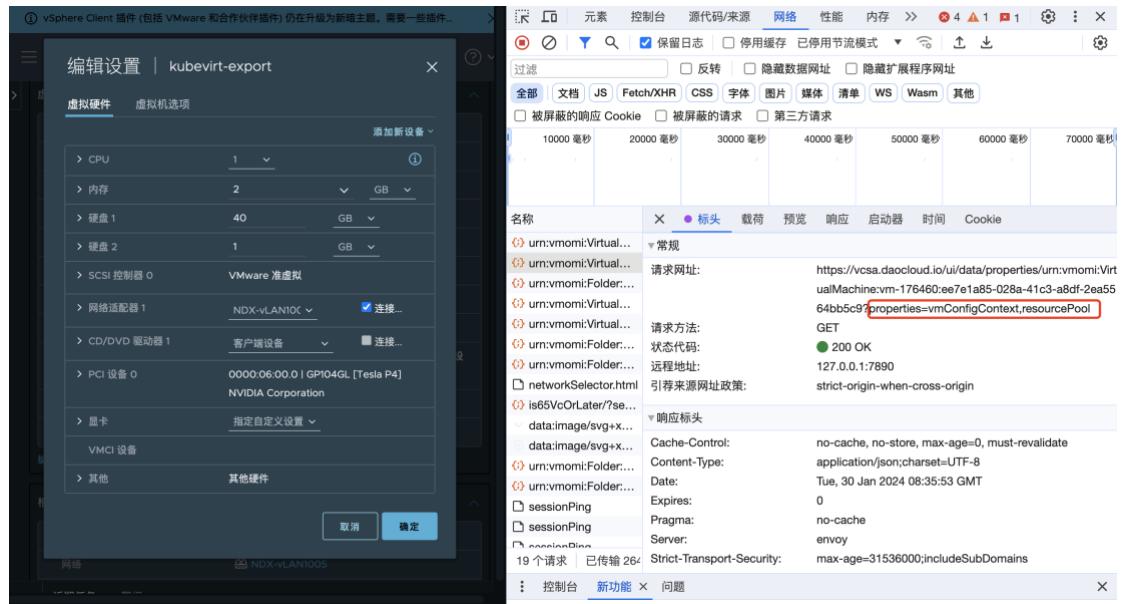
```
openssl s_client -connect 10.64.56.11:443 </dev/null | openssl x509 -in /dev/stdin -fingerprint -sha1 -noout
```

输出类似于：

```
Can't use SSL_get_servername
depth=0 CN = vcsa.daocloud.io
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = vcsa.daocloud.io
verify error:num=21:unable to verify the first certificate
verify return:1
depth=0 CN = vcsa.daocloud.io
verify return:1
DONE
sha1 Fingerprint=C3:9D:D7:55:6A:43:11:2B:DE:BA:27:EA:3B:C2:13:AF:E4:12:62:4D #所需值
```

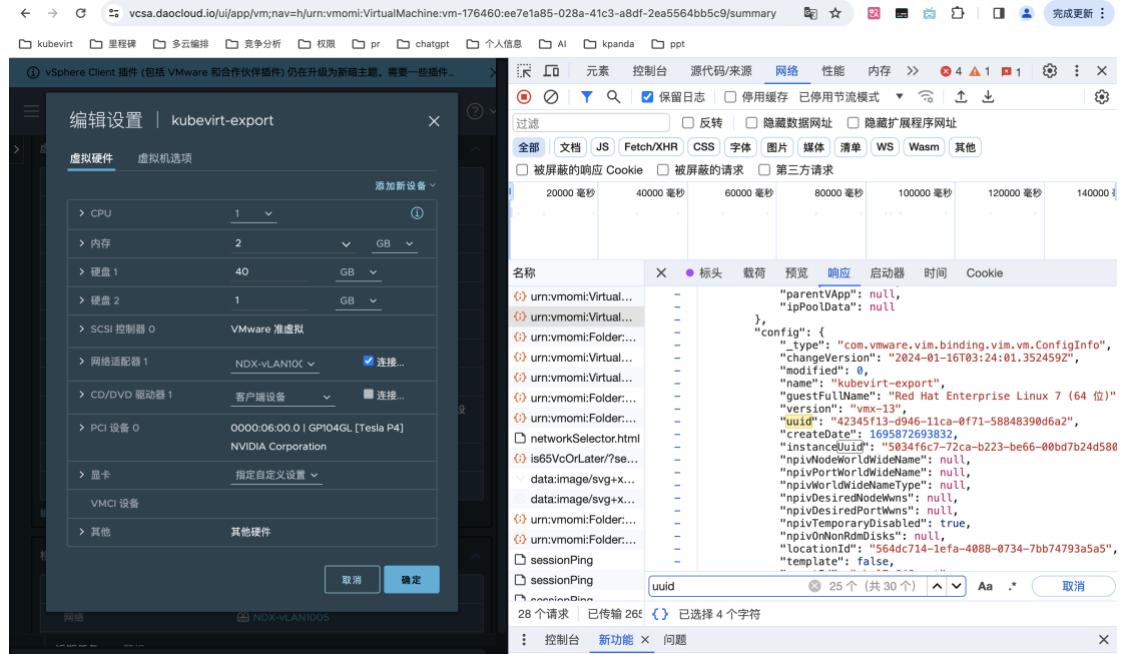
- vSphere 账号：获取 vSphere 的账号信息，注意权限问题

- vSphere 密码：获取 vSphere 的密码信息
- 需要导入虚拟机的 UUID (需要在 vSphere 的 web 页面获取)
  - 进入 Vsphere 页面中，进入被导入虚拟机的详情页面，点击 编辑配置，此时打开浏览器的开发者控制台，点击 网络 → 标头 找到如下图所示的 URL。



找到 URL

- 点击 响应，定位到 `vmConfigContext -> config`，最终找到目标值 `uuid`。



找到 uuid

- 需要导入虚拟机的 vmdk 文件 path

## 网络配置

需要根据网络模式的不同配置不同的信息，若有固定 IP 的需求，需要选择 Bridge 网络模式

- 创建 ovs 类型的 Multus CR，可参考[创建 Multus CR](#)
- 创建子网及 IP 池，参考[创建子网和 IP 池](#)

```

apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderIPPool
metadata:
  name: test2
spec:
  ips:
    - 10.20.3.90
  subnet: 10.20.0.0/16
  gateway: 10.20.0.1

