华为云欧拉操作系统（HCE）

用户指南

文档版本 01

发布日期 2025-06-06

华为云计算技术有限公司

商标声明


和其他华为商标均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。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 HCE 使用方法

您可通过下列方法使用Huawei Cloud EulerOS。

- 首次创建弹性云服务器和裸金属服务器实例时，推荐使用HCEOS公共镜像。

  如果没有HCE公共镜像请联系管理员创建HCE公共镜像。详细步骤请参见**注册HCE公共镜像**。
- 将操作系统切换为HCEOS。

  如果现有的弹性云服务器配置（网卡、磁盘、VPN等配置的类型和数量）都不需要改变，仅需要修改弹性云服务器的操作系统镜像，并且您的软件和原操作系统耦合度较低，适配到HCEOS改动较小，建议使用系统切换，可快速切换到HCEOS。
- 将操作系统迁移为HCEOS。

  如果现有的弹性云服务器或裸金属服务器配置（网卡、磁盘、VPN等配置的类型和数量）都不需要改变，操作系统软件的配置参数希望保留，可以通过操作系统迁移的方式迁移到HCEOS。

表 1-1 系统切换和迁移的区别

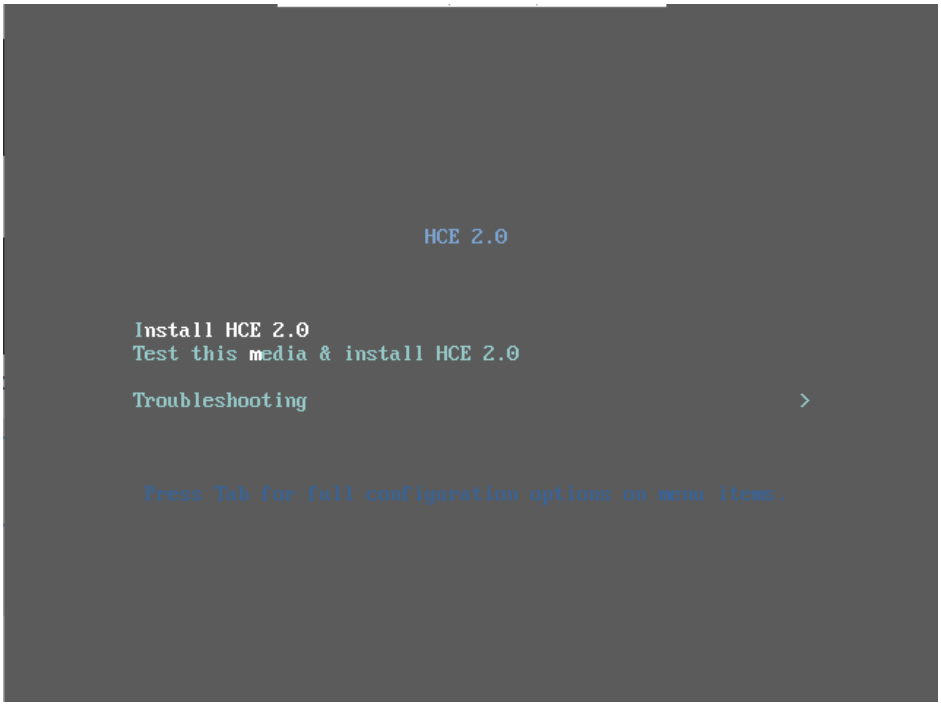
区别	系统切换	系统迁移
数据备份	• 切换操作系统会清除系统盘数据，包括系统盘上的系统分区和所有其它分区。 • 切换操作系统不影响数据盘数据。	• 迁移操作系统不会清除系统盘数据，为避免系统软件的数据丢失，建议将其备份。 • 迁移操作系统不影响数据盘数据。
个性化设置	切换操作系统后，当前操作系统内的个性化设置（如DNS、主机名等）将被重置，需重新配置。	迁移操作系统后，当前操作系统内的个性化设置（如DNS、主机名等）不需重新配置。

注册 HCE 公共镜像

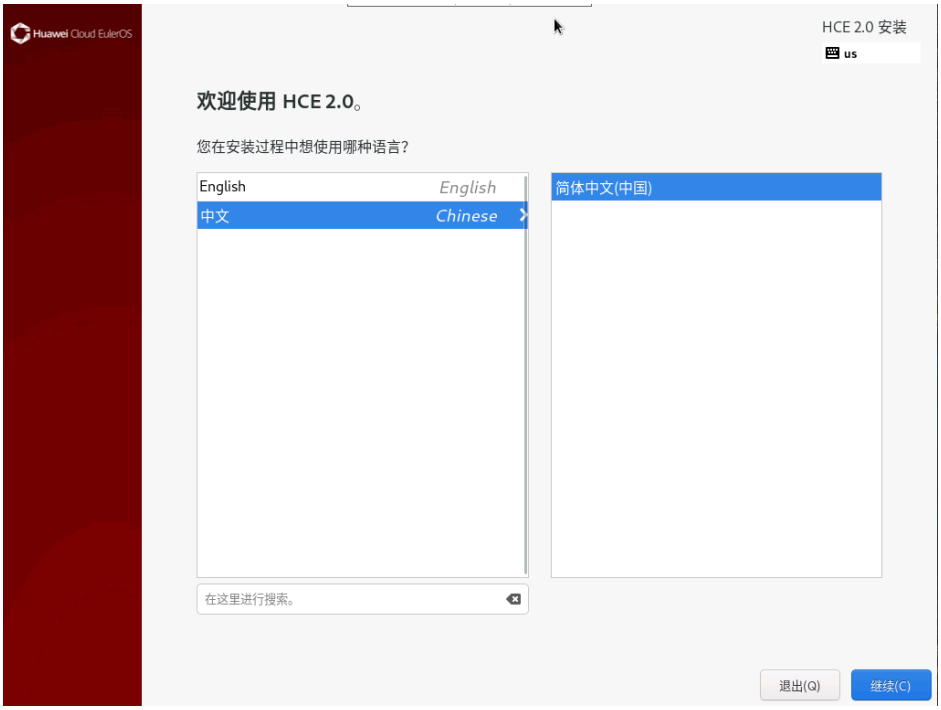
请参见**《华为云Stack 8.5.0 新建和扩容场景软件包下载列表》**中的《华为云Stack 8.5.0 IaaS软件包下载列表 04（DownloadCenter自动下载）》获取HCE的ISO发布包。

请参见《华为云Stack 8.5.0 资源发放指南》中的“计算服务 > 镜像服务（IMS）> 通过ISO镜像制作KVM公共镜像”章节制作HCE系统镜像。其中根据界面提示安装操作系统时可参见以下步骤配置，可根据实际规划进行调整。

1. 在ISO启动界面，使用键盘上下键选择“Install HCE 2.0”并回车。



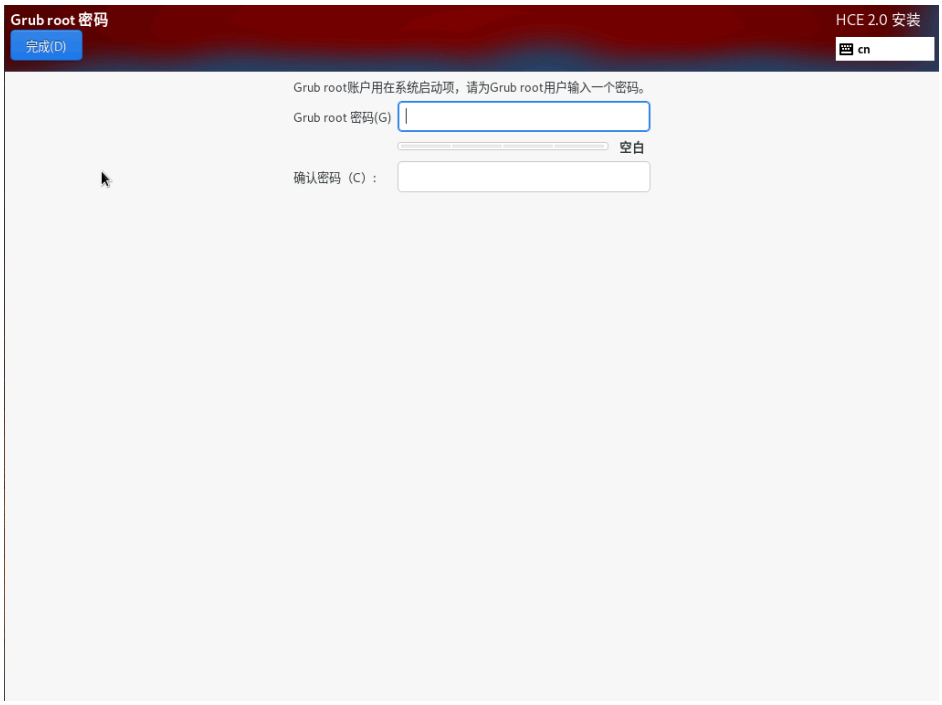
2. 等待系统加载并进入语言选择界面，按实际的需求进行语言与键盘布局选择，并单击“继续（C）”。



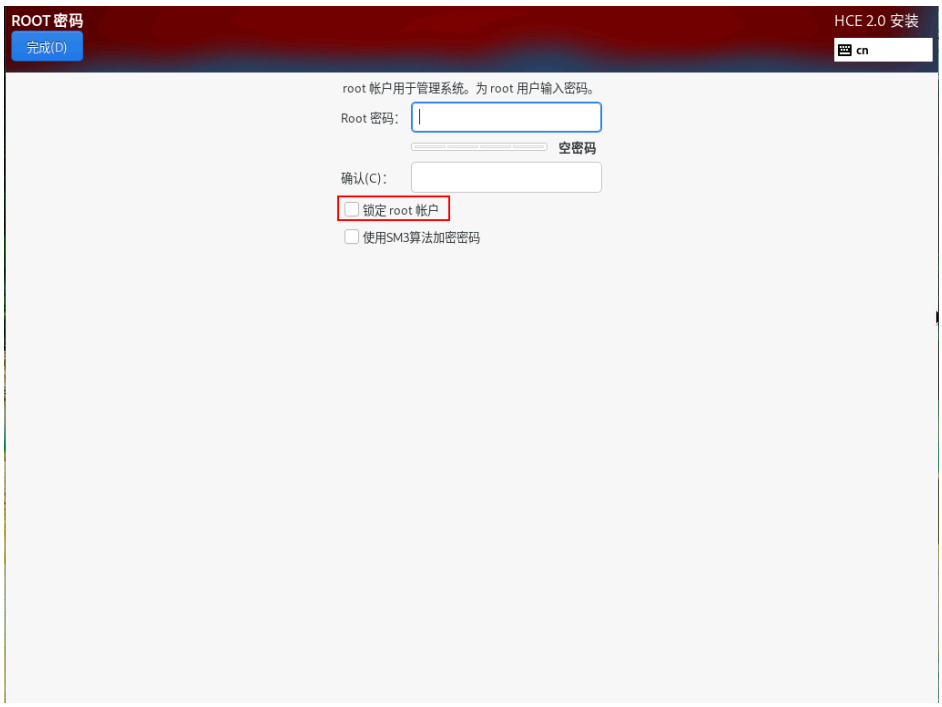
3. 在界面上，请按界面提示信息进行配置。其中，标注感叹号的内容为当前场景下的必配项，在配置修改后，必配项的内容可能发生改变。

4. 单击“安装目的地（D）”，选择需要安装的“磁盘”，并单击“完成（D）”

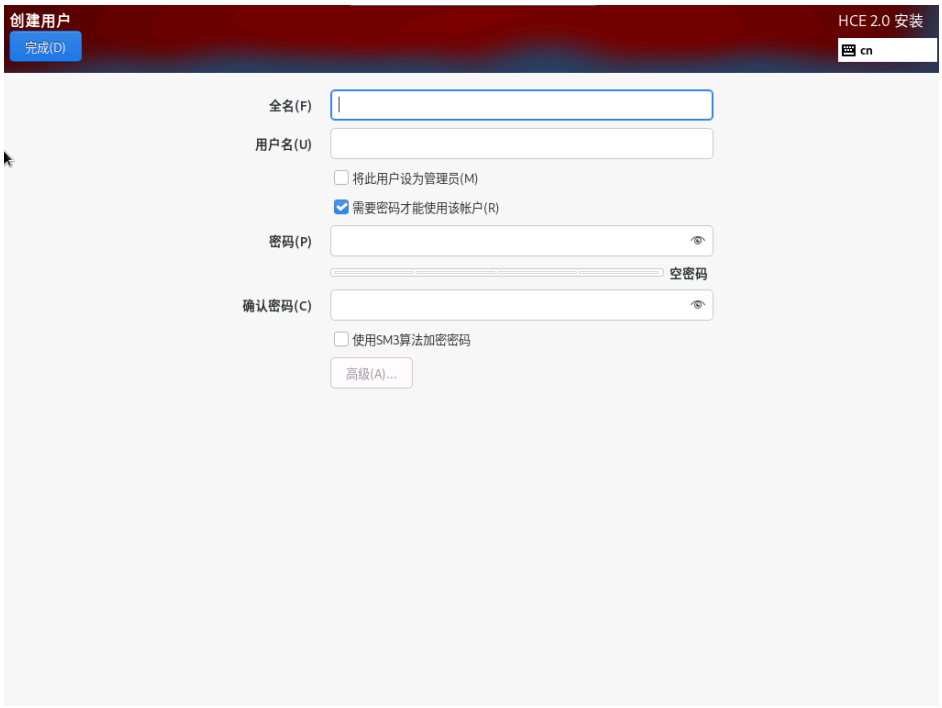
5. 单击“Grub2”，并配置Grub2 root密码，单击“完成（D）”确认配置并返回上一级菜单。



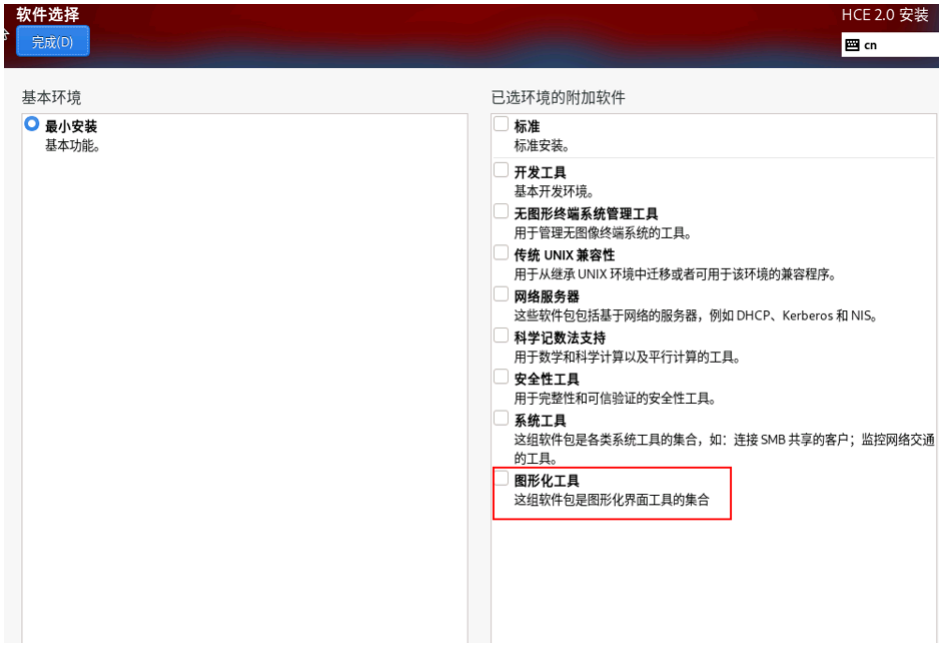
6. 单击“根密码（R）”，取消勾选“锁定root账户”，并配置root密码。单击“完成（D）”确认配置并返回上一级菜单。



7. 如需创建新用户，请单击“创建用户（U）”并填写用户信息。单击“完成（D）”确认配置并返回上一级菜单。

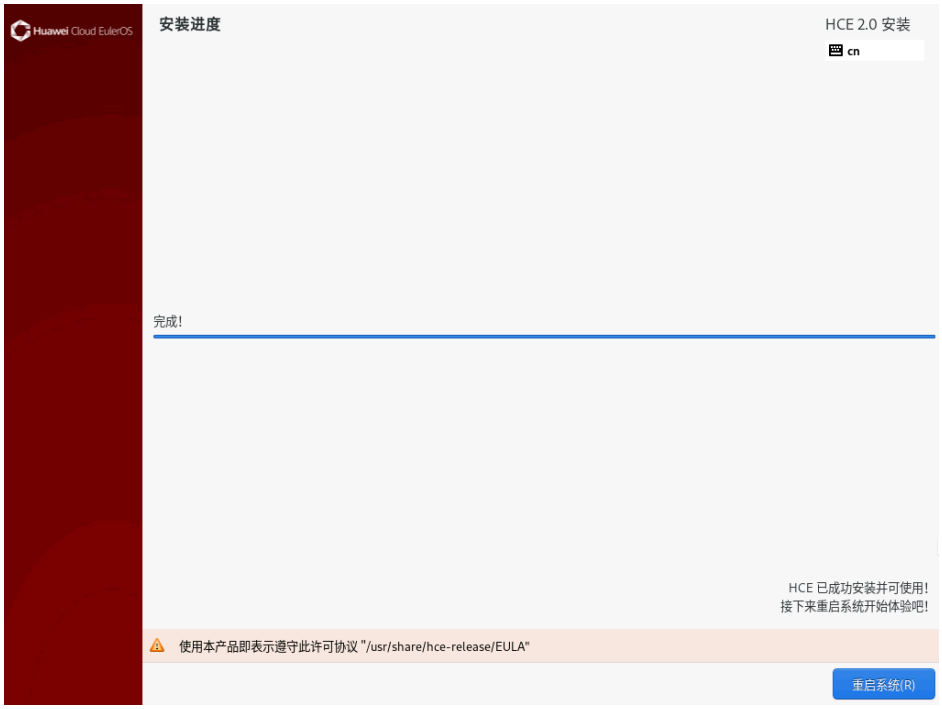


8. 如需启用OS图形化界面功能，请单击“软件选择（S）”，勾选“图形化工具”



