EulerOS V2.0SP5 管理员指南

文档版本 01

发布日期 2019-08-12

华为技术有限公司

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，华为公司对本文档内容不做任何明示或默示的声明或保证。

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。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： http://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 前言

概述

华为欧拉服务器操作系统软件V2.0是华为公司研发的企业级linux服务器操作系统，具有较高的性能、可靠性、易维护以及安全性，与业界软硬件良好兼容，能够满足您日常业务运维和管理的需求。

本手册为初次使用华为欧拉服务器操作系统软件V2.0产品的用户提供日常使用和配置维护的指引，文档包括基础配置、软件包管理、用户管理等相关说明，用户可以通过本文档对华为欧拉服务器操作系统软件V2.0进行日常的维护和相关配置。

华为欧拉服务器操作系统软件V2.0的软件简称为EulerOS V2.0，本手册及软件产品相关界面均使用了软件简称EulerOS V2.0。

读者对象

本手册适用于使用EulerOS V2.0 产品的用户，特别是初次使用或想了解EulerOS V2.0的用户，包括系统工程师、管理员及维护人员等。

使用本手册的用户需要具备基础的linux系统管理知识。

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
 危险	用于警示紧急的危险情形，若不避免，将会导致人员死亡或严重的人身伤害。
 警告	用于警示潜在的危险情形，若不避免，可能会导致人员死亡或严重的人身伤害。
 小心	用于警示潜在的危险情形，若不避免，可能会导致中度或轻微的人身伤害。

符号	说明
 注意	用于传递设备或环境安全警示信息，若不避免，可能会导致设备损坏、数据丢失、设备性能降低或其他不可预知的结果。“注意”不涉及人身伤害。
	用于突出重要/关键信息、最佳实践和小窍门等。“说明”不是安全警示信息，不涉及人身、设备及环境伤害。

2 基础配置

2.1 通过命令设置

2.1.1 设置语言环境

您可以通过localectl修改系统的语言环境，对应的参数设置保存在/etc/locale.conf文件中。这些参数，会在系统启动的早期被systemd的守护进程读取。

显示当前语言环境状态

要显示当前语言环境，使用命令如下：

```
localectl status
```

例如显示系统当前的设置，命令如下：

```
$ localectl status
   System Locale: LANG=zh_CN.UTF-8
       VC Keymap: cn
      X11 Layout: cn
```

列出可用的语言环境

要显示当前可用的语言环境，使用命令如下：

```
localectl list-locales
```

例如显示当前系统中所有可用的中文环境，命令如下：

```
$ localectl list-locales | grep zh
zh_CN
zh_CN.gb18030
zh_CN.gb2312
zh_CN.gbk
zh_CN.utf8
zh_HK
zh_HK.big5hkscs
zh_HK.utf8
```



```
zh_SG
zh_SG.gb2312
zh_SG.gbk
zh_SG.utf8
zh_TW
zh_TW.big5
zh_TW.euctw
zh_TW.utf8
```

设置语言环境

要设置语言环境，在root权限下使用命令如下：

```
localectl set-locale LANG=locale
```

例如设置为简体中文语言环境，在root权限下执行如下命令：

```
# localectl set-locale LANG=zh_CN.utf8
```

2.1.2 设置键盘

您可以通过localectl修改系统的键盘设置，对应的参数设置保存在/etc/locale.conf文件中。这些参数，会在系统启动的早期被systemd的守护进程读取。

显示当前设置

要显示当前键盘设置，使用命令如下：

```
localectl status
```

例如显示系统当前的设置，命令如下：

```
$ localectl status
   System Locale: LANG=zh_CN.UTF-8
       VC Keymap: cn
      X11 Layout: cn
```

列出可用的键盘布局

要显示当前可用的键盘布局，使用命令如下：

```
localectl list-keymaps
```

例如显示系统当前的中文键盘布局，命令如下：

```
$ localectl list-keymaps | grep cn
cn
```

设置键盘布局

要设置键盘布局，在root权限下使用命令如下：

```
localectl set-keymap map
```

此时设置的键盘布局同样也会应用到图形界面中。

设置完成后，查看当前状态：

```
$ localectl status
   System Locale: LANG=zh_CN.UTF-8
       VC Keymap: cn
      X11 Layout: us
```


2.1.3 设置日期和时间

本节介绍了如何通过timedatectl、date、hwclock命令来设置系统的日期、时间和时区等。

2.1.3.1 使用 timedatectl 命令设置

显示日期和时间

要显示当前的日期和时间，使用命令如下：

```
timedatectl
```

例如显示系统当前的日期和时间，如下：

```
$ timedatectl
      Local time: 五2015-08-14 15:57:24 CST
  Universal time: 五2015-08-14 07:57:24 UTC
        RTC time: 五2015-08-14 07:57:24
        Timezone: Asia/Shanghai (CST, +0800)
     NTP enabled: yes
NTP synchronized: no
 RTC in local TZ: no
      DST active: n/a
```

修改时间

要修改当前的时间，在root权限下使用命令如下：

```
timedatectl set-time HH:MM:SS
```

例如修改当前的时间，例如修改当前的时间为 15:57:24，如下：

```
# timedatectl set-time 15:57:24
```

修改日期

要修改当前的日期，在root权限下使用命令如下：

```
timedatectl set-time YYYY-MM-DD
```

例如修改当前的日期，例如修改当前的日期为2015年8月14号，如下：

```
# timedatectl set-time '2015-08-14'
```

修改时区

显示当前可用时区，使用命令如下：

```
timedatectl list-timezones
```

要修改当前的时区，在root权限下使用命令如下：

```
timedatectl set-timezone time_zone
```

例如修改当前的时区，首先查询所在地域的可用时区（以Asia为例）：

```
# timedatectl list-timezones | grep Asia
Asia/Aden
Asia/Almaty
Asia/Amman
Asia/Anadyr
Asia/Aqtau
```



```
Asia/Aqtobe
Asia/Ashgabat
Asia/Baghdad
Asia/Bahrain
……

Asia/Seoul
Asia/Shanghai
Asia/Singapore
Asia/Srednekolymsk
Asia/Taipei
Asia/Tashkent
Asia/Tbilisi
Asia/Tehran
Asia/Thimphu
Asia/Tokyo
```

修改当前的时区为“Asia/Shanghai”，如下：

```
# timedatectl set-timezone Asia/Shanghai
```

通过远程服务器进行时间同步

您可以启用NTP远程服务器进行系统时钟的自动同步。是否启用NTP，在root权限下使用命令如下：

```
timedatectl set-ntp boolean
```

例如启动自动远程时间同步，如下：

```
# timedatectl set-ntp yes
```

2.1.3.2 使用 date 命令设置

显示当前的日期和时间

要显示当前的日期和时间，使用命令如下：

```
date
```

默认情况下， date命令显示本地时间。要显示UTC时间，添加--utc或-u参数：

```
date --utc
```

要自定义对应的输出信息格式，添加 +"format" 参数：

```
date +"format"
```

表 2-1 参数说明

格式参数	说明
%H	小时以HH格式（例如 17）。
%M	分钟以MM格式（例如 37）。
%S	秒以SS格式（例如 25）。
%d	日期以DD格式（例如 15）。
%m	月份以MM格式（例如 07）。
%Y	年以YYYY格式（例如 2015）。

格式参数	说明
%Z	时区缩写（例如CEST）。
%F	日期整体格式为YYYY-MM-DD（例如 2015-7-15），和%Y-%m-%d等同。
%T	时间整体格式为HH:MM:SS（例如 18:30:25），和%H:%M:%S等同。

实际使用示例如下：

- 显示当前的日期和本地时间。
```
$ date
2015年 08月 17日 星期一 17:26:34 CST
```
- 显示当前的日期和UTC时间。
```
$ date --utc
2015年 08月 17日 星期一 09:26:18 UTC
```
- 自定义date命令的输出。
```
$ date +"%Y-%m-%d %H:%M"
2015-08-17 17:24
```

修改时间

要修改当前的时间，添加--set或者-s参数。在root权限下使用命令如下：

```
date --set HH:MM:SS
```

默认情况下， date命令设置本地时间。要设置UTC时间，添加--utc或-u参数：

```
date --set HH:MM:SS --utc
```

例如修改当前的时间，在root权限下使用命令如下：

```
# date --set 23:26:00
```

修改日期

要修改当前的日期，添加--set或者-s参数。在root权限下使用命令如下：

```
date --set YYYY-MM-DD
```

例如修改当前的日期为2015年11月2日，如下：

```
# date --set 2015-11-02
```

2.1.3.3 使用 hwclock 命令设置

hwclock用来进行硬件的时钟设置（RTC，Real Time Clock）。

硬件时钟和系统时钟

Linux将时钟分为系统时钟(System Clock)和硬件时钟(Real Time Clock，RTC)两种。系统时间是指当前Linux Kernel中的时钟，而硬件时钟则是主板上由电池供电的主板硬件时钟，这个时钟可以在BIOS的"Standard BIOS Feature"项中进行设置。

