



**openEuler**  
**1.0**

# **虚拟化用户指南**

发布日期      2020-01-18

# 目 录

法律声明.....	iv
前言.....	v
1 认识虚拟化.....	6
2 安装指南.....	10
2.1 最低硬件要求 .....	10
2.2 安装虚拟化核心组件 .....	10
2.2.1 安装方法 .....	10
2.2.2 验证安装是否成功 .....	11
3 用户和管理员指南.....	13
3.1 准备环境 .....	13
3.1.1 准备虚拟机镜像 .....	13
3.1.2 准备虚拟机网络 .....	15
3.1.3 准备 UEFI 引导工具集 EDK II.....	19
3.2 虚拟机配置 .....	20
3.2.1 总体介绍 .....	20
3.2.2 虚拟机描述 .....	21
3.2.3 虚拟 CPU 和虚拟内存.....	21
3.2.4 配置虚拟设备 .....	22
3.2.4.1 存储设备 .....	22
3.2.4.2 网络设备 .....	24
3.2.4.3 总线配置 .....	25
3.2.4.4 其它常用设备 .....	28
3.2.5 其他常见配置项 .....	29
3.2.6 XML 配置文件示例 .....	30
3.3 管理虚拟机 .....	33
3.3.1 虚拟机生命周期 .....	33
3.3.1.1 总体介绍 .....	33
3.3.1.2 管理命令 .....	35
3.3.1.3 示例 .....	36

3.3.2 在线修改虚拟机配置 .....	37
3.3.3 查询虚拟机信息 .....	37
3.3.4 登录虚拟机 .....	40
3.3.4.1 使用 VNC 密码登录 .....	40
3.3.4.2 配置 VNC TLS 登录 .....	41
3.4 热迁移虚拟机 .....	43
3.4.1 总体介绍 .....	43
3.4.2 应用场景 .....	43
3.4.3 注意事项和约束限制 .....	44
3.4.4 热迁移操作 .....	44
3.5 管理系统资源 .....	46
3.5.1 管理虚拟 CPU .....	47
3.5.1.1 CPU 份额 .....	47
3.5.1.2 绑定 qemu 进程至物理 CPU .....	48
3.5.1.3 调整虚拟 CPU 绑定关系 .....	49
3.5.2 管理虚拟内存 .....	50
3.5.2.1 NUMA 简介 .....	50
3.5.2.2 配置 Host NUMA .....	50
3.5.2.3 配置 Guest NUMA .....	51
3.6 管理设备 .....	52
3.6.1 配置虚拟机 PCIe 控制器 .....	52
3.6.2 管理虚拟磁盘 .....	53
3.6.3 管理虚拟网卡 .....	54
3.6.4 配置虚拟串口 .....	55
3.7 最佳实践 .....	55
3.7.1 halt-polling .....	55
3.7.2 内存大页 .....	56
<b>A 附录 .....</b>	<b>58</b>
A.1 术语和缩略语 .....	58

---

## 法律声明

---

版权所有 © 2020 华为技术有限公司。

您对“本文档”的复制、使用、修改及分发受知识共享(Creative Commons)署名一相同方式共享 4.0 国际公共许可协议(以下简称“CC BY-SA 4.0”)的约束。为了方便用户理解，您可以通过访问 <https://creativecommons.org/licenses/by-sa/4.0/> 了解 CC BY-SA 4.0 的概要 (但不是替代)。CC BY-SA 4.0 的完整协议内容您可以访问如下网址获取：  
<https://creativecommons.org/licenses/by-sa/4.0/legalcode>。

### 商标声明

openEuler 为华为技术有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

### 免责声明

本文档仅作为使用指导，除非适用法强制规定或者双方有明确书面约定，华为技术有限公司对本文档中的所有陈述、信息和建议不做任何明示或默示的声明或保证，包括但不限于不侵权，时效性或满足特定目的的担保。



# 前言

## 概述

本文档给出虚拟化介绍，并给出基于 openEuler 的虚拟化安装方法以及如何使用虚拟化，让用户了解虚拟化，并指导用户和管理员安装和使用虚拟化。

## 符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 须知	用于传递设备或环境安全警示信息。如不可避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

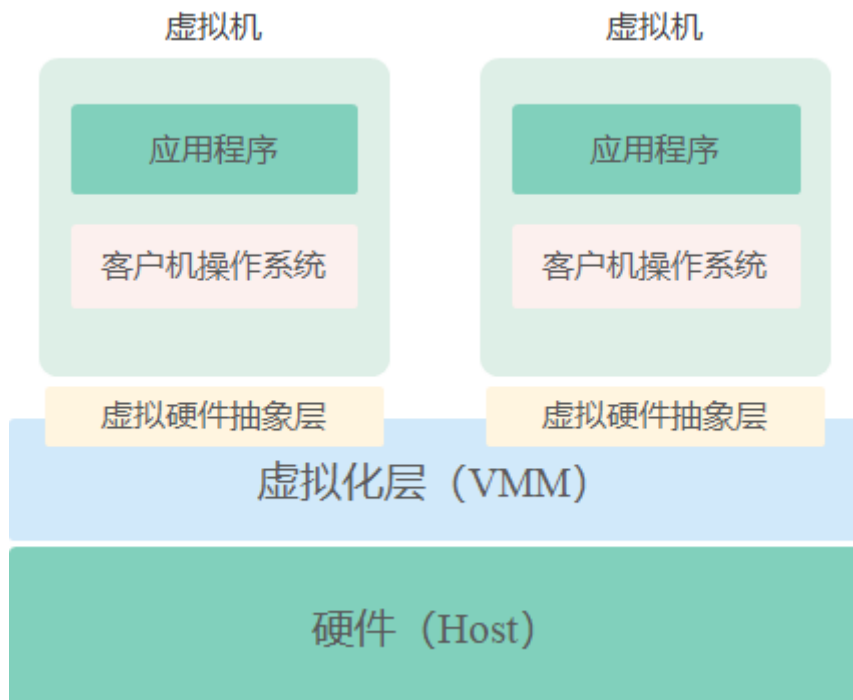
# 1 认识虚拟化

## 简介

在计算机技术中，虚拟化是一种资源管理技术，它将计算机的各种实体资源（处理器、内存、磁盘、网络适配器等）予以抽象，转换后呈现并可供分割、组合为一个或多个计算机配置环境。这种资源管理技术打破了实体结构不可分割的障碍，使这些资源在虚拟化后不受现有资源的架设方式、地域或物理配置限制，从而让用户可以更好地应用计算机硬件资源，提高资源利用率。

虚拟化使得一台物理服务器上可以运行多台虚拟机，虚拟机共享物理机的处理器、内存、I/O 资源等，但逻辑上虚拟机之间是互相隔离的。在虚拟化技术中，通常将这个物理服务器称为宿主机，宿主机上运行的虚拟机也叫客户机，虚拟机内部运行的操作系统称为客户机操作系统。在宿主机和虚拟机之间存在一层叫虚拟化层的软件，用于实现虚拟硬件的模拟，通常这个虚拟化层被称为虚拟机监视器，如下图所示：

图1-1 虚拟化架构



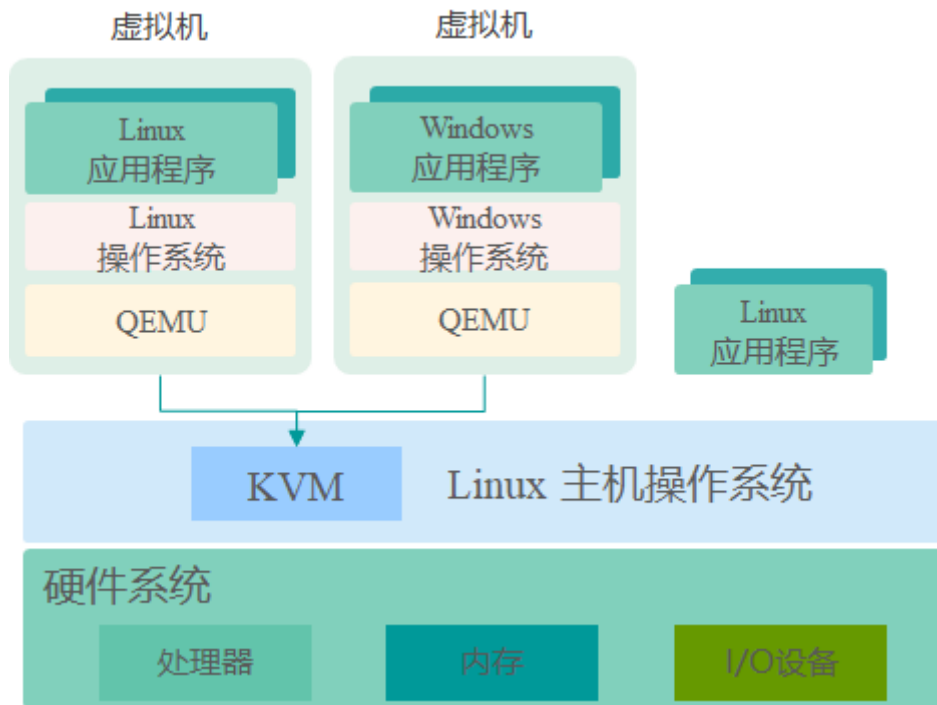
## 虚拟化架构

当前的主流虚拟化技术按照 VMM（Virtual Machine Monitor）实现结构不同分为两种：

- **Hypervisor 模型**  
在这种模型中，VMM 被看做是一个完备的操作系统，同时还具备虚拟化功能，VMM 直接管理所有的物理资源，包括处理器，内存和 IO 设备等，如 XEN，VMware vSphere。
- **宿主模型**  
这种模型中，物理资源是由宿主机操作系统管理，宿主机操作系统是传统的操作系统，如 Linux，Windows 等，宿主机操作系统不提供虚拟化能力，提供虚拟化能力的 VMM 作为系统的一个驱动或者软件运行在宿主操作系统上，VMM 通过调用 Host OS 的服务获得资源，实现处理器，内存和 IO 设备的模拟，这种模型的虚拟化实现有 KVM、Virtual Box 等。

KVM（Kernel-based Virtual Machine）即基于内核的虚拟机，是 Linux 的一个内核模块，该内核模块使 Linux 成为一个 Hypervisor。KVM 架构如图 1-2 所示。KVM 本身未模拟任何硬件设备，它用于使能硬件提供的虚拟化能力，比如 Intel VT-x, AMD-V, ARM virtualization extensions 等。主板、内存及 I/O 等设备的模拟由用户态的 QEMU 完成。用户态 QEMU 配合内核 KVM 模块共同完成虚拟机的硬件模拟，客户操作系统运行在 QEMU 和 KVM 模拟的硬件上。

图1-2 KVM 架构图



## 虚拟化组件

openEuler 软件包中提供的虚拟化相关组件：

- **KVM**：提供核心的虚拟化基础设施，使 Linux 系统成为一个 Hypervisor，支持多个虚拟机同时在该主机上运行。
- **QEMU**：模拟处理器并提供一组设备模型，配合 KVM 实现基于硬件的虚拟化模拟加速。
- **Libvirt**：为管理虚拟机提供工具集，主要包含统一、稳定、开放的应用程序接口（API）、守护进程（Libvirtd）和一个默认命令行管理工具（virsh）。
- **Open vSwitch**：为虚拟机提供虚拟网络的工具集，支持编程扩展，以及标准的管理接口和协议（如 NetFlow，sFlow，IPFIX，RSPAN，CLI，LACP，802.1ag）。

## 虚拟化特点

业界普遍认可虚拟化有以下特点：

- **分区**  
虚拟化可以对一台物理服务器进行软件逻辑分割，实现运行多台不同规格的虚拟机（虚拟服务器）。
- **隔离**  
虚拟化能够模拟虚拟硬件，为虚拟机运行完整操作系统提供硬件条件，每个虚拟机内部操作系统都是独立的，互相隔离的。例如一台虚拟机的操作系统由于故障



或者受到恶意破坏而崩溃，其他虚拟机内部的操作系统和应用不会受到任何影响。

- 封装性

以虚拟机为粒度封装，优秀的封装性使得虚拟机比物理机更灵活，可以实现虚拟机的热迁移、快照、克隆等功能，实现数据中心的快速部署和自动化运维。

- 硬件无关

经过虚拟化层的抽象后，虚拟机与底层的硬件没有直接的绑定关系，可以在其他服务器上不加修改地运行虚拟机。

## 虚拟化优势

虚拟化为 IT 基础设施带来了众多优势：

- 灵活性和可扩展性

用户可以根据需求进行动态资源分配和回收，满足动态变化的业务需求，同时也可以根据不同的产品需求，规划不同的虚拟机规格，在不改变物理资源配置的情况下进行规模调整。

- 更高的可用性和更好的运维手段

虚拟化提供热迁移，快照，热升级，容灾自动恢复等运维手段，可以在不影响用户的情况下对物理资源进行删除、升级或变更，提高了业务连续性，同时可以实现自动化运维。

- 提高安全性

虚拟化提供了操作系统级的隔离，同时实现基于硬件提供的处理器操作特权级控制，相比简单的共享机制具有更高的安全性，可实现对数据和服务进行可控和安全的访问。

- 更高的资源利用率

虚拟化可支持实现物理资源和资源池的动态共享，提高资源利用率。

## openEuler 虚拟化

openEuler 提供了支持 AArch64 处理器架构的 KVM 虚拟化组件。

# 2 安装指南

本章介绍在 openEuler 中安装虚拟化组件的方法。

## 2.1 最低硬件要求

## 2.2 安装虚拟化核心组件

## 2.1 最低硬件要求

在 openEuler 系统中安装虚拟化组件，最低硬件要求：

- AArch64 处理器架构（建议 ARMv8 处理器）
- 2 核 CPU
- 4GB 的内存
- 16GB 可用磁盘空间

## 2.2 安装虚拟化核心组件

### 2.2.1 安装方法

#### 前提条件

- 已经配置 yum 源。配置方式请参见《openEuler 1.0 管理员指南》。
- 安装操作需要管理员权限。

#### 安装步骤

步骤 1 安装 qemu 组件。

```
# yum install -y qemu
```

步骤 2 安装 libvirt 组件。

```
# yum install -y libvirt
```

步骤 3 启动 libvirtd 服务。

```
# systemctl start libvirtd
```

----结束

### 说明

KVM 模块已经集成在 openEuler 内核中，因此不需要单独安装。

## 2.2.2 验证安装是否成功

步骤 1 查看内核是否支持 KVM 虚拟化，即查看/dev/kvm 和/sys/module/kvm 文件是否存在，命令和回显如下：

```
# ls /dev/kvm
/dev/kvm
# ls /sys/module/kvm
parameters uevent
```