9. 单击“开始安装(B)”

10. 等待系统安装完成，并单击“重启系统（R）”完成安装。



2 系统迁移

3 更新 HCEOS 系统和 RPM 包

4 对 HCEOS 进行安全更新

4.1 安全更新概述

本节主要介绍如何使用yum或dnf命令查询并安装Huawei Cloud EulerOS中的安全更新。

各版本对yum和dnf命令的支持情况不同，本节以yum命令为例介绍。

说明

dnf作为yum的替代者，提供更好的性能，dnf和yum命令的使用方法相同。

- Huawei Cloud EulerOS 2.0及之后版本支持yum和dnf命令。

4.2 关于通用漏洞披露（CVE）

CVE（Common Vulnerabilities and Exposures）是已公开披露的各种计算机安全漏洞，所发现的每个漏洞都有一个专属的CVE编号。Huawei Cloud EulerOS为保障系统安全性，紧密关注业界发布的CVE信息，并会及时修复系统内各类软件漏洞，增强系统的安全性。您可在Huawei Cloud EulerOS的安全公告中查看安全更新记录。

根据CVSS（Common Vulnerability Scoring System）评分，Huawei Cloud EulerOS将安全更新分为四个等级：

- Critical（高风险，必须安装）
- Important（较高风险，强烈建议安装）
- Moderate（中等风险，推荐安装）
- Low（低风险，可选安装）

若您目前已安装Huawei Cloud EulerOS，您可以根据以下操作查询并安装安全更新，修复系统漏洞。本章节以Huawei Cloud EulerOS 2.0为例举例说明。

说明

Huawei Cloud EulerOS版本不同，部分显示可能与本文档存在差异，以实际显示为准。

4.3 yum 命令参数

命令格式：yum <command> [option]

表 4-1 command 的主要参数

参数	说明
help	显示帮助信息。
updateinfo	显示软件包更新信息摘要。
upgrade	升级软件包。
check-update	检查可用的软件包更新。

表 4-2 option 的主要参数

参数	说明
-h, --help, --help-cmd	显示命令帮助信息。
--security	显示安全相关的软件包信息。
--advisory ADVISORY	指定ADVISORY相关的软件包，可用于安装、查询或更新。 多个软件包之间使用英文逗号分隔。
--cve CVES	指定CVE相关的软件包，可用于安装、查询或更新。 多个软件包之间使用英文逗号分隔。
--sec-severity {Critical,Important,Moderate,Low}	指定安全更新等级相关的软件包，可用于安装、查询或更新。 {}中的安全更新等级参数可任意组合。

说明

更多详细信息，请使用**yum --help**获取帮助信息。

4.4 查询安全更新

命令格式：yum updateinfo <command> [option]

- 执行**yum updateinfo**命令，查询全部可用的安全更新信息。

```
[root@localhost ~]# yum updateinfo
Last metadata expiration check: 0:03:05 ago on Thu 08 Sep 2022 05:30:23 PM CST.
Updates Information Summary: available
12 Security notice(s)
4 Critical Security notice(s)
```

```
6 Important Security notice(s)
2 Moderate Security notice(s)
```

- <command>的主要参数。
  - list：查询当前可用的安全更新列表。

    ```
    [root@localhost ~]# yum updateinfo list
    Last metadata expiration check: 0:03:32 ago on Thu 08 Sep 2022 05:30:23 PM CST.
    HCE2-SA-2022-0006 Critical/Sec.  curl-7.79.1-2.h6.hce2.x86_64
    HCE2-SA-2022-0011 Moderate/Sec.  gnupg2-2.2.32-1.h6.hce2.x86_64
    HCE2-SA-2022-0002 Important/Sec. kernel-5.10.0-60.18.0.50.h425_2.hce2.x86_64
    ```

  - info <SA ID>：查询指定SA ID的安全更新详情。

    ```
    [root@localhost ~]# yum updateinfo info HCE2-SA-2022-0029
    Last metadata expiration check: 5:09:15 ago on Tue 13 Sep 2022 09:43:13 AM CST.
    ==================================================================
    An update for python3 is now available for HCE 2.0
    ==================================================================
    Update ID: HCE2-SA-2022-0029
    Type: security
    Updated: 2022-09-08 22:08:34
    CVEs: CVE-2021-28861
    Description: Security Fix(es):
    : Python 3.x through 3.10 has an open redirection vulnerability in lib/http/server.py due to no protection against multiple (/) at the beginning of URI path which may leads to information disclosure. (CVE-2021-28861)
    Severity: Important
    ```

- [option]的主要参数。
  - --sec-severity={Critical,Important,Moderate,Low}：指定安全更新级别，{}中的安全更新等级参数可任意组合。

    例如，使用--sec-severity=Critical查询某个安全更新级别。

    ```
    [root@localhost ~]# yum updateinfo list --sec-severity=Critical
    Last metadata expiration check: 0:10:15 ago on Thu 08 Sep 2022 05:30:23 PM CST.
    HCE2-SA-2022-0006 Critical/Sec. curl-7.79.1-2.h6.hce2.x86_64
    HCE2-SA-2022-0003 Critical/Sec. libarchive-3.5.2-1.h2.hce2.x86_64
    HCE2-SA-2022-0006 Critical/Sec. libcurl-7.79.1-2.h6.hce2.x86_64
    ......
    ```

    例如，使用--sec-severity={Critical,Moderate}查询多个安全更新级别。

    ```
    [root@localhost ~]# yum updateinfo list --sec-severity={Critical,Moderate}
    Last metadata expiration check: 0:11:07 ago on Thu 08 Sep 2022 05:30:23 PM CST.
    HCE2-SA-2022-0006 Critical/Sec. curl-7.79.1-2.h6.hce2.x86_64
    HCE2-SA-2022-0011 Moderate/Sec. gnupg2-2.2.32-1.h6.hce2.x86_64
    HCE2-SA-2022-0003 Critical/Sec. libarchive-3.5.2-1.h2.hce2.x86_64
    ......
    ```

  - --cve=<CVE ID>：查询指定的CVE ID。

    ```
    [root@localhost ~]#  yum updateinfo info --cve=CVE-2021-28861
    Last metadata expiration check: 5:10:38 ago on Tue 13 Sep 2022 09:43:13 AM CST.
    ==================================================================
    An update for python3 is now available for HCE 2.0
    ==================================================================
    Update ID: HCE2-SA-2022-0029
    Type: security
    Updated: 2022-09-08 22:08:34
    CVEs: CVE-2021-28861
    Description: Security Fix(es):
    : Python 3.x through 3.10 has an open redirection vulnerability in lib/http/server.py due to no protection against multiple (/) at the beginning of URI path which may leads to information disclosure. (CVE-2021-28861)
    Severity: Important
    ```

说明

更多详细信息，请使用**yum updateinfo --help**获取帮助信息。

4.5 检查安全更新

- 执行**yum check-update --security**命令，检查系统当前可用的安全更新。

```
[root@localhost ~]# yum check-update --security
Last metadata expiration check: 0:11:39 ago on Thu 08 Sep 2022 05:30:23 PM CST.
curl.x86_64             7.79.1-2.h6.hce2               hce2
gnupg2.x86_64           2.2.32-1.h6.hce2               hce2
kernel.x86_64           5.10.0-60.18.0.50.h425_2.hce2              hce2
unbound-libs.x86_64     1.13.2-3.h2.hce2               hce2
```

- 执行**yum check-update --sec-severity={Critical,Important,Moderate,Low}**命令，检查指定级别的安全更新。

  {}中的安全更新等级参数可任意组合。

```
[root@localhost ~]# yum check-update --sec-severity=Moderate
Last metadata expiration check: 0:23:57 ago on Thu 08 Sep 2022 05:30:23 PM CST.
gnupg2.x86_64               2.2.32-1.h6.hce2                hce2
python3-unbound.x86_64          1.13.2-3.h2.hce2                hce2
unbound-libs.x86_64         1.13.2-3.h2.hce2                hce2
```

4.6 安装安全更新

- 执行**yum upgrade --security**命令，安装全部安全更新。

```
[root@localhost ~]# yum upgrade --security
Last metadata expiration check: 5:21:24 ago on Tue 13 Sep 2022 09:43:13 AM CST.
Dependencies resolved.
================================================================
Package       Arch       Version                Repository    Size
================================================================
Installing:
Kernel        x86_64       5.10.0-60.18.0.50.h498_2.hce2    hce2       49 M
Upgrading:
Curl          x86_64       7.79.1-2.h6.hce2          hce2       147 k
......
Transaction Summary
================================================================
Install   1 Package
Upgrade  22 Packages
Total download size: 69 M
Is this ok [y/N]:
```

- 执行**yum upgrade --sec-severity={Critical,Important,Moderate,Low}**命令，安装指定级别的安全更新。

  {}中的安全更新等级参数可任意组合。

```
[root@localhost ~]# yum upgrade --sec-severity=Moderate
Last metadata expiration check: 0:32:27 ago on Thu 08 Sep 2022 05:30:23 PM CST.
Dependencies resolved.
================================================================
Package                 Architecture          Version            Repository       Size
================================================================
Upgrading:
gnupg2              x86_64              2.2.32-1.h6.hce2      hce2          2.2 M
python3-unbound         x86_64              1.13.2-3.h2.hce2         hce2            96 k
unbound-libs            x86_64              1.13.2-3.h2.hce2      hce2           505 k
Transaction Summary
================================================================Upgrade       3
Packages
Total download size: 2.8 M
Is this ok [y/N]:
```

- 执行**yum upgrade --advisory =<SA ID>**命令，安装指定SA ID的安全更新。

  多个软件包之间使用英文逗号分隔。

```
root@localhost ~]# yum upgrade --advisory=HCE2-SA-2022-0033
Last metadata expiration check: 1:48:44 ago on Tue 13 Sep 2022 09:43:13 AM CST.
Dependencies resolved.
================================================================
Package             Architecture          Version               Repository          Size
================================================================
Upgrading:
python3-rpm           x86_64        4.17.0-8.h13.hce2        hce2        87 k
rpm           x86_64        4.17.0-8.h13.hce2        hce2        498 k
rpm-libs        x86_64        4.17.0-8.h13.hce2        hce2        376 k
Transaction Summary
================================================================
Upgrade  3 Packages
Total download size: 962 k
Is this ok [y/N]:
```

- 执行**yum upgrade --cve=<CVE ID>**命令，安装指定CVE ID的安全更新。

  多个软件包之间使用英文逗号分隔。

```
[root@localhost ~]# yum upgrade --cve=CVE-2021-28861
Last metadata expiration check: 5:16:36 ago on Tue 13 Sep 2022 09:43:13 AM CST.
Dependencies resolved.
================================================================
Package                 Architecture    Version            Repository    Size
================================================================
Upgrading:
python3               x86_64          3.9.9-7.h10.hce2           hce2           8.0 M
python3-unversioned-command    x86_64          3.9.9-7.h10.hce2           hce2           3.9 k
Transaction Summary
================================================================
Upgrade  2 Packages
Total download size: 8.0 M
Is this ok [y/N]:
```

5 HCEOS 获取 openEuler 扩展软件包

HCEOS默认不加载开源社区openEuler的repo源，避免openEuler的软件包和HCEOS的软件包冲突。

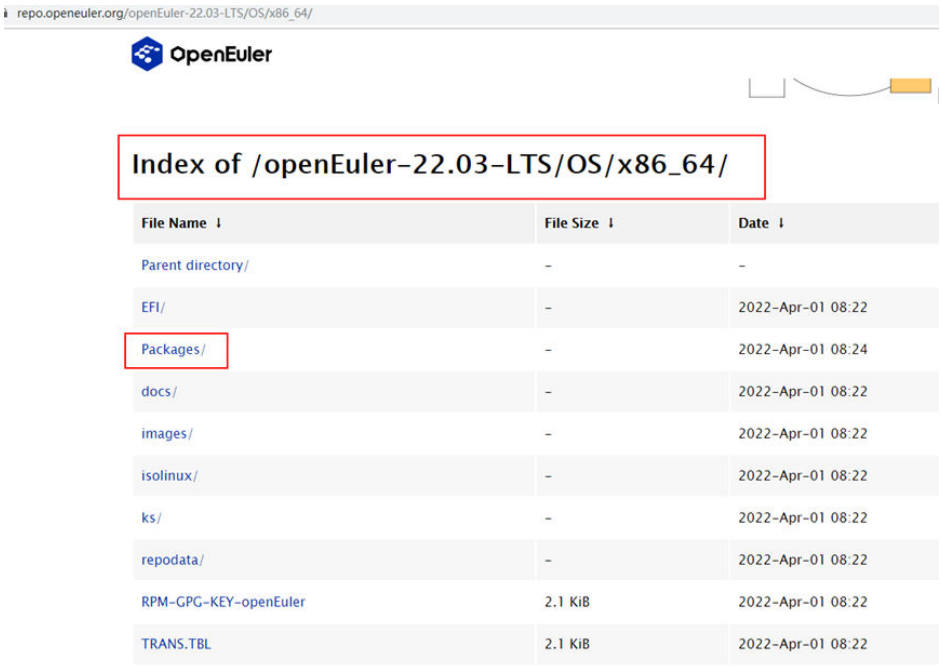
当前HCEOS 2.0版本仅兼容openEuler 22.03 LTS版本。本节介绍HCEOS 2.0如何获取openEuler 22.03 LTS的扩展软件包。获取方法有两种，请根据需要选择合适的方法。

获取方法	适用场景	安装依赖RPM包	repo文件的备份及恢复
通过wget命令下载RPM包	适用于少量RPM包的安装。	需要手动下载并安装依赖的RPM包。	不涉及
通过repo文件批量下载RPM包	适用于较多RPM包的安装。	自动安装依赖的RPM包，无需再次下载。	须先将/etc/yum.repos.d目录下原有的repo文件进行备份，并删除此目录下原有的repo文件。安装RPM包后，须再次恢复这些repo文件。

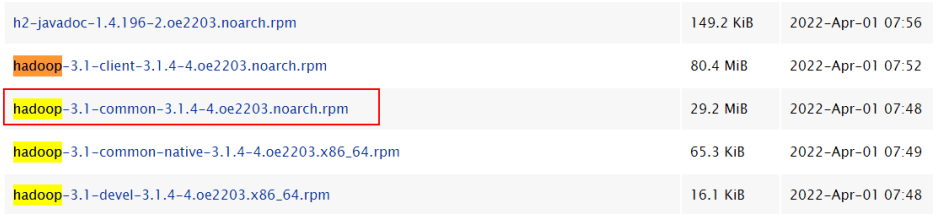
通过 wget 命令下载 RPM 包

本节以下载hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm为例，介绍使用**wget**命令下载并安装RPM包。

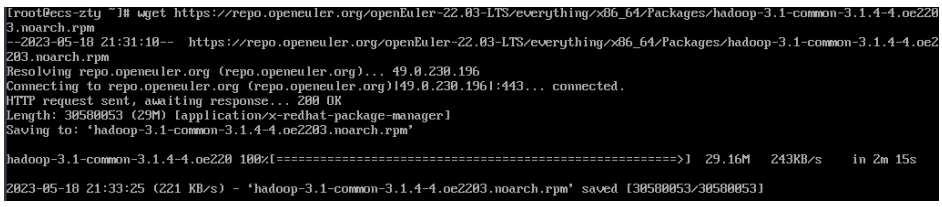
1. 单击**这里**登录openEuler社区。
2. 在OS/everything目录下，选择aarch64/或者x86_64/系统架构目录，并打开“Packages/”目录。



3. 查找所需要的RPM包，例如hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm。



4. 选择此包后右击复制下载链接，执行**wget**命令下载RPM包。



5. 检查是否下载成功。如下所示表示下载成功。

6. 使用**rpm -ivh hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm**命令安装RPM包，如下所示表示安装成功。

   如果安装过程中提示需要依赖其他的安装包，请根据同样的操作步骤先安装所依赖的安装包。


```
[root@ecs-zty ~]# rpm -ivh hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm
warning: hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm: Header V3 RSA/SHA1 Signature, key ID b25e7f66: NOKEY
error: Failed dependencies:
        apache-zookeeper is needed by hadoop-3.1-common-3.1.4-4.oe2203.noarch
        leveldb is needed by hadoop-3.1-common-3.1.4-4.oe2203.noarch
        protobuf2-java is needed by hadoop-3.1-common-3.1.4-4.oe2203.noarch
[root@ecs-zty ~]# rpm -ivh hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm --force
warning: hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm: Header V3 RSA/SHA1 Signature, key ID b25e7f66: NOKEY
error: Failed dependencies:
        apache-zookeeper is needed by hadoop-3.1-common-3.1.4-4.oe2203.noarch
        leveldb is needed by hadoop-3.1-common-3.1.4-4.oe2203.noarch
        protobuf2-java is needed by hadoop-3.1-common-3.1.4-4.oe2203.noarch
[root@ecs-zty ~]# rpm -ivh hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm --nodeps
warning: hadoop-3.1-common-3.1.4-4.oe2203.noarch.rpm: Header V3 RSA/SHA1 Signature, key ID b25e7f66: NOKEY
Verifying...                          ################################# [100%]
Preparing...                          ################################# [100%]
Updating / installing...
   1:hadoop-3.1-common-3.1.4-4.oe2203 ################################# [100%]
[root@ecs-zty ~]#
```