当Linux启动时，会读取硬件时间，并根据硬件系统时间来设置系统时间。

显示日期和时间

要显示当前硬件的日期和时间，在root权限下使用命令如下：

```
hwclock
```

例如显示当前硬件的日期和时间，如下：

```
# hwclock
2015年08月17日  星期一 14时34分42秒. -0.094973 秒
```

设置日期和时间

要修改当前硬件的日期和时间，在root权限下使用命令如下：

```
hwclock --set --date "dd mmm yyyy HH:MM"
```

例如修改当前的时间，如下：

```
# hwclock --set --date "21 Oct 2015 21:17" --utc
```

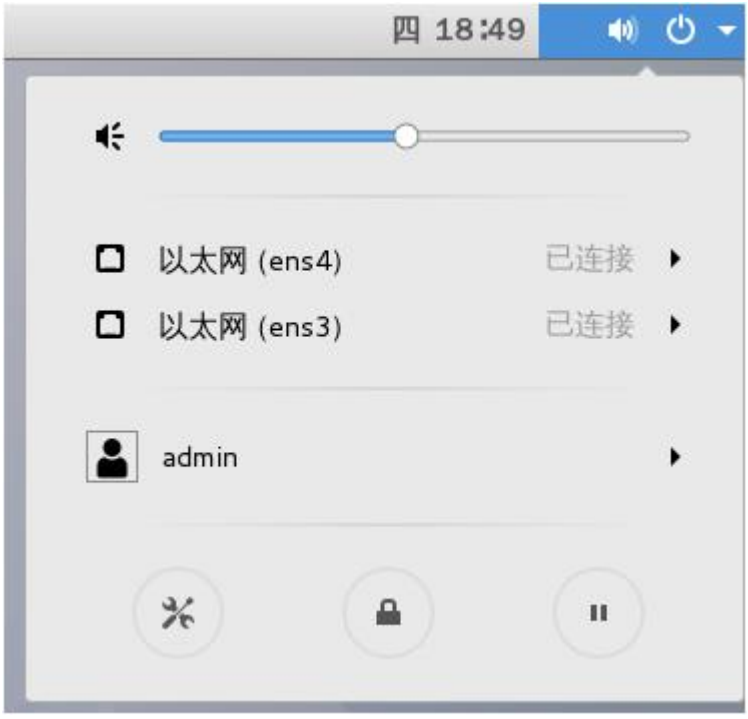
2.2 通过图形界面设置

本节主要介绍在图形界面环境下针对一些系统基本设置选项进行说明，指导用户操作。

2.2.1 进入设置界面

在桌面环境中，单击右上角的蓝色区域，在弹出的菜单中，单击左下角的设置图标，如图2-1。

图 2-1 设置界面



弹出用户设置界面，如图2-2所示。用户可以对背景、语言、网络等一系列配置项进行设置。

图 2-2 设置详细界面

2.2.2 设置语言

在所示的选项中，单击“区域和语言”。在弹出的设置界面中，如图2-3所示，用户可以根据需要，分别设置语言、格式和输入源。

图 2-3 语言

2.2.3 设置键盘

在设置界面所示的选项中，单击“键盘”。在弹出的设置界面中，如图2-4所示，用户可以根据需要，分别设置重复键、光标闪烁和快捷键等功能。

图 2-4 键盘

2.2.4 设置日期和时间

在设置界面所示的选项中，单击“日期和时间”，弹出修改日期和时间的主界面。

设置时，非root用户需要单击界面右上角“解锁”，输入管理员密码进行解锁，然后进行设置修改，如图2-5所示。

说明

如果安装过程中未创建管理员用户，只创建了普通用户，则此时需要输入root账户密码。

图 2-5 时间和日期

完成认证后，用户可以对地区、城市以及时间等进行修改，如图2-6和图2-7和图2-8所示。

图 2-6 日期和时间 1

图 2-7 日期和时间 2

图 2-8 设置时区

3 编译环境使用指导

3.1 前提条件

请在EulerOS环境或者redhat 7.0/7.1、centOS7.0/7.1环境上使用编译环境。

3.2 获取编译环境

EulerOS编译环境在发布包中的位置为Software。

```
[Software]
     |-[x86_64]
          |-[Compile Tools]
                |-Euler_compile_env.tar.gz    //通用版本的编译环境（包含9000和SVP）
                |-Euler_compile_env.tar.gz.sha256
                |-Euler_compile_env_pangea.tar.gz  //给统一存储（包含EVR）用户的X86架构编译
包
                |-Euler_compile_env_pangea.tar.gz.sha256
          |-[Debug Tools]
             |-[standard]//给通用版本（包含9000和SVP）用户的调试包
                |-debug_tool.tar.gz
                |-debug_tool.tar.gz.sha256
                |-perf_oprofile_tool.tar.gz
                |-perf_oprofile_tool.tar.gz.sha256
                |-vmcore_debug_tool.tar.gz
                |-vmcore_debug_tool.tar.gz.sha256
             |-[storage]//给统一存储（包含EVR）用户的调试包
                |-debug_tool.tar.gz
                |-debug_tool.tar.gz.sha256
                |-perf_oprofile_tool.tar.gz
                |-perf_oprofile_tool.tar.gz.sha256
                |-systemtap_run_tool.tar.gz
                |-systemtap_run_tool.tar.gz.sha256
                |-vmcore_debug_tool.tar.gz
                |-vmcore_debug_tool.tar.gz.sha256
     |-[aarch64]
          |-[Compile Tools]
```



```
                |compile_tools.tar.gz
                |compile_tools.tar.gz.sha256
                |-Euler_compile_env_cross.tar.gz
                |-Euler_compile_env_cross.tar.gz.sha256
                |-Euler_compile_env_cross_storage.tar.gz
                |-Euler_compile_env_cross_storage.tar.gz.sha256
          |-[Debug Tools]
             |-[standard]
                |-vmcore_debug_tool.tar.gz
                |-vmcore_debug_tool.tar.gz.sha256
             |-[storage]
                |-vmcore_debug_tool.tar.gz
                |-vmcore_debug_tool.tar.gz.sha256
```

3.3 操作步骤

3.3.1 root 用户操作步骤

步骤1 获取Euler_compile_env.tar.gz和Euler_compile_env.tar.gz.sha256。验证获取的编译环境是否完整。

```
sha256sum -c Euler_compile_env.tar.gz.sha256
Euler_compile_env.tar.tgz: OK     //表示编译环境完整
```

若显示如下，表示编译环境不完整，请重新获取：

```
sha256sum -c Euler_compile_env.tar.gz.sha256
Euler_compile_env.tar.gz: FAILED
sha256sum: WARNING: 1 computed checksum did NOT match
```

步骤2 执行tar xzvf Euler_compile_env.tar.gz，解压后目录为Euler_compile_env。

步骤3 拷贝自己的代码到Euler_compile_env的任意目录下。

步骤4 进入编译环境目录Euler_compile_env下，执行chroot ./ /bin/bash -login。

说明

如果要查询编译环境中的内核版本，请使用"uname -r"命令。

步骤5 cd到代码目录，执行编译。

此处以编译Hello World程序进行举例说明。编写Hello World程序，保存为helloworld.c，示例如下：

```
#include <stdio.h>
int main()
{
       printf("Hello World!\n");
       return 0;
}
```

执行命令：

```
gcc helloworld.c -o helloworld
```

编译执行未报错，表明执行通过，然后就可以将编译好的文件拷贝到对应的系统环境中执行。

说明

编译内核模块的时候，-C 可以指定内核源码目录“/lib/modules/$(uname -r)/build”。

步骤6 编译完成后，如果要删除编译环境，请按照如下步骤操作。

1. 执行exit命令退出编译环境。
2. 卸载proc：umount proc。
3. 执行cd ..命令退出当前目录。
4. 执行rm -rf Euler_compile_env命令删除编译环境。

说明

执行步骤4时会默认挂载/proc伪文件系统，所以在结束后需要执行卸载步骤；如果需要手动挂载和卸载其他伪文件系统，请参考**3.4 挂载及卸载/sys、/proc、/dev伪文件系统**。

----结束

3.3.2 普通用户操作步骤

步骤1 获取Euler_compile_env.tar.gz和Euler_compile_env.tar.gz.sha256。验证获取的编译环境是否完整。

```
sha256sum -c Euler_compile_env.tar.gz.sha256
Euler_compile_env.tar.tgz: OK     //表示编译环境完整
```

若显示如下，表示编译环境不完整，请重新获取：

```
sha256sum -c Euler_compile_env.tar.gz.sha256
Euler_compile_env.tar.gz: FAILED
sha256sum: WARNING: 1 computed checksum did NOT match
```

步骤2 需要使用root权限解压编译环境：sudo tar xzvf Euler_compile_env.tar.gz，解压后目录为Euler_compile_env。

说明

如果使用普通用户解压会失败，报错：Cannot mknod: Operation not permitted。

步骤3 下载或者编写要编译的代码，保证代码的所属用户是普通用户。

步骤4 进入编译环境目录Euler_compile_env，执行sudo chroot ./ /bin/bash -login。

说明

1 需要使用root权限执行chroot命令。

2 如果要查询编译环境中的内核版本，请使用"uname -r"命令。

步骤5 在编译环境中添加普通用户，添加的用户的UID和用户名需要和主机上的普通用户UID和用户名保持一致，执行useradd -u UID -m USERNAME。

说明

UID为主机上普通用户的UID，USERNAME为主机上的普通用户的用户名。

步骤6 执行exit命令退出编译环境，拷贝自己的代码到编译环境的/home/USERNAME下面；进入编译环境目录Euler_compile_env，执行sudo chroot ./ /bin/bash -login。

步骤7 在编译环境中切换成普通用户：su USERNAME。

步骤8 cd到代码目录，执行编译。

此处以编译Hello World程序进行举例说明。编写Hello World程序，保存为helloworld.c，示例如下：

```
#include <stdio.h>
int main()
{
        printf("Hello World!\n");
        return 0;
}
```


执行命令：

```
gcc helloworld.c -o helloworld
```

编译执行未报错，表明执行通过，编译过程中需要使用root权限的步骤，请使用sudo进行提权操作。然后就可以将编译好的文件拷贝到对应的系统环境中执行。

说明

编译内核模块的时候，-C 可以指定内核源码目录“/lib/modules/$(uname -r)/build”。

步骤9 编译完成后，如果要删除编译环境，请按照如下步骤操作。

1. 执行exit命令退出编译环境中的普通用户。
2. 执行exit命令退出编译环境。
3. 使用root权限卸载proc：sudo umount proc。
4. 执行cd ..命令退出当前目录。
5. 执行sudo rm -rf Euler_compile_env命令删除编译环境。

   说明

   如果需要手动挂载和卸载其他伪文件系统，请参考**3.4 挂载及卸载/sys、/proc、/dev伪文件系统**，该节涉及的操作都需要root权限。

----结束

3.4 挂载及卸载/sys、/proc、/dev 伪文件系统

本节只涉及如何挂载及卸载，具体/sys、/proc、/dev的功能及用法，请参考linux内核Documentation目录下的文档，例如/dev/shm文档在"linux内核源码目录/Documentation/filesystems/tmpfs.txt"。

挂载及卸载的前提是切换到编译环境（例如，在Euler_compile_env目录下执行chroot . ./bin/bash -login）。

1. 挂载/proc
   ```
   mount -t proc proc /proc
   ```
2. 挂载/sys
   ```
   mount -t sysfs  sysfs  /sys
   ```
3. 挂载/dev/shm
   ```
   mount -t tmpfs tmpfs /dev/shm
   ```
   或：

   指定tmpfs的size，但建议是物理内存的一半，详情见Linux内核文档tmpfs.txt，例如设置容量是1.5G，inode是1000000：
   ```
   mount -t tmpfs tmpfs /dev/shm -o size=1500M -o nr_inodes=1000000
   ```
   这意味着最多可存入一百万个小文件。
4. 挂载/dev/pts
   ```
   mount -t devpts devpts /dev/pts
   ```
5. 卸载使用umount，例如：
   ```
   umount /proc
   ```
6. 备注：
   - 当挂载好/proc后，就可以使用cat /proc/filesystems查看当前系统支持的文件系统。
   - 当挂载时，报类似“filesystem xxxx not supported by kernel”错误时，执行cat /proc/filesystems，然后从输出中查找是否有xxxx。

- 当挂载到的目录不存在时，例如/dev下不存在shm，请使用mkdir -p /dev/shm创建shm目录。
- 记得exit退出chroot时卸载登入编译环境后执行的挂载（例如，umount /proc），否则删除编译环境时，挂载的目录会提示busy。
- 实现自动挂载，需要把挂载命令行写到etc/profile里，当这样配置后，一旦用户进入编译环境，就会自动挂载。例如，将挂载命令放到Euler_compile_env/etc/profile里后，先cd Euler_compile_env，然后执行chroot . ./bin/bash -login，就会执行编译环境的profile实现自动挂载。
- 自动卸载，可以写在有权限卸载的用户的家目录下的.bash_logout，例如，将umount /proc添加到/root/.bash_logout里，如果.bash_logout不存在，可以新建一个。只有chroot带有-login时，exit退出才会执行.bash_logout。

4 管理用户

在linux中，每个普通用户都有一个账户，包括用户名、密码和主目录等信息。除此之外，还有一些系统本身创建的特殊用户，它们具有特殊的意义。其中最重要的是管理员账户，它默认用户名是root。同时linux也提供了用户组，使每一个用户至少属于一个组，这样便于进行权限的管理。

用户和组管理是系统安全管理的重要组成部分，本章主要介绍EulerOS提供的用户管理和组管理命令，如何建立多个管理员账户以及为普通用户分配特权的方法。

4.1 增加用户

useradd 命令

通过useradd命令可以为系统添加新用户信息。

```
useradd [options] user_name
```

用户信息文件

与用户账号信息有关的文件如下：

- /etc/passwd——用户账号信息。
- /etc/shadow——用户账号信息加密文件。
- /etc/group——组信息文件。
- /etc/defaut/useradd——定义默认设置文件。
- /etc/login.defs——系统广义设置文件。
- /etc/skel——默认的初始配置文件目录。

创建用户实例

例如新建一个用户XXX。命令如下：

```
[root@localhost ~]# useradd XXX
```

说明

没有任何提示，表明用户建立成功。这时并没有设置用户的口令，必须使用passwd命令修改用户的密码，没有设置密码的新账号将不能使用。

使用id命令查看新建的用户信息，命令如下：

```
[root@localhost ~]# id user_example
uid=502(user_example)    gid=502(user_example)
```