若上述文件存在，说明内核支持 KVM 虚拟化。若上述文件不存在，则说明系统内核编译时未开启 KVM 虚拟化，需要更换支持 KVM 虚拟化的 Linux 内核。

步骤 2 确认 qemu 是否安装成功。若安装成功则可以看到 qemu 软件包信息，命令和回显如下：

```
# rpm -qi qemu
Name       : qemu
Epoch     : 10
Version    : 4.0.0
Release    : 1
Architecture: aarch64
Install Date: Wed 24 Jul 2019 04:04:47 PM CST
Group      : Unspecified
Size       : 16869484
License    : GPLv2 and BSD and MIT and CC-BY
Signature  : (none)
Source RPM : qemu-4.0.0-1.src.rpm
Build Date : Wed 24 Jul 2019 04:03:52 PM CST
Build Host : localhost
Relocations : (not relocatable)
URL        : http://www.qemu.org
Summary    : QEMU is a generic and open source machine emulator and virtualizer
Description :
QEMU is a generic and open source processor emulator which achieves a good
emulation speed by using dynamic translation. QEMU has two operating modes:

* Full system emulation. In this mode, QEMU emulates a full system (for
  example a PC), including a processor and various peripherals. It can be
  used to launch different Operating Systems without rebooting the PC or
  to debug system code.
* User mode emulation. In this mode, QEMU can launch Linux processes compiled
  for one CPU on another CPU.

As QEMU requires no host kernel patches to run, it is safe and easy to use.
```

**步骤 3** 确认 libvirt 是否安装成功。若安装成功则可以看到 libvirt 软件包信息，命令和回显如下：

```
# rpm -qi libvirt
Name       : libvirt
Version    : 5.5.0
Release    : 1
Architecture: aarch64
Install Date: Tue 30 Jul 2019 04:56:21 PM CST
Group      : Unspecified
Size       : 0
License    : LGPLv2+
Signature  : (none)
Source RPM : libvirt-5.5.0-1.src.rpm
Build Date : Mon 29 Jul 2019 08:14:57 PM CST
Build Host : 71e8c1c149f
Relocations : (not relocatable)
URL        : https://libvirt.org/
Summary    : Library providing a simple virtualization API
Description :
Libvirt is a C toolkit to interact with the virtualization capabilities
of recent versions of Linux (and other OSes). The main package includes
the libvirtd server exporting the virtualization support.
```

**步骤 4** 查看 libvirt 服务是否启动成功。若服务处于“Active”状态，说明服务启动成功，可以正常使用 libvirt 提供的 virsh 命令行工具，命令和回显如下：

```
# systemctl status libvirtd
● libvirtd.service - Virtualization daemon
   Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled; vendor preset:
   enabled)
   Active: active (running) since Tue 2019-08-06 09:36:01 CST; 5h 12min ago
     Docs: man:libvirtd(8)
           https://libvirt.org
    Main PID: 40754 (libvirtd)
      Tasks: 20 (limit: 32768)
     Memory: 198.6M
    CGroup: /system.slice/libvirtd.service
            -40754 /usr/sbin/libvirtd
```

----结束

# 3 用户和管理员指南

本章介绍如何在虚拟化平台上创建虚拟机，并对虚拟机进行生命周期管理、信息查询等操作，便于用户和管理员进行相应操作。

- 3.1 准备环境
- 3.2 虚拟机配置
- 3.3 管理虚拟机
- 3.4 热迁移虚拟机
- 3.5 管理系统资源
- 3.6 管理设备
- 3.7 最佳实践

## 3.1 准备环境

### 3.1.1 准备虚拟机镜像

#### 概述

虚拟机镜像是一个文件，包含了已经完成安装并且可启动操作系统的虚拟磁盘。虚拟机镜像具有不同格式，常见的有 raw 格式和 qcow2 格式。qcow2 格式镜像相比 raw 格式，具有占用更小的空间，支持快照、Copy-On-Write、AES 加密、zlib 压缩等特性，但性能略逊于 raw 格式镜像。镜像文件的制作借助于 qemu-img 工具，本节以 qcow2 格式镜像文件为例，介绍虚拟机镜像制作过程。

#### 制作镜像

制作 qcow2 格式镜像文件的操作步骤如下：

步骤 1 安装 qemu-img 软件包。

```
# yum install -y qemu-img
```

步骤 2 使用 `qemu-img` 工具的 `create` 命令，创建镜像文件，命令格式为：

```
$ qemu-img create -f <imgFormat> -o <fileOption> <fileName> <diskSize>
```

其中，各参数含义如下：

- *imgFormat*: 镜像格式，取值为 `raw`, `qcow2` 等。
- *fileOption*: 文件选项，用于设置镜像文件的特性，如指定后端镜像文件，压缩，加密等特性。
- *fileName*: 文件名称。
- *diskSize*: 磁盘大小，用于指定块磁盘设备的大小，支持的单位有 `K`、`M`、`G`、`T`，分别代表 `KiB`、`MiB`、`GiB`、`TiB`。

例如，创建一个磁盘设备大小为 4GB、格式为 `qcow2` 的镜像文件 `openEuler-image.qcow2`，命令和回显如下：

```
$ qemu-img create -f qcow2 openEuler-image.qcow2 4G
Formatting 'openEuler-image.qcow2', fmt=qcow2 size=4294967296 cluster size=65536
lazy_refcounts=off refcount_bits=16
```

----结束

## 修改镜像磁盘空间大小

当虚拟机需要更大的磁盘空间时，可以使用 `qemu-img` 工具，修改虚拟机镜像磁盘空间的大小，修改方法如下。

步骤 1 查询当前虚拟机镜像磁盘空间大小，命令如下：

```
# qemu-img info <imgFileName>
```

例如，查询 `openEuler-image.qcow2` 镜像磁盘空间大小的命令和回显如下，说明该镜像磁盘空间大小为 4GiB。

```
# qemu-img info openEuler-image.qcow2
image: openEuler-image.qcow2
file format: qcow2
virtual size: 4.0G (4294967296 bytes)
disk size: 196K
cluster size: 65536
Format specific information:
  compat: 1.1
  lazy refcounts: false
  refcount bits: 16
  corrupt: false
```

步骤 2 修改镜像磁盘空间大小，命令如下，其中 *imgFileName* 为镜像名称，“+”和“-”分别表示需要增加或减小的镜像磁盘空间大小，单位为 `K`、`M`、`G`、`T`，代表 `KiB`、`MiB`、`GiB`、`TiB`。

```
# qemu-img resize <imgFileName> [+|-]<size>
```

例如，将上述 `openEuler-image.qcow2` 镜像磁盘空间大小扩展到 24GiB，即在原来 4GiB 基础上增加 20GiB，命令和回显如下：

```
# qemu-img resize openEuler-image.qcow2 +20G
Image resized.
```

**步骤 3** 查询修改后的镜像磁盘空间大小，确认是否修改成功，命令如下：

```
# qemu-img info <imgFileName>
```

例如，上述 openEuler-image.qcow2 镜像磁盘空间已扩展到 24GiB，命令和回显如下：

```
# qemu-img info openEuler-image.qcow2
image: openEuler-image.qcow2
file format: qcow2
virtual size: 24G (25769803776 bytes)
disk size: 200K
cluster size: 65536
Format specific information:
  compat: 1.1
  lazy refcounts: false
  refcount bits: 16
  corrupt: false
```

----结束

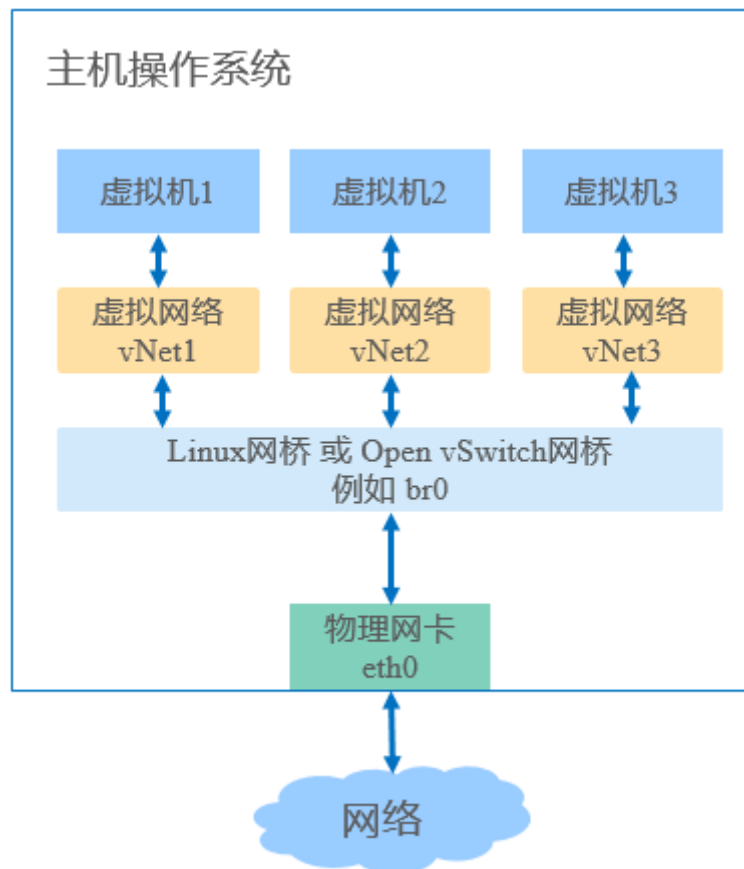
## 3.1.2 准备虚拟机网络

### 概述

为了使虚拟机可以与外部进行网络通信，需要为虚拟机配置网络环境。KVM 虚拟化支持 Linux 网桥、Open vSwitch 网桥等多种类型的网桥。如图 3-1 所示，数据传输路径为“虚拟机 -> 虚拟网卡设备 -> Linux 网桥或 Open vSwitch 网桥 -> 物理网卡”。创建网桥，除了为虚拟机配置虚拟网卡设备外，为主机创建网桥是连接虚拟化网络的关键。

本节给出搭建 Linux 网桥和 Open vSwitch 网桥的方法，使虚拟机连接到网络，用户可以根据情况选择搭建网桥的类型。

图3-1 虚拟网络结构图



## 搭建 Linux 网桥

以物理网卡 eth0 绑定到 Linux 网桥 br0 的操作为例，搭建 Linux 网桥的操作步骤如下：

步骤 1 安装 bridge-utils 软件包。

Linux 网桥通常通过 brctl 工具管理，其对应的安装包为 bridge-utils，安装命令如下：

```
# yum install -y bridge-utils
```

步骤 2 创建网桥 br0。

```
# brctl addbr br0
```

步骤 3 将物理网卡 eth0 绑定到 Linux 网桥。

```
# brctl addif br0 eth0
```

步骤 4 eth0 与网桥连接后，不再需要 IP 地址，将 eth0 的 IP 设置为 0.0.0.0。

```
# ifconfig eth0 0.0.0.0
```

步骤 5 设置 br0 的 IP 地址。



- 如果有 DHCP 服务器，可以通过 `dhclient` 设置动态 IP 地址。

```
# dhclient br0
```

- 如果没有 DHCP 服务器，给 `br0` 配置静态 IP，例如设置静态 IP 为 192.168.1.2，子网掩码为 255.255.255.0。

```
# ifconfig br0 192.168.1.2 netmask 255.255.255.0
```

----结束

## 搭建 Open vSwitch 网桥

Open vSwitch 网桥，具有更便捷的自动化编排能力。搭建 Open vSwitch 网桥需要安装网络虚拟化组件，这里介绍总体操作。

### 一、安装 Open vSwitch 组件

使用 Open vSwitch 提供虚拟网络，需要安装 Open vSwitch 网络虚拟化组件。

步骤 1 安装 Open vSwitch 组件。

```
# yum install -y openvswitch-kmod
# yum install -y openvswitch
```

步骤 2 启动 Open vSwitch 服务。

```
# systemctl start openvswitch
```

----结束

### 二、确认安装是否成功

确认 Open vSwitch 组件是否安装成功，需要检查 `openvswitch-kmod` 和 `openvswitch` 这两个组件是否安装成功。

步骤 1 确认 `openvswitch-kmod` 组件是否安装成功。若安装成功，可以看到软件包相关信息，命令和回显如下：

```
# rpm -qi openvswitch-kmod
Name       : openvswitch-kmod
Version    : 2.11.1
Release    : 1.oe3
Architecture: aarch64
Install Date: Thu 15 Aug 2019 05:07:49 PM CST
Group      : System Environment/Daemons
Size       : 15766774
License    : GPLv2
Signature  : (none)
Source RPM : openvswitch-kmod-2.11.1-1.oe3.src.rpm
Build Date : Thu 08 Aug 2019 04:33:08 PM CST
Build Host : armbuild10b175b113b44
Relocations : (not relocatable)
Vendor     : OpenSource Security Ralf Spennberg <ralf@os-s.net>
URL        : http://www.openvswitch.org/
Summary    : Open vSwitch Kernel Modules
Description:
Open vSwitch provides standard network bridging functions augmented with
```

```
support for the OpenFlow protocol for remote per-flow control of
traffic. This package contains the kernel modules.
```