通过 repo 文件批量下载 RPM 包

本节以openEuler-22.03-LTS/everything/x86_64为例，介绍下载openEuler-22.03-LTS/everything/x86_64目录下的RPM包并使用yum命令安装。

1. 首先确保虚拟机能访问**https://repo.openeuler.org/openEuler-22.03-LTS/**网址。
2. 配置yum源。

   进入/etc/yum.repos.d目录，新建一个openEuler.repo文件，并将以下内容复制到该文件里面。

   说明

   由于openEuler.repo文件和HCEOS系统repo文件有冲突，请先将/etc/yum.repos.d目录下HCEOS原有的repo文件进行备份，并删除HCEOS原有的repo文件，再创建openEuler.repo文件。

```
[openeuler]
name=openeuler
baseurl=https://repo.openeuler.org/openEuler-22.03-LTS/OS/x86_64/
gpgcheck=1
enabled=1
priority=3
gpgkey=https://repo.openeuler.org/openEuler-22.03-LTS/OS/x86_64/RPM-GPG-KEY-openEuler
[everything]
name=everything
baseurl=https://repo.openeuler.org/openEuler-22.03-LTS/everything/x86_64
gpgcheck=1
enabled=1
priority=3
gpgkey=https://repo.openeuler.org/openEuler-22.03-LTS/everything/x86_64/RPM-GPG-KEY-openEuler
```

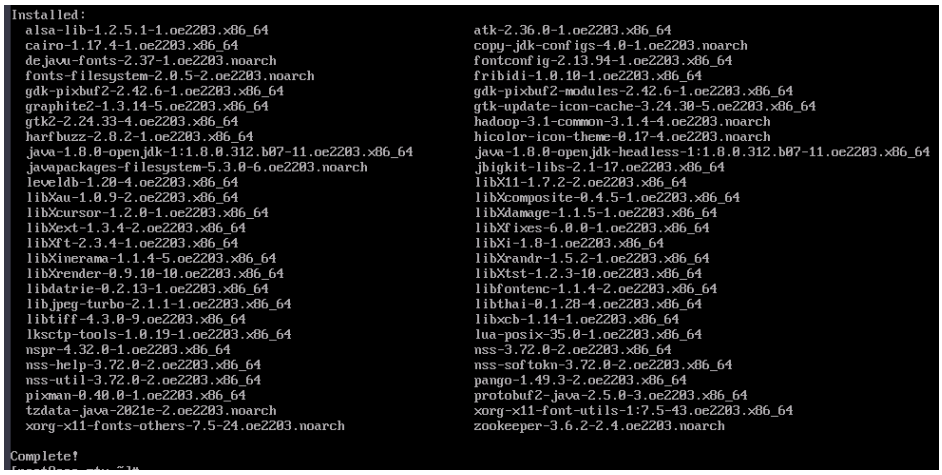
3. 执行**yum clean all**清除原来yum源的缓存信息。
4. 执行**yum makecache**连接新配置的源 ，如下图所示表示repo源连接成功。

```
[root@ecs-zty ~]# yum clean all
12 files removed
[root@ecs-zty ~]# yum makecache
openeuler                                                   235 kB/s | 3.4 MB     00:14
everything                                                  229 kB/s |  16 MB     01:12
HCE 2.0 base                                                 59 MB/s | 6.1 MB     00:00
HCE 2.0 updates                                             1.8 MB/s | 7.8 MB     00:04
Last metadata expiration check: 0:00:02 ago on Thu 18 May 2023 09:55:03 PM CST.
Metadata cache created.
```

5. 安装RPM包，以hadoop-3.1-common包为例。
   a. 执行**yum list**命令查看是否存在该包。

```
[root@ecs-zty ~]# yum list | grep hadoop-3.1-common
hadoop-3.1-common.noarch                    3.1.4-4.oe2203                  @System
hadoop-3.1-common-native.x86_64             3.1.4-4.oe2203                  everything
[root@ecs-zty ~]#
```

   b. 执行**yum -y install hadoop-3.1-common**命令来安装此包，如下所示表示该包已经安装成功。



6. 恢复repo文件。

   安装所需的openEuler包后，删除openEuler.repo文件，并将步骤**2**中删除的repo文件通过备份恢复。

6 HCE License 管理

操作背景

HCE提供HCE License的维保服务和软件升级功能。

- 在HCE操作系统需要进行维保、升级软件前，需要先执行License激活操作。
- 在已激活License的服务器被删除后，需要先对该服务器执行去激活操作，然后再在其他服务器上执行激活HCE License操作。

本节介绍在HCE操作系统上激活License、去激活License，查询License状态的方法。

约束限制

- 需要以root权限执行subscription-manager相关命令。
- 去激活操作只能在服务器已删除、License过期或License使用超过控制上限时执行。

  说明

  删除服务器前请先保存服务器的uuid值，即实例ID，以备去激活操作时使用。

前提条件

已创建装有HCE操作系统的弹性云服务器或裸金属服务器并登录。

弹性云服务器的详细创建步骤请参见**《弹性云服务器(ECS) 使用指南(for 华为云Stack 8.5.0)》**中《弹性云服务器(ECS) 8.5.0 用户指南(for 华为云Stack 8.5.0)》的“创建弹性云服务器”章节。

裸金属服务器的详细创建步骤请参见**《裸金属服务器(BMS) 使用指南(for 华为云Stack 8.5.0)》**中《裸金属服务器(BMS) 8.5.0 用户指南(for 华为云Stack 8.5.0)》的“创建裸金属服务器”章节。

使用方法

- 执行以下命令配置账户密码。各参数请根据实际情况修改。

  subscription-manager config --AK=xxx --SK=xxx --server=xxx --projectId=xxx

  - AK/SK：访问密钥id及其关联的私有访问密钥。请参见**《华为云欧拉操作系统(HCE) 使用指南(for 华为云Stack 8.5.0)》**中《华为云欧拉操作系统(HCE)

2.0.2409 API参考(for 华为云Stack 8.5.0) 》的“如何调用API > 认证鉴权”章节获取。

- server：https://服务端的endpoint终端节点。终端节点请参见**《 华为云欧拉操作系统(HCE) 使用指南(for 华为云Stack 8.5.0) 》**中《 华为云欧拉操作系统(HCE) 2.0.2409 API参考(for 华为云Stack 8.5.0) 》的“使用前必读” 章节获取。
- projectId：项目id。请参见**《 华为云欧拉操作系统(HCE) 使用指南(for 华为云Stack 8.5.0) 》**中《 华为云欧拉操作系统(HCE) 2.0.2409 API参考(for 华为云Stack 8.5.0) 》的“附录 > 获取项目ID”章节获取。

- 执行以下命令激活当前服务器HCE License：

  subscription-manager register
- 执行以下命令去激活服务器HCE License，参数uuid为目标服务器的唯一标识：

  subscription-manager unregister --uuid=xxxx

  uuid为实例的ID，请登录运营面云服务器控制台，单击实例名称获取实例ID。
- 执行以下命令查询服务器HCE License状态，参数uuid为目标服务器的唯一标识，缺省时目标服务器默认为本机：

  subscription-manager status --uuid=xxxx
- 执行以下命令查询账户下HCE License处于激活、宽限期和已过期的服务器：

  subscription-manager status --all

HCE License所有的状态及说明如下表所示。

表 6-1 HCE License 状态说明

License状态	说明
active	处于激活状态
grace	处于宽限期
inactive	未激活

7 制作 Docker 镜像并启动容器

本节介绍在HCEOS上制作HCEOS系统的Docker镜像并启动容器。

约束限制

- 运行容器镜像的HCEOS系统版本和制作的HCEOS容器镜像版本须保持一致。

制作镜像归档文件

1. 确认repo源配置正常。
   配置repo源请参见**HCE搭建REPO服务器**。
2. 新建临时目录作为Docker镜像的根系统文件，并将软件包安装到临时目录。
   ```
   rm -rf /tmp/docker_rootfs
   mkdir -p /tmp/docker_rootfs
   yum --setopt=install_weak_deps=False --installroot /tmp/docker_rootfs --releasever 2.0 install bash yum coreutils security-tool procps-ng vim-minimal tar findutils filesystem hce-repos hce-rootfiles cronie -y
   ```

   ⚠ 注意

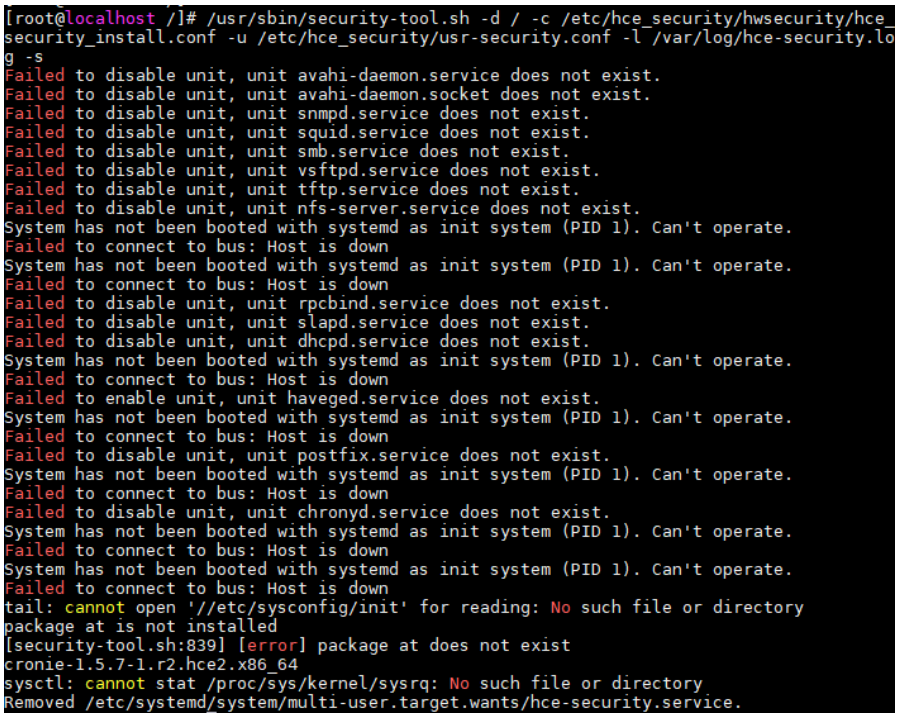
   - 上述操作中的**releasever**需替换为HCEOS对应的版本号。
   - 也可在此处安装其他所需的软件包，但是要确保/tmp文件下的空间足够。

3. chroot进入临时目录，进行如下配置。
   ```
   chroot /tmp/docker_rootfs
   ```
   ```
   [root@localhost tmp]# chroot /tmp/docker_rootfs
   [root@localhost /]#
   [root@localhost /]#
   ```

   a. 使用HCE security-tool.sh关闭不必要服务。
   ```
   export EULEROS_SECURITY=0
   echo "export TMOUT=300" >> /etc/bashrc
   /usr/sbin/security-tool.sh -d / -c /etc/hce_security/hwsecurity/hce_security_install.conf -u /etc/hce_security/usr-security.conf -l /var/log/hce-security.log -s
   ```
   执行过程中，有以下错误打印均为正常现象，报错的原因为：

   - 缺少服务文件。在chroot文件系统下没有启动服务导致。
   - 缺少引导系统的文件/etc/sysconfig/init。工具在系统启动阶段关闭服务，镜像rootfs不涉及系统启动。

- 缺少/proc/sys/kernel/sysrq，这是系统启动后生成的调用节点，在chroot文件系统下不存在。



b. 卸载security-tool、cronie、systemd软件包及其依赖软件包。

```
cp -af /etc/pam.d /etc/pam.d.bak
rm -f /etc/yum/protected.d/sudo.conf /etc/yum/protected.d/systemd.conf
yum remove -y security-tool cronie systemd
rpm -e --nodeps logrotate crontabs
rm -rf /etc/pam.d
mv /etc/pam.d.bak /etc/pam.d
sh -c 'shopt -s globstar; for f in $(ls /**/*.rpmsave); do rm -f $f; done'
[ -d /var/lib/dnf ] && rm -rf /var/lib/dnf/*
[ -d /var/lib/rpm ] && rm -rf /var/lib/rpm/__db.*
```

c. 移除/boot目录。

```
rm -rf /boot
```

d. 设置容器镜像语言为en_US。

```
cd /usr/lib/locale;rm -rf $(ls | grep -v en_US | grep -vw C.utf8 )
rm -rf /usr/share/locale/*
```

e. 移除共享文件 man、doc、info和mime。

```
rm -rf /usr/share/{man,doc,info,mime}
```

f. 移除缓存日志文件。

```
rm -rf /etc/ld.so.cache
[ -d /var/cache/ldconfig ] && rm -rf /var/cache/ldconfig/*
[ -d /var/cache/dnf ] && rm -rf /var/cache/dnf/*
[ -d /var/log ] && rm -rf /var/log/*.log
```

g. 移除java安全证书。

```
rm -rf /etc/pki/ca-trust/extracted/java/cacerts /etc/pki/java/cacerts
```

h. 移除/etc/machine-id。

```
rm -rf /etc/machine-id
```

i. 移除/etc/mtab。

```
rm -rf /etc/mtab
```

4. 退出chroot。

```
exit
```


5. 打包压缩临时目录，生成Docker镜像归档文件hce-docker.x86_64.tar.xz。

   归档路径为/tmp/docker_rootfs/hce-docker.x86_64.tar.xz

   ```
   pushd /tmp/docker_rootfs/
   tar cvf hce-docker.x86_64.tar .
   xz hce-docker.x86_64.tar
   popd
   ```

6. 将镜像归档文件转换为带有layer信息的文件。

   上述生成的镜像归档文件没有layer信息，不能使用**docker load**命令加载镜像。需要用**docker import**命令先将归档文件生成为镜像，再用**docker save**命令保存为一个带有layer信息的镜像文件，这样就可以使用**docker load**命令来加载镜像了。下面以镜像名为my_image，归档文件为docker_save.tar.xz举例。

   ```
   docker import hce-docker.x86_64.tar.xz my_image:v1
   docker save -o docker_save.tar.xz my_image:v1
   ```

   此时生成的docker_save.tar.xz归档文件就可以用**docker load**命令来加载镜像。

   ```
   docker load -i docker_save.tar.xz
   ```

使用镜像归档文件启动容器

1. 确认repo源配置正常。

   配置repo源请参见**HCE搭建REPO服务器**。

2. 安装docker软件包。

   ```
   yum install docker -y
   ```

3. 使用镜像归档文件创建容器镜像。

   ```
   mv /tmp/docker_rootfs/hce-docker.x86_64.tar.xz .
   docker import hce-docker.x86_64.tar.xz
   ```

   执行**docker images**命令可查看到容器镜像ID为6cfefae3a541。

   ```
   [root@localhost /]# mv /tmp/docker_rootfs/hce-docker.x86_64.tar.xz .
   [root@localhost /]# docker import hce-docker.x86_64.tar.xz
   sha256:6cfefae3a5410e3b48208f8a9e0a28fc08fa1b3a62ad39b27196a742969d5bfc
   [root@localhost /]#
   [root@localhost /]# docker images
   REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
   <none>              <none>              6cfefae3a541        6 seconds ago       181MB
   ```

   说明

   创建镜像可使用如下命令指定镜像的REPOSITORY和TAG参数。

   docker import [OPTIONS] file|URL|- [REPOSITORY[:TAG]]

4. 在容器中运行镜像bash文件。

   运行如下命令后，如果shell视图改变，表示成功进入容器bash。

   ```
   docker run -it 6cfefae3a541 bash
   ```

   ```
   [root@localhost /]# docker run -it 6cfefae3a541 bash
   [root@5c639b63fc0e /]#
   [root@5c639b63fc0e /]#
   ```

8 工具类

8.1 毕昇编译器

毕昇编译器（bisheng compiler）是一款提供高性能、高可信及易扩展的编译器工具链。毕昇编译器引入了多种编译技术，支持C/C++/Fortran编译语言。

约束限制

- 仅HCEOS 2.0 x86架构支持使用毕昇编译器。
- HCEOS原生的clang编译语言和毕昇编译器提供的clang编译语言不能同时使用。如果您已经安装原生的clang编译语言并需要使用它，就不能安装毕昇编译器。

  在安装了毕昇编译器之后，如果需要使用原生的clang编译语言，可执行**rpm -e bisheng-compiler**命令删除毕昇编译器，然后打开新终端。在新终端中，就可以使用原生的clang编译语言。

安装毕昇编译器

1. 确认repo源配置正常。

   配置repo源请参见**HCE搭建REPO服务器**。
2. 执行**yum install bisheng-compiler**命令安装工具。
3. 执行**source /usr/local/bisheng-compiler/env.sh**命令，导入环境变量。

   如果打开了新的终端，需要在新的终端重新导入环境变量才能正常使用毕昇编译器。
4. 检查工具是否安装成功。

   执行**clang -v**查看工具的版本号。若返回结果包含毕昇编译器版本信息，表示工具安装成功。

使用毕昇编译器

1. 编译运行C/C++程序。

   ```
   clang [command line flags] hello.c -o hello.o
   ./hello.o
   clang++ [command line flags] hello.cpp -o hello.o
   ./hello.o
   ```
2. 编译运行Fortran程序。


```
flang [command line flags] hello.f90 -o hello.o
./hello.o
```

3. 指定链接器。

   毕昇编译器指定的链接器是LLVM的lld，若不指定它则使用默认的ld。

   ```
   clang [command line flags] -fuse-ld=lld hello.c -o hello.o
   ./hello.o
   ```

8.2 应用加速工具

8.3 Pod 带宽管理工具

在业务混合部署的场景下，Pod带宽管理功能根据QoS分级对资源进行合理调度，提升网络带宽利用率。HCEOS提供oncn-tbwm带宽管理工具，使用tbwmcli命令对收发包方向的网络限速功能，实现网络QoS。