修改用户user_example的密码：

```
[root@localhost ~]# passwd user_example
```

根据提示两次输入新用户的密码，完成口令更改。过程如下：

```
[root@localhost ~]# passwd user_example
Changing password for user user_example.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

4.2 修改用户账号

修改密码

普通用户可以用passwd修改自己的密码，只有管理员才能用passwd usename为其他用户修改密码。

修改用户 shell 设置

使用chsh命令可以修改自己的shell，只有管理员才能用chsh usename为其他用户修改shell设置。

用户也可以使用usermod命令修改shell信息，命令如下：

```
usermod -s [new_shell_path] usename
```

其中new_shell_path和usename要取相应的值。

例：将用户user_example的shell改为csh。命令如下：

```
[root@localhost ~]# usermod -s /bin/csh user_example
```

修改主目录

```
usermod -d [new_home_directory] usename
```

将用户user_example的主目录更改为/home/user_example，命令如下。

```
[root@localhost ~]# usermod -d /home/user_example user_example
```

如果想将现有主目录的内容转移到新的目录，应该使用-m选项，如下所示：

```
usermod -d /new/home -m usename
```


修改 UID

```
usermod -u UID usename
```

该用户主目录中所拥有的文件和目录都将自动修改UID设置。但是，对于主目录外所拥有的文件，只能手工用chown命令修改所有权设置。

修改账号的有效期

如果使用了影子口令，则可以使用如下命令来修改一个账号的有效期：

```
usermod -e MM/DD/YY usename
```

4.3 删除用户

使用userdel命令可删除现有用户。

例：下面的命令将删除用户Test。命令如下：

```
[root@localhost ~]# userdel Test
```

如果想同时删除该用户的主目录以及其中所有内容，要使用-r参数来递归删除。

说明

- 无法删除已经进入系统的用户，如果想强行完成，需要先杀死有关的进程，然后再使用userdel命令。
- 删除账号之后，新创建的同名账号会继承之前未完全删除的相关联信息。所以在删除账号时，必须手动删除其关联的所有目录和文件。

4.4 管理员账户授权

4.4.1 创建多个管理员账户

大多数新系统管理员认为root用户是唯一的管理员账户，其实root只是系统默认的管理员账户。随便打开一个/etc/passwd文件的例子，您就会发现如下几行：

```
root:x:0:0:root: /root:/bin/bash
bin:x:1:1:bin: /bin: /sbin/nologin
daemon:x:2:2:daemon: /sbin: /sbin/nologin
adm:x:3:4:acim: /var/adm:/sbin/nologin
lp:x:4:7:lp: /va r/spool/ lpd: /sbin/nologin
sync:x:5:0:sync: /sbin: /bin/sync
shutdown:x:6:0:shutdown: /sbin: /sbin/shutdown
Test:x:0:0::/home/j ianCJzhonghua: /bin/bash
```

从上面的内容可以发现，root的UID和GID都为0。实际上，管理员账户的充要条件就是UID和GID都等于0，也就是说，只要用户的UID和GID都为0，就与常被称为root管理员账户一样，如上例中的Test也是一个管理员账户。

4.4.2 为普通用户分配特权

使用sudo命令可以允许普通用户执行管理员账户才能执行的命令。

sudo命令允许已经在/etc/sudoers文件中指定的用户运行管理员账户命令。例如，一个已经获得许可的普通用户可以运行：

```
sudo /usr/sbin/useradd newuserl
```


实际上，sudo的配置完全可以指定某个已经列入/etc/sudoers文件的普通用户可以做什么，不可以做什么。

/etc/sudoers的配置行如下所示。

- 空行或注释行（以#字符打头）：无具体功能的行。
- 可选的主机别名行：用来创建主机列表的简称。必须以Host_Alias关键词开头，列表中的主机必须用逗号隔开，如：

  ```
  Host_Alias  linux=ted1,ted2
  ```

  其中ted1和ted2是两个主机名，可使用linux（别名）称呼它们。
- 可选的用户别名行：用来创建用户列表的简称。用户别名行必须以User_Alias关键词开头，列表中的用户名必须以逗号隔开。其格式同主机别名行。
- 可选的命令别名行：用来创建命令列表的简称。必须以Cmnd_Alias开头，列表中的命令必须用逗号隔开。
- 可选的运行方式别名行：用来创建用户列表的简称。不同的是，使用这样的别名可以告诉sudo程序以列表中某一用户的身份来运行程序。
- 必要的用户访问说明行。

  用户访问的说明语法如下：

  ```
  user host = [ run as user ] command list
  ```

  在user处指定一个真正的用户名或定义过的别名，host也可以是一个真正的主机名或者定义过的主机别名。默认情况下，sudo执行的所有命令都是以root身份执行。如果您想使用其他身份可以指定。command list可以是以逗号分隔的命令列表，也可以是一个已经定义过的别名，如：

  ```
  ted1   ted2=/sbin/shutdown
  ```

  这一句说明ted1可以在ted2主机上运行关机命令。

  说明

  可以在一行定义多个别名，中间用冒号(:)隔开。

  可在命令或命令别名之前加上感叹号(!)，使该命令或命令别名无效。

  有两个关键词：ALL和NOPASSWD。ALL意味着“所有”（所有文件、所有主机或所有命令），NOPASSWD意味着不用密码。

下面是一个sudoers文件的例子：

```
#sudoers files
#User alias specification
User_Alias ADMIN=ted1:POWERUSER=globus,ted2
#user privilege specification
ADMIN ALL=ALL
POWERUSER ALL=ALL,!/bin/su
```

第3行定义了两个别名ADMIN和POWERUSER，第5行说明在所有主机上ADMIN都可以以root身份执行所有命令。第6行给POWERUSER除了运行su命令外等同ADMIN的权限。

4.5 通过图形界面设置

在图2-2所示的选项中，单击“用户”，弹出用户设置的主界面。

设置时，非root用户需要单击界面右上角“解锁”，输入管理员密码进行解锁，然后进行设置修改，如图4-1所示。

说明

如果安装过程中未创建管理员用户，只创建了普通用户，则此时需要输入root账户密码。

图 4-1 认证

完成认证后，用户单击左下角的“+”或者“-”，进行增加/删除用户操作。

4.5.1 添加用户

用户单击图4-2左下角的“+”（非root用户需要完成认证），添加用户，弹出如图4-3界面。

图 4-2 用户

用户需要设置账户类型（有“标准”和“管理员”两种），输入全名并设置用户名。设置完成后，单击“添加”即可，如图4-3所示。

图 4-3 添加用户

用户创建后，默认是被禁用的，需要设置登录选项。单击**图4-2**的“下次登入时设置”，弹出密码设置对话框，用户可以根据实际情况选择是否设置密码，如**图4-4**所示。

图 4-4 密码

4.5.2 删除用户

用户选择需要删除的用户，单击左下角的“-”（非root用户需要完成认证）删除用户，弹出如图4-5所示。删除用户时，系统会提示是否删除相关的用户数据，用户需要根据实际情况进行选择。

说明

当前正在登录的用户无法删除。另外，如果当前登录的是普通用户，要删除的是唯一的管理员用户，则操作无法执行。

图 4-5 删除用户

5 使用 yum 管理软件包

yum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。

5.1 配置 yum

5.1.1 修改配置文件

yum的主要配置文件是/etc/yum.conf，这个文件里面包含“main”部分，保存着yum的全局设置；也可以包含一个或者多个repository部分，用来设置需要安装的软件源位置。在/etc/yum.repos.d目录中有一些repo源相关文件，它们定义了各个仓库。

yum的配置一般有两种方式，一种是直接配置/etc目录下的yum.conf文件，另外一种是在/etc/yum.repos.d目录下增加.repo文件。

修改 main 部分

/etc/yum.conf文件包含一个“main”部分，配置文件示例如下：

```
[main]
cachedir=/var/cache/yum/$basearch/$releasever
keepcache=0
debuglevel=2
logfile=/var/log/yum.log
exactarch=1
obsoletes=1
gpgcheck=1
plugins=1
installonly_limit=3
```

说明

关于配置文件的完整说明，请参见man帮助信息的yum.conf(5) 。

常用选项说明：

表 5-1 main 参数说明

参数	说明
cachedir	Yum的缓存目录，该目录用于存储RPM包和数据库文件。
keepcache	可选值是1和0，表示是否要缓存已安装成功的那些RPM包及头文件，默认值为0，即不缓存。
debuglevel	设置Yum生成的debug信息。取值范围：[0-10]，数值越大会输出越详细的debug信息。默认值为2，设置为0表示不输出debug信息。
logfile	指定日志文件的输出位置。
exactarch	可选值1和0，设置yum在升级已安装的软件包时是否考虑体系架构。默认值为1，表示需要考虑体系架构，如果系统已经安装了一个32位的包，那么就不会通过安装相同的64位的包进行升级。
obsoletes	可选值1和0，设置是否允许更新陈旧的RPM包。认值为1，表示允许更新。
gpgcheck	可选值1和0，设置是否进行gpg校验。默认值为1，表示需要进行校验。
plugins	可选值1和0，表示启用或禁用yum插件。默认值为1，表示启用yum插件。
installonly_limit	设置可以同时安装“installonlypkgs”指令列出包的数量。默认值为3，不建议降低此值2。

修改 repository 部分

repository部分允许您定义定制化的yum软件源仓库，各个仓库的名称不能相同，否则会引起冲突。下面是[repository]部分的一个最小配置示例：

```
[repository]
name=repository_name
baseurl=repository_url
```

选项说明：

表 5-2 repository 参数说明

参数	说明
name=repository_name	软件仓库（repository ）描述的字符串。
baseurl=repository_url	软件仓库（repository ）的地址。 ● 使用http协议的网络位置：http://path/to/repo ● 使用ftp协议的网络位置：ftp://path/to/repo ● 本地位置：file:///path/to/local/repo

显示当前配置

要显示当前yum参数的值，执行如下命令：

```
yum-config-manager
```

要显示配置文件某一部分参数的值，执行如下命令：

```
yum-config-manager section…
```

您也可以使用一个全局正则表达式，来显示所有匹配部分的配置：

```
yum-config-manager glob_expression…
```

例如要列出的“repo”部分的所有配置选项及其对应的值，命令如下：

```
$ yum-config-manager repo \*
================================ repo: EulerOS-base ================================
[EulerOS-base]
async = True
bandwidth = 0
base_persistdir = /var/lib/yum/repos/x86_64/2.0SP5
baseurl = http://developer.huawei.com/ict/site-euleros/euleros/repo/yum/2.x/os/x86_64/
cache = 0
cachedir = /var/tmp/yum-euler-BafhEh/x86_64/2.0SP5/EulerOS-base
check_config_file_age = True
cost = 1000
deltarpm_metadata_percentage = 100
deltarpm_percentage =
enabled = True
enablegroups = True
exclude =
failovermethod = priority
……
```

5.1.2 创建 yum 源

要建立一个yum源，请按照下列步骤操作。

1. 安装createrepo软件包。在root权限下执行如下命令：
   ```
   yum install createrepo
   ```
2. 将需要的软件包复制到一个目录下，如/mnt/local_repo/ 。
3. 创建yum源，执行以下命令：
   ```
   createrepo --database /mnt/local_repo
   ```

5.1.3 添加、启用和禁用 yum 源

本节将介绍如何通过“yum-config-manager”命令添加、启用和禁用yum源。

添加 yum 源

要定义一个新的yum源，您可以在/etc/yum.conf文件中添加“repository”部分，或者在/etc/yum.repos.d/目录下添加“.repo文件”进行说明。建议您通过添加“.repo”的方式来定义yum源，每个yum源都有自己对应的“.repo文件”。

要在您的系统中添加一个这样的源，请在root权限下执行如下命令：

```
yum-config-manager --add-repo repository_url
```

例如要添加位于http://www.example.com/example.repo的源，命令如下：

```
# yum-config-manager --add-repo http://www.example.com/example.repo
Loaded plugins: langpacks, product-id, subscription-manager
adding repo from: http://www.example.com/example.repo
```