```

```

apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderIPPool

```

```

metadata:
  name: test3

spec:
  ips:
    - 10.20.240.1
  subnet: 10.20.0.0/16
  gateway: 10.20.0.1

---

apiVersion: spiderpool.spidernet.io/v2beta1
kind: SpiderMultusConfig
metadata:
  name: test1
  namespace: kube-system
spec:
  cniType: ovs
  coordinator:
    detectGateway: false
    detectIPConflict: false
    mode: auto
    tunePodRoutes: true
  disableIPAM: false
  enableCoordinator: true
  ovs:
    bridge: br-1
    ippools:
      ipv4:
        - test1
        - test2

```

## 获取 vSphere 的账号密码 secret

```

apiVersion: v1
kind: Secret
metadata:
  name: vsphere # 可更改
  labels:
    app: containerized-data-importer # 请勿更改
type: Opaque
data:
  accessKeyId: "username-base64"
  secretKey: "password-base64"

```

## 编写 kubevirt vm yaml 创建 vm

!!! tip

若有固定 IP 需求，则该 yaml 与使用默认网络的 yaml 有一些区别，已标注。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
    virtnest.io/alias-name: ""
    virtnest.io/image-secret: ""
  creationTimestamp: "2024-05-23T06:46:28Z"
  finalizers:
  - kubevirt.io/virtualMachineControllerFinalize
  generation: 1
  labels:
    virtnest.io/os-family: Ubuntu
    virtnest.io/os-version: "22.04"
  name: export-ubuntu
  namespace: default
spec:
  dataVolumeTemplates:
  - metadata:
      creationTimestamp: null
      name: export-ubuntu-rootdisk
      namespace: default
    spec:
      pvc:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
        storageClassName: local-path
      source:
      vddk:
        backingFile: "[A05-09-ShangPu-Local-DataStore] virtnest-export-ubuntu/virtnest-export-ubuntu.vmdk"
        url: "https://10.64.56.21"
        uuid: "421d6135-4edb-df80-ee54-8c5b10cc4e78"
        thumbprint: "D7:C4:22:E3:6F:69:DA:72:50:81:12:FA:42:18:3F:29:5C:7F:41:CA"
```

```

secretRef: "vsphere"
initImageURL: "release.daocloud.io/virtnest/vddk:v8"
runStrategy: Manual
template:
  metadata:
    annotations:
      ipam.spider.net/ippools: '[{"cleangateway":false,"ipv4":["test2"]}]' // 这里添加 spider
pool 网络
  creationTimestamp: null
spec:
  architecture: amd64
  domain:
  devices:
  disks:
    - bootOrder: 1
      disk:
        bus: virtio
        name: rootdisk
  interfaces: // 修改
这里的网络配置
  - bridge: {}
    name: ovs-bridge0
machine:
  type: q35
resources:
  requests:
    memory: 4Gi
networks: // 修
改这里的网络配置
  - multus:
    default: true
    networkName: kube-system/test1
    name: ovs-bridge0
volumes:
  - dataVolume:
    name: export-ubuntu-rootdisk
    name: rootdisk

```

## 进入 VNC 检查是否成功运行

1. 修改虚拟机的网络配置

## 2. 查看当前网络

在实际导入完成时，如下图所示的配置已经完成。然而，需要注意的是，`enp1s0` 接口并没有包含 `inet` 字段，因此无法连接到外部网络。

```
root@ubuntu:/home/ubuntu# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
2: enp1s0: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1480 qdisc fq_codel state UP group default qlen 1000
    link/ether 52:54:00:b4:48:fe brd ff:ff:ff:ff:ff:ff
        inet 10.0.2.2/24 brd 10.0.2.255 scope global dynamic enp1s0
            valid_lft 86312663sec preferred_lft 86312663sec
        inet6 fe80::5054:ff:feb4:48fe/64 scope link
            valid_lft forever preferred_lft forever
```

查看网络配置

## 3. 配置 netplan

在上图所示的配置中，将 `ethernets` 中的对象更改为 `enp1s0`，并使用 DHCP 获得 IP 地址。

```
root@ubuntu:/home/ubuntu# cat /etc/netplan/00-installer-config.yaml
# This is the network config written by 'subiquity'
network:
  ethernets:
    enp1s0:
      dhcp4: true
  version: 2
```

配置 netplan

## 4. 将 netplan 配置应用到系统网络配置中

`sudo netplan apply`

## 5. 对外部网络进行 ping 测试

```
root@ubuntu:/home/ubuntu# ping www.qq.com
PING ins-r23tsuuf.ias.tencent-cloud.net (58.246.163.58) 56(84) bytes of data.
64 bytes from 58.246.163.58 (58.246.163.58): icmp_seq=1 ttl=48 time=3.19 ms
64 bytes from 58.246.163.58 (58.246.163.58): icmp_seq=2 ttl=48 time=3.42 ms
64 bytes from 58.246.163.58 (58.246.163.58): icmp_seq=5 ttl=48 time=3.28 ms
```

ping 网络

## 6. 通过 SSH 在节点上访问虚拟机。

```
[root@master ~]# kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS   AGE     IP          NODE   NOMINATED NODE   READI
NESS GATES
virt-launcher-export-centos-vddk-2-disk-wqmlld 1/1    Running   0          5h18m   10.233.70.120 master <none>      1/1
virt-launcher-export-debian-vddk-2-disk-697fr  1/1    Running   0          45m    10.233.70.133 master <none>      1/1
virt-launcher-export-ubuntu-vddk-hs84c       1/1    Running   0          24m    10.233.70.134 master <none>      1/1
virt-launcher-testvm-7k5bx                 2/2    Running   0          2d     10.233.70.76  master <none>      1/1
virt-launcher-ubuntu-9qj9k                 1/1    Running   0          58m    10.233.70.132 master <none>      1/1
[root@master ~]# ssh ubuntu@10.233.70.134
The authenticity of host '10.233.70.134 (10.233.70.134)' can't be established.
ECDSA key fingerprint is SHA256:2/B4j9XvZ/DpaljjjvYyBFrsw7CZ87FEvsUV0jZR07E.
ECDSA key fingerprint is MD5:b7:4c:e0:a0:22:79:5e:b8:9e:27:d2:de:45:a3:b3:02.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.233.70.134' (ECDSA) to the list of known hosts.
ubuntu@10.233.70.134's password:
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-156-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Thu Dec 21 09:59:41 UTC 2023

System load: 0.28      Processes:           100
Usage of /: 38.3% of 8.79GB  Users logged in: 1
Memory usage: 13%        IP address for enp1s0: 10.0.2.2
Swap usage: 0%

0 updates can be applied immediately.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your Internet connection or proxy settings