前提条件

本功能固定使用ifb0，使用前请确定虚拟网卡ifb0未被使用，并加载ifb驱动。

约束与限制

- 仅HCEOS 2.0 x86架构支持使用tbwmcli命令。
- 仅允许root用户执行tbwmcli命令。
- tbwmcli命令同一时间只能在一个网卡使能Qos功能，多个网卡不支持并行使能网络QoS。
- 网卡被插拔重新恢复后，原来设置的QoS规则会丢失，需要手动重新配置网络QoS功能。
- 不支持cgroup v2。
- 升级oncn-tbwm软件包不会影响升级前的使能状态。卸载oncn-tbwm软件包会关闭对所有设备的使能。
- 仅支持识别数字、英文字母、中划线“-” 和下划线“_”四类字符类型的网卡名，其他字符类型的网卡不被识别。
- 实际使用过程中，带宽限速有可能造成协议栈内存积压，此时依赖传输层协议自行反压，对于udp等无反压机制的协议场景，可能出现丢包、ENOBUFS、限流不准等问题。
- 收包方向的网络限速依赖于TCP的反压能力，在非TCP协议的场景中，网络包已经收至目标网卡，不支持对于收包方向的网络限速。
- 不支持tbwmcli、tc命令和网卡命令混用，只能单独使用tbwmcli工具进行限速。例如，某个网卡上已经设置过tc qdisc规则的情况下，对此网卡使能网络QoS功能可能会失败。

使用方法

1. 安装oncn-tbwm软件包。

   a. 确认repo源配置正常。

      配置repo源请参见**HCE搭建REPO服务器**。

b. 执行**yum install oncn-tbwm**命令安装oncn-tbwm软件包。

c. 验证oncn-tbwm软件包正确性。

- 执行**tbwmcli -v**命令，正确安装则结果显示如下。
```
version: 1.0
```
- 确认以下oncn-tbwm服务组件，正常情况下以下服务组件均存在。
```
/usr/bin/tbwmcli
/usr/share/tbwmcli
/usr/share/tbwmcli/README.md
/usr/share/tbwmcli/bwm_prio_kern.o
/usr/share/tbwmcli/tbwm_tc.o
```

2. 根据需要执行tbwmcli命令。

表 8-1 tbwmcli 命令说明

命令	说明
tbwmcli -e ethx tbwmcli -d ethx egress	使能/关闭某个网卡的发包方向Qos功能。 示例：使能eth0网卡的发包方向Qos功能。 **tbwmcli -e eth0** enable eth0 egress success 示例：关闭eth0网卡的发包方向Qos功能。 **tbwmcli -d eth0 egress** disable eth0 egress success
tbwmcli -i ethx online/ offline tbwmcli -d ethx ingress	使能/关闭某个网卡的收包方向Qos功能。 示例：使能eth0网卡收包方向Qos功能，并设置为在线网卡。 **tbwmcli -i eth0 online** enable eth0 ingress success, dev is online 示例：使能eth0网卡收包方向Qos功能，并设置为离线网卡。 **tbwmcli -i eth0 offline** enable eth0 ingress success, dev is offline **说明** 收包方向不支持同时设置为多个离线网卡的情况，支持一个离线网卡和多个在线网卡。 示例：关闭eth0网卡收包方向Qos功能。 **tbwmcli -d eth0 ingress** disable eth0 ingress success
tbwmcli -d ethx	强制关闭某个网卡的所有Qos功能，并关闭ifb功能。 示例：强制关闭eth0网卡的所有Qos功能，并关闭ifb功能。 **tbwmcli -d eth0** disable eth0 success

命令	说明
tbwmcli -p istats/estats	打印收/发包方向内部统计信息。 示例：打印收包方向内部统计信息。 **tbwmcli -p istats** offline_target_bandwidth: 94371840online_pkts: 3626190offline_pkts: 265807online_rate: 0offline_rate: 13580offline_prio: 0 示例：打印发包方向内部统计信息。 **tbwmcli -p estats** offline_target_bandwidth: 94371840online_pkts: 4805452offline_pkts: 373961online_rate: 0offline_rate: 19307offline_prio: 1
tbwmcli -s path <prio> tbwmcli -p path	设置/查询cgroup的QoS优先级。 当前仅支持设置离线和在线两个QoS优先级。 ● 0：设置cgroup为在线QoS优先级。 ● -1：设置cgroup为离线QoS优先级。 示例：设置test_online cgroup的优先级为0。 **tbwmcli -s /sys/fs/cgroup/test_online 0** set prio success 示例：查询test_online cgroup的优先级。 **tbwmcli -p /sys/fs/cgroup/test_online** prio is 0
tbwmcli -s bandwidth <low，high> tbwmcli -p bandwidth	设置/查询离线带宽范围。 示例：设置离线宽带范围为30mb~100mb。 **tbwmcli -s bandwidth 30mb,100mb** set bandwidth success 示例：查询离线带宽范围。 **tbwmcli -p bandwidth** bandwidth is 31457280(B),104857600(B)
tbwmcli -s waterline <val> tbwmcli -p waterlin	设置/查询在线网络带宽水线。 示例：设置在线网络带宽水线为20mb。 **tbwmcli -s waterline 20mb** set waterline success 示例：查询在线网络带宽水线。 **tbwmcli -p waterline** waterline is 20971520 (B)
tbwmcli -p devs	查看系统上所有网卡的使能状态。 **tbwmcli -p devs** lo Egress : disabled lo Ingress : disabled eth0 Egress : disabled eth0 Ingress : enabled, it's offline ifb0 Egress : enabled
tbwmcli -c	强制删除所有网卡的qos，谨慎使用。
modprobe ifb numifbs=1	加载ifb。
rmmod ifb	卸载ifb。

8.4 安全加固工具

概述

HCE 2.0作为面向用户的通用Linux发行版，默认发布的OS ISO安装后未进行安全加固。

security-tool是符合相关基础安全加固要求的加固工具包，默认不随HCE 2.0安装，当需要执行加固时选择性安装，安装完成后在OS首次启动时执行自动化加固。

详细加固内容请参见security-tool RPM包内的相关加固配置，概括起来包括：

- 系统服务：例如SSH配置、删除postfix.service、启用haveged.service
- 内核参数：例如内核网络协议栈加固
- 账号口令：例如PAM参数加固
- 授权认证：例如warning banner、umask
- 文件权限：例如cron配置

security-tool 工具使用

步骤1 确认repo源配置正常。

配置repo源请参见**HCE搭建REPO服务器**。

步骤2 安装security-tool工具包

```
yum install -y security-tool
```

步骤3 在/etc/hce_security/hce_enhance_type.conf中写入需要加固的配置类型。

当前支持3种配置：cybersecurity（等保加固配置）、hwsecurity（云服务加固配置）、general（通用加固配置），推荐使用general（通用加固配置）。本示例以general为例。

```
echo general > /etc/hce_security/hce_enhance_type.conf
```

步骤4 启动 hce-security服务

```
systemctl start hce-security
```

执行完成后使用systemctl status hce-security查看服务状态，状态为 active(exited)即为加固成功。

```
[root@localhost ~]# systemctl status hce-security
● hce-security.service - HCE Security Tool
     Loaded: loaded (/usr/lib/systemd/system/hce-security.service; disabled; vendor preset: disabled)
     Active: active (exited) since Sat 2023-10-28 14:13:09 CST; 7min ago
    Process: 42457 ExecStart=/usr/sbin/hce_security-tool.sh (code=exited, status=0/SUCCESS)
   Main PID: 42457 (code=exited, status=0/SUCCESS)
```

加固日志见/var/log/hce_security.log。

用户可自行修改 /etc/hce_security/usr-security.conf 配置自己的加固项，完成个性化加固，配置文件具体修改方式如下：

```
################################################################################
#
# HowTo:
```

```
#       # delete key, and difference caused by blankspace/tab on key is ignored
#       id@d@file@key
#
#       # modify option: find line started with key, and get the value changed
#       id@m@file@key[@value]
#
#       # modify sub-option: find line started with key, and then change the value of key2 to
value2(prepostive separator should not be blank characters) in the line
#       id@M@file@key@key2[@value2]
#
#       # check existence of commands
#       id@which@command1 [command2 ...]
#
#       # execute command on the files found
#       id@find@dir@condition@command
#
#       # any command(with or without parameter), such as 'rm -f','chmod 700','which','touch', used to
extend functions, return 0 is ok
#       id@command@file1 [file2 ...]
#
# Notes:
#       1. The comment line should start with '#'
#       2. "value" related with "key" should contain prepositive separator("=," " and so on), if there is any.
#       3. When item starts with "d", "m" or "M", "file" should be a single normal file, otherwise multi-
objects(separated by blankspace) are allowed.
#
################################################################################
```

----结束

general/hwsecurity/cybersecurity 三种类型的差异

检查项类型	检查项名称	检查内容	general	hwsecurity	cybersecurity	是否默认满足
初始配置	文件系统配置	应当对系统关键目录进行分区挂载	-	-	-	否
		确保禁用不需要的文件系统	-	-	-	否
		确保无需修改的分区以只读方式挂载	-	-	-	否
		确保无需挂载设备的分区以nodev方式挂载	-	-	-	否
		确保无可执行文件的分区以noexec方式挂载	-	-	-	否
		确保无需SUID和SGID的分区以nosuid方式挂载	-	-	-	否
		避免使用USB存储	√	-	-	是

	软件服务配置	禁止安装X Window系统	-	-	-	是
		禁止启用debug-shell服务	√	-	-	是
		禁止启用rsync服务	√	-	-	是
		禁止启用avahi服务	√	√	-	是
		禁止启用SNMP服务	√	√	-	是
		禁止启用squid服务	√	√	-	是
		避免启用samba服务	√	√	-	是
		禁止启用FTP服务	√	√	-	是
		禁止启用TFTP服务	√	√	-	是
		禁止启用DNS服务	√	-	-	是
		禁止启用NFS服务	√	√	-	是
		禁止启用rpcbind服务	√	√	√	否
		禁止启用LDAP服务	√	√	-	是
		禁止启用DHCP服务	√	√	-	是
		禁止安装CUPS服务软件	-	-	-	是
		禁止安装NIS服务软件	-	-	-	是
		禁止安装telnet软件	-	-	-	是
		禁止安装NIS客户端	-	-	-	是
		禁止安装LDAP客户端	-	-	-	是
		禁止安装调测类工具	-	-	-	是
		禁止安装开发编译类工具	-	-	-	是
		禁止安装网络嗅探类工具	-	-	-	是
	软件升级配置	确保配置GPG公钥	-	-	-	是
		确保配置启用gpgcheck	-	-	-	是
		确保配置软件仓库源	-	-	-	是
	文件完整性检查	确保安装AIDE	-	-	-	否

		应当定期检查文件完整性	-	-	-	否
	通用进程加固	确保启用ASLR	√	-	-	是
		确保core dump配置正确	√	-	-	是
		应当合理限制用户可打开文件数量	-	-	-	否
		确保链接文件保护配置正确	√	-	-	是
		确保dmesg访问权限配置正确	√	√	-	否
		确保内核符号地址受限访问	√	√	-	是
		应当合理限制进程ptrace能力	-	-	-	否
		禁止全局加解密策略配置为LEGACY	-	-	-	是
系统服务	时间同步服务	应当正确配置ntpd服务	-	-	-	否
		应当正确配置chronyd服务	-	-	-	是
	定时任务服务	确保cron服务正常运行	√	-	-	是
		确保cron配置权限正确	√	√	-	否
	SSH服务	确保/etc/ssh/sshd_config权限配置正确	√	√	-	是
		确保SSH私钥文件权限配置正确	√	√	√	否
		确保SSH公钥文件权限配置正确	√	√	√	否
		确保启用IgnoreRhosts	√	-	-	是
		应当合理配置认证黑白名单	-	-	-	否
		确保SSH使能PAM认证	√	-	-	是
		禁止SSH root登录	-	-	√	否

		禁止SSH空口令登录	√	-	-	是
		禁止使用HostbasedAuthentication	√	-	-	是
		确保配置Warning Banner文件路径	√	√	-	否
		确保正确配置SSH日志级别	√	√	-	是
		应当配置SSH服务侦听IP	-	-	-	否
		应当正确配置SSH并发未认证连接数	√	-	-	否
		禁止使用X11Forwarding	√	√	-	否
		应当配置SSH MaxSessions不超过10	√	-	-	是
		应当正确配置MaxAuthTries	√	-	-	否
		禁止使用PermitUserEnvironment	√	-	-	是
		应当配置LoginGraceTime不超过60秒	√	√	-	否
		确保配置空闲超时间隔时间	√	√	-	否
		禁止使用AllowTcpForwarding	√	√	-	否
		确保SSH KexAlgorithms配置强算法	√	√	-	是
		确保SSH MACs配置强算法	√	√	-	是
		确保SSH Ciphers配置强算法	√	√	-	是
		禁止配置SSH将弃用的选项	√	-	-	是
网络服务	禁用不使用的网络协议和设备	避免使用不常见网络协议	-	-	-	否

		避免使用无线网络	-	-	-	是
	内核网络协议栈	禁止系统响应ICMP广播报文	√	√	-	是
		禁止接收ICMP重定向报文	√	√	-	否
		禁止转发ICMP重定向报文	√	-	-	是
		应当忽略所有ICMP请求	-	-	-	否
		确保忽略伪造的ICMP报文	√	-	-	是
		确保启用反向地址过滤	√	√	-	否
		禁止IP转发	√	√	-	是
		禁止接收源路由报文	√	√	-	否
		确保启用TCP-SYN cookie保护	√	√	-	是
		应当启用日志记录可疑的网络包	√	-	-	否
		避免启用tcp_timestamps	-	-	-	否
		确保TIME_WAIT TCP协议等待时间已配置	√	-	-	是
		应当合理配置SYN_RECV状态队列数量	-	-	-	否
		禁止使用ARP代理	-	-	-	是
防火墙配置	配置firewalld服务	应当启用firewalld服务	-	-	-	是
		确保iptables未启用	-	-	-	是
		确保nftables未启用	-	-	-	是
		应当配置正确的默认区域	-	-	-	否
		应当确保网络接口绑定正确区域	-	-	-	否
		避免开启不必要的服务和端口	-	-	-	否

	配置iptables服务	应当启用iptables服务	-	-	-	否
		确保firewalld未启用	-	-	-	否
		确保nftables未启用	-	-	-	是
		应当正确配置iptables默认拒绝策略	-	-	-	否
		应当正确配置iptables loopback策略	-	-	-	否
		应当正确配置iptables INPUT策略	-	-	-	否
		应当正确配置iptables OUTPUT策略	-	-	-	否
		应当正确配置iptables INPUT、OUTPUT关联策略	-	-	-	否
	配置nftables服务	应当启用nftables服务	-	-	-	否
		确保iptables未启用	-	-	-	是
		确保firewealld未启用	-	-	-	否
		应当配置nftables默认拒绝策略	-	-	-	否
		应当配置nftables loopback策略	-	-	-	否
		应当正确配置nftables input策略	-	-	-	否
		应当正确配置nftables output策略	-	-	-	否
		应当正确配置nftables input、output关联策略	-	-	-	否
日志审计	auditd	确保auditd审计已启用	√	-	-	是
		应当在启动阶段启用auditd	-	-	-	否
		应当正确配置audit_backlog_limit	-	-	-	否

		确保配置单个日志大小限制	-	-	-	是
		确保审计日志rotate已启用	-	-	-	否
		确保审计日志不被自动删除	-	-	-	是
		应当合理配置磁盘空间阈值	-	-	-	是
		避免配置审计日志限速阈值过小	-	-	-	是
		应当配置sudoers审计规则	-	√	√	否
		应当配置登录审计规则	-	-	-	是
		应当配置会话审计规则	-	-	-	是
		应当配置时间修改审计规则	-	√	√	否
		应当配置SELinux审计规则	-	-	-	否
		应当配置网络环境审计规则	-	-	√	否
		应当配置文件访问控制权限审计规则	-	-	-	否
		应当配置文件访问失败审计规则	-	-	-	否
		应当配置文件删除审计规则	-	-	-	否
		应当配置账号信息修改审计规则	-	√	√	否
		应当配置文件系统挂载审计规则	-	-	-	否
		应当配置提权命令审计规则	-	-	-	否
		应当配置内核模块变更审计规则	-	-	-	是
		应当配置修改sudo日志文件审计规则	-	-	-	否

	rsyslog	确保rsyslog服务已启用	√	√	√	否
		确保系统认证相关事件日志已记录	-	-	-	是
		确保cron服务日志已记录	√	-	-	是
		应当正确配置各服务日志记录	-	-	-	是
		应当正确配置rsyslog默认文件权限	√	√	√	否
		确保rsyslog日志rotate已配置	-	-	-	否
		应当配置发送日志到远程日志服务器	-	-	-	否
		应当仅在指定的日志主机上接收远程rsyslog消息	-	-	-	否
		确保rsyslog转储journald日志已配置	-	-	-	否
账号与口令管理	账号管理	禁止无需登录的账号拥有登录能力	-	-	-	否
		禁止存在不使用的账号	-	-	-	否
		应当正确设置账号有效期	-	-	-	否
		禁止存在UID为0的非root账号	-	-	-	是
		确保UID唯一	-	-	-	是
		确保GID唯一	-	-	-	是
		确保账号名唯一	-	-	-	是
		确保组名唯一	-	-	-	是
		确保/etc/passwd中的组都存在	-	-	-	是
		确保账号拥有自己的Home目录	-	-	-	是
		确保账号Home目录权限是750或更严格	-	-	-	是