**步骤 2** 确认 openvswitch 组件是否安装成功。若安装成功，可以看到软件包相关信息，命令和回显如下：

```
# rpm -qi openvswitch
Name       : openvswitch
Version    : 2.11.1
Release    : 1
Architecture: aarch64
Install Date: Thu 15 Aug 2019 05:08:35 PM CST
Group      : System Environment/Daemons
Size       : 6051185
License    : ASL 2.0
Signature  : (none)
Source RPM : openvswitch-2.11.1-1.src.rpm
Build Date : Thu 08 Aug 2019 05:24:46 PM CST
Build Host : armbuild10b247b121b105
Relocations : (not relocatable)
Vendor     : Nicira, Inc.
URL        : http://www.openvswitch.org/
Summary    : Open vSwitch daemon/database/utilities
Description :
Open vSwitch provides standard network bridging functions and
support for the OpenFlow protocol for remote per-flow control of
traffic.
```

**步骤 3** 查看 Open vSwitch 服务是否启动成功。若服务处于“Active”状态，说明服务启动成功，可以正常使用 Open vSwitch 提供的命令行工具，命令和回显如下：

```
# systemctl status openvswitch
● openvswitch.service - LSB: Open vSwitch switch
   Loaded: loaded (/etc/rc.d/init.d/openvswitch; generated)
   Active: active (running) since Sat 2019-08-17 09:47:14 CST; 4min 39s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 54554 ExecStart=/etc/rc.d/init.d/openvswitch start (code=exited,
status=0/SUCCESS)
    Tasks: 4 (limit: 9830)
   Memory: 22.0M
    CGroup: /system.slice/openvswitch.service
            └─54580 ovsdb-server: monitoring pid 54581 (healthy)
              └─54581 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer -vsyslog:err
-vfile:info --remote=punix:/var/run/openvswitch/db.sock --private-
key=db:Open vSwitch,SSL,private key --certificate>
                └─54602 ovs-vswitchd: monitoring pid 54603 (healthy)
                  └─54603 ovs-vswitchd unix:/var/run/openvswitch/db.sock -vconsole:emer -
vsyslog:err -vfile:info --mlockall --no-chdir --log-file=/var/log/openvswitch/ovs-
vswitchd.log --pidfile=/var/run/open>
```

----结束

### 三、搭建 Open vSwitch 网桥

以创建 Open vSwitch 一层网桥 br0 为例，介绍搭建方法。

步骤 1 创建 Open vSwitch 网桥 br0。

```
# ovs-vsctl add-br br0
```

步骤 2 将物理网卡 eth0 添加到 br0。

```
# ovs-vsctl add-port br0 eth0
```

步骤 3 eth0 与网桥连接后，不再需要 IP 地址，将 eth0 的 IP 设置为 0.0.0.0。

```
# ifconfig eth0 0.0.0.0
```

步骤 4 为 OVS 网桥 br0 分配 IP。

- 如果有 DHCP 服务器，可以通过 dhclient 设置动态 IP 地址。

```
# dhclient br0
```

- 如果没有 DHCP 服务器，给 br0 配置静态 IP，例如 192.168.1.2。

```
# ifconfig br0 192.168.1.2
```

----结束

### 3.1.3 准备 UEFI 引导工具集 EDK II

#### 概述

统一的可扩展固件接口 UEFI（Unified Extensible Firmware Interface）是一种全新类型的接口标准，用于开机自检、引导操作系统的启动，是传统 BIOS 的一种替代方案。EDK II 是一套实现了 UEFI 标准的开源代码，在虚拟化场景中，通常利用 EDK II 工具集，通过 UEFI 的方式启动虚拟机。使用 EDK II 工具需要在虚拟机启动之前安装对应的软件包，本节介绍 EDK II 的安装方法。

#### 安装方法

安装工具集 EDK II 的操作步骤如下：

步骤 1 安装 edk 软件包，命令如下：

```
# yum install -y edk2-aarch64
```

步骤 2 查询 edk 软件是否安装成功，命令如下：

```
# rpm -qi edk2-aarch64
```

若 edk 软件安装成功，命令和回显如下：

```
# rpm -qi edk2-aarch64
Name       : edk2-aarch64
Version    : 20180815gitcb5f4f45ce
Release    : 1.oe3
Architecture: noarch
Install Date: Mon 22 Jul 2019 04:52:33 PM CST
Group      : Applications/Emulators
```

----结束

## 3.2 虚拟机配置

### 3.2.1 总体介绍

#### 概述

Libvirt 工具采用 XML 格式的文件描述一个虚拟机特征，包括虚拟机名称、CPU、内存、磁盘、网卡、鼠标、键盘等信息。用户可以通过修改配置文件，对虚拟机进行管理。本章介绍 XML 配置文件各个元素的含义，指导用户完成虚拟机配置。

#### 基本格式

虚拟机 XML 配置文件以 **domain** 为根元素，**domain** 根元素中包含多个其他元素。XML 配置文件中的部分元素可以包含对应属性和属性值，用以详细地描述虚拟机信息，同一元素的不同属性使用空格分开。

XML 配置文件的基本格式如下，其中 **label** 代表具体标签名，**attribute** 代表属性，**value** 代表属性值，需要根据实际情况修改。

```
<domain type='kvm'>
  <name>VMName</name>
  <memory attribute='value'>8</memory>
  <vcpu>4</vcpu>
  <os>
    <label attribute='value' attribute='value'>
      ...
    </label>
  </os>
  <label attribute='value' attribute='value'>
    ...
  </label>
</domain>
```

#### 配置流程

1. 创建一个根元素为 **domain** 的 XML 配置文件。
2. 使用标签 **name**，根据命名规则指定唯一的虚拟机名称。
3. 配置虚拟 CPU 和虚拟内存等系统资源。
4. 配置虚拟设备。
  - a. 配置存储设备。
  - b. 配置网络设备。
  - c. 配置外部总线结构。
  - d. 配置鼠标等外部设备。
5. 保存 XML 配置文件。

## 3.2.2 虚拟机描述

### 概述

本节介绍虚拟机 **domain** 根元素和虚拟机名称的配置。

### 元素介绍

- **domain**: 虚拟机 XML 配置文件的根元素，用于配置运行此虚拟机的 hypervisor 的类型。  
属性 **type**: 虚拟化中 domain 的类型。openEuler 虚拟化中属性值为 **kvm**。
- **name**: 虚拟机名称。  
虚拟机名称为一个字符串，同一个主机上的虚拟机名称不能重复，虚拟机名称必须由数字、字母、“\_”、“-”、“:” 组成，但不支持全数字的字符串，且虚拟机名称不超过 64 个字符。

### 配置示例

例如，虚拟机名称为 **openEuler** 的配置如下：

```
<domain type='kvm'>
  <name>openEuler</name>
  ...
</domain>
```

## 3.2.3 虚拟 CPU 和虚拟内存

### 概述

本节介绍虚拟 CPU 和虚拟内存的常用配置。

### 元素介绍

- **vcpu**: 虚拟处理器的个数。
- **memory**: 虚拟内存的大小。  
属性 **unit**: 指定内存单位，属性值支持 **KiB** ( $2^{10}$  字节)，**MiB** ( $2^{20}$  字节)，**GiB** ( $2^{30}$  字节)，**TiB** ( $2^{40}$  字节) 等。
- **cpu**: 虚拟处理器模式。  
属性 **mode**: 表示虚拟 CPU 的模式，属性值 **host-passthrough** 表示虚拟 CPU 的架构和特性与主机保持一致。  
子元素 **topology**: 元素 **cpu** 的子元素，用于描述虚拟 CPU 模式的拓扑结构。
  - 子元素 **topology** 的属性 **socket**、**cores**、**threads** 分别描述了虚拟机具有多少个 **cpu socket**，每个 **cpu socket** 中包含多少个处理核心 (**core**)，每个处理器核心具有多少个超线程 (**thread**)，属性值为正整数且三者的乘积等于虚拟 CPU 的个数。

配置示例

例如，虚拟 CPU 个数为 4，处理模式为 host-passthrough，虚拟内存为 8GiB，4 个 CPU 分布在两个 CPU socket 中，且不支持超线程的配置如下：

```
<domain type='kvm'>
...
<vcpu>4</vcpu>
<memory unit='GiB'>8</memory>
<cpu mode='host-passthrough'>
  <topology sockets='2' cores='2' threads='1' />
</cpu>
...
</domain>
```

3.2.4 配置虚拟设备

虚拟机 XML 配置文件使用 devices 元素配置虚拟设备，包括存储设备、网络设备、总线、鼠标等，本节介绍常用的虚拟设备如何配置。

3.2.4.1 存储设备

概述

XML 配置文件可以配置虚拟存储设备信息，包括软盘、磁盘、光盘等存储介质及其存储类型等信息，本节介绍存储设备的配置方法。

元素介绍

XML 配置文件使用 disk 元素配置存储设备，disk 常见的属性如表 3-1 所示，常见子元素及子元素属性如表 3-2 所示。

表3-1 元素 disk 的常用属性

元素	属性	含义	属性值及其含义
disk	type	指定后端存储介质类型	block: 块设备 file: 文件设备 dir: 目录路径
	device	指定呈现给虚拟机的存储介质	disk: 磁盘（默认） floppy: 软盘 cdrom: 光盘

表3-2 元素 disk 的常用子元素及属性说明

子元素	子元素含义	属性说明
-----	-------	------

子元素	子元素含义	属性说明
source	指定后端存储介质，与 disk 元素的属性“type”指定类型相对应	file: 对应 file 类型，值为对应文件的完全限定路径。 dev: 对应 block 类型，值为对应主机设备的完全限定路径。 dir: 对应 dir 类型，值为用作磁盘目录的完全限定路径。
driver	指定后端驱动器的详细信息	type: 磁盘格式的类型，常用的有“raw”和“qcow2”，需要与 source 的格式一致。 io: 磁盘 IO 模式，支持“native”和“threads”选项。 cache: 磁盘的 cache 模式，可选项有“none”、“writethrough”、“writeback”、“directsync”等。 iothread: 指定为磁盘分配的 IO 线程。
target	指磁盘呈现给虚拟机的总线和设备	dev: 指定磁盘的逻辑设备名称，如 SCSI、SATA、USB 类型总线常用命令习惯为 sd[a-p]，IDE 类型设备磁盘常用命名习惯为 hd[a-d]。 bus: 指定磁盘设备的类型，常见的有“scsi”、“usb”、“sata”、“virtio”等类型。
boot	表示此磁盘可以作为启动盘使用	order: 指定磁盘的启动顺序。
readonly	表示磁盘具有只读属性，磁盘内容不可以被虚拟机修改，通常与光驱结合使用	-

## 配置示例

按照 3.1.1 准备虚拟机镜像操作完成虚拟机镜像准备后，可以使用如下 XML 配置文件示例，为虚拟机配置虚拟磁盘。

例如，该示例为虚拟机配置了两个 IO 线程，一个块磁盘设备和一个光盘设备，第一个 IO 线程分配给块磁盘设备使用。该块磁盘设备的后端介质为 qcow2 格式，且被作为优先启动盘。

```
<domain type='kvm'>
  ...
  <iothreads>2</iothreads>
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native' iothread="1"/>
      <source file='/mnt/openEuler-image.qcow2' />
      <target dev='vda' bus='virtio' />
      <boot order='1' />
    </disk>
  </devices>
</domain>
```

```
</disk>
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw' cache='none' io='native' />
  <source file='/mnt/openEuler-dvd.iso' />
  <target dev='sdb' bus='scsi' />
  <readonly />
  <boot order='2' />
</disk>
...
</devices>
</domain>
```

### 3.2.4.2 网络设备

#### 概述

XML 配置文件可以配置虚拟网络设备，包括 ethernet 模式、bridge 模式、vhostuser 模式等，本节介绍虚拟网卡设备的配置方法。

#### 元素介绍

XML 配置文件中元素“interface”，其属性“type”表示虚拟网卡的模式，可选的值有“ethernet”、“bridge”、“vhostuser”等，下面以“bridge”模式虚拟网卡为例介绍其子元素以及对应的属性。

表3-3 bridge 模式虚拟网卡常用子元素

子元素	子元素含义	属性及含义
mac	虚拟网卡的 mac 地址	address: 指定 mac 地址，若不配置，会自动生成。
target	后端虚拟网卡名	dev: 创建的后端 tap 设备的名称。
source	指定虚拟网卡后端	bridge: 与 bridge 模式联合使用，值为网桥名称。
boot	表示此网卡可以作为远程启动	order: 指定网卡的启动顺序。
model	表示虚拟网卡的类型	type: bridge 模式网卡通常使用 virtio。
virtualport	端口类型	type: 若使用 OVS 网桥，需要配置为 openvswitch。
driver	后端驱动类型	name: 驱动名称，通常取值为 vhost。 queues: 网卡设备队列数。



## 配置示例

- 按照 3.1.2 准备虚拟机网络创建了 Linux 网桥 br0 后，配置一个桥接在 br0 网桥上的 virtio 类型的虚拟网卡设备，对应的 XML 配置如下：

```
<domain type='kvm'>
...
<devices>
  <interface type='bridge'>
    <source bridge='br0' />
    <model type='virtio' />
  </interface>
  ...
</devices>
</domain>
```

- 如果按照 3.1.2 准备虚拟机网络创建了 OVS 网桥，配置一个后端使用 vhost 驱动，且具有四个队列的 virtio 虚拟网卡设备。

```
<domain type='kvm'>
...
<devices>
  <interface type='bridge'>
    <source bridge='br0' />
    <virtualport type='openvswitch' />
    <model type='virtio' />
    <driver name='vhost' queues='4' />
  </interface>
  ...
</devices>
</domain>
```

### 3.2.4.3 总线配置

#### 概述

总线是计算机各个部件之间进行信息通信的通道。外部设备需要挂载到对应的总线上，每个设备都会被分配一个唯一地址（由子元素 `address` 指定），通过总线网络完成与其他设备或中央处理器的信息交换。常见的设备总线有 ISA 总线、PCI 总线、USB 总线、SCSI 总线、PCIe 总线。

PCIe 总线是一种典型的树结构，具有比较好的扩展性，总线之间通过控制器关联，这里以 PCIe 总线为例介绍如何为虚拟机配置总线拓扑。

#### 说明

总线的配置相对比较繁琐，若不需要精确控制设备拓扑结构，可以使用 Libvirt 自动生成的缺省总线配置。

#### 元素介绍

在 Libvirt 的 XML 配置中，每个控制器元素（使用 `controller` 元素表示）可以表示一个总线，根据虚拟机架构的不同，一个控制器上通常可以挂载一个或多个控制器或设备。这里介绍常用属性和子元素。