Last login: Thu Dec 21 09:50:03 2023
ubuntu@ubuntu:~$ ls
ubuntu@ubuntu:~$ ll
total 36
drwxr-xp-x  5 ubuntu  ubuntu  4096 Dec 21 09:19 .
drwxr-xp-x  3 root   root   4096 Dec 15 02:43 ..
-rw-----  1 ubuntu  ubuntu  272 Dec 21 09:34 .bash_history
-rw-r--r--  1 ubuntu  ubuntu  220 Apr  4  2018 .bash_logout
-rw-r--r--  1 ubuntu  ubuntu 3771 Apr  4  2018 .bashrc
drwxr-xp-x  2 ubuntu  ubuntu  4096 Dec 18 02:01 .cache/
drwxr-xp-x  3 ubuntu  ubuntu  4096 Dec 18 02:01 .gnupg/
drwxrwxr-x  3 ubuntu  ubuntu  4096 Dec 21 08:52 .local/
-rw-r--r--  1 ubuntu  ubuntu  887 Apr  4  2018 .profile
-rw-r--r--  1 ubuntu  ubuntu  0 Dec 21 08:55 sudo_as_admin_successful
```

访问虚拟机

## 如何从 VMWare 导入传统 Windows 虚拟机到云原生虚拟机平台

本文将详细介绍如何通过命令行将外部平台 VMware 上的虚拟机导入到 DCE 5.0 的虚拟机中。

### !!! info

本文档外部虚拟平台是 VMware vSphere Client，后续简写为 vSphere。

技术上是依靠 kubevirt cdi 来实现的。操作前，vSphere 上被导入的虚拟机需要关机。以 Windows 操作系统的虚拟机为例。

## 环境准备

导入前，需要参考[网络配置](#)准备环境。

## 获取 Windows 虚拟机的信息

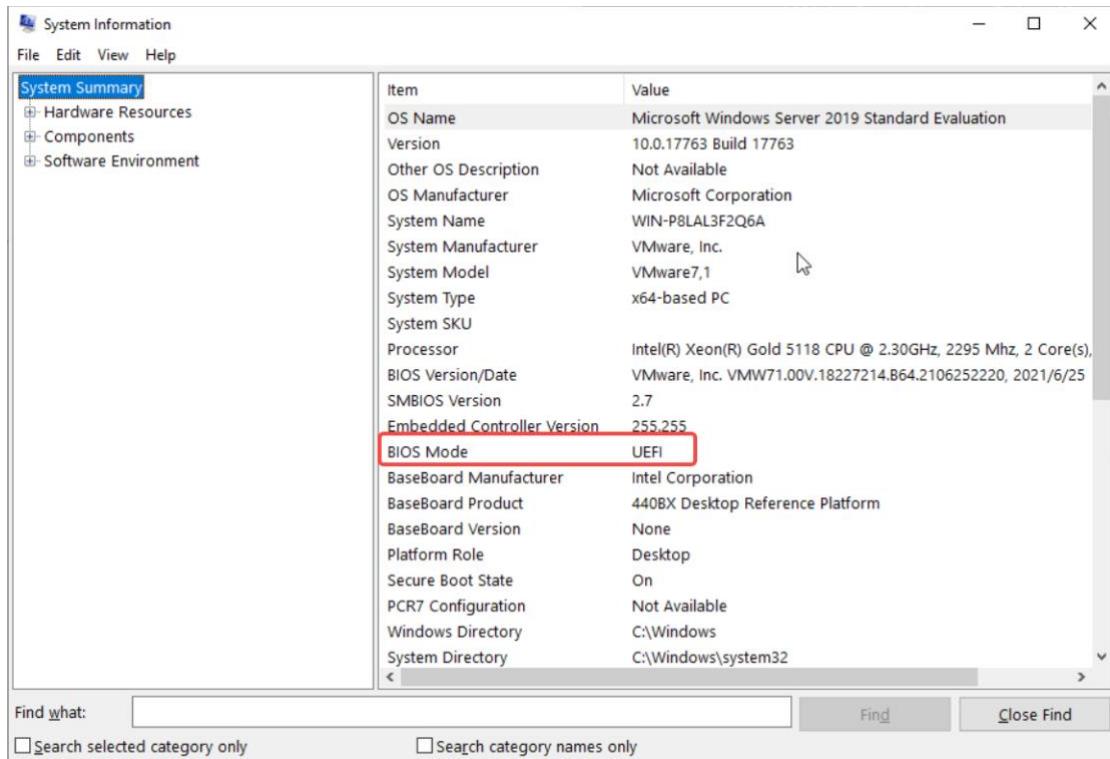