		避免账号Home目录下存在.forward文件	-	-	-	是
		避免账号Home目录下存在.netrc文件	-	-	-	是
		确保用户PATH变量被严格定义	-	-	-	是
	口令管理	确保配置口令复杂度	√	√	√	否
		确保限制重用历史口令次数	√	√	√	否
		确保口令中不包含账号字符串	-	-	-	是
		确保口令使用SHA512算法加密	√	√	√	否
		确保口令过期时间设置正确	√	√	√	否
		确保口令过期告警时间设置正确	√	√	-	是
		应当设置口令修改周期设置正确	√	√	√	否
		确保不活跃口令锁定时间不超过30天	√	-	-	是
		确保Grub已设置口令保护	-	-	-	是
		确保单用户模式已设置口令保护	-	-	-	是
身份认证	登录管理	确保登录失败一定次数后锁定账号	√	√	√	否
		避免root用户本地接入系统	-	-	-	否
		确保会话超时时间设置正确	√	√	√	否
	确保Warning Banner包含合理的信息	确保本地登录Warning Banner包含合理的信息	√	√	-	否
		确保远程登录Warning Banner包含合理的信息	√	√	-	否
		确保motd文件包含合理的信息	√	√	-	否

		确保/etc/issue权限配置正确	√	√	-	是
		确保/etc/issue.net权限配置正确	√	√	-	是
		确保/etc/motd权限配置正确	√	√	-	是
访问控制	SELinux	应当启用enforce模式	-	-	-	是
		应当正确配置SELinux策略	-	-	-	是
		避免标签为unconfined_service_t的服务存在	-	-	-	否
		确保SETroubleshoot服务未安装	-	-	-	是
		确保MCS转换服务未安装	-	-	-	是
	特权命令	确保su受限使用	√	√	√	否
		确保su命令继承用户环境变量不会引入提权	√	√	√	否
		确保普通用户通过sudo运行特权程序	-	-	-	否
		确保配置sudo日志文件	√	-	-	否
		禁止使用SysRq键	-	√	-	否
	系统文件权限	确保/etc/passwd权限配置正确	√	-	-	是
		确保/etc/passwd-权限配置正确	√	-	-	是
		确保/etc/shadow权限配置正确	√	-	-	是
		确保/etc/shadow-权限配置正确	√	-	-	是
		确保/etc/group权限配置正确	√	-	-	是
		确保/etc/group-权限配置正确	√	-	-	是

		确保/etc/gshadow权限配置正确	√	-	-	是
		确保/etc/gshadow-权限配置正确	√	-	-	是
		确保全局可写目录已设置sticky位	-	-	-	是
		禁止存在无属主或属组的文件或目录	-	-	-	是
		禁止存在全局可写的文件	-	-	-	是
		禁止存在空链接文件	-	-	-	是
		禁止存在隐藏的可执行文件	-	-	-	是
		确保删除文件非必要的SUID和SGID位	-	-	-	是
		确保umask是027或更严格	√	√	√	否

说明

- “√”表示执行。
- “-”表示不执行。

9 内核功能与接口

9.1 内核 memory 的 OOM 进程控制策略

背景信息

现有操作系统中，支持配置离线业务和在线业务。当内存发生OOM时，会优先选择离线业务控制组中的消耗内存最多的进程，结束进程回收内存，但是对于某些离线业务也有核心业务，因此会造成很大的影响。

针对这个问题，HCEOS调整了OOM时回收内存的策略，增加了配置cgroup优先级的功能。 内存紧张情况下内核会遍历cgroup，对低优先级的cgroup结束进程，并回收内存，使离线业务中重要的业务可以存活下来。

前提条件

vm.panic_on_oom接口默认开启，系统OOM时panic。故使用memcg OOM优先级配置时（即memcg_qos_enable配置为1或2），须先执行**sysctl -w vm.panic_on_oom=0**命令，关闭系统参数vm.panic_on_oom。

memcg OOM 优先级接口功能说明

接口	说明	取值
memcg_qos_enable	memcg OOM优先级策略开关。 ● 0：不开启优先级配置。当OOM时，按照系统原有的OOM操作结束进程，结束内存消耗最大的进程，回收内存。 ● 1：开启优先级配置并以cgroup为粒度。当OOM时，结束优先级低的cgroup所有进程，并回收内存。 ● 2：开启优先级配置并以单个进程个为粒度。当OOM时，结束优先级低的cgroup中的最大的一个进程，并回收内存。	整数形式，取值范围为0~2，默认值为0。

接口	说明	取值
memory.qos_level	配置cgroup组优先级。值越小cgroup组优先级越低。 ● 当memcg OOM时，会以当前cgroup组为父节点，查找子节点优先级最低的cgroup组中内存使用最大的进程，结束该进程，回收内存。 ● 当OOM时，对于优先级相等的cgroup组，会根据组的内存使用量进行二次排序，选择内存使用最大的进行OOM操作。 **说明** ● 使用memory.qos_level的前提条件为memcg_qos_enable取值须为1或2。 ● 新创建的cgroup组的memory.qos_level值默认会继承父节点的memory.qos_level的值，但是子节点的优先级不受父节点的限制。 ● 如果修改cgroup组父节点的优先级，子节点的优先级会自动调整，和父节点保持一致。	整数形式，取值范围为-1024~1023，默认值为0。

接口配置示例

按如下所示创建6个cgroup子节点A、B、C、D、E、F，配置memcg_qos_enable接口，并通过memory.qos_level接口设置OOM的优先级，优先级取值如图所示。

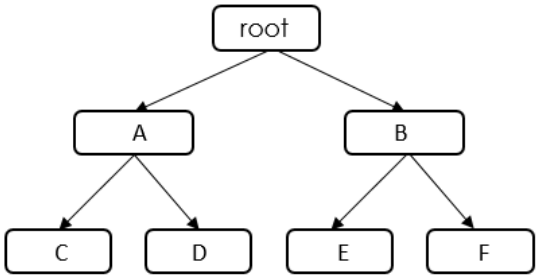


表 9-1 数据规划

cgroup组	memory.qos_level取值	说明
A	**-8**	当在root中进行OOM操作时，内核遍历root所有cgroup组，最终选择优先级最低的A、E。由于A和E优先级相同，内核继续对A和E使用的内存进行比较。 ● 如果设置memcg_qos_enable=1，优先对A/E中内存使用量大的一个cgroup组回收内存，结束cgroup组中的所有进程，并回收内存。 ● 如果设置memcg_qos_enable=2，优先对A/E中内存使用量最大的一个进程回收内存。
B	10	
C	1	
D	2	
E	**-8**	
F	3	

1. 关闭系统参数vm.panic_on_oom。
```
sysctl -w vm.panic_on_oom=0
```
2. 开启memcg OOM优先级策略功能。
```
echo 1 > /proc/sys/vm/memcg_qos_enable
```
3. 运行以下命令创建两个cgroup节点 A、B，并分别设置A、B节点的memcg OOM优先级值为-8、10。
```
mkdir /sys/fs/cgroup/memory/A
mkdir /sys/fs/cgroup/memory/B
cd /sys/fs/cgroup/memory/A
echo -8 > memory.qos_level
cd /sys/fs/cgroup/memory/B
echo 10 > memory.qos_level
```
4. 运行以下命令分别在A节点下创建C、D子节点，在B节点下创建E、F子节点， 并分别设置C、D、E、F子节点的memcg OOM优先级值为1、2、-8、3。
```
mkdir /sys/fs/cgroup/memory/A/C
mkdir /sys/fs/cgroup/memory/A/D
mkdir /sys/fs/cgroup/memory/B/E
mkdir /sys/fs/cgroup/memory/B/F
cd /sys/fs/cgroup/memory/A/C
echo 1 > memory.qos_level
cd /sys/fs/cgroup/memory/A/D
echo 2 > memory.qos_level
cd /sys/fs/cgroup/memory/B/E
echo -8 > memory.qos_level
cd /sys/fs/cgroup/memory/B/F
echo 3 > memory.qos_level
```

9.2 内核 memory 的多级内存回收策略

需求背景

在容器高密度混合部署场景中，IO读写较多的离线业务消耗大量page cache，导致系统空闲内存降低，达到全局空闲内存水位线后触发全局内存回收，使得在线任务申请内存时进入内存回收的慢路径，引发时延抖动。

为解决此问题，HCEOS 2.0新增支持多级内存回收策略。申请内存时，设置内存警示值，可触发内存回收任务，保证可用内存空间；回收内存时，设置多级内存保护水位线，保护任务可用性。

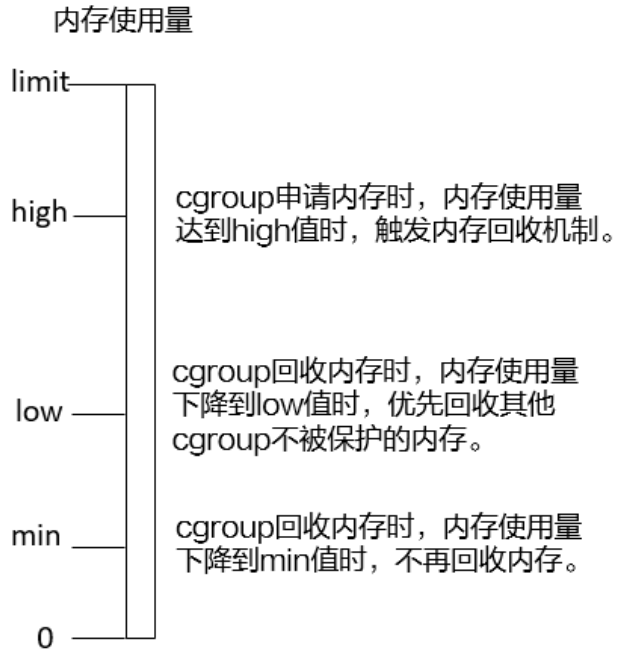
约束与限制

memory.min和memory.low只在叶子节点生效。创建memory cgroup时，会将父节点的memory.min和memory.low清零。

多级内存回收接口说明

memory.min、memory.low和memory.high接口在非根的memory cgroup下面默认存在，可以向文件内写值配置，也可以读取当前配置。合理的取值大小顺序为memory.min≤memory.low<memory.high，三者可独立使用，也可联合使用。

内存回收机制如下图。



接口	说明
memory.min	硬保护内存保护值，默认值为0。系统没有可回收内存的时候，也不会回收在该值边界及以下的内存。读写说明如下： ● 读该接口可以查看硬保护内存大小，单位为byte。 ● 写该接口可以设置硬保护内存大小，单位不做限制。 ● 配置范围：0-memory.limit_in_bytes。

接口	说明
memory.low	尽力而为的内存保护值，默认值为0。 系统优先回收未被保护的cgroup组的内存。如果内存还不足，再回收memory.min到memory.low之间的内存。 读写说明如下： ● 读该接口可以查看Best-effort内存保护值，单位为byte。 ● 写该接口可以设置Best-effort内存保护值，单位不做限制。 ● 配置范围：0-memory.limit_in_bytes。
memory.high	内存回收警示值，默认为max。当cgroup组内存使用量达到high值，会触发对该cgroup及子节点的同步内存回收任务，将内存尽量限制在high之下，避免触发memory.limit限制导致OOM。读写说明如下： ● 读该接口可以查看Throttle limit内存保护值，单位为byte。 ● 写该接口可以设置Throttle limit内存保护值，单位不做限制。 ● 配置范围：0-memory.limit_in_bytes

接口示例

按如下所示创建6个cgroup节点A、B、C、D、E、F，配置memory.min接口，示例说明多级内存回收策略。

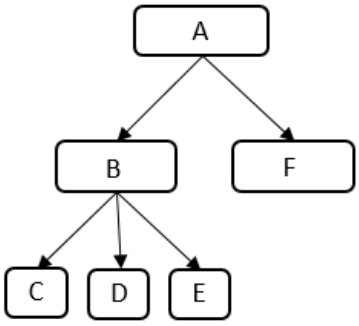


表 9-2 数据规划

cgroup组	memory.limit_in_bytes	memory.min	memory.usage_in_bytes
A	200M	0	-
B	-	0	-
C	-	75M	50M
D	-	25M	50M
E	-	0	50M
F	-	125M	-

1. 创建cgroup节点A，并配置memory.limit_in_bytes=200M。
```
mkdir /sys/fs/cgroup/memory/A
echo 200M > /sys/fs/cgroup/memory/A/memory.limit_in_bytes
```
2. 创建cgroup节点B。
```
mkdir /sys/fs/cgroup/memory/A/B
```
3. 创建cgroup节点C，并配置memory.min=75M，在当前节点创建申请50M cache的进程。
```
mkdir /sys/fs/cgroup/memory/A/B/C
echo 75M > /sys/fs/cgroup/memory/A/B/C/memory.min
```
4. 创建cgroup节点D，并配置memory.min=25M，在当前节点创建申请50M cache的进程。
```
mkdir /sys/fs/cgroup/memory/A/B/D
echo 25M > /sys/fs/cgroup/memory/A/B/D/memory.min
```
5. 创建cgroup节点E，并配置memory.min=0，在当前节点创建申请50M cache的进程。
```
mkdir /sys/fs/cgroup/memory/A/B/E
```
6. 创建cgroup节点F，并配置memory.min=125M，申请125M保护内存，触发内存回收。
```
mkdir /sys/fs/cgroup/memory/A/F
echo 125M > /sys/fs/cgroup/memory/A/F/memory.min
```

返回结果：

C节点memory.min=75M，memory.usage_in_bytes=50M。

D节点memory.min=25M，memory.usage_in_bytes=25M。

E节点memory.min=0，memory.usage_in_bytes=0。

B节点memory.usage_in_bytes=75M。

9.3 内核 cpu cgroup 的多级混部调度

需求背景

在业务混部场景中，Linux内核调度器需要为高优先级任务赋予更多的调度机会，并需要把低优先级任务对内核调度带来的影响降到最低。原有的在线、离线两级混部调度无法满足业务需求。

为解决此问题，HCEOS 2.0内核cpu cgroup支持多级混部调度，提供cgroup接口/sys/fs/cgroup/cpu/cpu.qos_level将任务调度级别扩展到5个级别，支持用户对每个cgroup组单独设置优先级。

约束与限制

内核cpu cgroup的多级混部调度基于5.10.0-60.18.0.50.r692_16.hce2.x86_64内核版本开发，当前cpu.qos_level仅支持cgroup-v1， 不支持cgroup-v2。

多级混部调度接口说明

cpu.qos_level的生效规则：

- CFS调度器自上而下逐层选择task_group，同一个父节点内的子节点之间cpu.qos_level生效。

- 子cgroup创建时默认继承父cgroup的cpu.qos_level，支持重新配置cpu.qos_level值。
- 同优先级的qos_level之间的资源竞争服从CFS调度器的策略。
- 同一个cpu上，qos_level < 0 的任务始终会被qos_level >= 0的任务无条件抢占，不受层级约束。

在调度高优先级任务时：

- 在线任务可无条件抢占离线任务，在多核调度时，在线任务可优先抢占其他核上的离线任务。超线程（Hyper Thread）场景，优先级为2的在线任务可驱逐SMT上的离线任务。
- 高优先级的任务被唤醒时获得一定的时间片加速，可立刻抢占低优先级的任务（忽略CFS的最小运行时间片），获得更好的低时延响应。

表 9-3 cpu.qos_level 接口说明

接口	说明
cpu.qos_level	配置cgroup的cpu优先级。取值类型为整数形式，取值范围 为[-2, 2]，默认值为0。 ● cpu.qos_level >= 0 标识cgroup组内任务为在线任务，在线任务可无条件抢占离线任务。 优先级 0 < 1 < 2，同为在线业务的任务，高优先级的在线相比低优先级的在线可获取更多的CPU资源抢占机会。 ● cpu.qos_level < 0 标识cgroup组内的任务为离线任务，-1优先级高于-2。 -1级别的任务比-2级别的任务可获得更多的CPU资源抢占机会。 父节点为离线任务时，子节点只能继承父节点的优先级，不支持修改为其他优先级。

接口示例

按如下所示创建3个cgroup节点A、B、C，配置并查看qos_level接口。

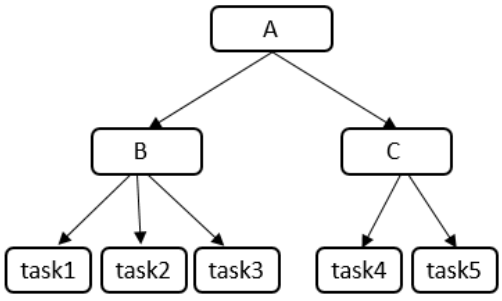


表 9-4 数据规划

cgroup组	cpu.qos_level
A	1
B	-2
C	2

1. 创建cgroup A及子节点B、C，依次设置A、B、C的cpu调度优先级为1、-2、2。

   cgroup A和cgroup C中的任务可无条件抢占cgroup B任务的CPU资源，cgroup C优先级大于cgroup A。

   ```
   mkdir -p /sys/fs/cgroup/cpu/A
   echo 1 > /sys/fs/cgroup/cpu/A/cpu.qos_level
   mkdir -p /sys/fs/cgroup/cpu/A/B
   echo -2 > /sys/fs/cgroup/cpu/A/B/cpu.qos_level
   mkdir -p /sys/fs/cgroup/cpu/A/C
   echo 2 > /sys/fs/cgroup/cpu/A/C/cpu.qos_level
   ```

2. 将task1、task2、task3进程加入cgroup B。

   task1、task2、task3进程加入cgroup B后，task1、task2、task3进程的cpu调度优先级为-2。

   ```
   echo $PID1 > /sys/fs/cgroup/cpu/A/B/tasks
   echo $PID2 > /sys/fs/cgroup/cpu/A/B/tasks
   echo $PID3 > /sys/fs/cgroup/cpu/A/B/tasks
   ```

3. 将task4、task5进程加入cgroup C。

   task4、task5进程加入cgroup C后，task4、task5进程的cpu调度优先级为2。

   ```
   echo $PID4 > /sys/fs/cgroup/cpu/A/C/tasks
   echo $PID5 > /sys/fs/cgroup/cpu/A/C/tasks
   ```