```
grabbing file http://www.example.com/example.repo to /etc/yum.repos.d/example.repo
example.repo                                               |  413 B     00:00
repo saved to /etc/yum.repos.d/example.repo
```

启用 yum 源

要启用yum源，请在root权限下执行如下命令：

```
yum-config-manager --enable repository…
```

您也可以使用一个全局正则表达式，来启用所有匹配的yum源：

```
yum-config-manager --enable glob_expression…
```

例如要启用已定义的example、example-debuginfo和example-source源，命令如下：

```
# yum-config-manager --enable example\*
Loaded plugins: langpacks, product-id, subscription-manager
============================== repo: example ==============================
[example]
bandwidth = 0
base_persistdir = /var/lib/yum/repos/x86_64/6Server
baseurl = http://www.example.com/repo/6Server/x86_64/
cache = 0
cachedir = /var/cache/yum/x86_64/6Server/example
[output truncated]
```

禁用 yum 源

要禁用yum源，请在root权限下执行如下命令：

```
yum-config-manager --disable repository…
```

同样的，您也可以使用一个全局正则表达式，来禁用所有匹配的yum源：

```
yum-config-manager --disable glob_expression…
```

5.2 管理软件包

使用yum能够让您方便的进行查询、安装、删除软件包等操作。

搜索软件包

您可以使用RPM包名称、缩写或者描述搜索需要的RPM包，使用命令如下：

```
yum search term…
```

示例如下：

```
$   yum search httpd
========================================== N/S matched: httpd
==========================================
httpd.x86_64 : Apache HTTP Server
httpd-devel.x86_64 : Development interfaces for the Apache HTTP server
httpd-manual.noarch : Documentation for the Apache HTTP server
httpd-tools.x86_64 : Tools for use with the Apache HTTP Server
libmicrohttpd.i686 : Lightweight library for embedding a webserver in applications
libmicrohttpd.x86_64 : Lightweight library for embedding a webserver in applications
mod_auth_mellon.x86_64 : A SAML 2.0 authentication module for the Apache Httpd Server
mod_dav_svn.x86_64 : Apache httpd module for Subversion server
```

列出软件包清单

要列出系统中所有已安装的以及可用的RPM包信息，使用命令如下：

```
yum list all
```

要列出系统中特定的RPM包信息，使用命令如下：

```
yum list glob_expression…
```

示例如下：

```
$ yum list httpd
Available Packages
httpd.x86_64              2.4.6-40.4.h1           EulerOS-base
[root@localhost yum.repos.d]#
```

显示 RPM 包信息

要显示一个或者多个RPM包信息，使用命令如下：

```
yum info package_name…
```

例如搜索，命令如下：

```
$ yum info httpd
Available Packages
Name        : httpd
Arch        : x86_64
Version     : 2.4.6
Release     : 40.4.h1
Size        : 1.2 M
Repo        : EulerOS-base
Summary     : Apache HTTP Server
URL         : http://httpd.apache.org/
License     : ASL 2.0
Description : The Apache HTTP Server is a powerful, efficient, and extensible
            : web server.
```

安装 RPM 包

要安装一个软件包及其所有未安装的依赖，请在root权限下执行如下命令：

```
yum install package_name
```

您也可以通过添加软件包名字同时安装多个软件包。请在root权限下执行如下命令：

```
yum install package_name package_name…
```

示例如下：

```
yum install httpd-devel.x86_64
```

下载软件包

在使用yum安装过程中，可能会有提示信息需要您确认，如下：

```
...
Total size: 1.2 M
Is this ok [y/d/N]:
...
```

输入d，yum会下载相应的软件包，但是并不会安装。您可以根据自己需要在离线状态下安装这些软件包。下载的软件包默认保存在“/var/cache/yum/$basearch/$releasever/*repo-name*/packages/”目录（*repo-name*为系统repo源配置文件中repo的名称）。

删除软件包

要卸载软件包以及相关的依赖软件包，请在root权限下执行如下命令：

```
yum remove package_name…
```

示例如下：

```
yum remove totem
```

5.3 管理软件包组

软件包集合是服务于一个共同的目的一组软件包，例如系统工具集等。使用yum可以对软件包组进行安装/删除等操作，使相关操作更高效。

列出软件包组清单

使用summary参数，可以列出系统中所有已安装软件包组、可用的组，可用的环境组的数量，命令如下：

```
yum groups summary
```

使用示例如下：

```
$ yum groups summary
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)
EulerOS-base                                | 4.2 kB     00:00
(1/3): EulerOS-base/updateinfo              |  10 kB   00:00
(2/3): EulerOS-base/group_gz                |  15 kB   00:00
(3/3): EulerOS-base/primary_db              | 5.5 MB   00:07
Available environment groups: 4
Available Groups: 4
Done
```

要列出所有软件包组和它们的组ID ，命令如下：

```
yum group list ids
```

使用示例如下：

```
$ yum group list ids
Available environment groups:
   Base System (base-sys)
   Developer Mode (developer-mode)
   Cloud Server (cloud-server)
   Server with GUI (graphical-server-environment)
Installed groups:
   Development Tools (development)
Available Groups:
   Compatibility Libraries (compat-libraries)
   Security Tools (security-tools)
   Smart Card Support (smart-card)
Done
```

显示软件包组信息

要列出包含在一个软件包组中必须安装的包和可选包，使用命令如下：

```
yum group info glob_expression…
```

例如显示Development Tools信息，示例如下：

```
$  yum group info "Development Tools"
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)

Group: Development Tools
 Group-Id: development
```



```
Description: A basic development environment.
Mandatory Packages:
   autoconf
   automake
   binutils
  +bison
  +flex
   gcc
  +gcc-c++
  +gcc-go
   gettext
   libtool
   make
  +patch
   pkgconfig
  +redhat-rpm-config
  +rpm-build
  +rpm-sign
Default Packages:
  +byacc
  +cscope
  +ctags
  +diffstat
  +doxygen
   elfutils
  +gcc-gfortran
   git
  +indent
  +intltool
  +patchutils
  +rcs
  +subversion
  +swig
  +systemtap
Optional Packages:
  ElectricFence
  ant
  babel
  bzr
  chrpath
  cmake
  compat-gcc-44
  compat-gcc-44-c++
  cvs
  dejagnu
  expect
  gcc-gnat
  gcc-objc
  gcc-objc++
  imake
  javapackages-tools
  libstdc++-docs
  mercurial
  mod_dav_svn
  nasm
  perltidy
  python-docs
  rpmdevtools
  rpmlint
  systemtap-sdt-devel
  systemtap-server
```

安装软件包组

每一个软件包组都有自己的名称以及相应的ID（groupid），您可以使用软件包组名称或它的ID进行安装。

要安装一个软件包组，请在root权限下执行如下命令：


```
yum group install group_name
yum group install groupid
```

例如安装Development Tools相应的软件包组，命令如下：

```
# yum group install "Development Tools"
# yum group install development
```

删除软件包组

要卸载软件包组，您可以使用软件包组名称或它的ID，在root权限下执行如下命令：

```
yum group remove group_name
yum group remove groupid
```

例如删除Development Tools相应的软件包组，命令如下：

```
# yum group remove "Development Tools"
# yum group remove development
```

5.4 检查并更新

yum可以检查您的系统中是否有软件包需要更新。您可以通过yum列出需要更新的软件包，一次性全部更新或者选择对应的包单独更新。

检查更新

如果您需要显示当前系统可用的更新，使用命令如下：

```
yum check-update
```

使用实例如下：

```
# yum check-update
base                                     | 2.9 kB     00:00
updates                                  | 2.9 kB     00:00
(1/2): updates/primary_db                | 2.9 MB   00:00
(2/2): base/primary_db                   | 4.4 MB   00:00

anaconda-core.x86_64         19.31.123-1.14             updates
anaconda-gui.x86_64          19.31.123-1.14             updates
anaconda-tui.x86_64          19.31.123-1.14             updates
anaconda-user-help.x86_64    19.31.123-1.14             updates
anaconda-widgets.x86_64      19.31.123-1.14             updates
bind-libs.x86_64             32:9.9.4-29.3              updates
bind-libs-lite.x86_64        32:9.9.4-29.3              updates
bind-license.noarch          32:9.9.4-29.3              updates
bind-utils.x86_64            32:9.9.4-29.3              updates
euleros-config.x86_64        1.0-6                      updates
……
```

升级

如果您需要升级单个软件包，在root权限下使用命令如下：

```
yum update package_name
```

例如升级rpm包，示例如下：

```
# yum update anaconda-gui.x86_64
Resolving Dependencies
--> Running transaction check
---> Package anaconda-gui.x86_64 0:19.31.123-1.13 will be updated
---> Package anaconda-gui.x86_64 0:19.31.123-1.14 will be an update
```



```
--> Processing Dependency: anaconda-widgets = 19.31.123-1.14 for package: anaconda-
gui-19.31.123-1.14.x86_64
--> Processing Dependency: anaconda-user-help = 19.31.123-1.14 for package: anaconda-
gui-19.31.123-1.14.x86_64
--> Processing Dependency: anaconda-core = 19.31.123-1.14 for package: anaconda-
gui-19.31.123-1.14.x86_64
--> Running transaction check
---> Package anaconda-core.x86_64 0:19.31.123-1.13 will be updated
--> Processing Dependency: anaconda-core = 19.31.123-1.13 for package: anaconda-
tui-19.31.123-1.13.x86_64
---> Package anaconda-core.x86_64 0:19.31.123-1.14 will be an update
---> Package anaconda-user-help.x86_64 0:19.31.123-1.13 will be updated
---> Package anaconda-user-help.x86_64 0:19.31.123-1.14 will be an update
---> Package anaconda-widgets.x86_64 0:19.31.123-1.13 will be updated
---> Package anaconda-widgets.x86_64 0:19.31.123-1.14 will be an update
--> Running transaction check
---> Package anaconda-tui.x86_64 0:19.31.123-1.13 will be updated
---> Package anaconda-tui.x86_64 0:19.31.123-1.14 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                  Arch         Version              Repository     Size
================================================================================
Updating:
 anaconda-gui             x86_64       19.31.123-1.14       updates       461 k
Updating for dependencies:
 anaconda-core            x86_64       19.31.123-1.14       updates       1.4 M
 anaconda-tui             x86_64       19.31.123-1.14       updates       274 k
 anaconda-user-help       x86_64       19.31.123-1.14       updates       315 k
 anaconda-widgets         x86_64       19.31.123-1.14       updates       748 k

Transaction Summary
================================================================================
Upgrade  1 Package (+4 Dependent packages)

Total download size: 3.1 M
Is this ok [y/d/N]:
```

类似的，如果您需要升级软件包组，在root权限下使用命令如下：