与导入 Linux 操作系统的虚拟机类似，可参考[如何从 VMWare 导入传统 Linuxs 虚拟机到云原生虚拟机平台](#)获取以下信息：

- 获取 vSphere 账号密码
- 获取 vSphere 虚拟机信息

## 检查 Windows 的引导类型

将外部平台的虚拟机导入到 DCE 5.0 的虚拟化平台中时，需要根据虚拟机的启动类型（BIOS 或 UEFI）进行相应的配置，以确保虚拟机能够正确启动和运行。

可以通过“系统信息”检查 Windows 是 BIOS 还是 UEFI 引导。如果是 UEFI 则需要在 YAML 文件中添加相关信息。



## 系统信息

## 导入过程

准备 window.yaml 文件，注意以下配置项

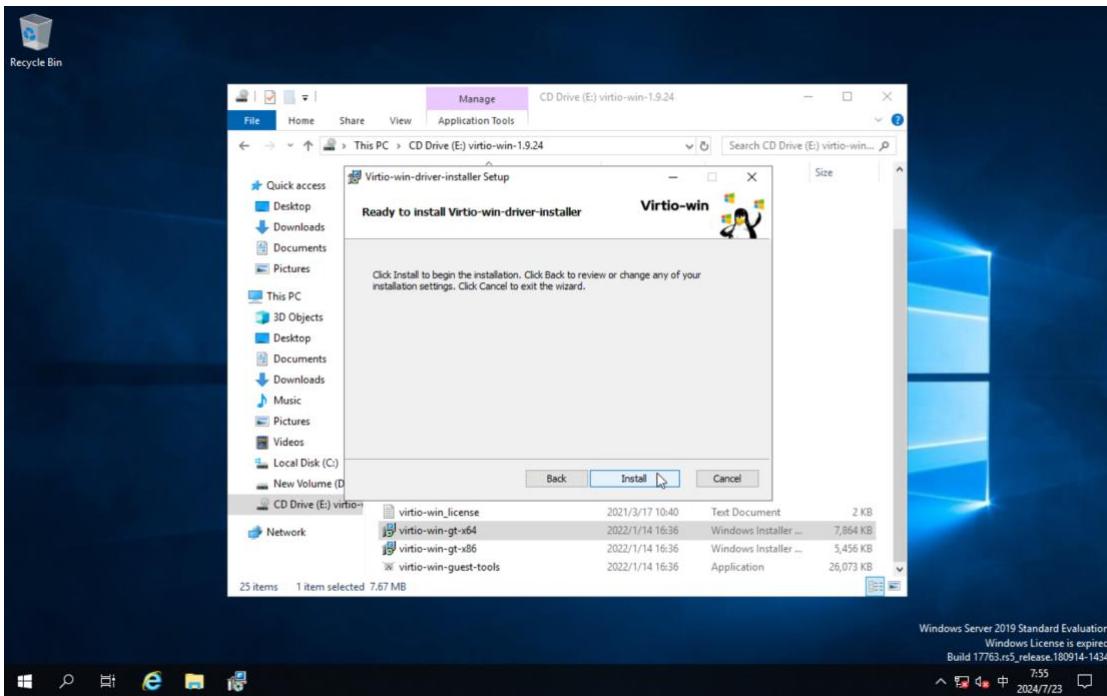
- 引导 Virtio 驱动的 PVC
- 磁盘总线类型，根据引导类型设置为 sata 或 virtio
- 如果使用 UEFI，需要添加 UEFI 配置

点击查看 window.yaml 示例

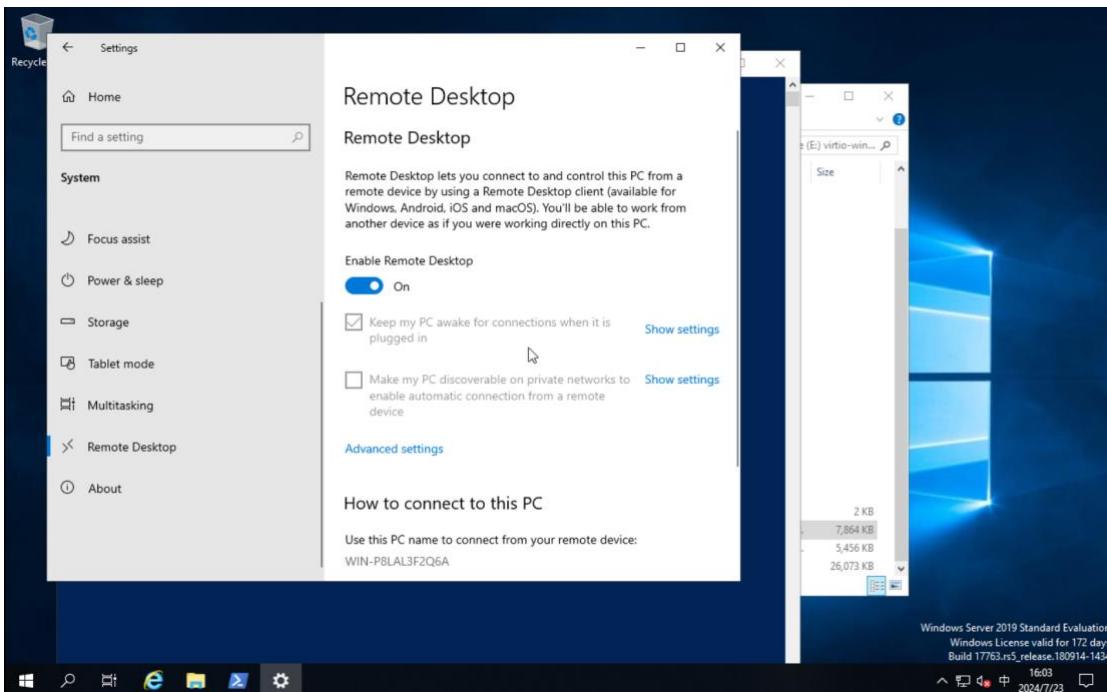
```
yaml title="window.yaml" apiVersion: kubevirt.io/v1 kind: VirtualMachine metadata: labels:
virtnest.io/os-family: windows virtnest.io/os-version: "server2019" name: export-
window-21 namespace: default spec: dataVolumeTemplates:
- metadata:
name: export-window-21-rootdisk spec: pvc: accessModes:
- ReadWriteOnce resources: requests: storage:
22Gi storageClassName: local-path source: vddk:
backingFile: "[A05-09-ShangPu-Local-DataStore]" virtnest-export-window/virtnest-export-