**controller:** 控制器元素，表示一个总线。

- 属性 **type**: 控制器必选属性, 表示总线类型。常用取值有 “pci”、“usb”、“scsi”、“virtio-serial”、“fdc”、“ccid”。
- 属性 **index**: 控制器必选属性, 表示控制器的总线 “bus” 编号 (编号从 0 开始), 可以在地址元素 “address” 元素中使用。
- 属性 **model**: 控制器必选属性, 表示控制器的具体型号, 其可选择的值与控制器类型 “type” 的值相关, 对应关系及含义请参见表 3-4。
- 子元素 **address**: 为设备或控制器指定其在总线网络中的挂载位置。
  - 属性 **type**: 设备地址类型。常用取值有 “pci”、“usb”、“drive”。address 的 type 类型不同, 对应的属性也不同, 常用 type 属性值及其该取值下 address 的属性请参见表 3-5。
- 子元素 **model**: 控制器具体型号的名称。
  - 属性 **name**: 指定控制器具体型号的名称, 和父元素 controller 中的属性 model 对应。

表3-4 controller 属性 type 常用取值和 model 取值对应关系

type 属性值	model 属性值	简介
pci	pcie-root	PCIe 根节点, 可挂载 PCIe 设备或控制器
	pcie-root-port	只有一个 slot, 可以挂载 PCIe 设备或控制器
	pcie-to-pci-bridge	PCIe 转 PCI 桥控制器, 可挂载 PCI 设备
usb	ehci	USB 2.0 控制器, 可挂载 USB 2.0 设备
	nec-xhci	USB 3.0 控制器, 可挂载 USB 3.0 设备
scsi	virtio-scsi	virtio 类型 SCSI 控制器, 可以挂载块设备, 如磁盘, 光盘等
virtio-serial	virtio-serial	virtio 类型串口控制器, 可挂载串口设备, 如 pty 串口

表3-5 address 元素不同设备类型下的属性说明

类型 type 属性值	含义	对应地址属性
pci	地址类型为 PCI 地址, 表示该设备在 PCI 总线网络中的挂载位置。	domain: PCI 设备的域号 bus: PCI 设备的 bus 号 slot: PCI 设备的 device 号 function: PCI 设备的 function 号 multifunction: controller 元素可选, 是否开启 multifunction 功能

类型 type 属性值	含义	对应地址属性
usb	地址类型为 USB 地址，表示该设备在 USB 总线中的位置。	bus: USB 设备的 bus 号 port: USB 设备的 port 号
drive	地址类型存储设备地址，表示所属的磁盘控制器，及其在总线中的位置。	controller: 指定所属控制器号 bus: 设备输出的 channel 号 target: 存储设备 target 号 unit: 存储设备 lun 号

## 配置示例

该示例给出一个 PCIe 总线的拓扑结构。PCIe 根节点（BUS 0）下挂载了三个 PCIe-Root-Port 控制器。第一个 PCIe-Root-Port 控制器（BUS 1）开启了 multifunction 功能，并在其下挂载一个 PCIe-to-PCI-bridge 控制器，形成了一个 PCI 总线（BUS 3），该 PCI 总线上挂载了一个 virtio-serial 设备和一个 USB 2.0 控制器。第二个 PCIe-Root-Port 控制器（BUS 2）下挂载了一个 SCSI 控制器。第三个 PCIe-Root-Port 控制器（BUS 0）下无挂载设备。配置内容如下：

```
<domain type='kvm'>
...
<devices>
  <controller type='pci' index='0' model='pcie-root'/>
  <controller type='pci' index='1' model='pcie-root-port'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0'
multifunction='on'/>
  </controller>
  <controller type='pci' index='2' model='pcie-root-port'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='pci' index='3' model='pcie-to-pci-bridge'>
    <model name='pcie-pci-bridge'/>
    <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
  </controller>
  <controller type='pci' index='4' model='pcie-root-port'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
  </controller>
  <controller type='scsi' index='0' model='virtio-scsi'>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
  </controller>
  <controller type='virtio-serial' index='0'>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x02' function='0x0'/>
  </controller>
  <controller type='usb' index='0' model='ehci'>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x01' function='0x0'/>
  </controller>
...
```

```
</devices>  
</domain>
```

### 3.2.4.4 其它常用设备

#### 概述

除存储设备、网络设备外，XML 配置文件中还需要指定一些其他外部设备，本节介绍这些元素的配置方法。

#### 元素介绍

- **serial**: 串口设备  
属性 **type**: 用于指定串口类型。常用属性值为 **pty**、**tcp**、**pipe**、**file**。
- **video**: 媒体设备  
属性 **type**: 媒体设备类型。常用属性值为 **virtio**。  
子元素 **model**: **video** 的子元素，用于指定媒体设备类型。
- **input**: 输出设备  
属性 **type**: 指定输出设备类型。常用属性值为 **tablet**、**keyboard**，分别表示输出设备为写字板、键盘。  
属性 **bus**: 指定挂载的总线。常用属性值为 **USB**。
- **emulator**: 模拟器应用路径
- **graphics**: 图形设备  
属性 **type**: 指定图形设备类型。常用属性值为 **vnc**。  
属性 **listen**: 指定侦听的 IP 地址。

#### 配置示例

例如，在下面的示例中，配置了虚拟机的模拟器路径，**pty** 串口、**virtio** 媒体设备、**USB** 写字板、**USB** 键盘以及 **VNC** 图形设备。

#### 说明

**graphics** 的 **type** 配置为 **VNC** 时，建议配置属性 **passwd**，即使用 **VNC** 登录时的密码。

```
<domain type='kvm'>  
  ...  
  <devices>  
    <emulator>/usr/libexec/qemu-kvm</emulator>  
    <console type='pty'/>  
    <video>  
      <model type='virtio'/>  
    </video>  
    <input type='tablet' bus='usb'/>  
    <input type='keyboard' bus='usb'/>  
    <graphics type='vnc' listen='0.0.0.0' passwd='n8VfjbFK'/>  
    ...  
  </devices>  
</domain>
```

## 3.2.5 其他常见配置项

### 概述

除系统资源和虚拟设备外，XML 配置文件还需要配置一些其他元素，本节介绍这些元素的配置方法。

### 元素介绍

- **os**: 定义虚拟机启动参数。  
子元素 **type**: 指定虚拟机类型，属性 **arch** 表示架构类型，如 **aarch64**，属性 **machine** 表示虚拟机的芯片组类型，如 **aarch64** 结构使用 “**virt-4.0**” 类型。  
子元素 **loader**: 指定加载固件，如配置 EDK 提供的 EFI 文件，属性 **readonly** 表示是否是只读文件，值为 “**yes**” 或 “**no**”，属性 **type** 表示 loader 的类型，常用的值有 “**rom**”、“**pflash**”。  
子元素 **nvr**am: 指定 **nvr**am 文件路径，用于存储 EFI 启动配置。
- **features**: Hypervisor 支持控制一些虚拟机 CPU/machine 的特性，如高级电源管理接口 “**acpi**”，ARM 处理器指定 GICv3 中断控制器等。
- **iothreads**: 指定 **iothread** 数量，可以用于加速存储设备性能。
- **on\_poweroff**: 虚拟机关闭时采取的动作。
- **on\_reboot**: 虚拟机重启时采取的动作。
- **on\_crash**: 虚拟机崩溃时采取的动作。
- **clock**: 采用的时钟类型。  
属性 **offset**: 设置虚拟机时钟的同步类型，可选的值有 “**localtime**”、“**utc**”、“**timezone**”、“**variable**” 等。

### 配置示例

虚拟机的类型为 **aarch64** 结构，使用 **virt-4.0** 芯片组，利用 UEFI 启动的虚拟机配置如下：

```
<domain type='kvm'>
...
<os>
  <type arch='aarch64' machine='virt-4.0'>hvm</type>
  <loader readonly='yes' type='pflash'>/usr/share/edk2/aarch64/QEMU_EFI-
pflash.raw</loader>
  <nvr>am</nvr>/var/lib/libvirt/qemu/nvr/
am/openEulerVM.fd</nvr>
</os>
...
</domain>
```

为虚拟机配置 **ACPI** 和 **GIC V3** 中断控制器特性。

```
<features>
  <acpi/>
  <gic version='3' />
</features>
```

为虚拟机配置两个 `iothread`，用于加速存储设备性能。

```
<iothreads>2</iothreads>
```

虚拟机关闭时，销毁虚拟机。

```
<on_poweroff>destroy</on_poweroff>
```

虚拟机重启时，重新启动虚拟机。

```
<on_reboot>restart</on_reboot>
```

虚拟机崩溃时，重新启动虚拟机。

```
<on_crash>restart</on_crash>
```

时钟采用“`utc`”的同步方式。

```
<clock offset='utc' />
```

## 3.2.6 XML 配置文件示例

### 概述

本节给出两个基本的虚拟机 XML 配置文件示例，供用户参考。

### 示例一

一个包含基本元素的 XML 配置文件，其内容示例如下：

```
<domain type='kvm'>
  <name>openEulerVM</name>
  <memory unit='GiB'>8</memory>
  <vcpu>4</vcpu>
  <os>
    <type arch='aarch64' machine='virt-4.0'>hvm</type>
    <loader readonly='yes' type='pflash'>/usr/share/edk2/aarch64/QEMU EFI-
pflash.raw</loader>
    <nvram>/var/lib/libvirt/qemu/nvram/openEulerVM.fd</nvram>
  </os>
  <features>
    <acpi/>
    <gic version='3' />
  </features>
  <cpu mode='host-passthrough'>
    <topology sockets='2' cores='2' threads='1' />
  </cpu>
  <iothreads>1</iothreads>
  <clock offset='utc' />
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>restart</on_crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' iothread="1"/>
      <source file='/mnt/openEuler-image.qcow2' />
    </disk>
  </devices>
</domain>
```

```
<target dev='vda' bus='virtio'>
  <boot order='1'>
</disk>
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw'>
  <source file='/mnt/openEuler-1.0-aarch64-dvd.iso'>
  <readonly>
  <target dev='sdb' bus='scsi'>
  <boot order='2'>
</disk>
<interface type='bridge'>
  <source bridge='br0'>
  <model type='virtio'>
</interface>
<console type='pty'>
  <video>
    <model type='virtio'>
  </video>
  <controller type='scsi' index='0' model='virtio-scsi'>
<controller type='usb' model='ehci'>
<input type='tablet' bus='usb'>
<input type='keyboard' bus='usb'>
<graphics type='vnc' listen='0.0.0.0' passwd='n8VfjbFK'>
</devices>
</domain>
```

## 示例二

一个包含基本元素及总线元素的 XML 配置文件，其配置示例如下：

```
<domain type='kvm'>
  <name>openEulerVM</name>
  <memory unit='GiB'>8</memory>
  <vcpu>4</vcpu>
  <os>
    <type arch='aarch64' machine='virt-4.0'>hvm</type>
    <loader readonly='yes' type='pflash'>/usr/share/edk2/aarch64/QEMU_EFI-
pflash.raw</loader>
    <nvram>/var/lib/libvirt/qemu/nvram/openEulerVM.fd</nvram>
  </os>
  <features>
    <acpi/>
    <gic version='3'>
  </features>
  <cpu mode='host-passthrough'>
    <topology sockets='2' cores='2' threads='1'>
  </cpu>
  <iothreads>1</iothreads>
  <clock offset='utc'>
  <on poweroff>destroy</on poweroff>
  <on reboot>restart</on reboot>
  <on crash>restart</on crash>
  <devices>
    <emulator>/usr/libexec/qemu-kvm</emulator>
    <disk type='file' device='disk'>
```

```
<driver name='qemu' type='qcow2' iothread="1"/>
<source file='/mnt/openEuler-image.qcow2'/>
<target dev='vda' bus='virtio'/>
<boot order='1'/>
</disk>
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source file='/mnt/openEuler-1.0-aarch64-dvd.iso'/>
  <readonly/>
  <target dev='sdb' bus='scsi'/>
  <boot order='2'/>
</disk>
  <controller type='pci' index='0' model='pcie-root'/>
  <controller type='pci' index='1' model='pcie-root-port'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0'
multifunction='on'/>
  </controller>
  <controller type='pci' index='2' model='pcie-root-port'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
  </controller>
  <controller type='pci' index='3' model='pcie-to-pci-bridge'>
    <model name='pcie-pci-bridge'/>
    <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
  </controller>
  <controller type='pci' index='4' model='pcie-root-port'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x2'/>
  </controller>
  <controller type='scsi' index='0' model='virtio-scsi'>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
  </controller>
  <controller type='virtio-serial' index='0'>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x02' function='0x0'/>
  </controller>
  <controller type='usb' index='0' model='ehci'>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x01' function='0x0'/>
  </controller>
  <interface type='bridge'>
    <source bridge='br0'/>
    <model type='virtio'/>
  </interface>
  <console type='pty'/>
  <video>
    <model type='virtio'/>
  </video>
  <input type='tablet' bus='usb'/>
  <input type='keyboard' bus='usb'/>
  <graphics type='vnc' listen='0.0.0.0' passwd='n8VfjbFK'/>
</devices>
</domain>
```