```
yum group update group_name
```

更新所有的包和它们的依赖

要更新所有的包和它们的依赖，在root权限下使用命令如下：

```
yum update
```


6 管理服务

本章介绍如何使用systemd进行系统和服务管理。

6.1 简介

systemd是在Linux下，与SysV和LSB初始化脚本兼容的系统和服务管理器。systemd使用socket和D-Bus来开启服务，提供基于守护进程的按需启动策略，支持快照和系统状态恢复，维护挂载和自挂载点，实现了各服务间基于从属关系的一个更为精细的逻辑控制，拥有更高的并行性能。

概念介绍

systemd开启和监督整个系统是基于unit的概念。unit是由一个与配置文件对应的名字和类型组成的（例如：avahi.service unit有一个具有相同名字的配置文件，是守护进程Avahi的一个封装单元）。unit有多重类型，如**表6-1**所示。

表 6-1 unit 说明

unit名称	后缀名	描述
Service unit	.service	系统服务。
Target unit	.target	一组systemd units。
Automount unit	.automount	文件系统挂载点。
Device unit	.device	内核识别的设备文件。
Mount unit	.mount	文件系统挂载点。

unit名称	后缀名	描述
Path unit	.path	在一个文件系统中的文件或目录。
Scope unit	.scope	外部创建的进程。
Slice unit	.slice	一组用于管理系统进程分层组织的units。
Snapshot unit	.snapshot	systemd manager的保存状态。
Socket unit	.socket	一个进程间通信的Socket。
Swap unit	.swap	swap设备或者swap文件。
Timer unit	.timer	systemd计时器。

所有的可用systemd unit类型，可在如表6-2所示的路径下查看。

表 6-2 可用 systemd unit 类型

路径	描述
/usr/lib/systemd/system/	随安装的RPM产生的systemd units。
/run/systemd/system/	在运行时创建systemd units。
/etc/systemd/system/	由系统管理员创建和管理的systemd units。

6.2 特性说明

更快的启动速度

systemd提供了比UpStart更激进的并行启动能力，采用了socket/D-Bus activation等技术启动服务，带来了更快的启动速度。

为了减少系统启动时间，systemd的目标是：

- 尽可能启动更少的进程
- 尽可能将更多进程并行启动

同样地，UpStart也试图实现这两个目标。UpStart采用事件驱动机制，服务可以暂不启动，当需要的时候才通过事件触发其启动，这符合第一个设计目标；此外，不相干的服务可以并行启动，这也实现了第二个目标。

提供按需启动能力

当sysvinit系统初始化的时候，它会将所有可能用到的后台服务进程全部启动运行。并且系统必须等待所有的服务都启动就绪之后，才允许用户登录。这种做法有两个缺点：首先是启动时间过长；其次是系统资源浪费。

某些服务很可能在很长一段时间内，甚至整个服务器运行期间都没有被使用过。比如CUPS，打印服务在多数服务器上很少被真正使用到。您可能没有想到，在很多服务器

上SSHD也是很少被真正访问到的。花费在启动这些服务上的时间是不必要的；同样，花费在这些服务上的系统资源也是一种浪费。

systemd可以提供按需启动的能力，只有在某个服务被真正请求的时候才启动它。当该服务结束，systemd可以关闭它，等待下次需要时再次启动它。

采用 cgroup 特性跟踪和管理进程的生命周期

init系统的一个重要职责就是负责跟踪和管理服务进程的生命周期。它不仅可以启动一个服务，也能够停止服务。这看上去没有什么特别的，然而在真正用代码实现的时候，您或许会发现停止服务比一开始想的要困难。

服务进程一般都会作为精灵进程（daemon）在后台运行，为此服务程序有时候会派生(fork)两次。在UpStart中，需要在配置文件中正确地配置expect小节。这样UpStart通过对fork系统调用进行计数，从而获知真正的精灵进程的PID号。

cgroup已经出现了很久，它主要用来实现系统资源配额管理。cgroup提供了类似文件系统的接口，使用方便。当进程创建子进程时，子进程会继承父进程的cgroup。因此无论服务如何启动新的子进程，所有的这些相关进程都会属于同一个cgroup，systemd只需要简单地遍历指定的cgroup即可正确地找到所有的相关进程，将它们逐一停止即可。

启动挂载点和自动挂载的管理

传统的Linux系统中，用户可以用/etc/fstab文件来维护固定的文件系统挂载点。这些挂载点在系统启动过程中被自动挂载，一旦启动过程结束，这些挂载点就会确保存在。这些挂载点都是对系统运行至关重要的文件系统，比如HOME目录。和sysvinit一样，systemd管理这些挂载点，以便能够在系统启动时自动挂载它们。systemd还兼容/etc/fstab文件，您可以继续使用该文件管理挂载点。

有时候用户还需要动态挂载点，比如打算访问DVD内容时，才临时执行挂载以便访问其中的内容，而不访问光盘时该挂载点被取消(umount)，以便节约资源。传统地，人们依赖autofs服务来实现这种功能。

systemd内建了自动挂载服务，无需另外安装autofs服务，可以直接使用systemd提供的自动挂载管理能力来实现autofs的功能。

实现事务性依赖关系管理

系统启动过程是由很多的独立工作共同组成的，这些工作之间可能存在依赖关系，比如挂载一个NFS文件系统必须依赖网络能够正常工作。systemd虽然能够最大限度地并发执行很多有依赖关系的工作，但是类似“挂载NFS”和“启动网络”这样的工作还是存在天生的先后依赖关系，无法并发执行。对于这些任务，systemd维护一个“事务一致性”的概念，保证所有相关的服务都可以正常启动而不会出现互相依赖，以至于死锁的情况。

与 SysV 初始化脚本兼容

和UpStart一样，systemd引入了新的配置方式，对应用程序的开发也有一些新的要求。如果systemd想替代目前正在运行的初始化系统，就必须和现有程序兼容。任何一个Linux发行版都很难为了采用systemd而在短时间内将所有的服务代码都修改一遍。

systemd提供了和sysvinit以及LSB initscripts兼容的特性。系统中已经存在的服务和进程无需修改。这降低了系统向systemd迁移的成本，使得systemd替换现有初始化系统成为可能。

能够对系统进行快照和恢复

systemd支持按需启动，因此系统的运行状态是动态变化的，人们无法准确地知道系统当前运行了哪些服务。systemd快照提供了一种将当前系统运行状态保存并恢复的能力。

比如系统当前正运行服务A和B，可以用systemd命令行对当前系统运行状况创建快照。然后将进程A停止，或者做其他的任意的对系统的改变，比如启动新的进程C。在这些改变之后，运行systemd的快照恢复命令，就可立即将系统恢复到快照时刻的状态，即只有服务A，B在运行。一个可能的应用场景是调试：比如服务器出现一些异常，为了调试用户将当前状态保存为快照，然后可以进行任意的操作，比如停止服务等等。等调试结束，恢复快照即可。

6.3 管理系统服务

systemd提供systemctl命令来运行、关闭、重启、显示、启用/禁用系统服务。

sysvinit 命令和 systemd 命令

systemd提供systemctl命令与sysvinit命令的功能类似。当前版本中依然兼容service和chkconfig命令，相关说明如表 1，但建议用systemdemd进行系统服务管理。

表 6-3 sysvinit 命令和 systemd 命令的对照表

sysvinit命令	systemd命令	备注
service foo start	systemctl start foo.service	用来启动一个服务 (并不会重启现有的)。
service foo stop	systemctl stop foo.service	用来停止一个服务 (并不会重启现有的)。
service foo restart	systemctl restart foo.service	用来停止并启动一个服务。
service foo reload	systemctl reload foo.service	当支持时，重新装载配置文件而不中断等待操作。
service foo condrestart	systemctl condrestart foo.service	如果服务正在运行那么重启它。
service foo status	systemctl status foo.service	汇报服务是否正在运行。
chkconfig foo on	systemctl enable foo.service	在下次启动时或满足其他触发条件时设置服务为启用。
chkconfig foo off	systemctl disable foo.service	在下次启动时或满足其他触发条件时设置服务为禁用。
chkconfig foo	systemctl is-enabled foo.service	用来检查一个服务在当前环境下被配置为启用还是禁用。

sysvinit命令	systemd命令	备注
chkconfig –list	systemctl list-unit-files --type=service	输出在各个运行级别下服务的启用和禁用情况。
chkconfig foo –list	ls /etc/systemd/system/*.wants/foo.service	用来列出该服务在哪些运行级别下启用和禁用。
chkconfig foo –add	systemctl daemon-reload	当您创建新服务文件或者变更设置时使用。

显示所有当前服务

如果您需要显示当前正在运行的服务，使用命令如下：

```
systemctl list-units --type service
```

如果您需要显示所有的服务（包括未运行的服务），需要添加-all参数，使用命令如下：

```
systemctl list-units --type service --all
```

例如显示当前正在运行的服务，命令如下：

```
$ systemctl list-units --type service
UNIT                            LOAD   ACTIVE SUB     DESCRIPTION
abrt-ccpp.service               loaded active exited  Install ABRT coredump hook
abrt-oops.service               loaded active running ABRT kernel log watcher
abrt-vmcore.service             loaded active exited  Harvest vmcores for ABRT
abrt-xorg.service               loaded active running ABRT Xorg log watcher
abrtd.service                   loaded active running ABRT Automated Bug Reporting Tool
...systemd-vconsole-setup.service loaded active exited  Setup Virtual Consoletog-
pegasus.service             loaded active running OpenPegasus CIM Server

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

46 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'
```

显示服务状态

如果您需要显示某个服务的状态，可执行如下命令：

```
systemctl status name.service
```

相关状态显示参数说明如表6-4所示。

表 6-4 状态参数说明

参数	描述
Loaded	说明服务是否被加载，并显示服务对应的绝度路径以及是否启用。
Active	说明服务是否正在运行，并显示时间节点。
Main PID	相应的系统服务的PID值。

参数	描述
Status	相关系统服务的其他信息。
Process	相关进程的其他信息。
CGroup	相关控制组（CGroup）的其他信息。

如果您需要鉴别某个服务是否运行，可执行如下命令：

```
systemctl is-active name.service
```

同样，如果您需要判断某个服务是否被启用，可执行如下命令：

```
systemctl is-enabled name.service
```

例如查看gdm.service服务状态，命令如下：

```
# systemctl status gdm.service
gdm.service - GNOME Display Manager   Loaded: loaded (/usr/lib/systemd/system/gdm.service;
enabled)   Active: active (running) since Thu 2013-10-17 17:31:23 CEST; 5min ago
 Main PID: 1029 (gdm)
   CGroup: /system.slice/gdm.service
           ├─1029 /usr/sbin/gdm
           ├─1037 /usr/libexec/gdm-simple-slave --display-id /org/gno...           └─
1047 /usr/bin/Xorg :0 -background none -verbose -auth /r...Oct 17 17:31:23 localhost systemd[1]:
Started GNOME Display Manager.
```

运行服务

如果您需要运行某个服务，请在root权限下执行如下命令：

```
systemctl start name.service
```

例如运行httpd服务，命令如下：

```
# systemctl start httpd.service
```

关闭服务

如果您需要关闭某个服务，请在root权限下执行如下命令：

```
systemctl stop name.service
```

例如关闭蓝牙服务，命令如下：

```
# systemctl stop bluetooth.service
```

重启服务

如果您需要重启某个服务，请在root权限下执行如下命令：

```
systemctl restart name.service
```

执行命令后，当前服务会被关闭，但马上重新启动。如果您指定的服务，当前处于关闭状态，执行命令后，服务也会被启动。

例如重启蓝牙服务，命令如下：

```
# systemctl restart bluetooth.service
```


启用服务

如果您需要在开机时启用某个服务，请在root权限下执行如下命令：

```
systemctl enable name.service
```

例如设置httpd服务开机时启动，命令如下：

```
# systemctl enable httpd.serviceln -s '/usr/lib/systemd/system/httpd.service' '/etc/systemd/system/multi-user.target.wants/httpd.service'
```

禁用服务

如果您需要在开机时禁用某个服务，请在root权限下执行如下命令：

```
systemctl disable name.service
```

例如在开机时禁用蓝牙服务启动，命令如下：

```
# systemctl disable bluetooth.servicerm '/etc/systemd/system/dbus-org.bluez.service'rm '/etc/systemd/system/bluetooth.target.wants/bluetooth.service'
```

6.4 改变运行级别

Target 和运行级别

systemd用目标（target）替代了运行级别的概念，提供了更大的灵活性，如您可以继承一个已有的目标，并添加其他服务，来创建自己的目标。下表列举了systemd下的目标和常见runlevel的对应关系：

表 6-5 运行级别和 systemd 目标

运行级别	systemd目标（target）	描述
0	runlevel0.target，poweroff.target	关闭系统。
1	runlevel1.target，rescue.target	单用户模式。
2	runlevel2.target，multi-user.target	用户定义/域特定运行级别。默认等同于3。
3	runlevel3.target，multi-user.target	多用户，非图形化。用户可以通过多个控制台或网络登录。
4	runlevel4.target，multi-user.target	用户定义/域特定运行级别。默认等同于3。
5	runlevel5.target，graphical.target	多用户，图形化。通常为所有运行级别3的服务外加图形化登录。
6	runlevel6.target，reboot.target	重启系统。

查看系统默认目标

查看当前系统默认的启动级别，使用如下命令：