window.vmdk" url: "https://10.64.56.21" uuid: "421d40f2-21a2-
cfab-d5c9-e7f8abfc2faa" thumbprint:
"D7:C4:22:E3:6F:69:DA:72:50:81:12:FA:42:18:3F:29:5C:7F:41:CA" secretRef:
```







## 下载 VirtIO 2



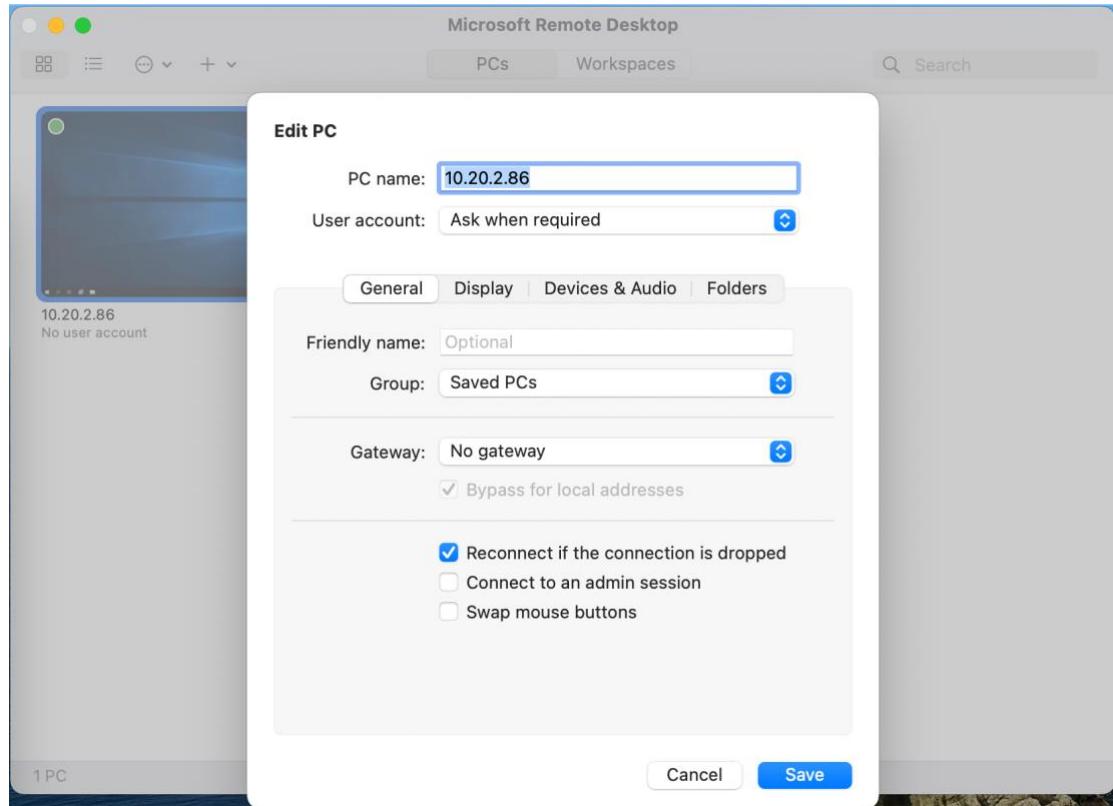
## 开启远程桌面 Remote Desktop

# 重启后更新 YAML

[点击查看修改后的 window.yaml 示例](#)



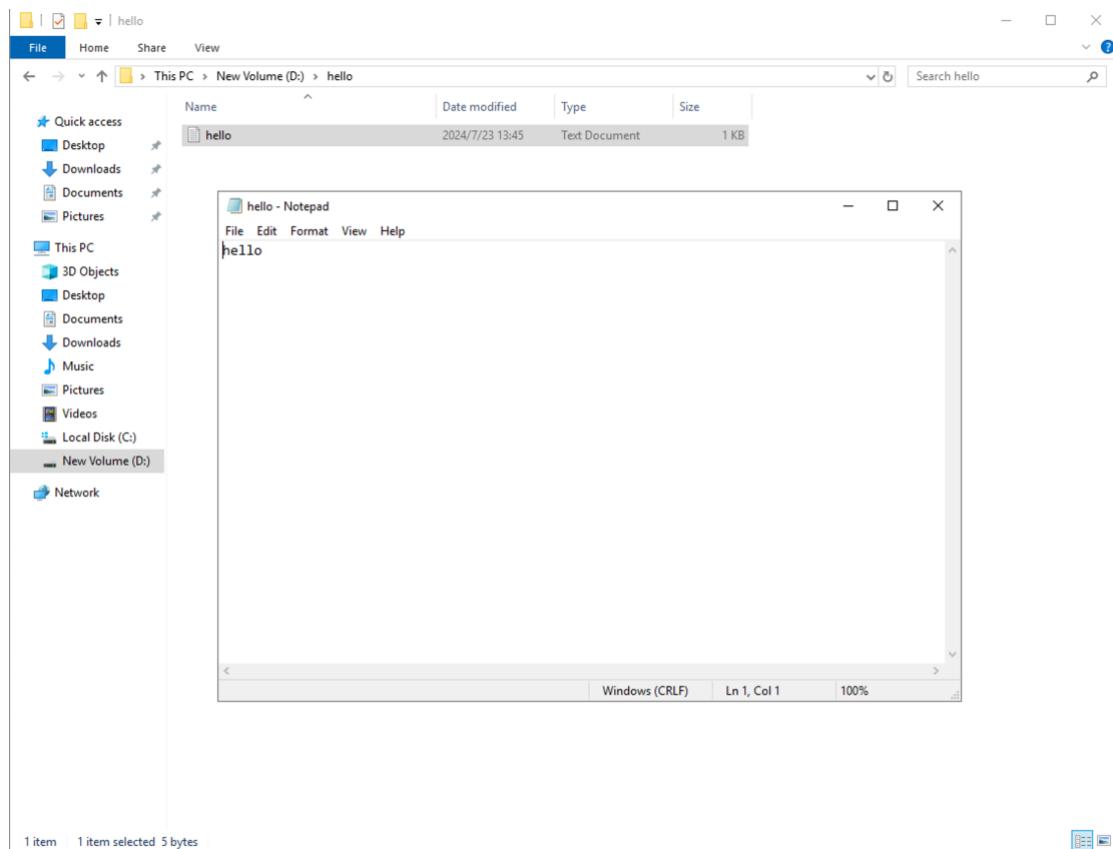
登录。



访问虚拟机

- 验证网络访问和数据盘数据

## 验证网络



查看数据盘数据

## 对比导入 Linux 和 Windows 虚拟机的差异

- Windows 可能需要 UEFI 配置。
- Windows 通常需要安装 VirtIO 驱动。
- Windows 多磁盘导入通常不需要重新挂载磁盘。

## 创建 Windows 虚拟机

本文将介绍如何通过命令行创建 Windows 虚拟机。

## 前提条件

1. 创建 Windows 虚拟机之前，需要先参考[安装虚拟机模块的依赖和前提](#)确定您的环境已经准备就绪。
2. 创建过程建议参考官方文档：[安装 windows 的文档](#)、[安装 Windows 相关驱动程序](#)。
3. Windows 虚拟机建议使用 VNC 的访问方式。

## 导入 ISO 镜像

创建 Windows 虚拟机需要导入 ISO 镜像的主要原因是为了安装 Windows 操作系统。与 Linux 操作系统不同，Windows 操作系统安装过程通常需要从安装光盘或 ISO 镜像文件中引导。因此，在创建 Windows 虚拟机时，需要先导入 Windows 操作系统的安装 ISO 镜像文件，以便虚拟机能够正常安装。

以下介绍两个导入 ISO 镜像的办法：

1. (推荐) 制作 Docker 镜像，建议参考[构建镜像](#)
2. (不推荐) 使用 virtctl 将镜像导入到 PVC 中

可参考如下命令