## 3.3 管理虚拟机

### 3.3.1 虚拟机生命周期

#### 3.3.1.1 总体介绍

##### 概述

为了更好地利用硬件资源，降低成本，用户需要合理地管理虚拟机。本节介绍虚拟机生命周期过程中的基本操作，包括虚拟机创建、使用、删除等，指导用户更好地管理虚拟机。

##### 虚拟机状态

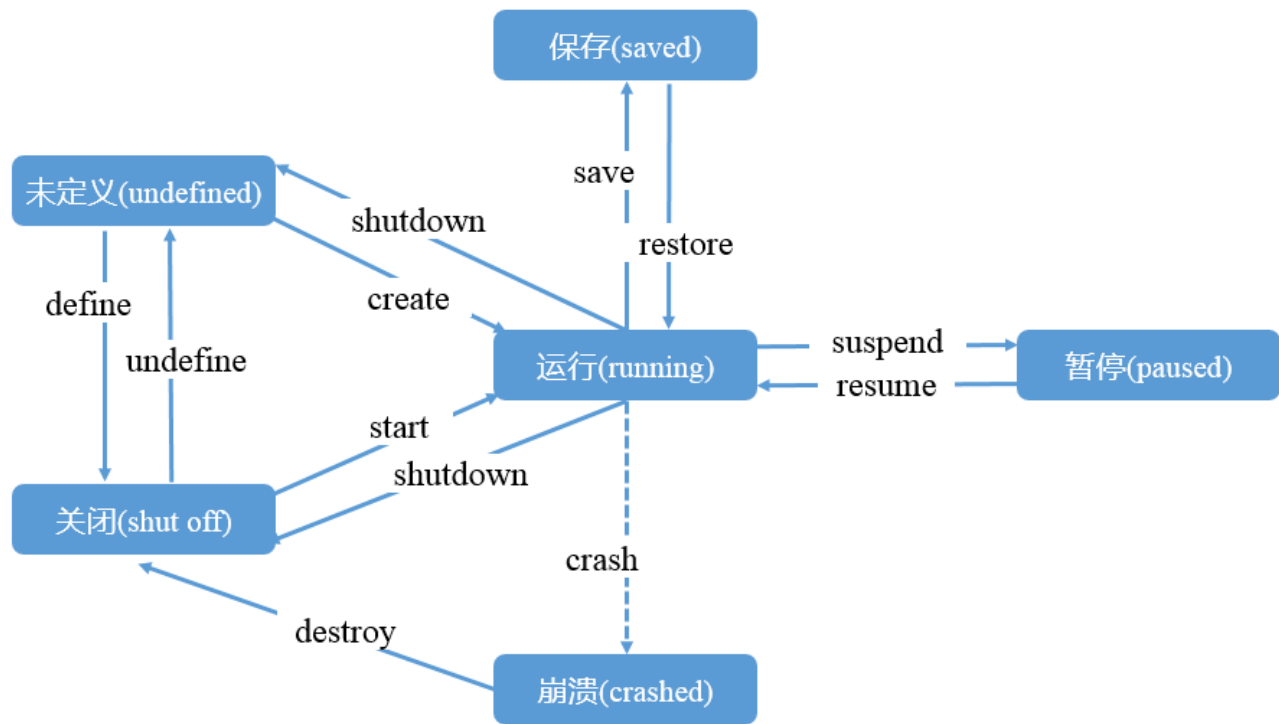
虚拟机主要有如下几种状态：

- 未定义（undefined）：虚拟机未定义或未创建，即 libvirt 认为该虚拟机不存在。
- 关闭状态（shut off）：虚拟机已经被定义但未运行，或者虚拟机被终止。
- 运行中（running）：虚拟机处于运行状态。
- 暂停（paused）：虚拟机运行被挂起，其运行状态被临时保存在内存中，可以恢复到运行状态。
- 保存（saved）：与暂停（paused）状态类似，其运行状态被保存在持久性存储介质中，可以恢复到运行状态。
- 崩溃（crashed）：通常是由于内部错误导致虚拟机崩溃，不可恢复到运行状态。

##### 状态转换

虚拟机不同状态之间可以相关转换，但必须满足一定规则。虚拟机不同状态之间的转换常用规则如图 3-2 所示。

图3-2 状态转换图



虚拟机标识

在 Libvirt 中，完成创建的虚拟机实例称做一个“domain”，其描述了虚拟机的 CPU、内存、网络设备、存储设备等各种资源的配置信息。在同一个主机上，每个 domain 具有唯一标识，通过虚拟机名称 Name、UUID、Id 表示，对应含义请参见表 3-6。在虚拟机生命周期期间，可以通过虚拟机标识对特定虚拟机进行操作。

表3-6 domain 标识说明

标识	含义
Name	虚拟机名称
UUID	通用唯一识别码
Id	虚拟机运行标识
	说明 关闭状态的虚拟机无此标识。

说明

可通过 virsh 命令查询虚拟机 Id 和 UUID，操作方法请参见 3.3.3 查询虚拟机信息章节内容。

### 3.3.1.2 管理命令

#### 概述

用户可以使用 `virsh` 命令工具管理虚拟机生命周期。本节介绍生命周期相关的命令以指导用户使用。

#### 前提条件

- 执行虚拟机生命周期操作之前，需要查询虚拟机状态以确定可以执行对应操作。状态之间的基本转换关系请参见“3.3.1.1 总体介绍”中的“[状态转换](#)”。
- 具备管理员权限。
- 准备好虚拟机 XML 配置文件。

#### 命令使用说明

用户可以使用 `virsh` 命令管理虚拟机生命周期，命令格式为：

```
virsh <operate> <obj> <options>
```

各参数含义如下：

- *operate*：管理虚拟机生命周期对应操作，例如创建、销毁、启动等。
- *obj*：命令操作对象，如指定需要操作的虚拟机。
- *options*：命令选项，该参数可选。

虚拟机生命周期管理各命令如表 3-7 所示。其中 *VMInstance* 为虚拟机名称、虚拟机 ID 或者虚拟机 UUID，*XMLFile* 是虚拟机 XML 配置文件，*DumpFile* 为转储文件，请根据实际情况修改。

表3-7 虚拟机生命周期管理命令

命令	含义
<b>virsh define</b> <XMLFile>	定义持久化虚拟机，定义完成后虚拟机处于关闭状态，虚拟机被看作为一个 <i>domian</i> 实例
<b>virsh create</b> <XMLFile>	创建一个临时性虚拟机，创建完成后虚拟机处于运行状态
<b>virsh start</b> <VMInstance>	启动虚拟机
<b>virsh shutdown</b> <VMInstance>	关闭虚拟机。启动虚拟机关机流程，若关机失败可使用强制关闭
<b>virsh destory</b> <VMInstance>	强制关闭虚拟机
<b>virsh reboot</b> <VMInstance>	重启虚拟机
<b>virsh save</b> <VMInstance>	将虚拟机的运行状态转储到文件中

命令	含义
<DumpFile>	
<b>virsh restore</b> <DumpFile>	从虚拟机状态转储文件恢复虚拟机
<b>virsh suspend</b> <VMInstance>	暂停虚拟机的运行，使虚拟机处于 <b>pasued</b> 状态
<b>virsh resume</b> <VMInstance>	唤醒虚拟机，将处于 <b>paused</b> 状态的虚拟机恢复到运行状态
<b>virsh undefine</b> <VMInstance>	销毁持久性虚拟机，虚拟机生命周期完结，不能继续对该虚拟机继续操作

3.3.1.3 示例

本节给出虚拟机生命周期管理相关命令的示例。

- 创建虚拟机  
虚拟机 XML 配置文件为 **openEulerVM.xml**，命令和回显如下：

```
# virsh define openEulerVM.xml
Domain openEulerVM defined from openEulerVM.xml
```
- 启动虚拟机  
启动名称为 **openEulerVM** 的虚拟机，命令和回显如下：

```
# virsh start openEulerVM
Domain openEulerVM started
```
- 重启虚拟机  
重启名称为 **openEulerVM** 的虚拟机，命令和回显如下：

```
# virsh reboot openEulerVM
Domain openEulerVM is being rebooted
```
- 关闭虚拟机  
关闭名称为 **openEulerVM** 的虚拟机，命令和回显如下：

```
# virsh shutdown openEulerVM
Domain openEulerVM is being shutdown
```
- 销毁虚拟机
  - 若虚拟机启动时未使用 **nvr**am 文件，销毁虚拟机命令如下：

```
# virsh undefine <VMInstance>
```
  - 若虚拟机启动时使用了 **nvr**am 文件，销毁该虚拟机需要指定 **nvr**am 的处理策略，命令如下：

```
# virsh undefine <VMInstance> <strategy>
```

其中<strategy>为销毁虚拟机的策略，可取值：  
--nvr**a**m: 销毁虚拟机的同时删除其对应的 **nvr**am 文件。  
--keep-nvr**a**m: 销毁虚拟机，但保留其对应的 **nvr**am 文件。  
例如，删除虚拟机 **openEulerVM** 及其 **nvr**am 文件，命令和回显如下：

```
# virsh undefine openEulerVM --nvram
Domain openEulerVM has been undefined
```

## 3.3.2 在线修改虚拟机配置

### 概述

虚拟机创建之后用户可以修改虚拟机的配置信息，称为在线修改虚拟机配置。在线修改配置以后，新的虚拟机配置文件会被持久化，并在虚拟机关闭、重新启动后生效。

修改虚拟机配置命令格式如下：

```
virsh edit <VMInstance>
```

virsh edit 命令通过编辑“domain”对应的 XML 配置文件，完成对虚拟机配置的更新。virsh edit 使用 vi 程序作为默认的编辑器，可以通过修改环境变量“EDITOR”或“VISUAL”指定编辑器类型。virsh edit 默认优先使用“VISUAL”环境变量指定的文本编辑器。

### 操作步骤

步骤 1（可选）设置 virsh edit 命令的编辑器为 vim。

```
# export VISUAL=vim
```

步骤 2 使用 virsh edit 打开虚拟机名称为 openEulerVM 对应的 XML 配置文件。

```
# virsh edit openEulerVM
```

步骤 3 修改虚拟机配置文件。

步骤 4 保存虚拟机配置文件并退出。

步骤 5 重启虚拟机使配置修改生效。

```
# virsh reboot openEulerVM
```

----结束

## 3.3.3 查询虚拟机信息

### 概述

管理员在管理虚拟机的过程中经常需要知道一些虚拟机信息，libvirt 提供了一套命令行工具用于查询虚拟机的相关信息。本章介绍相关命令的使用方法，便于管理员来获取虚拟机的各种信息。

### 前提条件

查询虚拟机信息需要：

- libvirtd 服务处于运行状态。
- 命令行操作需要拥有管理员权限。

查询主机上的虚拟机信息

- 查询主机上处于运行和暂停状态的虚拟机列表。

```
# virsh list
```

例如，下述回显说明当前主机上存在 3 台虚拟机，其中 openEulerVM01、openEulerVM02 处于运行状态，openEulerVM03 处于暂停状态。

Id	Name	State
-----		
39	openEulerVM01	running
40	openEulerVM02	running
69	openEulerVM03	paused

- 查询主机上已经定义的所有虚拟机信息列表。

```
# virsh list --all
```

例如，下述回显说明当前主机上定义了 4 台虚拟机，其中虚拟机 openEulerVM01 处于运行状态，openEulerVM02 处于暂停状态，openEulerVM03 和 openEulerVM04 处于关机状态。

Id	Name	State
-----		
39	openEulerVM01	running
69	openEulerVM02	paused
-	openEulerVM03	shut off
-	openEulerVM04	shut off

查询虚拟机基本信息

Libvirt 组件提供了一组查询虚拟机状态信息的命令，包括虚拟机运行状态、设备信息或者调度属性等，使用方法请参见表 3-8。

表3-8 查询虚拟机基本信息

查询的信息内容	命令行	说明
基本信息	<b>virsh dominfo</b> <VMInstance>	包括虚拟机 ID、UUID，虚拟机规格等信息。
当前状态	<b>virsh domstate</b> <VMInstance>	可以使用--reason 选项查询虚拟机变为当前状态的原因。
调度信息	<b>virsh schedinfo</b> <VMInstance>	包括 vCPU 份额信息。
vCPU 数目	<b>virsh vcpucount</b> <VMInstance>	查询虚拟机 vCPU 的个数。
虚拟块设备状态	<b>virsh domblkstat</b> <VMInstance>	查询块设备名称可以使用 virsh domblklist 命令。
虚拟网卡状态	<b>virsh domifstat</b> <VMInstance>	查询网卡名称可以使用 virsh domiflist 命令。

查询的信息内容	命令行	说明
I/O 线程	<b>virsh iothreadinfo</b> <i>&lt;VMInstance&gt;</i>	虚拟机 I/O 线程及其 CPU 亲和性。

## 示例

- 使用 **virsh dominfo** 查询一个定义好的虚拟机的基本信息，从查询结果可知，虚拟机 ID 为 5，UUID 为 ab472210-db8c-4018-9b3e-fc5319a769f7，内存大小为 8GiB，vCPU 数目为 4 个等。

```
# virsh dominfo openEulerVM
Id:          5
Name:        openEulerVM
UUID:        ab472210-db8c-4018-9b3e-fc5319a769f7
OS Type:     hvm
State:       running
CPU(s):      4
CPU time:    6.8s
Max memory:  8388608 KiB
Used memory: 8388608 KiB
Persistent:  no
Autostart:   disable
Managed save: no
Security model: none
Security DOI: 0
```

- 使用 **virsh domstate** 查询虚拟机的当前状态，从查询结果可知，虚拟机 *openEulerVM* 当前处于运行状态。

```
# virsh domstate openEulerVM
running
```

- 使用 **virsh schedinfo** 查询虚拟机的调度信息，从查询结果可知，虚拟机 CPU 预留份额为 1024。

```
# virsh schedinfo openEulerVM
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
global_period  : 100000
global_quota   : -1
iothread period: 100000
iothread_quota : -1
```

- 使用 **virsh vcpucount** 查询虚拟机的 vCPU 数目，从查询结构可知，虚拟机有 4 个 CPU。

```
# virsh vcpucount openEulerVM
maximum   live    4
current   live    4
```

- 使用 `virsh domblklist openEulerVM` 查询虚拟机磁盘设备信息，从查询结构可知，虚拟机有 2 个磁盘，`sda` 是 `qcow2` 格式的虚拟磁盘，`sdb` 是一个 `cdrom` 设备。

```
# virsh domblklist openEulerVM
Target    Source
-----
sda       /home/openeuler/vm/centos76_aarch64.qcow2
sdb       /home/openeuler/vm/CentOS-7-aarch64-Everything-1810.iso
```

- 使用 `virsh domiflist openEulerVM` 查询虚拟机网卡信息，从查询结果可知，虚拟机有 1 张网卡，对应的后端是 `vnet0` 在主机 `br0` 网桥上，MAC 地址为 `00:05:fe:d4:f1:cc`。

```
# virsh domiflist openEulerVM
Interface Type      Source    Model    MAC
-----
vnet0     bridge   br0       virtio   00:05:fe:d4:f1:cc
```