4. 查看cgroup B的cpu调度优先级及进程。

   ```
   [root@localhost cpu_qos]# cat /sys/fs/cgroup/cpu/A/B/cpu.qos_level
   -2
   [root@localhost boot]# cat /sys/fs/cgroup/cpu/A/B/tasks
   1879
   1880
   1881
   ```

5. 查看cgroup C的cpu调度优先级及进程。

   ```
   [root@localhost cpu_qos]# cat /sys/fs/cgroup/cpu/A/C/cpu.qos_level
   2
   [root@localhost boot]# cat /sys/fs/cgroup/cpu/A/C/tasks
   1882
   1883
   ```

9.4 内核异常事件分析指南

背景说明

HCEOS运行时，不可避免地会出现一些内核事件，例如**soft lockup**、**RCU(Read-Copy Update) stall**、**hung task**、**global OOM**、**cgroup OOM**、**page allocation failure**、**list corruption**、**Bad mm_struct**、**I/O error**、**EXT4-fs error**、**MCE (Machine Check Exception)**、**fatal signal**、**warning**、**panic**。本节为您介绍这些内核事件的原理及触发方法。

soft lockup

soft lockup是内核检测CPU在一定时间内（默认20秒）没有发生调度切换时，上报的异常。

- 原理

  soft lockup利用Linux内核watchdog机制触发。内核会为每一个CPU启动一个优先级最高的FIFO实时内核线程watchdog，名称为watchdog/0、watchdog/1以此类推。这个线程会定期调用watchdog函数，默认每4秒执行一次。同时调用过后会重置一个hrtimer定时器在2倍的watchdog_thresh时间后到期。watchdog_thresh是内核参数，对应默认超时时间为20秒。

  在超时时间内，如果内核线程watchdog没被调度，hrtimer定时器到期，即触发内核打印类似如下的soft lockup异常。

  ```
  BUG: soft lockup - CPU#3 stuck for 23s! [kworker/3:0:32]
  ```

- 触发方法

  关闭中断或关闭抢占，软件执行死循环。

RCU(Read-Copy Update) stall

RCU stall是一种rcu宽限期内rcu相关内核线程没有得到调度的异常。

- 原理

  在RCU机制中，reader不用等待，可以任意读取数据，RCU记录reader的信息；writer更新数据时，先复制一份副本，在副本上完成修改，等待所有reader退出后，再一次性地替换旧数据。

  writer需要等所有reader都停止引用“旧数据”才能替换旧数据。这相当于给了这些reader一个优雅退出的宽限期，这个等待的时间被称为grace-period，简称GP。

  当reader长时间没有退出，writer等待的时间超过宽限期时，即上报RCU Stall。

- 触发方法

  内核在Documentation/RCU/stallwarn.txt文档列出了可能触发RCU stall的场景：cpu在rcu reader临界区一直循环，cpu在关闭中断或关闭抢占场景中一直循环等。

hung task

当内核检测到进程处于D状态超过设定的时间时，上报hung task异常。

- 原理

  进程其中一个状态是TASK_UNINTERRUPTIBLE，也叫D状态，处于D状态的进程只能被wake_up唤醒。内核引入D状态时，是为了让进程等待IO完成。正常情况下，IO正常处理，进程不应该长期处于D状态。

  hung task检测进程长期处于D状态的原理，内核会创建一个线程khungtaskd，用来定期遍历系统中的所有进程，检查是否存在处于D状态超过设置时长（默认120秒）的进程。如果存在这样的进程，则打印并上报相关警告和进程堆栈。如果配置了hung_task_panic（通过proc或内核启动参数配置），则直接发起panic。

- 触发方法

  创建内核线程，设成D状态，scheduler释放时间片。

global OOM

Linux的OOM killer特性是一种内存管理机制，在系统可用内存较少的情况下，内核为保证系统还能够继续运行下去，会选择结束一些进程释放掉一些内存。

- 原理

  通常oom_killer的触发流程是：内核为某个进程分配内存，当发现当前物理内存不够时，触发OOM。OOM killer遍历当前所有进程，根据进程的内存使用情况进行打分，然后从中选择一个分数最高的进程，终止进程释放内存。

  OOM killer的处理主要集中在mm/oom_kill.c，核心函数为out_of_memory，函数处理流程为：

  a. 通知系统中注册了oom_notify_list的模块释放一些内存，如果从这些模块中释放出了一些内存，直接结束oom killer流程；如果回收失败，进入下一步。
  b. 触发oom killer通常是由当前进程进行内存分配所引起。如果当前进程已经挂起了一个SIG_KILL信号或者正在退出，直接选中当前进程，终止进程释放内存；否则进入下一步。
  c. 检查panic_on_oom系统管理员的设置，决定OOM时是进行oom killer还是panic。如果选择panic，则系统崩溃并重启；如果选择oom killer，进入下一步。
  d. 进入oom killer，检查系统设置，系统管理员可设置终止当前尝试分配内存、引起OOM的进程或其它进程。如果选择终止当前进程，oom killer结束；否则进入下一步。
  e. 调用select_bad_process选中合适进程，然后调用oom_kill_process终止选中的进程。如果select_bad_process没有选出任何进程，内核进入panic。

- 触发方法

  执行占用大内存的程序，直到内存不足。

cgroup OOM

- 与global OOM的区别

  cgroup OOM与global OOM的内存范围不同。当FS cgroup组内进程使用内存超过了设置的上限，cgroup通过KILL相应的进程来释放内存。

- 触发方法

  执行占用大内存的程序，直到内存不足。

page allocation failure

page allocation failure是申请空闲页失败时，系统上报的错误。当程序申请某个阶数（order）的内存，但系统内存中，没有比申请阶数高的空闲页，即触发内核报错。

- 原理

  Linux使用伙伴系统（buddy system）内存分配算法。将所有的空闲页表（一个页表的大小为4K）分别链接到包含了11个元素的数组中，数组中的每个元素将大小相同的连续页表组成一个链表，页表的数量为1、2、4、8、16、32、64、128、256、512、1024，所以一次性可以分配的最大连续内存为1024个连续的4k页表，即4MB的内存。

  假设申请一个包括256个页表的内存，指定阶数order为6，系统会依次查找数组中的第9、10、11个链表，上一个为空，表示没有此阶数的空闲内存，查找下一个，直到最后一个链表。

如果所有链表均为空，申请失败，则内核上报错误page allocation failure。输出报错信息，描述申请阶数为6的内存页失败：

```
page allocation failure:order:6
```

- 触发方法

  用alloc_pages连续申请高阶数内存页（例如order=10），不释放，直到申请失败。

list corruption

list corruption是内核检查链表有效性失败的报错，报错类型分为list_add corruption和list_del corruption。

- 原理

  内核提供list_add和list_del，对传入的链表先检查链表的有效性（valid），检查通过后，修改链表增加或删除节点。如果检查链表失败，则上报corruption错误。检查和报错代码在内核lib/list_debug.c。

```
bool __list_add_valid(struct list_head *new, struct list_head *prev,
              struct list_head *next)
{
    if (CHECK_DATA_CORRUPTION(next->prev != prev,
            "list_add corruption. next->prev should be prev (%px), but was %px. (next=%px).\n",
            prev, next->prev, next) ||
        CHECK_DATA_CORRUPTION(prev->next != next,
            "list_add corruption. prev->next should be next (%px), but was %px. (prev=%px).\n",
            next, prev->next, prev) ||
        CHECK_DATA_CORRUPTION(new == prev || new == next,
            "list_add double add: new=%px, prev=%px, next=%px.\n",
            new, prev, next))
        return false;

    return true;
}
EXPORT_SYMBOL(__list_add_valid);

bool __list_del_entry_valid(struct list_head *entry)
{
    struct list_head *prev, *next;

    prev = entry->prev;
    next = entry->next;

    if (CHECK_DATA_CORRUPTION(next == LIST_POISON1,
            "list_del corruption, %px->next is LIST_POISON1 (%px)\n",
            entry, LIST_POISON1) ||
        CHECK_DATA_CORRUPTION(prev == LIST_POISON2,
            "list_del corruption, %px->prev is LIST_POISON2 (%px)\n",
            entry, LIST_POISON2) ||
        CHECK_DATA_CORRUPTION(prev->next != entry,
            "list_del corruption. prev->next should be %px, but was %px\n",
            entry, prev->next) ||
        CHECK_DATA_CORRUPTION(next->prev != entry,
            "list_del corruption. next->prev should be %px, but was %px\n",
            entry, next->prev))
        return false;

    return true;

}
EXPORT_SYMBOL(__list_del_entry_valid);
```

  这种错误通常为内存异常操作导致，例如内存踩踏、内存损坏等。

- 触发方法

  用list.h的内核标准接口创建链表，非法修改链表节点的prev或next指针，再调用内核list_add/list_del接口。

Bad mm_struct

Bad mm_struct错误通常是由于内核中的一个或多个mm_struct数据结构被破坏或损坏所导致。

- 原理

  mm_struct是Linux内核中的一个重要数据结构，用于跟踪进程的虚拟内存区域。如果该数据结构被破坏，可能会导致进程崩溃或系统崩溃。这种错误通常内存异常导致，例如mm_struct区域的内存被踩踏、内存越界等。

- 触发方法

  无人为触发方法，当硬件错误，或者Linux系统内核代码错误时会触发Bad mm_struct。

I/O error

Linux I/O error报错通常表示输入/输出操作失败，在网卡、磁盘等IO设备驱动异常，或文件系统异常都可能打印这个错误。

- 原理

  错误原因取决于代码执行失败的条件。常见的触发异常的原因是硬件故障、磁盘损坏、文件系统错误、驱动程序问题、权限问题等。例如当系统尝试读取或写入磁盘上的数据时，如果发生错误，就会出现I/O错误。

- 触发方法

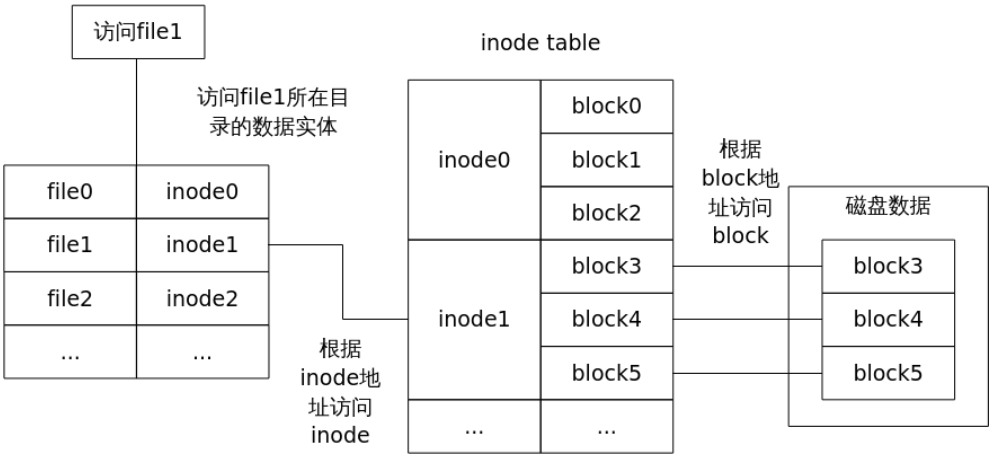
  系统读写磁盘过程，拔出磁盘，导致磁盘数据损坏。

EXT4-fs error

EXT4-fs error是由于ext4格式的文件系统中，文件节点的错误导致。

- 原理

  文件储存的最小存储单位叫做“扇区”（sector），连续多个扇区组成“块”（block）。inode节点储存文件的元信息，包括文件的创建者、创建日期、大小、属性、实际存储的数据块（block number）。EXT4格式的inode信息校验失败会触发EXT4-fs error。



  内核ext4校验使用checksum校验inode信息，当出现分区表错误、磁盘硬件损坏时，内核返回-EIO错误码，系统上报EXT4-fs error checksum invalid错误。

- 触发方法

  使用磁盘过程中强行拔盘，重新接入读盘。

MCE (Machine Check Exception)

Machine Check Exception (MCE) 是CPU发现硬件错误时触发的异常（exception），上报中断号是18，异常的类型是abort。

- 原理

导致MCE的原因主要有：总线故障、内存ECC校验错、cache错误、TLB错误、内部时钟错误等。不仅硬件故障会引起MCE，不恰当的BIOS配置、firmware bug、软件bug也有可能引起MCE。

MCE中断上报，操作系统检查一组寄存器称为Machine-Check MSR，根据寄存器的错误码执行对应的处理函数（函数实现依赖不同的芯片架构实现）。

- 触发方法

  无人为触发方法，当总线故障、内存ECC校验错、cache错误、TLB错误、内部时钟错误等时会触发MCE。

fatal signal

fatal signal指信号处理方式不能被设置为忽略或执行自定义处理函数的信号类型，包括SIGKILL、SIGSTOP、SIGILL等。

- 原理

  Linux信号（signal）机制，用于系统中进程间通讯，是一种异步的通知机制。当一个信号发送给一个进程，而操作系统中断了进程正常的控制流程时，任何非原子操作都将被中断。

  如果SIG符合条件，即为fatal信号：

  ```
  #define sig_fatal(t, signr) \
      (!siginmask(signr, SIG_KERNEL_IGNORE_MASK|SIG_KERNEL_STOP_MASK) && \
  ```

- 触发方法

  用户态程序执行非法指令、kill -9杀进程。

warning

Warning是操作系统在运行时，检测到需要立即注意的内核问题（issue），而采取的上报动作，打印发生时的调用栈信息。上报后，系统继续运行。

- 原理

  Warning是通过调用WARN、WARN_ON、WARN_ON_ONCE等宏来触发的。

  导致Warning的原因有多种，需要根据调用栈回溯，找到调用Warning宏的具体原因。Warning宏并不会导致系统运行状态发生改变，也不提供处理Warning的指导。

- 触发方法

  根据系统调用构造Warning条件。

panic

Kernel panic是指操作系统在监测到内部的致命错误，并无法安全处理此错误时采取的动作。内核触发到某种异常情况，运行kernel_panic函数，并尽可能把异常发生时获取的全部信息打印出来。

- 原理

  导致异常的原因多种多样，通过异常打印的调用信息，找到调用kernel_panic的原因。常见的原因包括内核堆栈溢出、内核空间的除0异常、内存访问越界、内核陷入死锁等。

- 触发方法

内核态读0地址。

10 XGPU 共享技术

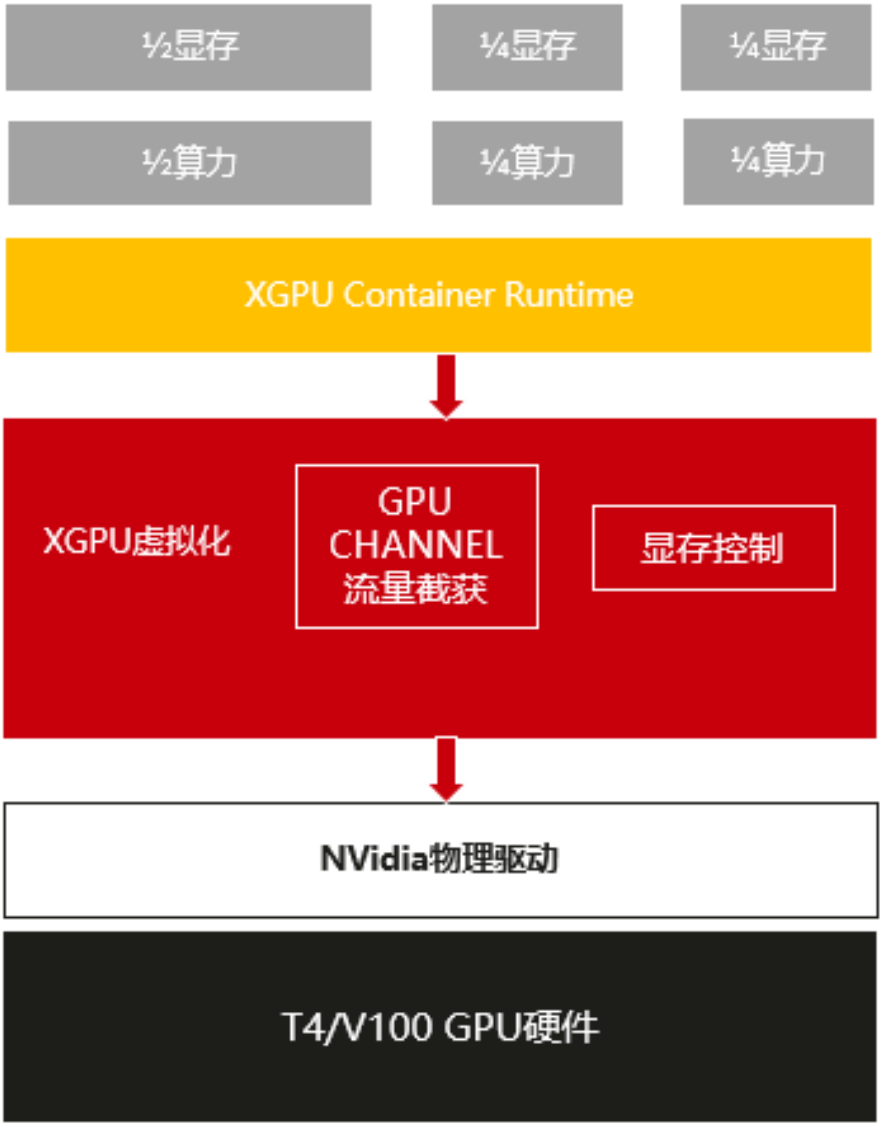
10.1 XGPU 共享技术概述

XGPU共享技术是基于内核虚拟GPU开发的共享技术。XGPU服务可以隔离GPU资源，实现多个容器共用一张显卡，从而实现业务的安全隔离，提高GPU硬件资源的利用率并降低使用成本。