```
virtctl image-upload -n <命名空间> pvc <PVC 名称> \
--image-path=<ISO 文件路径> \
--access-mode=ReadWriteOnce \
--size=6G \
--uploadproxy-url=<https://cdi-uploadproxy ClusterIP 和端口> \
--force-bind \
--insecure \
--wait-secs=240 \
--storage-class=<SC>
```

例如：

```
virtctl image-upload -n <命名空间> pvc <PVC 名称> \
--image-path=<ISO 文件路径> \
```

```
--access-mode=ReadWriteOnce \
--size=6G \
--uploadproxy-url=<https://cdi-uploadproxy> ClusterIP 和端口 \
--force-bind \
--insecure \
--wait-secs=240 \
--storage-class=<SC>
```

## YAML 创建 Windows 虚拟机

使用 yaml 创建 Windows 虚拟机，更加灵活并且更易编写和维护。以下介绍三种参考的

yaml：

### 1. 推荐使用 Virtio 驱动 + Docker 镜像的方式

- 如果你需要使用存储能力-挂载磁盘，请安装 [vistor 驱动程序](#)
- 如果你需要使用网络能力，请安装 [NetKVM 驱动程序](#)

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  labels:
    virtnest.io/os-family: Windows
    virtnest.io/os-version: '10'
  name: windows10-virtio
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        name: win10-system-virtio
        namespace: default
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 32Gi
            storageClassName: local-path
          source:
```

```
blank: {}

running: true

template:

metadata:

labels:
    app: windows10-virtio
    version: v1
    kubevirt.io/domain: windows10-virtio

spec:

architecture: amd64

domain:

cpu:
    cores: 8
    sockets: 1
    threads: 1

devices:

disks:
    - bootOrder: 1
        disk:
            bus: virtio # 使用 virtio
            name: win10-system-virtio
    - bootOrder: 2
        cdrom:
            bus: sata # 对于 ISO 镜像, 使用 sata
            name: iso-win10
    - bootOrder: 3
        cdrom:
            bus: sata # 对于 containerdisk, 使用 sata
            name: virtiocontainerdisk

interfaces:
    - name: default
      masquerade: {}

machine:

type: q35

resources:

requests:
    memory: 8G

networks:
    - name: default
      pod: {}

volumes:
    - name: iso-win10

persistentVolumeClaim:
    claimName: iso-win10
```

```
- name: win10-system-virtio
  persistentVolumeClaim:
    claimName: win10-system-virtio
- containerDisk:
    image: kubevirt/virtio-container-disk
    name: virtiocontainerdisk
```

2. (不推荐) 使用 Virtio 驱动和 virtctl 工具的组合方式, 将镜像导入到 Persistent Volume Claim (PVC) 中。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  labels:
    virtnest.io/os-family: Windows
    virtnest.io/os-version: '10'
  name: windows10-virtio
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        name: win10-system-virtio
        namespace: default
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 32Gi
          storageClassName: local-path
        source:
          blank: {}
  running: true
  template:
    metadata:
      labels:
        app: windows10-virtio
        version: v1
        kubevirt.io/domain: windows10-virtio
  spec:
```

```
architecture: amd64
domain:
  cpu:
    cores: 8
    sockets: 1
    threads: 1
  devices:
    disks:
      - bootOrder: 1
        # 请使用 virtio
        disk:
          bus: virtio
          name: win10-system-virtio
          # ISO 镜像请使用 sata
      - bootOrder: 2
        cdrom:
          bus: sata
          name: iso-win10
          # containerdisk 请使用 sata
      - bootOrder: 3
        cdrom:
          bus: sata
          name: virtiocontainerdisk
  interfaces:
    - name: default
      masquerade: {}
  machine:
    type: q35
  resources:
    requests:
      memory: 8G
  networks:
    - name: default
      pod: {}
  volumes:
    - name: iso-win10
      persistentVolumeClaim:
        claimName: iso-win10
    - name: win10-system-virtio
      persistentVolumeClaim:
        claimName: win10-system-virtio
    - containerDisk:
        image: kubevirt/virtio-container-disk
        name: virtiocontainerdisk
```

3. (不推荐) 不使用 Virtio 驱动的情况下, 使用 virtctl 工具将镜像导入到 Persistent Volume Claim (PVC) 中。虚拟机可能使用其他类型的驱动或默认驱动来操作磁盘和网络设备。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
    kubevirt.io/latest-observed-api-version: v1
    kubevirt.io/storage-observed-api-version: v1
  labels:
    virtnest.io/os-family: Windows
    virtnest.io/os-version: '10'
  name: windows10
  namespace: default
spec:
  dataVolumeTemplates:
    # 创建系统盘, 你创建多个 PVC (磁盘)
    - metadata:
        name: win10-system
        namespace: default
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 32Gi
            storageClassName: local-path
          source:
            blank: {}
        running: true
      template:
        metadata:
          labels:
            app: windows10
            version: v1
            kubevirt.io/domain: windows10
        spec:
          architecture: amd64
          domain:
            cpu:
```

```

cores: 8
sockets: 1
threads: 1
devices:
  disks:
    - bootOrder: 1
      # 无 virtio 驱动, 请使用 sata
    cdrom:
      bus: sata
      name: win10-system
      # ISO 镜像, 请使用 sata
    - bootOrder: 2
      cdrom:
        bus: sata
        name: iso-win10
  interfaces:
    - name: default
      masquerade: {}
  machine:
    type: q35
  resources:
    requests:
      memory: 8G
  networks:
    - name: default
      pod: {}
  volumes:
    - name: iso-win10
      persistentVolumeClaim:
        claimName: iso-win10
    - name: win10-system
      persistentVolumeClaim:
        claimName: win10-system

```

## 云桌面

Windows 版本的虚拟机大多数情况是需要远程桌面控制访问的，建议使用 [Microsoft 远程桌面](#) 控制您的虚拟机。

### !!! note

- 你的 Windows 版本需支持远程桌面控制，才能使用 [Microsoft 远程桌面](<https://learn.microsoft.com/zh-cn/windows-server/remote/remote-desktop-ser>

vices/clients/remote-desktop-mac#get-the-remote-desktop-client)。

- 需要关闭 Windows 的防火墙。

## 增加数据盘

Windows 虚拟机添加数据盘的方式和 Linux 虚拟机一致。你可以参考下面的 YAML 示例：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
<...>
spec:
  dataVolumeTemplates:
    # 添加一块数据盘
    - metadata:
        name: win10-disk
      namespace: default
    spec:
      pvc:
        accessModes:
          - ReadWriteOnce
      resources:
        requests:
          storage: 16Gi
      storageClassName: hwameistor-storage-lvm-hdd
      source:
        blank: {}
    template:
      spec:
        domain:
          devices:
            disks:
              - bootOrder: 1
                disk:
                  bus: virtio
                  name: win10-system
                  # 添加一块数据盘
              - bootOrder: 2
                disk:
                  bus: virtio
                  name: win10-disk
    <...>
  volumes:
    <...>
    # 添加一块数据盘
```