- 使用 `virsh iothreadinfo openEulerVM` 查询虚拟机 I/O 线程信息，从查询结果可知虚拟机有 5 个 I/O 线程，在物理 CPU7-10 上进行调度。

```
# virsh iothreadinfo openEulerVM
IOThread ID    CPU Affinity
-----
3             7-10
4             7-10
5             7-10
1             7-10
2             7-10
```

## 3.3.4 登录虚拟机

本章介绍使用 VNC 登录虚拟机的方法。

### 3.3.4.1 使用 VNC 密码登录

#### 概述

当虚拟机操作系统安装部署完成之后，用户可以通过 VNC 协议远程登录虚拟机，从而对虚拟机进行管理操作。

#### 前提条件

使用 RealVNC、TightVNC 等客户端登录虚拟机，在登录虚拟机之前需要获取如下信息：

- 虚拟机所在主机的 IP 地址。
- 确保客户端所在的环境可以访问到主机的网络。
- 虚拟机的 VNC 侦听端口，该端口一般在客户机启动时自动分配，一般为  $5900 + x$ （ $x$  为正整数，按照虚拟机启动的顺序递增，且 5900 对用户不可见）。
- 如果 VNC 设置了密码，还需要获取虚拟机的 VNC 密码。

#### 说明

为虚拟机 VNC 配置密码，需要编辑虚拟机 XML 配置文件，即为 `graphics` 元素新增一个 `passwd` 属性，属性的值为要配置的密码。例如，将虚拟机的 VNC 密码配置为 `n8VfjbFK` 的 XML 配置参考如下：



```
<graphics type='vnc' port='5900' autoport='yes' listen='0.0.0.0' keymap='en-us' passwd='n8VfjbFK'>
  <listen type='address' address='0.0.0.0' />
</graphics>
```

## 操作步骤

步骤 1 查询虚拟机使用的 VNC 端口号。例如名称为 *openEulerVM* 的虚拟机，命令如下：

```
# virsh vncdisplay openEulerVM
:3
```

### 说明

登录 VNC 需要配置防火墙规则，允许 VNC 端口的连接。参考命令如下，其中 *X* 为数值“5900 + 端口号”，例如本例中为 5903。

```
firewall-cmd --zone=public --add-port=X/tcp
```

步骤 2 打开 VncViewer 软件，输入主机 IP 和端口号。格式为“主机 IP:端口号”，例如：“10.133.205.53:3”。

步骤 3 单击“确定”输入 VNC 密码（可选），登录到虚拟机 VNC 进行操作。

----结束

## 3.3.4.2 配置 VNC TLS 登录

### 概述

VNC 服务端和客户端默认采用明文方式进行数据传输，因此通信内容可能被第三方截获。为了提升安全性，openEuler 支持 VNC 服务端配置 TLS 模式进行加密认证。TLS（Transport Layer Security）即传输层安全，可以实现 VNC 服务端和客户端之间加密通信，从而防止通信内容被第三方截获。

### 说明

- 使用 TLS 加密认证模式需要 VNC 客户端支持 TLS 模式（例如 TigerVNC），否则无法连接到 VNC 客户端。
- TLS 加密认证模式配置粒度为主机服务器级别，开启该特性后，主机上正在运行的所有虚拟机对应的 VNC 客户端都将开启 TLS 加密认证模式。

## 操作步骤

VNC 开启 TLS 加密认证模式的操作步骤如下：

步骤 1 登录 VNC 服务端所在主机登录 VNC 服务端所在主机，开启或修改服务端配置文件 */etc/libvirt/qemu.conf* 中对应的配置项。相关配置内容如下所示：

```
vnc listen = "x.x.x.x"           # "x.x.x.x"为 VNC 的侦听地址，请用户根据实际配置，VNC 服务端只允许该地址或地址段范围内的客户端连接请求
vnc tls = 1                      # 配置为 1，表示开启 VNC TLS 支持
vnc tls x509 cert dir = "/etc/pki/libvirt-vnc"      #指定证书存放的路径为
/etc/pki/libvirt-vnc
vnc_tls_x509_verify = 1          #配置为 1，表示 TLS 认证使用 x509 证书
```

步骤 2 为 VNC 创建证书和私钥文件。此处以 GNU TLS 为例进行说明。

### 说明

使用 GNU TLS，请提前安装好 gnu-utils 软件包。

1. 制作证书颁发机构 CA（Certificate Authority）的证书文件。

```
# certtool --generate-privkey > ca-key.pem
```

2. 制作自签名的 CA 证书公私钥。其中 *Your organization name* 为机构名，由用户指定。

```
# cat > ca.info<<EOF
cn = Your organization name
ca
cert_signing_key
EOF
# certtool --generate-self-signed \
    --load-privkey ca-key.pem \
    --template ca.info \
    --outfile ca-cert.pem
```

上述生成文件，ca-cert.pem 是生成的 CA 公钥，ca-key.pem 是生成的 CA 私钥，需要由 CA 妥善保管防止泄露。

3. 为 VNC 服务端颁发证书。其中 *Client Organization Name* 为实际的服务名称，例如 cleint.foo.com，需要用户根据实际指定。

```
# cat > server.info<<EOF
cn = Server Organization Name
tls_www_server
encryption_key
signing_key
EOF
# certtool --generate-privkey > server-key.pem
# certtool --generate-certificate \
    --load-ca-certificate ca-cert.pem \
    --load-ca-privkey ca-key.pem \
    --load-privkey server-key.pem \
    --template server.info \
    --outfile server-cert.pem
```

上述生成文件，server-key.pem 是 VNC 服务端的私钥，server-cert.pem 是 VNC 服务端的公钥。

4. 为 VNC 客户端颁发证书。

```
# cat > client.info<<EOF
cn = Client Organization Name
tls_www_client
encryption_key
signing_key
EOF
# certtool --generate-privkey > client-key.pem
# certtool --generate-certificate \
    --load-ca-certificate ca-cert.pem \
    --load-ca-privkey ca-key.pem \
    --load-privkey client-key.pem \
    --template client.info \
    --outfile client-cert.pem
```

上述生成文件，client-key.pem 是 VNC 客户端的私钥，client-cert.pem 是 VNC 客户端的公钥，生成的公私钥对需要拷贝到 VNC 客户端。

步骤 3 关闭需要被登录的虚拟机，重启 VNC 服务端所在主机的 libvirtd 服务。

```
# systemctl restart libvirtd
```

步骤 4 将生成的服务端证书放置到 VNC 服务端指定目录并将证书的权限改为只允许当前用户读写。

```
# sudo mkdir -m 750 /etc/pki/libvirt-vnc
# cp ca-cert.pem /etc/pki/libvirt-vnc/ca-cert.pem
# cp server-cert.pem /etc/pki/libvirt-vnc/server-cert.pem
# cp server-key.pem /etc/pki/libvirt-vnc/server-key.pem
# chmod 0600 /etc/pki/libvirt-vnc/*
```

步骤 5 将生成的客户端证书 ca-cert.pem，client-cert.pem 和 client-key.pem 拷贝到 VNC 客户端。配置 VNC 客户端的 TLS 证书后即可使用 VNC TLS 登录。

#### 说明

- VNC 客户端证书的配置请参见各客户端对应的使用说明，由用户自行配置。
- 登录虚拟机的方式请参见“使用 VNC 密码登录”。

----结束

## 3.4 热迁移虚拟机

### 3.4.1 总体介绍

#### 概述

当虚拟机在物理机上运行时，物理机可能存在资源分配不均，造成负载过重或过轻的情况。另外，物理机存在硬件更换、软件升级、组网调整、故障处理等操作，如何在不中断业务的情况下完成这些操作十分重要。虚拟机热迁移技术可以在业务连续前提下，完成负载均衡或上述操作，提升用户体验和工作效率。虚拟机热迁移通常是将整个虚拟机的运行状态完整保存下来，同时可以快速恢复到原有的甚至不同的硬件平台上。虚拟机恢复后，仍然可以平滑运行，用户感知不到任何差异。根据虚拟机数据存储在当前主机还是远端存储设备（共享存储）的不同，openEuler 支持共享存储热迁移和非共享存储热迁移两种方式。

### 3.4.2 应用场景

共享存储和非共享存储热迁移的共同应用场景有：

- 当物理机故障或者负载过重时，可以将运行的虚拟机迁移到另一台物理机上，以避免业务中断，保证业务的正常运行。
- 当多数的物理机负载过轻时，可以将虚拟机迁移整合，以减少物理机数量，提高资源的利用率。
- 当物理服务器硬件设备成为瓶颈，比如 CPU、内存、硬盘等，需要更换性能更好的硬件，或者需要增加设备，但是又不能关闭虚拟机或者停止业务。

- 服务器软件升级，比如虚拟化平台升级，就可以把虚拟机从旧版本虚拟化平台热迁移到新版本虚拟化平台。

对于非共享存储热迁移，还可以应用在如下场景：

- 当物理机故障存储空间不足，需要将运行的虚拟机迁移到另一台物理机上，可以避免业务中断，保证业务的正常运行。
- 当物理机存储设备老化，性能不能支撑当前业务数据处理，成为系统性能的瓶颈，需要更换性能更强的存储，但是又不能关闭虚拟机或者停止虚拟机，这需要将运行的虚拟机迁移到一个具有更好性能的物理机上。

### 3.4.3 注意事项和约束限制

- 热迁移过程中，需要保证网络状态良好。如果发生网络中断，热迁移会暂停，直到网络恢复后才会继续，当发生超时，热迁移会失败。
- 迁移过程中，不允许对虚拟机进行生命周期和管理虚拟机硬件设备等操作。
- 虚拟机正在迁移的过程中，应尽可能保证源端、目的端服务器不被意外下电或重启，否则会导致热迁移失败，甚至可能导致虚拟机被下电。
- 虚拟机正在迁移的过程中，不允许对虚拟机做关机、重启或恢复操作，否则可能会导致热迁移失败，当执行 ACPI 方式重启时，再执行热迁移会导致虚拟机关闭。
- 只支持同构热迁移，即源端和目的端 CPU 型号需要相同。
- 跨业务网段虚拟机迁移可以成功，但是到目的端后会出现网络异常，为了防止该情况发生，需要用户保证迁移业务网段一致。
- 如果源端虚拟机 vCPU 数大于目的端的物理机 CPU 核数，则迁移后将会影响到虚拟机的性能，应保证目的端物理机 CPU 核数大于等于源端虚拟机 vCPU 数。

非共享存储热迁移过程中的额外注意事项：

- 不支持迁移源端和目的端为同一个磁盘镜像文件的迁移，用户需要对该类迁移进行特殊处理，提防覆盖写坏数据而导致镜像损坏。
- 不支持对共享磁盘的迁移，用户需要对该类迁移进行防呆处理。
- 迁移的目的端镜像只支持文件，不支持裸设备，用户需要对目的端是裸设备的迁移进行防呆处理。
- 目的端需要创建与源端大小、数量相同的磁盘镜像，否则迁移失败。
- 混合迁移场景，需要传入迁移的磁盘，不能包括共享和只读的磁盘。

### 3.4.4 热迁移操作

#### 前提条件

- 进行热迁移之前要确保源端和目的端主机之间的网络是互通的，并且源端和目的获得资源权限是对等的，即两端同时能够访问到相同的存储资源和网络资源。
- 在执行虚拟机热迁移前应当对虚拟机进行健康检查，并确保目的端主机有足够的 CPU、内存和存储资源。

## 设置热迁移参数（可选）

在执行热迁移之前，可以通过使用 `virsh migrate-setmaxdowntime` 命令来指定虚拟机热迁移过程中能够容忍的最大停机时间，这是一个可选的配置项。

例如，指定名为 *openEulerVM* 的虚拟机最大停机时间为 500ms：

```
# virsh migrate-setmaxdowntime openEulerVM 500
```

同时，可以通过调用 `virsh migrate-setspeed` 来限制虚拟机热迁移占用的带宽大小，防止当前虚拟机热迁移的时候占用带宽过大，对主机上的其他虚拟机或者业务造成影响，这个选择同样也是热迁移的一个可选项。

例如，指定名为 *openEulerVM* 的虚拟机热迁带宽为 500Mbps：

```
# virsh migrate-setspeed openEulerVM --bandwidth 500
```

用户可以使用 `migrate-getspeed` 来查询虚拟机热迁移过程中的最大带宽。

```
# virsh migrate-getspeed openEulerVM
500
```

## 热迁移操作（共享存储场景）

**步骤 1** 确认是否为共享存储。

```
# virsh domblklist <VMInstance>
Target    Source
-----
sda       /dev/mapper/open_euleros_disk
sdb       /mnt/nfs/images/openeuler-test.qcow2
```

首先，使用 `virsh domblklist` 命令查询虚拟机的存储设备信息，例如上面的查询结果显示虚拟机配置有 2 个存储设备：*sda* 盘和 *sdb* 盘，然后再分别查询一下这两个设备对应后端存储是本地存储还是远端存储，如果虚拟机的所有存储设备都在远端共享存储之上，则说明该虚拟机为共享存储虚拟机，否则为非共享存储虚拟机。

**步骤 2** 执行如下命令，进行虚拟机热迁移。

例如，将虚拟机 *openEulerVM* 迁移到目的主机上使用 `virsh migrate` 命令。

```
# virsh migrate --live --unsafe openEulerVM qemu+ssh://<destination-host-ip>/system
```