```
systemctl get-default
```

查看当前系统的目标

查看当前系统默认的启动级别，使用如下命令：

```
systemctl list-units --type target
```

改变默认目标

改变系统默认的目标，在root权限下使用如下命令：

```
systemctl set-default name.target
```

改变当前目标

改变当前系统的目标，在root权限下使用如下命令：

```
systemctl isolate name.target
```

切换到救援模式

改变当前系统为救援模式，在root权限下使用如下命令：

```
systemctl rescue
```

这条命令和“systemctl isolate rescue.target”类似，但它会给当前所有的登录用户发送一条提示消息。如果想避免systemd发送这个消息，您可以添加“--no-wall”参数。具体命令如下：

```
systemctl --no-wall rescue
```

说明

用户需要重启系统，从救援模式进入正常模式。

切换到紧急模式

改变当前系统为紧急模式，在root权限下使用如下命令：

```
systemctl emergency
```

这条命令和“systemctl isolate emergency.target”类似，但它会给当前所有的登录用户发送一条提示消息。如果想避免systemd发送这个消息，您可以添加“--no-wall”参数。具体命令如下：

```
systemctl --no-wall emergency
```

说明

用户需要重启系统，从紧急模式进入正常模式。

6.5 关闭、暂停和休眠系统

systemctl 命令

systemd通过systemctl命令可以对系统进行关机、重启、休眠等已系列操作。当前仍兼容linux常用管理命令，对应关系如下表。建议用户使用systemctl命令进行操作。

表 6-6 命令对应关系

之前命令	systemctl命令	描述
halt	systemctl halt	关闭系统
poweroff	systemctl poweroff	关闭电源
reboot	systemctl reboot	重启
pm-suspend	systemctl suspend	待机
pm-hibernate	systemctl hibernate	休眠
pm-suspend-hybrid	systemctl hybrid-sleep	缓和休眠模式

关闭系统

要关闭系统并下电，在root权限下执行如下命令：

```
systemctl poweroff
```

要关闭系统但不下电机器，在root权限下执行如下命令：

```
systemctl halt
```

执行上述会给当前所有的登录用户发送一条提示消息。如果想避免systemd发送这个消息，您可以添加“--no-wall”参数。具体命令如下：

```
systemctl --no-wall poweroff
```

重启系统

要关重启系统，在root权限下执行如下命令：

```
systemctl reboot
```

执行上述会给当前所有的登录用户发送一条提示消息。如果想避免systemd发送这个消息，您可以添加“--no-wall”参数。具体命令如下：

```
systemctl --no-wall reboot
```

使系统待机

要使系统待机，在root权限下执行如下命令：

```
systemctl suspend
```


使系统休眠

要使系统休眠，在root权限下执行如下命令：

```
systemctl hibernate
```

要使系统待机且处于休眠状态，在root权限下执行如下命令：

```
systemctl hybrid-sleep
```


7 管理进程

本章介绍了Linux内核的进程管理方式，然后以实例的方式讲解了Linux提供的常用的进程控制命令、at和cron服务，以及进程查看命令。

7.1 管理系统进程

操作系统管理多个用户的请求和多个任务。大多数系统都只有一个CPU和一个主要存储，但一个系统可能有多个二级存储磁盘和多个输入/输出设备。操作系统管理这些资源并在多个用户间共享资源，当用户提出一个请求时，造成好像系统被用户独占的假象。实际上操作系统监控着一个等待执行的任务队列，这些任务包括用户任务、操作系统任务、邮件和打印任务等。本节将从用户的角度讲述如何控制进程。

7.1.1 调度启动进程

有时候需要对系统进行一些比较费时而且占用资源的维护工作，这些工作适合在深夜进行，这时候用户就可以事先进行调度安排，指定任务运行的时间或者场合，到时候系统会自动完成这些任务。要使用自动启动进程的功能，就需要掌握以下几个启动命令。

7.1.1.1 定时运行一批程序（at）

at 命令

用户使用at命令在指定时刻执行指定的命令序列。该命令至少需要指定一个命令和一个执行时间。at命令可以只指定时间，也可以时间和日期一起指定。

at命令的语法格式如下：

```
at [-V] [-q 队列] [-f 文件名] [-mldbv] 时间
at -c作业[作业…]
```

设置时间

at允许使用一套相当复杂的时间指定方法，比如：

- 接受在当天的hh:mm（小时：分钟）式的时间指定。如果该时间已经过去，那么就放存第二天执行。
- 使用midnight（深夜）、noon（中午）、teatime（饮茶时间，一般是下午4点）等比较模糊的词语来指定时间。
- 采用12小时计时制，即在时间后面加上AM（上午）或者PM（下午）来说明是上午还是下午。
- 指定命令执行的具体日期，指定格式为month day（月日）或者mm/dd/yy（月/日/年）或者dd.mm.yy（日.月.年）。指定的日期必须跟在指定时间的后面。

上面介绍的都是绝对计时法，其实还可以使用相对计时法，这对于安排不久就要执行的命令是很有好处的。指定格式为now+count time-units，now就是当前时间，time-units是时间单位，这里可以是minutes（分钟）、hours（小时）、days（天）、weeks（星期）。count是时间的数量，究竟是几天，还是几小时等。还有一种计时方法就是直接使用today（今天）、tomorrow（明天）来指定完成命令的时间。下面通过一些例子来说明具体用法。

例如指定在今天下午4:30执行某个命令。假设现在时间是中午12:30，2015年6月7日，可用命令格式如下：

```
at 4:30pm
at 16:30
at 16:30 today
at now+4 hours
at now+ 240 minutes
at 16:30 7.6.15
at 16:30 6/7/15
at 16:30 Jun 7
```

以上这些命令表达的意义是完全一样的，所以在安排时间的时候完全可以根据个人喜好和具体情况自由选择。一般采用绝对时间的24小时计时法可以避免由于用户自己的疏忽造成计时错误，例如上例可以写成：at 16:30 6/7/15。

执行权限

对于at命令来说，需要定时执行的命令是从标准输入或者使用-f选项指定的文件中读取并执行的。如果at命令是从一个使用su命令切换到用户shell中执行的，那么当前用户被认为是执行用户，所有的错误和输出结果都会送给这个用户。但是如果有邮件送出的话，收到邮件的将是原来的用户，也就是登录时shell的所有者。

例如在6月8日上午10点执行slocate -u命令。命令如下：

```
# at  10:00  6/8/15
at> slocate -u
at>
[1]+   Stopped    at  10:00  6/8/15
```

上面的结果中，输入at命令之后，会出现提示符at>，提示用户输入命令，在此输入了slocate -u，然后按回车键。还可以输入多条命令，当所有要执行的命令输入结束后，按Ctrl+d键结束at命令。

在任何情况下，管理员账户都可以使用这个命令。对于其他用户来说，是否可以使用就取决于/etc/at.allow和/etc/at.deny文件。

7.1.1.2 周期性运行一批程序（cron）

前面介绍at命令都会在一定时间内完成一定任务，但是它只能执一次。也就是说，当指定了运行命令后，系统在指定时间完成任务，以后就不再执行了。但是在很多情况下需要周期性重复执行一些命令，这时候就需要使用cron命令来完成任务。

运行机制

首先cron命令会搜索/var/spool/cron目录，寻找以/etc/passwd文件中的用户名命名的crontab文件，被找到的这种文件将装入内存。比如一个用户名为globus的用户，对应的crontab立件应该是/var/spool/cron/globus，即以该用户命名的crontab文件存放在/var/spool/cron目录下面。

cron命令还将搜索/etc/crontab文件，这个文件是用不同的格式写成的。cron启动以后，它将首先检查是否有用户设置了crontab文件，如果没有就转入睡眠状态，释放系统资源。所以该后台进程占用资源极少，它每分钟被换醒一次，查看当前是否有需要运行的命令。

命令执行结束后，任何输出都将作为邮件发送给crontab的所有者，或者是/etc/crontab文件中MAILTO环境变量中指定的用户。这是cron的工作原理，但是cron命令的执行不需要用户干涉，用户只需要修改crontab中要执行的命令。

crontab 命令

crontab命令用于安装、删除或者显示用于驱动cron后台进程的表格。用户把需要执行的命令序列放到crontab文件中以获得执行，而且每个用户都可以有自己的crontab文件。

crontab命令的常用方法如下：

- crontab -u //设置某个用户的cron服务，root用户在执行crontab时需要此参数。
- crontab -l //列出某个用户cron服务的详细内容。
- crontab -r //删除某个用户的cron服务。
- crontab -e //编辑某个用户的cron服务。

例如root查看自己的cron设置。命令如下：

```
crontab -u root -l
```

crontab 文件

在crontab文件中输入需要执行的命令和时间。该文件中每行都包括6个域，其中前5个域是指定命令被执行的时间，最后一个域是要被执行的命令。每个域之间使用空格或者制表符分隔。格式如下：

```
minute hour day-of-month month-of-year day-of-week commands
```

对于每一项的说明如所示。

表 7-1 参数说明

参数	描述
minute	分钟（0~59）。
hour	小时（0~23）。
day-of-month	一个月（）的第几天（1~31）。
month-of-year	一年的第几个月（1~12）。
day-of-week	一周的星期几（0~6），0代表星期天。
commands	需要执行的命令。

这些项都不能为空，必须指定值。除了数字还有几个特殊的符号“*”、“/”和“-”、“，”。其中，*代表所有的取值范围内的数字，/代表每的意思，“*/5”表示每5个单位，“-”代表从某个数字到某个数字，“，”分开几个离散时数字。对于要执行的命令，调用的时候需要写出命令的完整路径。

例如晚上11点到早上8点之间每两个小时，在/tmp/test.txt文件中加入sleepy文本。在crontab文件中对应的行如下：

```
* 23-8/2 * * * echo"sleepy" >> /tmp/test.txt
```

每次编辑完某个用户的cron设置后，cron自动在/var/spool/cron下生成一个与此用户同名的文件。此用户的cron信息都记录在这个文件中，这个文件是不可以直接编辑的，只可以用crontab -e来编辑。用户也可以另外建立一个文件，使用“cron文件名”命令导入cron设置。

假设有个用户名为globus，它需要为自己创建的一个crontab文件。步骤如下：

1. 首先可以使用任何文本编辑器建立一个新文件，并将向该文件加入需要运行的命令和要定期执行的时间，假发该文件为 ~/globus.cron。
2. 然后使用crontab命令安装这个文件，使用crontab命令使之成为该用户的crontab文件。命令如下：
   ```
   crontab  globus. -/globus.cron
   ```

这样crontab文件就建立好了，可以转到/var/spool/cron目录下面查看，发现多了一个globus文件。这个文件就是所需的crontab文件。

说明

cron启动后，每过一分钟读一次crontab文件，检查是否要执行里面的命令。因此该文件被修改后不需要重新启动cron服务。

编辑配置文件

cron服务每分钟不仅要读一次/var/spool/cron内的所有文件，还需要读一次/etc/crontab，因此通过配置这个文件也能得到cron的服务。用crontab配置是针对某个用户的，而编辑/etc/crontab是针对系统的任务。此文件的文件格式如下：