```

- name: win10-disk
  persistentVolumeClaim:
    claimName: win10-disk

```

## 快照、克隆、实时迁移

这些能力和 Linux 虚拟机一致，可直接参考配置 Linux 虚拟机的方式。

## 访问 Windows 虚拟机

1. 创建成功后，进入虚拟机列表页面，发现虚拟机正常运行。

名称	状态	命名空间	操作系统	节点	IP	CPU	内存	创建时间
vm-windows	运行中	virt-demo	Windows	nicole-centos8-25	10.246.151.221	4 Core	1.86 Gi	2024-01-25 10:...

2. 点击控制台访问（VNC），可以正常访问。

## 虚拟机常见问题

虚拟机 (virtnest) 包含 apiserver 和 agent 两个部分，遇到问题时应从这两部分进行排查。

## 页面 API 报错

若页面请求 API 报错 500 或 cluster 资源不存在，首先应检查[全局服务集群](#)中虚拟机相关服务的日志，寻找是否 kpanda 的关键词。若存在，需确认 kpanda 相关服务是否运行正常。

# VM 无法正常使用

若创建的 VM 无法正常使用，原因多样。以下是排查方向：

## VM 创建失败

VM 创建失败时，应在目标集群中查看 VM 的详细信息：

```
kubectl -n your-namespace describe vm your-vm
```

如果详细信息涉及存储，如 PVC、PV、SC 等，请检查 SC 状态。问题未解决时，应[咨询开发人员](#)。

如果详细信息涉及设备，如 KVM、GPU 等，请核实目标集群节点是否完成了[依赖条件](#)检查。

若所有依赖已安装，应[咨询开发人员](#)。

## 案例 1

- 现象：在资源充足的情况下还是报错：

名称	状态	命名空间
vm-cirros	● 错误 <a href="#">?</a>	<p><b>Unschedulable</b> 0/3 nodes are available: 3 Insufficient devices.kubevirt.io/kvm. preemption: 0/3 nodes are available: 3 No preemption victims found for incoming pod.</p>

创建虚拟机报错一

- 解决办法：[给节点启用硬件虚拟化](#)

## 案例 2

```
[root@yiting-virthost-m1 ~]# k get po
NAME          READY   STATUS    RESTARTS   AGE
virt-launcher-vm-cirros-snj4t   2/3     Error      0   2m59s
[root@yiting-virthost-m1 ~]# k logs virt-launcher-vm-cirros-snj4t
[{"component": "virt-launcher", "level": "info", "msg": "Collected all requested hook sidecar sockets", "pos": "manager.go:88", "timestamp": "2024-07-01T21:40:50.611895Z"}, {"component": "virt-launcher", "level": "info", "msg": "Sorted all collected sidecar sockets per hook point based on their priority and name: map[]", "pos": "manager.go:91", "timestamp": "2024-07-01T21:40:50.612031Z"}, {"component": "virt-launcher", "level": "info", "msg": "Connecting to libvirt daemon: qemu+unix:///session?socket=/var/run/libvirt/virtqemud-sock", "pos": "libvirt.go:565", "timestamp": "2024-07-01T21:40:50.612666Z"}, {"component": "virt-launcher", "level": "info", "msg": "Connecting to libvirt daemon failed: virError{Code=38, Domain=7, Message='Failed to connect socket to '/var/run/libvirt/virtqemud-sock': No such file or directory'}", "pos": "libvirt.go:573", "timestamp": "2024-07-01T21:40:50.613689Z"}, {"component": "virt-launcher", "level": "info", "msg": "libvirt version: 9.5.0, package: 6.el9 (builder@centos.org, 2023-08-25-08:53:56, )", "subcomponent": "libvirt", "thread": "40", "timestamp": "2024-07-01T21:40:50.649000Z"}, {"component": "virt-launcher", "level": "info", "msg": "hostname: vm-cirros", "subcomponent": "libvirt", "thread": "40", "timestamp": "2024-07-01T21:40:50.649000Z"}, {"component": "virt-launcher", "level": "error", "msg": "internal error: Unable to get session bus connection: Cannot spawn a message bus without a machine-id: Unable to load /var/lib/dbus/machine-id or /etc/machine-id. Failed to open file '/var/lib/dbus/machine-id': No such file or directory", "pos": "virQ0BusGetSessionBus:126", "subcomponent": "libvirt", "thread": "40", "timestamp": "2024-07-01T21:40:50.649000Z"}, {"component": "virt-launcher", "level": "error", "msg": "internal error: Unable to get system bus connection: Could not connect: No such file or directory", "pos": "virQ0BusGetSystemBus:99", "subcomponent": "libvirt", "thread": "40", "timestamp": "2024-07-01T21:40:50.649000Z"}, {"component": "virt-launcher", "level": "info", "msg": "Connected to libvirt daemon", "pos": "libvirt.go:581", "timestamp": "2024-07-01T21:40:51.119263Z"}, {"component": "virt-launcher", "level": "info", "msg": "Registered libvirt event notify callback", "pos": "client.go:563", "timestamp": "2024-07-01T21:40:51.156935Z"}, {"component": "virt-launcher", "level": "info", "msg": "Marked as ready", "pos": "virt-launcher.go:75", "timestamp": "2024-07-01T21:40:51.157749Z"}]
```

### 创建虚拟机报错二

解决办法：[升级节点操作系统内核](#)

## VM 创建成功但无法使用

若 VM 创建成功但无法使用，应在 DCE 页面检查 VM 的 VNC 页面是否正常。若显示但仅限启动信息，请检查[依赖条件](#)。若依赖条件齐全，应[咨询开发人员](#)。

若 VNC 页面显示异常，应使用以下命令查看 VM 详细信息：

```
kubectl -n your-namespace describe vm your-vm
```

当详细信息涉及存储信息，如 PVC、PV、SC 等，应检查 SC 状态。问题未解决时，应[咨询开发人员](#)。

## VNC 可以启动但网络无法访问

按照下面流程进行排查，将相关信息记录下来，反馈给[开发人员](#)。在 VM 所在集群中执行以下操作：

### 1. 获取 VM 的 Pod IP

```
kubectl -n your-namespace get vmi your-vm -o wide
```

### 2. 在节点上执行 ssh 登录你的 VM