其中，*<destination-host-ip>* 为目的主机 IP 地址，热迁移之前需要进行 `ssh` 认证以获取目的端主机管理源权限。

另外，`virsh migrate` 命令还有 `--auto-converge` 和 `--timeout` 子选项来保证迁移的顺利完成。

其中，相关子选项：

`--unsafe` 命令会强制进行热迁移，忽略安全检查步骤。

`--auto-converge` 命令会对 CPU 进行降频限速，确保热迁移流程能够收敛。

`--timeout` 选项会指定一个热迁移超时时间，热迁移超过指定时间后会强制挂起虚拟机让热迁移得以收敛。

步骤 3 热迁移完成后命令返回，虚拟机在目的端主机正常运行。

----结束

## 热迁移操作（非共享存储场景）

步骤 1 首先，先查询虚拟机存储设备列表，确保虚拟机使用的是非共享存储。

例如，通过 `virsh domblklist` 查询到准备迁移的虚拟机有一个 `qcow2` 格式的磁盘 `sda`，`sda` 的 `xml` 配置为：

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' cache='none' io='native' />
  <source file='/mnt/sdb/openeuler/openEulerVM.qcow2' />
  <target dev='sda' bus='scsi' />
  <address type='drive' controller='0' bus='0' target='0' unit='0' />
</disk>
```

执行热迁移之前需要在目的端主机相同磁盘目录下创建一个虚拟磁盘文件，注意磁盘的格式和大小必须保持一致。

```
# qemu-img create -f qcow2 /mnt/sdb/openeuler/openEulerVM.qcow2 20G
```

步骤 2 在源端使用 `virsh migrate` 命令来执行热迁移，迁移的时候会将存储也一并迁移到目的端。

```
# virsh migrate --live --unsafe --copy-storage-all --migrate-disks-all --migrate-
disks sda \
openEulerVM qemu+ssh://<dest-host-ip>/system
```

步骤 3 热迁移完成后命令返回，虚拟机在目的端主机运行正常，存储设备也被迁移到目的主机上。

----结束

## 3.5 管理系统资源

使用 `libvirt` 命令来管理虚拟机的系统资源，如 `vCPU`、虚拟内存资源等。

在开始前：

- 确保主机上运行了 `libvirtd` 守护进程。
- 用 `virsh list --all` 命令确认虚拟机已经被定义。

## 3.5.1 管理虚拟 CPU

### 3.5.1.1 CPU 份额

#### 概述

虚拟化环境下，同一主机上的多个虚拟机竞争使用物理 CPU。为了防止某些虚拟机占用过多的物理 CPU 资源，影响相同主机上其他虚拟机的性能，需要平衡虚拟机 vCPU 的调度，避免物理 CPU 的过度竞争。

CPU 份额表示一个虚拟机竞争物理 CPU 计算资源的能力大小总和。用户通过调整 `cpu_shares` 值能够设置虚拟机抢占物理 CPU 资源的能力。`cpu_shares` 值无单位，是一个相对值。虚拟机获得的 CPU 计算资源，是与其他虚拟机的 CPU 份额，按相对比例，瓜分物理 CPU 除预留外可用计算资源。通过调整 CPU 份额来保证虚拟机 CPU 计算资源服务质量。

#### 操作步骤

通过修改分配给虚拟机的运行时间的 `cpu_shares` 值，来平衡 vCPU 之间的调度。

- 查看虚拟机的当前 CPU 份额：

```
# virsh schedinfo <VMInstance>
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
global period  : 100000
global_quota   : -1
iothread period: 100000
iothread_quota : -1
```

- 在线修改：修改处于 `running` 状态的虚拟机的当前 CPU 份额，使用带 `--live` 参数的 `virsh schedinfo` 命令：

```
# virsh schedinfo <VMInstance> --live cpu_shares=<number>
```

比如将正在运行的虚拟机 `openEulerVM` 的 CPU 份额从 1024 改为 2048：

```
# virsh schedinfo openEulerVM --live cpu_shares=2048
Scheduler      : posix
cpu_shares     : 2048
vcpu_period    : 100000
vcpu_quota     : -1
emulator_period: 100000
emulator_quota : -1
global period  : 100000
global_quota   : -1
iothread period: 100000
iothread_quota : -1
```

对 `cpu_shares` 值的修改立即生效，虚拟机 `openEulerVM` 能得到的运行时间将是原来的 2 倍。但是这一修改将在虚拟机关机并重新启动后失效。

- 持久化修改：在 libvirt 内部配置中修改虚拟机的 CPU 份额，使用带 **--config** 参数的 **virsh schedinfo** 命令：

```
# virsh schedinfo <VMInstance> --config cpu_shares=<number>
```

比如将虚拟机 openEulerVM 的 CPU 份额从 1024 改为 2048：

```
# virsh schedinfo openEulerVM --config cpu_shares=2048
Scheduler      : posix
cpu shares     : 2048
vcpu period    : 0
vcpu quota     : 0
emulator period: 0
emulator quota : 0
global period  : 0
global quota   : 0
iothread period: 0
iothread_quota : 0
```

对 **cpu\_shares** 值的修改不会立即生效，在虚拟机 openEulerVM 下一次启动后才生效，并持久生效。虚拟机 **euler** 能得到的运行时间将是原来的 2 倍。

### 3.5.1.2 绑定 qemu 进程至物理 CPU

#### 概述

qemu 主进程绑定特性是将 qemu 主进程绑定到特定的物理 CPU 范围内，从而保证了运行不同业务的虚拟机不会干扰到邻位虚拟机。例如在一个典型的云计算场景中，一台物理机上会运行多台虚拟机，而每台虚拟机的业务不同，造成了不同程度的资源占用，为了避免存储 IO 密集的虚拟机对邻位虚拟机的干扰，需要将不同虚拟机处理 IO 的存储进程完全隔离，由于 qemu 主进程是处理前后端的主要服务进程，故需要实现隔离。

#### 操作步骤

通过 **virsh emulatorpin** 命令可以绑定 qemu 主进程到物理 CPU。

- 查看 qemu 进程当前绑定的物理 CPU 范围：

```
# virsh emulatorpin openEulerVM
emulator: CPU Affinity
-----
*: 0-63
```

这说明虚拟机 openEulerVM 对应的 qemu 主进程可以在主机的所有物理 CPU 上调度。

- 在线绑定：修改处于 running 状态的虚拟机对应的 qemu 进程的绑定关系，使用带 **-live** 参数的 **vcpu emulatorpin** 命令：

```
# virsh emulatorpin openEulerVM --live 2-3

# virsh emulatorpin openEulerVM
emulator: CPU Affinity
-----
*: 2-3
```



以上命令把虚拟机 *openEulerVM* 对应的 *qemu* 进程绑定到物理 CPU2、3 上，即限制了 *qemu* 进程只在这两个物理 CPU 上调度。这一绑定关系的调整立即生效，但在虚拟机关机并重新启动后失效。

- 持久化绑定：在 *libvirt* 内部配置中修改虚拟机对应的 *qemu* 进程的绑定关系，使用带 **--config** 参数的 *virsh emulatorpin* 命令：

```
# virsh emulatorpin openEulerVM --config 0-3,^1

# virsh emulatorpin euler
emulator: CPU Affinity
-----
*: 0,2-3
```

以上命令把虚拟机 *openEulerVM* 对应的 *qemu* 进程绑定到物理 CPU0、2、3 上，即限制了 *qemu* 进程只在这三个物理 CPU 上调度。这一绑定关系的调整不会立即生效，在虚拟机下一次启动后才生效，并持久生效。

### 3.5.1.3 调整虚拟 CPU 绑定关系

#### 概述

把虚拟机的 vCPU 绑定在物理 CPU 上，即 vCPU 只在绑定的物理 CPU 上调度，在特定场景下达到提升虚拟机性能的目的。比如在 NUMA 系统中，把 vCPU 绑定在同一个 NUMA 节点上，可以避免 vCPU 跨节点访问内存，避免影响虚拟机运行性能。如果未绑定，默认 vCPU 可在任何物理 CPU 上调度。具体的绑定策略由用户来决定。

#### 操作步骤

通过 *virsh vcpupin* 命令可以调整 vCPU 和物理 CPU 的绑定关系。

- 查看虚拟机的当前 vCPU 绑定信息：

```
# virsh vcpupin openEulerVM
VCPU  CPU Affinity
-----
0      0-63
1      0-63
2      0-63
3      0-63
```

这说明虚拟机 *openEulerVM* 的所有 vCPU 可以在主机的所有物理 CPU 上调度。

- 在线调整：修改处于 *running* 状态的虚拟机的当前 vCPU 绑定关系，使用带 **--live** 参数的 *virsh vcpupin* 命令：

```
# virsh vcpupin openEulerVM --live 0 2-3

# virsh vcpupin euler
VCPU  CPU Affinity
-----
0      2-3
1      0-63
2      0-63
3      0-63
```

以上命令把虚拟机 *openEulerVM* 的 vCPU0 绑定到 PCPU2、3 上，即限制了 vCPU0 只在这两个物理 CPU 上调度。这一绑定关系的调整立即生效，但在虚拟机关机并重新启动后失效。

- 持久化调整：在 libvirt 内部配置中修改虚拟机的 vCPU 绑定关系，使用带 **--config** 参数的 **virsh vcpupin** 命令：

```
# virsh vcpupin openEulerVM --config 0 0-3,^1

# virsh vcpupin openEulerVM
VCPU   CPU Affinity
-----
0      0,2-3
1      0-63
2      0-63
3      0-63
```

以上命令把虚拟机 *openEulerVM* 的 vCPU0 绑定到物理 CPU0、2、3 上，即限制了 vCPU0 只在这三个物理 CPU 上调度。这一绑定关系的调整不会立即生效，在虚拟机下一次启动后才生效，并持久生效。

## 3.5.2 管理虚拟内存

### 3.5.2.1 NUMA 简介

传统的多核运算使用 **SMP**（Symmetric Multi-Processor）模式：将多个处理器与一个集中的存储器和 I/O 总线相连。所有处理器只能访问同一个物理存储器，因此 **SMP** 系统也被称为一致存储器访问（UMA）系统。一致性指无论在什么时候，处理器只能为内存的每个数据保持或共享唯一一个数值。很显然，**SMP** 的缺点是伸缩性有限，因为在存储器和 I/O 接口达到饱和的时候，增加处理器并不能获得更高的性能。

**NUMA**（Non Uniform Memory Access Architecture）模式是一种分布式存储器访问方式，处理器可以同时访问不同的存储器地址，大幅度提高并行性。**NUMA** 模式下，处理器被划分成多个“节点”（NODE），每个节点分配一块本地存储器空间。所有节点中的处理器都可以访问全部的物理存储器，但是访问本节点内的存储器所需要的时间，比访问某些远程节点内的存储器所花的时间要少得多。

### 3.5.2.2 配置 Host NUMA

为提升虚拟机性能，在虚拟机启动前，用户可以通过虚拟机 XML 配置文件为虚拟机指定主机的 NUMA 节点，使虚拟机内存分配在指定的 NUMA 节点上。本特性一般与 vCPU 绑定一起使用，从而避免 vCPU 远端访问内存。

#### 须知

如果在指定的 NODE 上内存不够时，启动虚拟机会失败，并且存在系统 OOM（Out Of Memory）强制关闭进程的可能性。

## 操作步骤

- 查看 Host 的 NUMA 拓扑结构：

```
# numactl -H
available: 4 nodes (0-3)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
node 0 size: 31571 MB
node 0 free: 17095 MB
node 1 cpus: 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
node 1 size: 32190 MB
node 1 free: 28057 MB
node 2 cpus: 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
node 2 size: 32190 MB
node 2 free: 10562 MB
node 3 cpus: 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63
node 3 size: 32188 MB
node 3 free: 272 MB
node distances:
node  0  1  2  3
 0: 10 15 20 20
 1: 15 10 20 20
 2: 20 20 10 15
 3: 20 20 15 10
```

- 在虚拟机 XML 配置文件中添加 **numatune** 字段，创建并启动虚拟机。配置参数如下：

```
<numatune>
  <memory mode="strict" nodeset="0"/>
</numatune>
```

以上配置表示虚拟机的内存将在 Host 的 NUMA NODE0 上分配。假设虚拟机的 vCPU 也绑定在 NODE0 的 PCPU 上，就可以避免由于 vCPU 访问远端内存带来的性能下降。

### 3.5.2.3 配置 Guest NUMA

虚拟机中运行的很多业务软件都针对 NUMA 架构进行了性能优化，尤其是对于大规格虚拟机，这种优化的作用更明显。openEuler 提供了 Guest NUMA 特性，在虚拟机内部呈现出 NUMA 拓扑结构。用户可以通过识别这个结构，对业务软件的性能进行优化，从而保证业务更好的运行。

配置 Guest NUMA 时可以指定 vNode 的内存在 HOST 上的分配位置，实现内存的分块绑定，同时配合 vCPU 绑定，使 vNode 上的 vCPU 和内存在同一个物理 NUMA node 上。

## 操作步骤

在虚拟机的 XML 配置文件中，配置了 Guest NUMA 后，就可以在虚拟机内部查看 NUMA 拓扑结构。<numa>项是 Guest NUMA 的必配项。

```
<cputune>
  <vcpupin vcpu='0' cpuset='0-3'/>
  <vcpupin vcpu='1' cpuset='0-3'/>
  <vcpupin vcpu='2' cpuset='16-19'/>
  <vcpupin vcpu='3' cpuset='16-19'/>
</cputune>
<numatune>
  <memnode cellid="0" mode="strict" nodeset="0"/>
</numatune>
```

```
<memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
[...]
<cpu>
  <numa>
    <cell id='0' cpus='0-1' memory='2097152'/>
    <cell id='1' cpus='2-3' memory='2097152'/>
  </numa>
</cpu>
```

#### 说明

- <numa>项提供虚拟机内部呈现 NUMA 拓扑功能，“cell id”表示 vNode 编号，“cpus”表示 vCPU 编号，“memory”表示对应 vNode 上的内存大小。
- 如果希望通过 Guest NUMA 提供更好的性能，则需要配置<numatune>和<cputune>，使 vCPU 和对应的内存分布在同一个物理 NUMA NODE 上：
- <numatune>中的“cellid”和<numa>中的“cell id”是对应的；“mode”可以配置为“strict”（严格从指定 node 上申请内存，内存不够则失败）、“preferred”（优先从某一 node 上申请内存，如果不够则从其他 node 上申请）、“interleave”（从指定的 node 上交叉申请内存）；“nodeset”表示指定物理 NUMA NODE。
- <cputune>中需要将同一 cell id 中的 vCPU 绑定到与 memnode 相同的物理 NUMA NODE 上。

## 3.6 管理设备

### 3.6.1 配置虚拟机 PCIe 控制器

#### 概述

虚拟机内部的网卡、磁盘控制器、PCIe 直通设备都需要挂接到 PCIe Root Port 下面，每个 Root Port 对应一个 PCIe 插槽。Root Port 的下挂设备支持热插拔，但是 Root Port 本身不支持热插拔，因此需要用户考虑设备热插的需求，规划虚拟机需要预留的最大 PCIe Root Port 数量，在虚拟机启动之前完成 Root Port 的静态配置。

#### 配置 PCIe Root、PCIe Root Port 和 PCIe-PCI-Bridge

虚拟机 PCIe 控制器通过 XML 文件进行配置，PCIe Root、PCIe Root Port 和 PCIe-PCI-Bridge 对应 XML 中的 model 分别为 pcie-root、pcie-root-port、pcie-to-pci-bridge。

- 简化配置方法

在虚拟机的 XML 文件中写入以下内容，controller 的其他属性由 Libvirt 自动填充：