```
SHELL=/bin/sh
PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root  //如果出现错误，或者有数据输出，数据作为邮件发给这个账号
HOME=/
#  run-parts
01  * * * *   root run-parts /etc/cron.hourly     //每个小时执行一次/etc/cron.hourly里的脚本
02 4 * * *   root run-parts /etc/cron.daily    //每天执行一次/etc/cron.daily里的脚本
22 4 * * 0  root run-parts /etc/cron.weekly     //每周执行一次/etc/cron.weekly里的脚本
42 4 1  * *  root run-parts /etc/cron.monthly     //每月执行一次/etc/cron.monthly里的脚本
```

说明

如果去掉run-parts参数，其后面就是运行的某个脚本名，而不是目录名。

7.1.2 挂起/恢复进程

作业控制允许进程挂起并可以在需要时恢复进程的运行，被挂起的作业恢复后将从中止处开始继续运行。只要在键盘上按Ctrl+Z键，即可挂起当前的前台作业。在键盘上按Ctrl+Z键后，将挂起当前执行的命令cat。使用jobs命令可以显示shell的作业清单，包括具体的作业、作业号以及作业当前所处的状态。

恢复进程执行时，有两种选择：用fg命令将挂起的作业放回到前台执行；用bg命令将挂起的作业放到后台执行。灵活使用上述命令，将给自己带来很大的方便。

7.2 查看进程

Linux是一个多任务系统，经常需要对这些进程进行一些调配和管理。要进行管理，首先就要知道现在的进程情况：有哪些进程、进程的状态如何等。Linux提供了多种命令来了解进程的状况。

who 命令

who命令主要用于查看当前系统中的用户情况。如果用户想和其他用户建立即时通讯，比如使用talk命令，那么首先要确定的就是该用户确实在线上，不然talk进程就无法建立起来。又如，系统管理员希望监视每个登录的用户此时此刻的所作所为，也要使用who命令。who命令应用起来非常简单，可以比较准确地掌握用户的情况，所以使用非常广泛。

例如查看系统中的用户及其状态。使用如下：

```
# who
admin     tty1         Jul 28 15:55
admin     pts/0        Aug  5 15:46 (192.168.0.110)
admin     pts/2        Jul 29 19:52 (192.168.0.110)
root     pts/3        Jul 30 12:07 (192.168.0.110)
root     pts/4        Jul 31 10:29 (192.168.0.144)
root     pts/5        Jul 31 14:52 (192.168.0.11)
root     pts/6        Aug  6 10:12 (192.168.0.234)
root     pts/8        Aug  6 11:34 (192.168.0.234)
```

ps 命令

ps命令是最基本又非常强大的进程查看命令。使用该命令可以确定有哪些进程正在运行和运行的状态、进程是否结束、进程有没有僵尸、哪些进程占用了过多的资源等，大部分进程信息都是可以通过执行该命令得到的。

ps命令最常用的还是用来监控后台进程的工作情况，因为后台进程是不与屏幕、键盘这些标准输入/输出设备进行通信的，所以如果需要检测其状况，就可使用ps命令。ps命令的常见选项如表7-2所示。

表 7-2 选项说明

选项	描述
-e	显示所有进程。
-f	全格式。
-h	不显示标题。
-l	使用长格式。
-w	宽行输出。
-a	显示终端上的所有进程，包括其他用户的进程。
-r	只显示正在运行的进程。

选项	描述
-x	显示没有控制终端的进程。

例如显示系统中终端上的所有进行进程。命令如下：

```
# ps -a
  PID TTY          TIME CMD
12175 pts/6    00:00:00 bash
24526 pts/0    00:00:00 vsftpd
29478 pts/5    00:00:00 ps
32461 pts/0    1-01:58:33 sh
```

top 命令

top命令和ps命令的基本作用是相同的，显示系统当前的进程和其他状况，但是top是一个动态显示过程，即可以通过用户按键来不断刷新进程的当前状态，如果在前台执行该命令，它将独占前台，直到用户终止该程序为止。其实top命令提供了实时的对系统处理器的状态监视。它将显示系统中CPU的任务列表。该命令可以按CPU使用、内存使用和执行时间对任务进行排序，而且该命令的很多特性都可以通过交互式命令或者在定制文件中进行设定。

top命令输出的实例如图7-1所示：

图 7-1 top 显示

```
top - 19:04:08 up 9 days,  3:09,  8 users,  load average: 2.17, 2.08, 2.06
Tasks: 242 total,   8 running, 234 sleeping,   0 stopped,   0 zombie
Cpu(s):  8.3%us,  0.2%sy,  0.0%ni, 91.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:     19983M total,    19777M used,      206M free,      567M buffers
Swap:     2053M total,       10M used,     2043M free,    12326M cached

  PID USER      PR  NI  VIRT  RES  SHR S  %CPU %MEM    TIME+  COMMAND
32757 root      20   0 4462m 3.0g 5440 S   100 15.3   1542:40 qemu-kvm
32461 root      20   0 11580 1380 1120 R   100  0.0   1563:47 sh
31437 root      20   0 4626m 2.4g 5436 R     4 12.1  14:36.89 qemu-kvm
29553 root      20   0 17256 1392  932 R     0  0.0   0:00.02 top
31438 root      20   0     0    0    0 S     0  0.0   0:12.80 vhost-31437
32758 root      20   0     0    0    0 S     0  0.0   0:25.21 vhost-32757
    1 root      20   0 10540  796  748 S     0  0.0   0:04.59 init
    2 root      20   0     0    0    0 S     0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S     0  0.0   0:01.64 ksoftirqd/0
    6 root      RT   0     0    0    0 S     0  0.0   0:01.08 migration/0
    7 root      RT   0     0    0    0 S     0  0.0   0:01.66 watchdog/0
    8 root      RT   0     0    0    0 S     0  0.0   0:01.09 migration/1
    9 root      20   0     0    0    0 S     0  0.0   0:05.58 kworker/1:0
   10 root      20   0     0    0    0 S     0  0.0   0:01.31 ksoftirqd/1
   11 root      20   0     0    0    0 S     0  0.0   0:50.48 kworker/0:1
   12 root      RT   0     0    0    0 S     0  0.0   0:01.27 watchdog/1
   13 root      RT   0     0    0    0 S     0  0.0   0:01.64 migration/2
   14 root      20   0     0    0    0 S     0  0.0   0:00.00 kworker/2:0
   15 root      20   0     0    0    0 S     0  0.0   1:01.89 ksoftirqd/2
   16 root      RT   0     0    0    0 S     0  0.0   0:01.38 watchdog/2
   17 root      RT   0     0    0    0 R     0  0.0   0:01.12 migration/3
   18 root      20   0     0    0    0 S     0  0.0   0:00.00 kworker/3:0
   19 root      20   0     0    0    0 S     0  0.0   0:22.84 ksoftirqd/3
   20 root      RT   0     0    0    0 S     0  0.0   0:01.33 watchdog/3
   21 root      RT   0     0    0    0 S     0  0.0   0:01.56 migration/4
   22 root      20   0     0    0    0 S     0  0.0   0:00.00 kworker/4:0
   23 root      20   0     0    0    0 S     0  0.0   0:00.01 ksoftirqd/4
   24 root      RT   0     0    0    0 S     0  0.0   0:01.29 watchdog/4
```


kill 命令

当需要中断一个前台进程的时候，通常足使用“Ctrl+c”组合键，而对于后台进程不能用组合健来终止，这时就可以使用kill命令。该命令可以终止前台和后台进程。终止后台进程的原因包括：该进程占用CPU的时间过多、该进程已经死锁等。

kill命令是通过向进程发送指定的信号来结束进程的。如果没有指定发送的信号，那么默认值为TERM信号。TERM信号将终止所有不能捕获该信号的进程。至于那些可以捕获该信号的进程可能就需要使用KILL信号（它的编号为9），而该信号不能被捕捉。

kill命令的浯法格式有以下两种方式：

```
kill [-s 信号 | -p] [-a] 进程号…
kill -l [信号]
```

其中进程号可以通过ps命令的输出得到。-s选项是给程序发送指定的信号，详细的信号可以用“kill -l”命令查看；-p选项只显示指定进程的ID号，而发送信号。

杀死pid为1409的进程，示例如下：

```
# kill -9 1409
```

显示所有的信号及其编号对应关系，示例如下：

```
# kill -l
 1) SIGHUP? 2) SIGINT? 3) SIGQUIT? 4) SIGILL
 5) SIGTRAP? 6) SIGABRT? 7) SIGBUS? 8) SIGFPE
 9) SIGKILL?10) SIGUSR1?11) SIGSEGV?12) SIGUSR2
13) SIGPIPE?14) SIGALRM?15) SIGTERM?16) SIGSTKFLT
17) SIGCHLD?18) SIGCONT?19) SIGSTOP?20) SIGTSTP
21) SIGTTIN?22) SIGTTOU?23) SIGURG?24) SIGXCPU
25) SIGXFSZ?26) SIGVTALRM?27) SIGPROF?28) SIGWINCH
29) SIGIO?30) SIGPWR?31) SIGSYS?34) SIGRTMIN
35) SIGRTMIN+1?36) SIGRTMIN+2?37) SIGRTMIN+3?38) SIGRTMIN+4
39) SIGRTMIN+5?40) SIGRTMIN+6?41) SIGRTMIN+7?42) SIGRTMIN+8
43) SIGRTMIN+9?44) SIGRTMIN+10?45) SIGRTMIN+11?46) SIGRTMIN+12
47) SIGRTMIN+13?48) SIGRTMIN+14?49) SIGRTMIN+15?50) SIGRTMAX-14
51) SIGRTMAX-13?52) SIGRTMAX-12?53) SIGRTMAX-11?54) SIGRTMAX-10
55) SIGRTMAX-9?56) SIGRTMAX-8?57) SIGRTMAX-7?58) SIGRTMAX-6
59) SIGRTMAX-5?60) SIGRTMAX-4?61) SIGRTMAX-3?62) SIGRTMAX-2
```


8 HMI 使用说明

8.1 概述

8.1.1 HMI 介绍

Huawei Machine Image（简称HMI）基于EulerOS V2.0SP5（简称EulerOS 2.5）制作，提供云平台的启动实例（云中的虚拟服务器）。您在启动实例时指定 HMI，可以从该 HMI 启动所需任意数量的实例，您还可以根据需要从任意多个不同 HMI 启动实例。

HMI 包括以下内容：

- 一个用于实例（例如，操作系统、应用程序服务器和应用程序）根卷的模板。
- 控制可以使用 HMI 启动实例的账户的启动许可。
- 一个指定在实例启动时要附加到实例的卷的块储存设备映射。

8.1.2 产品特点

安全性

- 签名认证：对所有安装RPM进行签名认证。
- 证书登陆：默认情况下不允许用户通过密码进行远程登陆，只允许通过发放的证书远程登陆。

  说明

  密码远程登录需要用户通过本地登录服务器，将/etc/ssh/sshd_config文件中的PasswordAuthentication禁用取消，并将可用状态由no改为yes。

- 动态密码：不设置静态默认密码，全部证书密码通过cloud-init注入。

  说明

  此项需要云平台支持cloud-init服务。

- 补丁更新提醒：当repo源中有新版本或CVE修复补丁更新时，系统通过登录界面或邮件提醒用户进行更新。
- 安全加固：HMI默认经过完备的安全加固，系统更加安全可靠。

  说明

  详细加固项请参考《EulerOS V2.0SP5安全加固指南》。

易用性

- 集成云场景工具：默认集成云场景必要软件包(cloud-init、cloud-utils-growpart等)，避免用户重复安装。
- 设置常用策略：设置常用logrotate日志切割及转储策略，减少用户工作量。
- 提供丰富的业务软件：提供软件仓库（repo源），包含常用的业务软件，满足用户多样化的业务需求。
- 一键式安装：默认对接repo源，一键式安装所需软件，更快捷。

性能

- 组件最小化：全面梳理云服务场景，裁剪非必要的软件包及服务，实现组件最小化，系统更精简。
- 启动更迅速：系统启动速度比默认安装情况下，启动快约20%。
- IO性能：通过定制特定的内核参数，在华为公有云上实现IO性能的有效提升。
- 网络性能：设置特定内核参数，使得网络性能得到更好的发挥。

8.2 首次启动

8.2.1 远程登录

1. 进入华为企业云网页（**http://www.hwclouds.com/**），管理控制台界面，选择“弹性云服务器”，点击所要登录的vm的“远程登录”按钮，如**图 1**。

   说明

   这里以华为企业云登录为例，其他云平台登录界面请参考云平台自身的说明。

   图 8-1 远程登录

2. 输入用户名和口令登录系统。

   说明

   默认仅有root用户，root的默认密码为 Huawei@SYS3，强烈建议用户登录后第一时间修改root用户的口令。

8.2.2 查看系统信息

- 使用如下命令查看系统信息：