XGPU 共享技术架构

XGPU通过内核驱动为容器提供虚拟的GPU设备，在保证性能的前提下隔离显存和算力，为充分利用GPU硬件资源进行训练和推理提供有效保障。您可以通过命令方便地配置容器内的虚拟GPU设备。

图 10-1 XGPU 共享技术架构图



产品优势

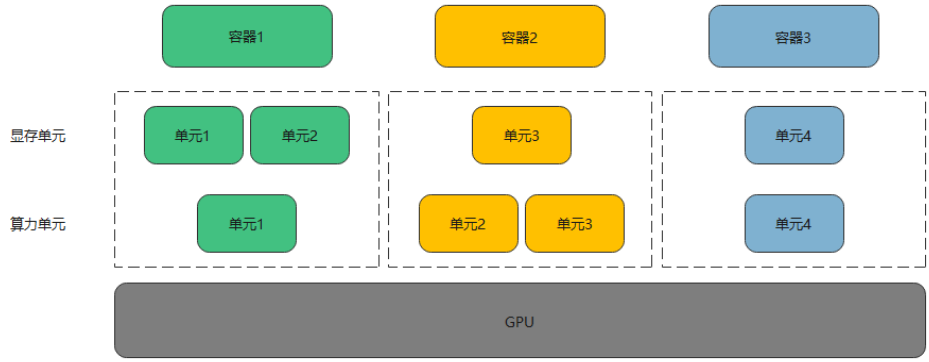
- 节约成本

  随着显卡技术的不断发展，单张GPU卡的算力越来越强，同时价格也越来越高。但在很多的业务场景下，一个AI应用并不需要一整张的GPU卡。XGPU的出现让多个容器共享一张GPU卡，从而实现业务的安全隔离，提升GPU利用率，节约用户成本。

- 可灵活分配资源

  XGPU实现了物理GPU的资源任意划分，您可以按照不同比例灵活配置。

  - 支持按照显存和算力两个维度划分，您可以根据需要灵活分配。



  - XGPU支持只隔离显存而不隔离算力的策略，同时也支持基于权重的算力分配策略。算力支持最小1%粒度的划分，推荐最小算力不低于4%。
- 兼容性好

  不仅适配标准的Docker和Containerd工作方式，而且兼容Kubernetes工作方式。
- 操作简单

  无需重编译AI应用，运行时无需替换CUDA库。

10.2 安装并使用 XGPU

本章节介绍如何安装和使用XGPU服务。

约束限制

- XGPU功能仅在Nvidia Tesla T4、V100上支持。
- HCEOS内核版本为5.10及以上版本。
- GPU实例已安装535.54.03版本的NVIDIA驱动。
- GPU实例已安装18.09.0-300或更高版本的docker。
- 受GPU虚拟化技术的限制，容器内应用程序初始化时，通过nvidia-smi监测工具监测到的实时算力可能超过容器可用的算力上限。
- 当CUDA应用程序创建时，会在GPU卡上申请一小部分UVM显存（在Nvidia Tesla T4上大约为3 MiB），这部分显存属于管理开销，不受XGPU服务管控。
- 暂不支持同时在裸机环境以及该环境直通卡的虚拟机中同时使用。
- XGPU服务的隔离功能不支持以UVM的方式申请显存，即调用CUDA API cudaMallocManaged()，更多信息，请参见**NVIDIA官方文档**。请使用其他方式申请显存，例如调用cudaMalloc()等。

  说明

  XGPU允许用户动态禁用UVM的方式申请显存，禁用方法参考uvm_disable接口说明。

安装 XGPU 服务

安装XGPU服务请联系客服。

XGPU 服务使用示例

影响XGPU服务的环境变量如下表所示，您可以在创建容器时指定环境变量的值。容器引擎可以通过XGPU服务获得算力和显存。

表 10-1 影响 XGPU 服务的环境变量

环境变量名称	取值类型	说明	示例
GPU_IDX	Integer	指定容器可使用的GPU显卡。	为容器分第一张显卡： GPU_IDX=0
GPU_CONTAINER_MEM	Integer	设置容器内可使用的显存大小，单位 MiB。	为容器分配的显存大小为5120MiB： GPU_CONTAINER_MEM=5120
GPU_CONTAINER_QUOTA_PERCENT	Integer	指定显卡算力分配百分比。 算力支持最小1%粒度的划分，推荐最小算力不低于4%。	为容器分配50%的算力比例： GPU_CONTAINER_QUOTA_PERCEN=50
GPU_POLICY	Integer	指定GPU使用的算力隔离的策略。 • 0：不隔离算力，即原生调度。 • 1：固定算力调度。 • 2：平均调度。 • 3：抢占调度。 • 4：权重抢占调度。 • 5：混合调度。 • 6：权重弱调度。 算力隔离策略示例详见**XGPU算力调度示例**。	设置算力隔离策略为固定算力调度： GPU_POLICY=1
GPU_CONTAINER_PRIORITY	Integer	指定容器的优先级。 • 0：低优先级 • 1：高优先级	创建高优先级容器： GPU_CONTAINER_PRIORITY=1

以nvidia的docker创建两个容器为例，介绍XGPU服务的使用方法，数据规划如下。

表 10-2 数据规划

参数	容器1	容器2	说明
GPU_IDX	0	0	指定两个容器使用第一张显卡。
GPU_CONTAINER_QUOTA_PERCENT	50	30	为容器1分配50%算力，为容器2分配30%算力。

参数	容器1	容器2	说明
GPU_CONTAINER_MEM	5120	1024	为容器1分配5120MiB显存，为容器2分配1024MiB显存。
GPU_POLICY	1	1	设置第一张显卡使用固定算力调度策略。
GPU_CONTAINER_PRIORITY	1	0	指定容器1为高优先级容器，容器2为低优先级容器。

配置示例：

```
docker run --rm -it --runtime=nvidia -e GPU_CONTAINER_QUOTA_PERCENT=50 -e GPU_CONTAINER_MEM=5120 -e GPU_IDX=0 -e GPU_POLICY=1 -e GPU_CONTAINER_PRIORITY=1 --shm-size 16g -v /mnt/:/mnt nvcr.io/nvidia/tensorrt:19.07-py3 bash
docker run --rm -it --runtime=nvidia -e GPU_CONTAINER_QUOTA_PERCENT=30 -e GPU_CONTAINER_MEM=1024 -e GPU_IDX=0 -e GPU_POLICY=1 -e GPU_CONTAINER_PRIORITY=0 --shm-size 16g -v /mnt/:/mnt nvcr.io/nvidia/tensorrt:19.07-py3 bash
```

查看 procfs 节点

XGPU服务运行时会在/proc/xgpu下生成并自动管理多个procfs节点，您可以通过procfs节点查看和配置XGPU服务相关的信息。下面介绍各procfs节点的用途。

1. 执行以下命令，查看节点信息。

   ```
   ls /proc/xgpu/
   0 container version uvm_disable
   ```

   目录内容说明如下表所示：

目录	读写类型	说明
0	读写	XGPU服务会针对GPU实例中的每张显卡生成一个的目录，并使用数字作为目录名称，例如0、1、2。本示例中只有一张显卡，对应的目录ID为0。
container	读写	XGPU服务会针对运行在GPU实例中的每个容器生成一个的目录。
version	只读	XGPU的版本。
uvm_disable	读写	是否禁用UVM的方式申请显存，全局粒度，默认值为0。 ● 0：不禁用 ● 1：禁用

2. 执行以下命令，查看第一张显卡对应的目录内容。

   ```
   ls /proc/xgpu/0/
   max_inst meminfo policy quota utilization_line utilization_rate xgpu1 xgpu2
   ```

   目录内容说明如下表所示：

目录	读写类型	说明
max_inst	读写	用于设置容器的最大数量，取值范围为1~25。仅在没有容器运行的情况下可修改。
meminfo	只读	此显卡总共可用的显存大小。
policy	读写	指定GPU使用的算力隔离的策略，默认值为1。 ● 0：不隔离算力，即原生调度。 ● 1：固定算力调度。 ● 2：平均调度。 ● 3：抢占调度。 ● 4：权重抢占调度。 ● 5：混合调度。 ● 6：权重弱调度。 算力隔离策略示例详见**XGPU算力调度示例**。
quota	只读	算力总权重。
utilization_line	读写	在离线混部的算力压制水位线。 当GPU整卡利用率超过该值时，在线容器完全压制离线容器，否则在线容器部分压制离线容器。
utilization_rate	只读	GPU整卡利用率。
xgpuIndex	读写	属于此显卡的xgpu子目录。 本示例中，属于第1张显卡的xgpu为xgpu1和xgpu2

3. 执行以下命令，查看container目录内容。

```
ls /proc/xgpu/container/
9418 9582
```

目录内容说明如下表所示：

目录	读写类型	说明
containerID	读写	容器的ID。 使用XGPU创建容器的时候分配的ID，每个容器对应一个ID。

4. 执行以下命令，查看containerID目录内容。

```
ls /proc/xgpu/container/9418/
xgpu1 uvm_disable
ls /proc/xgpu/container/9582/
xgpu2 uvm_disable
```

目录内容说明如下表所示：

目录	读写类型	说明
xgpuIndex	读写	属于此容器的xgpu子目录。 本示例中，属于容器9418的xgpu为xgpu1，属于容器9582的xgpu为xgpu2。
uvm_disable	读写	是否禁用UVM的方式申请显存，容器粒度，默认值为0。 ● 0：不禁用 ● 1：禁用

5. 执行以下命令，查看xgpuIndex目录内容。

```
ls /proc/xgpu/container/9418/xgpu1/
meminfo priority quota
```

目录内容说明如下表所示：

目录	读写类型	说明
meminfo	只读	此XGPU分配的可见显存大小和当前剩余可用显存大小。 如3308MiB/5120MiB, 64% free，指分配了5120MiB，剩余64%可使用。
priority	读写	用于设置容器的优先级，默认值为0。 ● 0：低优先级 ● 1：高优先级 该功能用于在线离线混合使用场景，高优先级容器可以抢占低优先级容器的算力。
quota	只读	此XGPU分配的算力百分比。 如50，指此XGPU分配了显卡50%的算力。

了解procfs节点的用途后，您可以在GPU实例中执行命令进行切换调度策略、查看权重等操作，示例命令如下表所示。

命令	效果
echo 1 > /proc/xgpu/0/policy	修改第一张显卡的调度策略为权重调度。
cat /proc/xgpu/container/$containerID/$xgpuIndex/meminfo	查看指定容器里xgpu分配的显存大小。
cat /proc/xgpu/container/$containerID/$xgpuIndex/quota	查看指定容器里xgpu分配的算力权重。
cat /proc/xgpu/0/quota	查看第一张显卡上剩余可用的算力权重。

命令	效果
echo 20 > /proc/xgpu/0/max_inst	设置第一张显卡最多可以创建20个容器。
echo 1 > /proc/xgpu/container/$containerID/$xgpuIndex/priority	设置指定容器里xgpu的优先级为高优先级。
echo 40 > /proc/xgpu/0/utilization_line	设置第一张显卡的在离线混部算力压制水位线为40%。
echo 1 > /proc/xgpu/container/$containerID/uvm_disable	设置指定容器里xgpu禁用UVM的方式申请显存

升级 XGPU 服务

XGPU服务采用冷升级的方式。

1. 关闭所有运行中的容器。
   docker ps -q | xargs -I {} docker stop {}
2. 升级xgpu的rpm包。
   rpm -U hce_xgpu

卸载 XGPU 服务

1. 关闭所有运行中的容器。
   docker ps -q | xargs -I {} docker stop {}
2. 卸载xgpu的rpm包。
   rpm -e hce_xgpu

通过 xgpu-smi 工具监控容器

您可以通过xgpu-smi工具查看XGPU容器的相关信息，包括容器ID、算力使用、分配情况、显存使用及分配情况等。

xgpu-smi的监控展示信息如下所示：

```
+=============================================================================+
|  HUAWEI CLOUD XGPU-SMI                        XGPU Version: 1.0             |
|=============================================================================|
|   Container-Id   |  GPU   |   GPU-Util/Limit   |   GPU-Memory-Usage/Limit   |
+------------------+--------+--------------------+----------------------------+
|     800a09ae74bc |      1 |           0% / 30% |                 0M / 2000M |
|     b8402c9316e4 |      0 |           0% / 10% |                 0M / 6000M |
+-----------------------------------------------------------------------------+
```

您可使用**xgpu-smi -h**命令查看xgpu-smi工具的帮助信息。

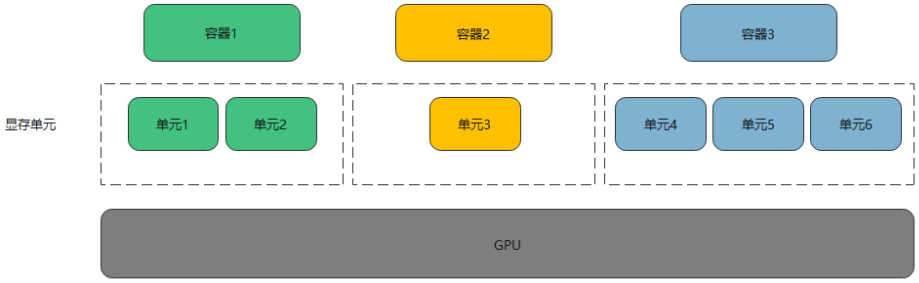
```
[root@localhost ~]#
[root@localhost ~]# xgpu-smi -h
xgpu-smi provides monitoring information about xgpu containers.
The data is presented in either a plain text or a json format, via stdout or a file.

Usage:
  xgpu-smi [flags]
  xgpu-smi [command]

Available Commands:
  clean       Release XGPU
  list        List XGPU containers and used GPUs
  query       Show XGPU containers' information and GPUs' available resource information in JSON format

Flags:
  -c, --container string                filter the specific containers' information
  -f, --filename string                 log to a specified file, rather than to stdout
  -g, --gpu string                      filter the specific GPUs' information
  -h, --help                            help for xgpu-smi
  -r, --remote-runtime-endpoint string   endpoint of remote runtime endpoint (default "unix:///var/run/containerd
/containerd.sock")

Use "xgpu-smi [command] --help" for more information about a command.
[root@localhost ~]#
[root@localhost ~]#
```

10.3 XGPU 算力调度示例

当使用XGPU服务创建XGPU时，XGPU服务会按照最大容器数量（**max_inst**）为每张显卡设置时间片（X ms）用于为容器分配GPU算力，以单元1、单元2…单元N表示。本节max_inst以20为例，介绍使用不同调度策略时对算力的调度示例。

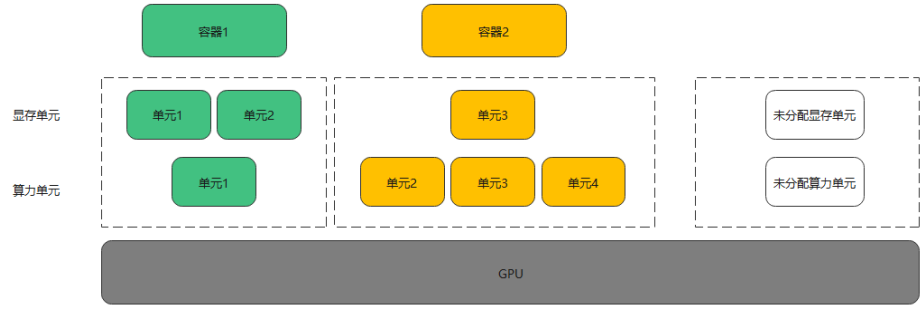
原生调度（policy=0）

原生调度表示使用NVIDIA GPU本身的算力调度方式。在原生调度策略下XGPU只用来做显存的隔离。



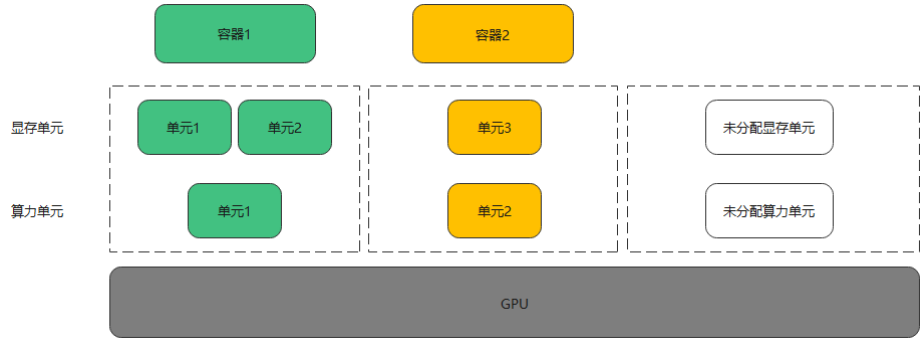
固定算力调度（policy=1）

固定算力调度表示以固定的算力百分比为容器分配算力。例如为容器1和容器2分别分配5%和15%的算力，如下图所示。



平均调度（policy=2）

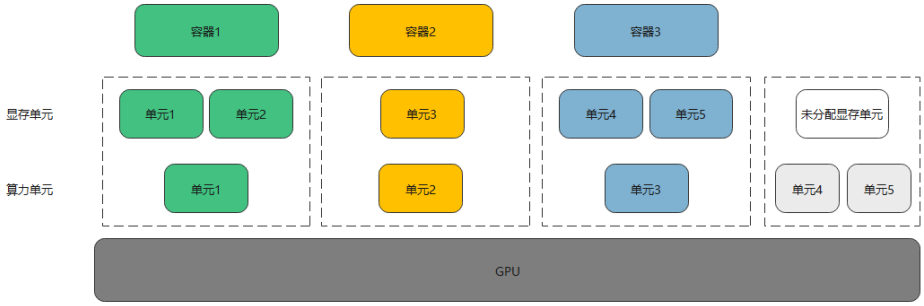
平均调度表示每个容器固定获得1/max_inst的算力。以max_inst=20为例，每个容器固定获得1/max_inst，即5%的算力，如下图所示。