```
ssh your-vm-username@xx.xx.xx.xx
```

如果无法访问，请[咨询开发人员](#)。

### 3. 检查 VM 使用的网络模式

如果是默认网络模式 (masquerade) , [咨询开发人员。](#)

如果是 bridge + ovs, 需要确认以下信息。

- 检查 Spiderpool 是否安装成功, 并且确保安装在 kube-system 命名空间下。
- ovs 安装成功, 并且 ovs bridge 配置成功。

若以上信息确认无误, 请[咨询开发人员。](#)

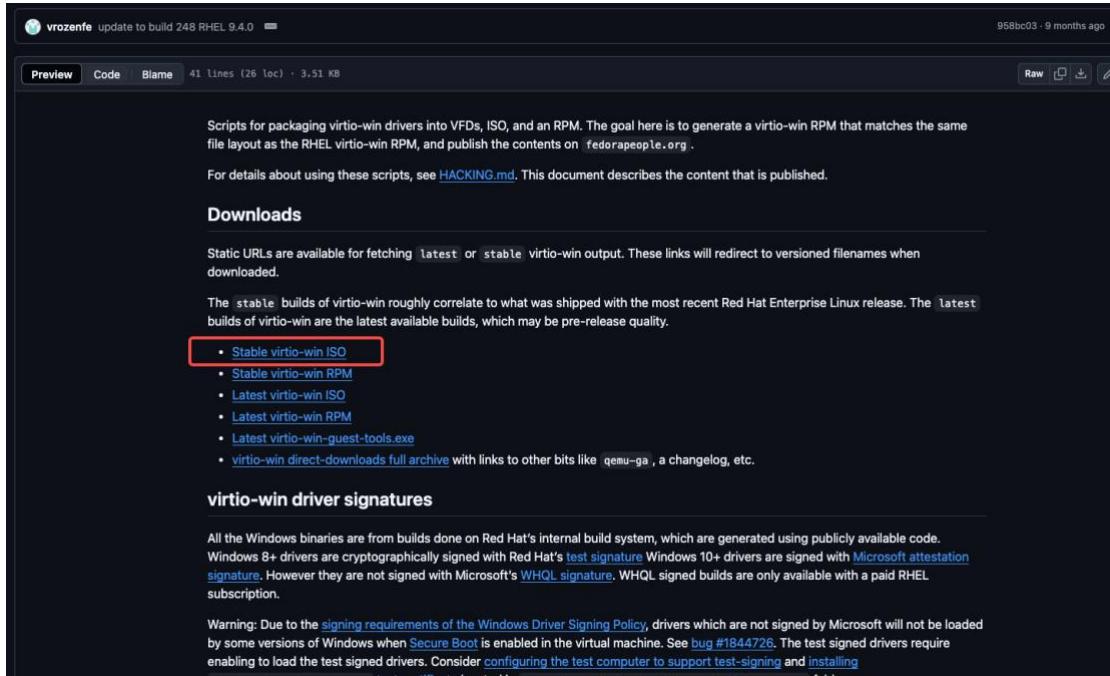
## Windows 虚拟机新增磁盘无法识别问题

通过 KubeVirt 在新安装的 Windows Server 上添加新磁盘时识别不到, 需要按照如下步骤

安装 virtio 驱动:

#### 1. 下载 stable virtio-win iso

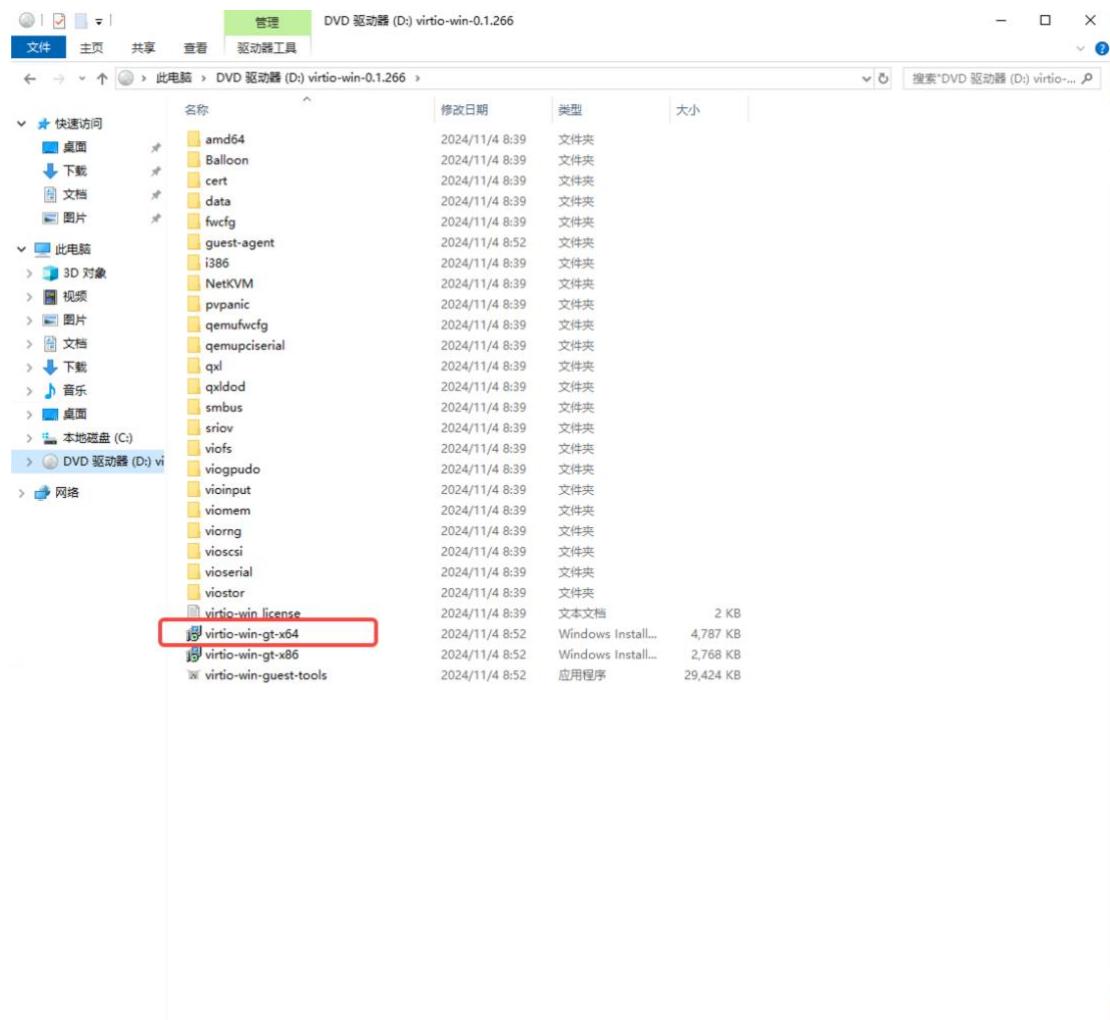
下 载 地 址 为 : <https://github.com/virtio-win/virtio-win-pkg-scripts/blob/master/README.md>



img

## 2. 安装 virtio-win iso

挂载 iso 镜像后，点击 virtio-win-gt-x64 进行安装，安装完成后，即可看到添加的磁盘



img

## 3. 添加磁盘后的效果