```
<controller type='pci' index='0' model='pcie-root'/>
<controller type='pci' index='1' model='pcie-root-port'/>
<controller type='pci' index='2' model='pcie-to-pci-bridge'/>
<controller type='pci' index='3' model='pcie-root-port'/>
<controller type='pci' index='4' model='pcie-root-port'/>
<controller type='pci' index='5' model='pcie-root-port'/>
```

其中：由于 pcie-root 和 pcie-to-pci-bridge 分别占用 1 个 index，因此最终的 index 等于需要的 Root Port 数量+1。

- 完整配制方法

在虚拟机的 XML 文件中写入以下内容：

```
<controller type='pci' index='0' model='pcie-root'/>
<controller type='pci' index='1' model='pcie-root-port'>
  <model name='pcie-root-port'/>
  <target chassis='1' port='0x8'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x0'
multifunction='on'/>
</controller>
<controller type='pci' index='2' model='pcie-to-pci-bridge'>
  <model name='pcie-pci-bridge'/>
  <address type='pci' domain='0x0000' bus='0x01' slot='0x00' function='0x0'/>
</controller>
<controller type='pci' index='3' model='pcie-root-port'>
  <model name='pcie-root-port'/>
  <target chassis='3' port='0x9'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/>
</controller>
<controller type='pci' index='3' model='pcie-root-port'>
```

其中：

- Root Port 的 chassis 和 port 属性必须依次递增，由于中间插入一个 PCIe-PCI-Bridge，chassis 编号跳过 2，但是 port 编号仍然连续。
- Root Port 的 address function 的取值范围为 0x0~0x7。
- 每个 slot 最多下挂 8 个 function，挂满之后需要递增 slot 编号。

由于完整配置方法相对复杂，建议采用简化配置方法。

## 3.6.2 管理虚拟磁盘

### 概述

虚拟磁盘类型主要包含 virtio-blk、virtio-scsi、vhost-scsi 等。virtio-blk 模拟的是一种 block 设备，virtio-scsi 和 vhost-scsi 模拟的是一种 scsi 设备。

- virtio-blk：普通系统盘和数据盘可用，该种配置下虚拟磁盘在虚拟机内部呈现为 vd[a-z]或 vd[a-z][a-z]。
- virtio-scsi：普通系统盘和数据盘建议选用，该种配置下虚拟磁盘在虚拟机内部呈现为 sd[a-z]或 sd[a-z][a-z]。
- vhost-scsi：对性能要求高的虚拟磁盘建议选用，该种配置下虚拟磁盘在虚拟机内部呈现为 sd[a-z]或 sd[a-z][a-z]。

### 操作步骤

虚拟磁盘的配置步骤，请参见 3.2.4.1 存储设备。本节以 virtio-scsi 磁盘为例，介绍挂载和卸载虚拟磁盘的简单方法。

- 挂载 virtio-scsi 磁盘：

使用 virsh attach-device 命令挂载 virtio-scsi 虚拟磁盘：

```
# virsh attach-device <VMInstance> <attach-device.xml>
```

上述命令可以为虚拟机在线挂载磁盘，其中磁盘信息由 `attach-device.xml` 文件指定。下面是一个 `attach-device.xml` 文件的例子：

```
### attach-device.xml ###
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2' cache='none' io='native' />
  <source file='/path/to/another/qcow2-file' />
  <backingStore />
  <target dev='sdb' bus='scsi' />
  <address type='drive' controller='0' bus='0' target='1' unit='0' />
</disk>
```

通过上述命令挂载的磁盘，在虚拟机关机重启后失效。如果需要为虚拟机持久化挂载虚拟磁盘，需要使用带 `--config` 参数的 `virsh attach-device` 命令。

- 卸载 `virtio-scsi` 磁盘：

通过在线挂载的磁盘，如果不需要再使用，可以通过 `virsh detach` 命令动态卸载：

```
# virsh detach-device <VMInstance> <detach-device.xml>
```

其中，`detach-device.xml` 指定了需要卸载的磁盘的 XML 信息，与动态挂载时的 XML 信息保持一致。

## 3.6.3 管理虚拟网卡

### 概述

虚拟网卡类型主要包含 `virtio-net`、`vhost-net`、`vhost-user` 等。用户在创建虚拟机后，可能会有挂载或者卸载虚拟网卡的需求。`openEuler` 提供了网卡热插拔的功能，通过网卡热插拔，能够改变网络的吞吐量，提高系统的灵活性和扩展性。

### 操作步骤

虚拟网卡的配置步骤，请参见 3.2.4.2 网络设备。本节以 `vhost-net` 网卡为例，介绍挂载和卸载虚拟网卡的简单方法。

- 挂载 `vhost-net` 网卡：

使用 `virsh attach-device` 命令挂载 `vhost-net` 虚拟网卡：

```
# virsh attach-device <VMInstance> <attach-device.xml>
```

上述命令可以为虚拟机在线挂载 `vhost-net` 网卡，其中网卡信息由 `attach-device.xml` 文件指定。下面是一个 `attach-device.xml` 文件的例子：

```
### attach-device.xml ###
<interface type='bridge'>
  <mac address='52:54:00:76:f2:bb' />
  <source bridge='br0' />
  <virtualport type='openvswitch' />
  <model type='virtio' />
  <driver name='vhost' queues='2' />
</interface>
```

通过上述命令挂载的 `vhost-net` 网卡，在虚拟机关机重启后失效。如果需要为虚拟机持久化挂载虚拟网卡，需要使用带 `--config` 参数的 `virsh attach-device` 命令。

- 卸载 `vhost-net` 网卡：

通过在线挂载的网卡，如果不需要再使用，可以通过 `virsh detach` 命令动态卸载：

```
# virsh detach-device <VMInstance> <detach-device.xml>
```

其中，`detach-device.xml` 指定了需要卸载的 XML 信息，与动态挂载时的 XML 信息保持一致。

## 3.6.4 配置虚拟串口

### 概述

在虚拟化环境下，由于管理和业务的需求，虚拟机与宿主机需要互相通信。但在云管理系统复杂的网络架构下，运行在管理平面的服务与运行在业务平面的虚拟机之间，不能简单的进行三层网络互相通信，导致服务部署和信息收集不够快速。因此需要提供虚拟串口，来达到虚拟机与宿主机之间互相通信的目的。通过在虚拟机的 XML 配置文件中增加相应串口的配置项，可以实现虚拟机与宿主机之间的互相通信。

### 操作步骤

Linux 虚拟机串口控制台，即虚拟机串口连接到宿主机的一个伪终端设备，通过宿主机的设备间接实现对虚拟机的交互式操作。在该场景下串口需配置为 `pty` 类型，本节介绍 `pty` 型串口的配置方法。

- 在虚拟机的 XML 配置文件中"devices"节点下添加如下所示的虚拟串口配置项：

```
<serial type='pty'>
</serial>
<console type='pty'>
  <target type='serial' />
</console>
```

- 使用 `virsh console` 命令连接到正在运行的虚拟机的 `pty` 串口。

```
# virsh console <VMInstance>
```

- 如果要确保没有遗漏任何串口消息，请在启动虚拟机时使用 `--console` 选项连接到串口。

```
# virsh start --console <VMInstance>
```

## 3.7 最佳实践

### 3.7.1 halt-polling

#### 概述

在计算资源充足的情况下，为使虚拟机获得接近物理机的性能，可以使用 `halt-polling` 特性。此时 `vCPU` 因 `halt` 触发 `vm-exit` 后，并不会立即调用 `scheduler`，把计算资源让给其他主机侧进程；而是会根据配置先 `polling` 一段时间，若该 `vCPU` 在 `polling` 期间被唤醒，那么 `vCPU` 可以直接 `vm-entry` 执行 `Guest` 任务，而不用从主机侧的 `idle` 进程唤醒，减少了调度流程的开销，一定程度上提高了虚拟机系统的性能。

## 说明

halt-polling 的机制保证虚拟机的 vCPU 线程的及时响应，但在虚拟机空载的时候，主机侧也会 polling，导致主机看到 vCPU 所在 CPU 占用率比较高，而实际虚拟机内部 CPU 占用率并不高。

## 操作步骤

通过 sysfs 可以动态修改 vCPU 用于 halt-polling 的时间，默认配置为 500000，单位为 ns。

```
# echo 400000 > /sys/module/kvm/parameters/halt_poll_ns
```

## 3.7.2 内存大页

### 概述

相比传统的 4K 内存分页，openEuler 也支持 2MB/1GB 的大内存分页。内存大页可以有效减少 TLB miss，显著提升内存访问密集型业务的性能。openEuler 使用两种技术来实现内存大页。

- 静态大页

静态大页要求宿主机操作系统在加载前提前预留一个静态大页池，虚拟机创建时通过修改 xml 配置文件的方式，指定虚拟机的内存从静态大页池中分配。静态大页能保证虚拟机的所有内存存在 host 上始终以大页形式存在，保证物理连续，但增加了部署的困难，修改静态大页池的页面大小后需要重启 host 才能生效。静态大页的页面大小支持 2M 或 1G。

- 透明大页

openEuler 默认开启透明大页模式（THP），虚拟机分配内存时自动选择可用的 2M 连续页，同时自动完成大页的拆分合并，当没有可用的 2M 连续页时，它会选择可用的 4K 页面进行分配。透明大页的好处是不需要用户感知，同时能尽量使用 2M 大页以提升内存访问性能。

在虚拟机完全使用静态大页的场景下，可以通过关闭透明大页的方法，减少宿主机操作系统的开销，以便虚拟机获得更稳定的性能。

## 操作步骤

- 使用静态大页

在创建虚拟机之前通过修改 XML 的方式，为虚拟机配置使用静态大页。

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB' />
  </hugepages>
</memoryBacking>
```

以上 XML 片段表示为虚拟机配置 1G 静态大页。

```
<memoryBacking>
  <hugepages>
    <page size='2' unit='MiB' />
  </hugepages>
</memoryBacking>
```



以上 XML 片段表示为虚拟机配置 2M 静态大页。

- 使用透明大页

通过 sysfs 可以动态开启使用透明大页：

```
# echo always > /sys/kernel/mm/transparent_hugepage/enabled
```

动态关闭使用透明大页：

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

# A 附录

## A.1 术语和缩略语

文档中使用的术语和缩略语请分别参见表 A-1 和表 A-2。

表A-1 术语表

术语	含义
AArch64	AArch64 是 ARMv8 架构的一种执行状态。AArch64 不仅仅是 32 位 ARM 构架的扩展，而是 ARMv8 内全新的构架，完全使用全新的 A64 指令集。
Domain	资源的一个可配置集合，包括内存、虚拟 CPU，网络设备和磁盘设备。在 Domain 中运行虚拟机。一个 Domain 被分配虚拟资源，可以独立地被启动、停止和重启。
Libvirt	一套用于管理虚拟化平台的工具集，可用于管理 KVM、QEMU、Xen 及其他虚拟化。
Guest OS	即客户机操作系统，指运行在虚拟机上的操作系统。
Host OS	即宿主机操作系统，指被虚拟的物理机的操作系统。
Hypervisor	即虚拟机监视器 VMM，是一种运行在基础物理服务器和操作系统之间的中间软件层，可允许多个操作系统和应用共享硬件。
客户机操作系统	即 Guest OS，指运行在虚拟机上的操作系统。
宿主机操作系统	即 Host OS，指被虚拟的物理机的操作系统。
虚拟机	使用虚拟化技术，通过软件模拟完整的计算机硬件系统功能，构造出的完整虚拟计算机系统。

表A-2 缩略语表

缩略语	英文全称	中文全称	含义
NUMA	Non Uniform Memory Access Architecture	非统一内存访问架构	NUMA 是一种为多处理器计算机设计的内存架构。在 NUMA 下，处理器访问它自己的本地内存的速度比非本地内存（内存位于另一个处理器，或者是处理器之间共享的内存）快一些。
KVM	Kernel-based Virtual Machine	基于内核的虚拟机	KVM 是基于内核的虚拟机，是 Linux 的一个内核模块，该模块使得 Linux 成为一个 Hypervisor。
OVS	Open vSwitch	开放虚拟交换标准	OVS 是一个高质量的多层虚拟交换机，使用开源 Apache2.0 许可协议。
QEMU	Quick Emulator	快速模拟器	QEMU 是一个通用的可执行硬件虚拟化的开源模拟器。
SMP	Symmetric Multi-Processor	对称多处理	SMP 是一种多处理器的计算机硬件架构。现在多数的处理器系统都采用对称多处理器架构。该架构系统拥有多个处理器，各处理器共享内存子系统和总线结构。
UEFI	Unified Extensible Firmware Interface	统一的可扩展固件接口	一种详细描述全新类型接口的标准。该接口用于操作系统自动从预启动的操作环境，加载到一种操作系统上。
VM	Virtual Machine	虚拟机	使用虚拟化技术，通过软件模拟完整的计算机硬件系统功能，构造出的完整虚拟计算机系统。
VMM	Virtual Machine Monitor	虚拟机监视器	是一种运行在基础物理服务器和操作系统之间的中间软件层，可允许多个操作系统和应用共享硬件。