抢占调度（policy=3）

抢占调度表示每个容器固定获得1个时间片，XGPU服务会从算力单元1开始调度。但如果某个算力单元没有分配给某个容器，或者容器内没有进程打开GPU设备，则跳过调度切换到下一个时间片。图中灰色部分的算力单元表示被跳过不参与调度。

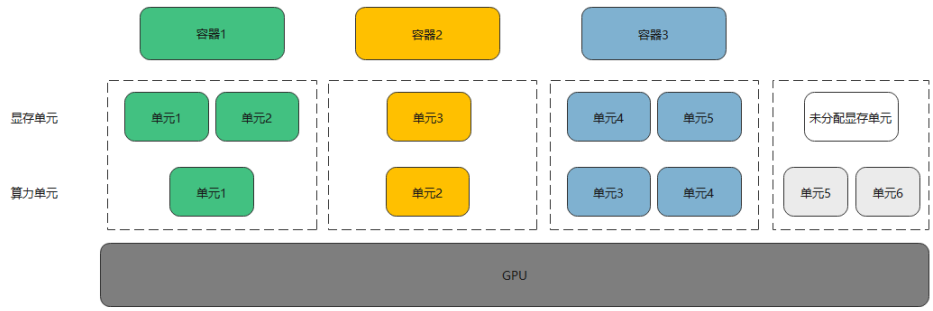
本例中容器1、2、3占用的实际算力百分比均为33.33%。



权重抢占调度（policy=4）

权重抢占调度表示按照每个容器的算力比例为容器分配时间片。XGPU服务会从算力单元1开始调度，但如果某个算力单元没有分配给某个容器，则跳过调度切换到下一个时间片。例如为容器1、2、3分别分配5%、5%、10%的算力，则容器1、2、3分别占用1、1、2个算力单元。图中灰色部分的算力单元表示被跳过不参与调度。

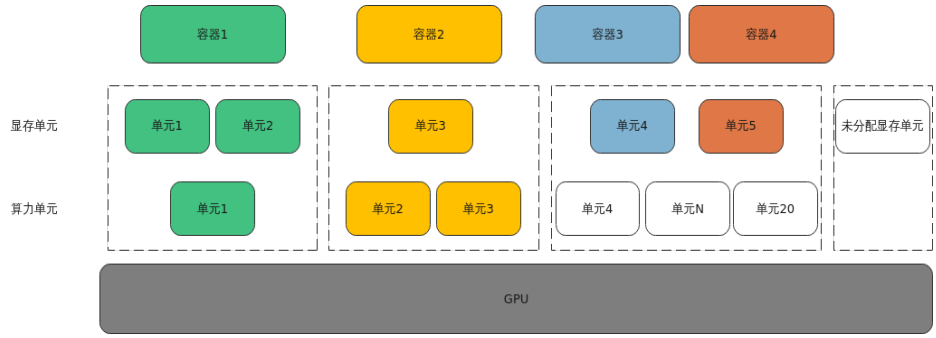
本例中容器1、2、3占用的实际算力百分比为25%、25%、50%。



混合调度（policy=5）

混合调度表示单张GPU卡支持单显存隔离和算力显存隔离类型。其中算力显存隔离的容器其隔离效果同固定算力（policy=1）完全一致，单显存隔离的容器共享算力显存隔离的容器分配后剩余的GPU算力。以max_inst=20为例，容器1、2为算力显存隔离容器，其分配的算力分别为5%、10%，容器3、4为单显存隔离的容器，则容器1、2分别占用1、2个算力单元，容器3、4共享剩余17个算力单元。此外，当容器2中没有进程打开GPU设备时，则容器1、2分别占用1、0个算力单元，容器3、4共享剩余19个算力单元。

在混合调度下，根据GPU_CONTAINER_QUOTA_PERCENT是否为0来区分容器是否开启算力隔离，GPU_CONTAINER_QUOTA_PERCENT为0的所有容器共享GPU的空闲算力。



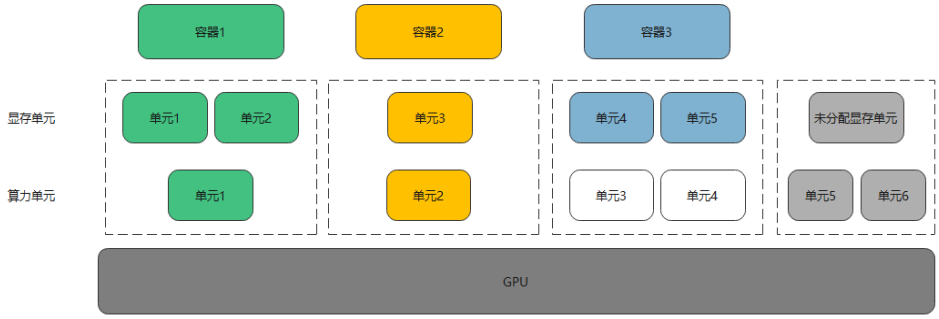
说明

混合调度策略不支持高优先级容器。

权重弱调度（policy=6）

权重弱调度表示按照每个容器的算力比例为容器分配时间片，隔离性弱于权重抢占调度。XGPU服务会从算力单元1开始调度，但如果某个算力单元没有分配给某个容器，或者容器内没有进程打开GPU设备，则跳过调度切换到下一个时间片。例如为容器1、2、3分别分配5%、5%、10%的算力，则容器1、2、3分别占用1、1、2个算力单元。图中白色部分的算力单元表示容器3的空闲算力，图中白色部分和灰色部分的算力单元表示被跳过不参与调度。

本例中容器1、2、3占用的实际算力百分比为50%、50%、0%。



说明

权重弱调度涉及空闲算力的抢占和抢回，因此容器在空闲和忙碌之间切换时会影响其他容器的算力，该算力波动属于正常情况。当某个容器从空闲切换到忙碌时，其抢回算力的时延不超过100ms。

11 HCE 搭建 REPO 服务器

将HCE提供的ISO发布包HCE-2.0-Enterprise-{arch}-dvd.iso创建为repo源，有如下两种方式，以下以x86_64架构ISO为例进行说明：

创建/更新本地 repo 源

使用mount挂载，将HCE的ISO发布包HCE-2.0-Enterprise-x86_64-dvd.iso创建为repo源，并能够对repo源进行更新。

获取ISO发布包

请参见**《华为云Stack 8.5.0 新建和扩容场景软件包下载列表》**中的《华为云Stack 8.5.0 IaaS软件包下载列表 04（DownloadCenter自动下载）》获取HCE的ISO发布包。

挂载ISO创建repo源

在root权限下使用mount命令挂载ISO发布包。示例如下：

```
# mount /home/HCE-2.0-Enterprise-x86_64-dvd.iso /mnt/
```

挂载好的mnt目录如下：

```
.
├── boot.catalog
├── docs
├── EFI
├── images
├── Packages
├── repodata
├── TRANS.TBL
└── RPM-GPG-KEY-HCE-2
```

其中，Packages为rpm包所在的目录，repodata为repo源元数据所在的目录，RPM-GPG-KEY-HCE-2为HCE的签名公钥。

创建本地repo源

可以拷贝ISO发布包中相关文件至本地目录以创建本地repo源，示例如下：

```
# mount /home/HCE-2.0-Enterprise-x86_64-dvd.iso /mnt/
# mkdir -p /home/hce/srv/repo/
# cp -r /mnt/Packages /home/hce/srv/repo/
# cp -r /mnt/repodata /home/hce/srv/repo/
# cp -r /mnt/RPM-GPG-KEY-HCE-2 /home/hce/srv/repo/
```

创建完成后本地repo目录如下：

```
.
├── Packages
├── repodata
└── RPM-GPG-KEY-HCE-2
```

Packages为rpm包所在的目录，repodata为repo源元数据所在的目录，RPM-GPG-KEY-HCE-2为HCE的签名公钥。

更新repo源

更新repo源有两种方式：

- 通过新版本的ISO更新已有的repo源，与创建repo源的方式相同，即挂载ISO发布包或重新拷贝ISO发布包至本地目录。
- 在repo源的Packages目录下添加rpm包，然后通过createrepo命令更新repo源。

  ```
  # createrepo --update --workers=10 ~/hce/srv/repo
  ```

  其中，--update表示更新，--workers表示线程数，可自定义。

  说明

  若命令打印信息为“createrepo：未找到命令”，则表示未安装createrepo软件，可在root权限下执行dnf install createrepo进行安装。

部署远端 repo 源

用户需首先准备好一台安装有HCE操作系统的ECS或其他机器，并且该机器与其他HCE机器网络互通。以下操作实现在HCE上通过nginx部署repo源。

nginx安装与配置

1. 请自行下载nginx工具并在root权限下安装nginx。
2. 安装nginx之后，在root权限下配置/etc/nginx/nginx.conf。

   说明

   文档中的配置内容仅供参考，请用户根据实际情况（例如安全加固需要）进行配置。

```
user  nginx;
worker_processes  auto;                          # 建议设置为core-1
error_log  /var/log/nginx/error.log  warn;       # log存放位置
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
}

http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;
    sendfile        on;
    keepalive_timeout  65;

    server {
        listen       80;
        server_name  localhost;                  # 服务器名（url）
        client_max_body_size 4G;
        root         /usr/share/nginx/repo;              # 服务默认目录
```

```
    location / {
        autoindex             on;           # 开启访问目录下层文件
        autoindex_exact_size on;
        autoindex_localtime   on;
    }

  }

}
```

启动nginx服务

1. 在root权限下通过systemctl命令启动nginx服务：

```
# systemctl enable nginx
# systemctl start nginx
```

2. nginx是否启动成功可通过下面命令查看：

```
# systemctl status nginx
```

nginx服务启动成功显示如下。

```
[root@localhost ~]# systemctl status nginx
● nginx.service - SYSV: Nginx is an HTTP(S) server, HTTP(S) reverse proxy and IMAP/POP3 p
roxy server
   Loaded: loaded (/etc/rc.d/init.d/nginx)
   Active: active (running) since Wed 2016-12-21 05:20:31 EST; 2 days ago
     Docs: man:systemd-sysv-generator(8)
 Main PID: 1965 (nginx)
   CGroup: /system.slice/system-hostos.slice/nginx.service
           ├─1965 nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.conf
           └─1967 nginx: worker process

Dec 21 05:20:30 localhost.localdomain systemd[1]: Starting SYSV: Nginx is an HTTP(S) s....
Dec 21 05:20:31 localhost.localdomain nginx[1446]: Starting nginx: [  OK  ]
Dec 21 05:20:31 localhost.localdomain systemd[1]: Started SYSV: Nginx is an HTTP(S) se....
Hint: Some lines were ellipsized, use -l to show in full.
```

若nginx服务启动失败，查看错误信息：

```
# systemctl status nginx.service --full
```

如下图显示nginx服务创建失败，是由于目录/var/spool/nginx/tmp/client_body创建失败，在root权限下手动进行创建，类似的问题也这样处理：

```
[root@localhost ~]# systemctl status nginx.service --full
● nginx.service - SYSV: Nginx is an HTTP(S) server, HTTP(S) reverse proxy and IMAP/POP3 p
roxy server
   Loaded: loaded (/etc/rc.d/init.d/nginx)
   Active: failed (Result: exit-code) since Thu 2016-12-08 06:13:45 EST; 3min 8s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 24340 ExecStart=/etc/rc.d/init.d/nginx start (code=exited, status=1/FAILURE)

Dec 08 06:13:45 localhost.localdomain systemd[1]: Starting SYSV: Nginx is an HTTP(S) serve
r, HTTP(S) reverse proxy and IMAP/POP3 proxy server...
Dec 08 06:13:45 localhost.localdomain nginx[24340]: Starting nginx: nginx: [emerg] mkdir()
 "/var/spool/nginx/tmp/client_body" failed (13: Permission denied)
Dec 08 06:13:45 localhost.localdomain nginx[24340]: [FAILED]
Dec 08 06:13:45 localhost.localdomain systemd[1]: nginx.service: control process exited, c
ode=exited status=1
Dec 08 06:13:45 localhost.localdomain systemd[1]: Failed to start SYSV: Nginx is an HTTP(S
) server, HTTP(S) reverse proxy and IMAP/POP3 proxy server.
Dec 08 06:13:45 localhost.localdomain systemd[1]: Unit nginx.service entered failed state.
Dec 08 06:13:45 localhost.localdomain systemd[1]: nginx.service failed.
```

```
# mkdir -p /var/spool/nginx/tmp/client_body
# mkdir -p /var/spool/nginx/tmp/proxy
# mkdir -p /var/spool/nginx/tmp/fastcgi
# mkdir -p /usr/share/nginx/uwsgi_temp
# mkdir -p /usr/share/nginx/scgi_temp
```

repo源部署

1. 在root权限下创建nginx配置文件/etc/nginx/nginx.conf中指定的目录/usr/share/nginx/repo：


```
# mkdir -p /usr/share/nginx/repo
```

2. 在root权限下修改目录/usr/share/nginx/repo的权限：
   ```
   # chmod -R 755 /usr/share/nginx/repo
   ```
3. 设置防火墙规则，开启nginx设置的端口（此处为80端口），在root权限下通过firewall设置端口开启：
   ```
   # firewall-cmd --add-port=80/tcp --permanent
   # firewall-cmd --reload
   ```
   在root权限下查询80端口是否开启成功，输出为yes则表示80端口开启成功：
   ```
   # firewall-cmd --query-port=80/tcp
   ```
   也可在root权限下通过iptables来设置80端口开启：
   ```
   # iptables -I INPUT -p tcp --dport 80 -j ACCEPT
   ```
4. nginx服务设置好之后，即可通过ip直接访问网页。
5. 通过下面几种方式将repo源放入到/usr/share/nginx/repo下：
   - 在root权限下拷贝镜像中相关文件至/usr/share/nginx/repo下，并修改目录权限。
     ```
     # mount /home/HCE-2.0-Enterprise-x86_64-dvd.iso  /mnt/
     # cp -r /mnt/Packages /usr/share/nginx/repo
     # cp -r /mnt/repodata /usr/share/nginx/repo
     # cp -r /mnt/RPM-GPG-KEY-HCE-2 /usr/share/nginx/repo
     # chmod -R 755 /usr/share/nginx/repo
     ```
     HCE-2.0-Enterprise-x86_64-dvd.iso存放在/home目录下。
   - 使用root在/usr/share/nginx/repo下创建repo源的软链接。
     ```
     # ln -s /mnt /usr/share/nginx/repo/os
     ```
     /mnt为已经创建好的repo源，/usr/share/nginx/repo/os将指向/mnt。

使用repo源

repo可配置为yum源，yum（全称为 Yellow dog Updater, Modified）是一个Shell前端软件包管理器。基于RPM包管理，能够从指定的服务器自动下载RPM包并且安装，可以自动处理依赖性关系，并且一次安装所有依赖的软件包，无需繁琐地一次次下载和安装。

repo配置为yum源（软件源）

构建好的repo可以配置为yum源使用，在/etc/yum.repos.d/目录下使用root权限创建***.repo的配置文件（必须以.repo为扩展名），分为本地和http服务器配置yum源两种方式：

- 配置本地yum源

  在/etc/yum.repos.d目录下创建hce.repo文件，使用构建的本地repo源作为yum源，hce.repo的内容如下：

  ```
  [base]
  name=base
  baseurl=file:///home/hce/srv/repo
  enabled=1
  gpgcheck=1
  gpgkey=file:///home/hce/srv/repo/RPM-GPG-KEY-HCE-2
  ```

说明

- [repoid]中的repoid为软件仓库（repository）的ID号，所有.repo配置文件中的各repoid不能重复，必须唯一。示例中repoid设置为**base**。
- name为软件仓库描述的字符串。
- baseurl为软件仓库的地址。
- enabled为是否启用该软件源仓库，可选值为1和0。缺省值为1，表示启用该软件源仓库。
- gpgcheck可设置为1或0，1表示进行gpg（GNU Private Guard）校验，0表示不进行gpg校验，gpgcheck可以确定rpm包的来源是有效和安全的。
- gpgkey为验证签名用的公钥。

- 配置http服务器yum源

  在/etc/yum.repos.d目录下创建hce.repo文件。

  若使用用户部署的http服务端的repo源作为yum源，hce.repo的内容如下：

```
[base]
name=base
baseurl=http://192.168.139.209/
enabled=1
gpgcheck=1
gpgkey=http://192.168.139.209/RPM-GPG-KEY-HCE-2
```

说明

“192.168.139.209”为示例地址，请用户根据实际情况进行配置。

dnf 相关命令

dnf命令在安装升级时能够自动解析包的依赖关系，一般的使用方式如下：

```
dnf <command> <packages name>
```

常用的命令如下：

- 安装，需要在root权限下执行。
  ```
  # dnf install <packages name>
  ```
- 升级，需要在root权限下执行。
  ```
  # dnf update <packages name>
  ```
- 回退，需要在root权限下执行。
  ```
  # dnf downgrade <packages name>
  ```
- 检查更新
  ```
  # dnf check-update
  ```
- 卸载，需要在root权限下执行。
  ```
  # dnf remove <packages name>
  ```
- 查询
  ```
  # dnf search <packages name>
  ```
- 本地安装，需要在root权限下执行。
  ```
  # dnf localinstall <absolute path to package name>
  ```
- 查看历史记录
  ```
  # dnf history
  ```
- 清除缓存目录
  ```
  # dnf clean all
  ```
- 更新缓存
  ```
  # dnf makecache
  ```