```
cat /etc/os-release
```

输出如下：

```
[root@localhost ~]# cat /etc/os-release
NAME="EulerOS"
VERSION="2.0 (SP5)"
ID="euleros"
ID_LIKE="rhel fedora centos"
VERSION_ID="2.0"
PRETTY_NAME="EulerOS 2.0 (SP5)"
ANSI_COLOR="0;31"
```

- 查看系统相关的资源信息，如下：

查看CPU信息，可使用如下命令：

```
lscpu
```

查看内存信息，可使用如下命令：

```
free
```

查看磁盘信息，可使用如下命令：

```
fdisk -l
```

8.2.3 配置 repo 源

- repo地址

http://developer.huawei.com/ict/site-euleros/euleros/repo/yum/

说明

此处的地址为示例，不同云平台的repo源地址请参考云平台的说明。

- repo配置

在/etc/yum.repos.d/下，EulerOS-2.5-Base.repo文件默认配置如下：

```
[base]
name=EulerOS-base
baseurl=http://developer.huawei.com/ict/site-euleros/euleros/
repo/yum/2.x/os/x86_64/
enabled=1
gpgcheck=1
gpgkey=http://developer.huawei.com/ict/site-euleros/euleros/
repo/yum/2.x/os/RPM-GPG-KEY-EulerOS
```

说明

用户若需要配置其他的yum源，请参照以上格式对EulerOS-2.5-Base.repo文件进行修改。

8.2.4 安装业务软件

配置repo源之后，即可通过 yum install 命令安装所需要软件。

如下示例，演示成功安装gcc软件，请按照提示进行操作。安装过程如下：

```
[root@localhost ~]# yum install gcc
Loaded plugins: priorities
base                                                    | 1.2 kB  00:00:00
base/primary                                            | 3.3 MB  00:00:00
base                                                   15946/15946
Resolving Dependencies
--> Running transaction check
---> Package gcc.x86_64 0:4.8.5-4.h1 will be installed
--> Processing Dependency: cpp = 4.8.5-4.h1 for package: gcc-4.8.5-4.h1.x86_64
```



```
--> Processing Dependency: glibc-devel >= 2.2.90-12 for package: gcc-4.8.5-4.h1.x86_64
--> Processing Dependency: libmpfr.so.4()(64bit) for package: gcc-4.8.5-4.h1.x86_64
--> Processing Dependency: libmpc.so.3()(64bit) for package: gcc-4.8.5-4.h1.x86_64
--> Running transaction check
---> Package cpp.x86_64 0:4.8.5-4.h1 will be installed
---> Package glibc-devel.x86_64 0:2.17-111.h9 will be installed
--> Processing Dependency: glibc-headers = 2.17-111.h9 for package: glibc-devel-2.17-111.h9.x86_64
--> Processing Dependency: glibc-headers for package: glibc-devel-2.17-111.h9.x86_64
---> Package libmpc.x86_64 0:1.0.1-3 will be installed
---> Package mpfr.x86_64 0:3.1.1-4 will be installed
--> Running transaction check
---> Package glibc-headers.x86_64 0:2.17-111.h9 will be installed
--> Processing Dependency: kernel-headers >= 2.2.1 for package: glibc-headers-2.17-111.h9.x86_64
--> Processing Dependency: kernel-headers for package: glibc-headers-2.17-111.h9.x86_64
--> Running transaction check
---> Package kernel-headers.x86_64 0:3.10.0-327.28.56.4 will be installed
--> Finished Dependency Resolution
Dependencies Resolved
================================================================================Package         Arch
Version           Repository       Size
================================================================================Installing:
gcc             x86_64       4.8.5-4.h1         base             16 M
Installing for dependencies:
cpp              x86_64       4.8.5-4.h1         base            5.9 M
glibc-devel      x86_64       2.17-111.h9        base            1.0 M
glibc-headers    x86_64      2.17-111.h9         base            664 k
kernel-headers   x86_64     3.10.0-327.28.56.4   base            3.2 M
libmpc           x86_64       1.0.1-3            base             51 k
mpfr           x86_64         3.1.1-4            base            204 k
Transaction Summary
================================================================================Install  1 Package (+6
Dependent packages)
Total download size: 27 M
Installed size: 59 M
Is this ok [y/d/N]: y
Downloading packages:
(1/7): cpp-4.8.5-4.h1.x86_64.rpm                    | 5.9 MB  00:00:00
(2/7): glibc-devel-2.17-111.h9.x86_64.rpm           | 1.0 MB  00:00:00
(3/7): glibc-headers-2.17-111.h9.x86_64.rpm         | 664 kB  00:00:00
(4/7): kernel-headers-3.10.0-327.28.56.4.x86_64.rpm| 3.2 MB  00:00:00
(5/7): libmpc-1.0.1-3.x86_64.rpm                    |  51 kB  00:00:00
(6/7): mpfr-3.1.1-4.x86_64.rpm                      | 204 kB  00:00:00
(7/7): gcc-4.8.5-4.h1.x86_64.rpm                    |  16 MB  00:00:00
--------------------------------------------------------------------------------
Total                                       44 MB/s |  27 MB  00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing : mpfr-3.1.1-4.x86_64                              1/7
Installing : libmpc-1.0.1-3.x86_64                            2/7
Installing : cpp-4.8.5-4.h1.x86_64                            3/7
Installing : kernel-headers-3.10.0-327.28.56.4.x86_64         4/7
Installing : glibc-headers-2.17-111.h9.x86_64                 5/7
Installing : glibc-devel-2.17-111.h9.x86_64                   6/7
Installing : gcc-4.8.5-4.h1.x86_64                            7/7
Verifying  : glibc-devel-2.17-111.h9.x86_64                   1/7
Verifying  : mpfr-3.1.1-4.x86_64                              2/7
Verifying  : glibc-headers-2.17-111.h9.x86_64                 3/7
Verifying  : kernel-headers-3.10.0-327.28.56.4.x86_64         4/7
Verifying  : gcc-4.8.5-4.h1.x86_64                            5/7
Verifying  : libmpc-1.0.1-3.x86_64                            6/7
Verifying  : cpp-4.8.5-4.h1.x86_64                            7/7
Installed:
gcc.x86_64 0:4.8.5-4.h1
Dependency Installed:
cpp.x86_64 0:4.8.5-4.h1    glibc-devel.x86_64 0:2.17-111.h9   glibc-headers.x86_64 0:2.17-111.h9
kernel-headers.x86_64 0:3.10.0-327.28.56.4       libmpc.x86_64 0:1.0.1-3      mpfr.x86_64 0:3.1.1-4
Complete!
```


8.2.5 开发应用程序

HMI 的repo源中提供了一套完整的 Linux 开发工具。要开发应用程序，请通过 yum 选择所需的开发工具。另外，大部分 CentOS 和其他类似发行版上开发的应用程可以直接运行在 HMI 上。

8.3 日常维护

8.3.1 日常更新

软件更新检测

配置repo源，参考**8.2.3 配置repo源**。检测是否有软件包需要更新，执行命令：

```
yum check-update
```

结果如下：

```
Loaded plugins: fastestmirror, langpacks, priorities
Loading mirror speeds from cached hostfile
ModemManager.x86_64            1.1.0-8.git20130913            base
ModemManager-glib.x86_64       1.1.0-8.git20130913            base
NetworkManager.x86_64          1:1.0.6-27                     base
NetworkManager-adsl.x86_64     1:1.0.6-27                     base
NetworkManager-glib.x86_64      1:1.0.6-27                    base
NetworkManager-libnm.x86_64        1:1.0.6-27                 base
NetworkManager-libreswan.x86_64    1.0.6-3                    base
NetworkManager-team.x86_64         1:1.0.6-27                 base
NetworkManager-tui.x86_64          1:1.0.6-27                 base
PackageKit.x86_64                    1.0.7-5                  base
PackageKit-command-not-found.x86_64  1.0.7-5                  base
PackageKit-glib.x86_64                 1.0.7-5                base
PackageKit-gstreamer-plugin.x86_64     1.0.7-5                base
PackageKit-gtk3-module.x86_64            1.0.7-5              base
PackageKit-yum.x86_64                    1.0.7-5              base
SDL.x86_64                               1.2.15-14            base
abattis-cantarell-fonts.noarch           0.0.16-3             base
abrt.x86_64                            2.1.11-36              base
abrt-addon-ccpp.x86_64               2.1.11-36                base
```

升级部分或全部软件包，参考**8.3.2 升级软件**。

配置安装 update-motd

update-motd已集成到EulerOS系统。

查看update-motd软件是否安装，执行命令：

```
rpm -qa | grep update-motd
```

没有安装update-motd，参考**8.2.4 安装业务软件**进行安装，执行命令：

```
yum install update-motd
```

安装完成后执行下面命令启用服务：

```
systemctl daemon-reload
systemctl enable update-motd.timer
systemctl start update-motd.timer
```

服务启动后，重新登录时会显示系统信息及需要更新软件数目。


```
Last login: Thu Jan  5 21:34:35 2017 from 192.168.1.230
*********************************************************************
*                                                                   *
* This system is for the use of authorized users only.  Usage of   *
* this system may be monitored and recorded by system personnel.    *
*                                                                   *
*********************************************************************

System load:   0.00
Memory usage:  8.2%             System uptime: 14 days, 1 hours
Usage on /:    90%              Swap usage:    0.0%
IP Addresses:  9.41.31.207

Pending Updates: 736
```

说明

该更新包括：组更新、已安装包的更新。

8.3.2 升级软件

1. 升级全部软件，执行命令如下：

   ```
   yum update
   ```

   执行结果如下：

   ```
   Loaded plugins: fastestmirror, langpacks, priorities
   Loading mirror speeds from cached hostfile
   Resolving Dependencies
   --> Running transaction check
   ---> Package ModemManager.x86_64 0:1.1.0-6.git20130913.el7 will be updated
   ---> Package ModemManager.x86_64 0:1.1.0-8.git20130913 will be an update
   ---> Package ModemManager-glib.x86_64 0:1.1.0-6.git20130913.el7 will be updated
   ---> Package ModemManager-glib.x86_64 0:1.1.0-8.git20130913 will be an update
   ---> Package NetworkManager.x86_64 1:1.0.0-14.git20150121.b4ea599c.el7 will be updated
   ---> Package NetworkManager.x86_64 1:1.0.6-27 will be an update
   ---> Package NetworkManager-adsl.x86_64 1:1.0.0-14.git20150121.b4ea599c.el7 will be updated
   ---> Package NetworkManager-adsl.x86_64 1:1.0.6-27 will be an update
   ---> Package NetworkManager-glib.x86_64 1:1.0.0-14.git20150121.b4ea599c.el7 will be updated
   ---> Package NetworkManager-glib.x86_64 1:1.0.6-27 will be an update
   ---> Package NetworkManager-libnm.x86_64 1:1.0.0-14.git20150121.b4ea599c.el7 will be updated
   ```

2. 升级部分软件包，执行命令如下：

   ```
   yum update -y aaa bbb ccc
   ```

   示例结果如下：

   ```
   Loaded plugins: fastestmirror, langpacks, priorities
   Loading mirror speeds from cached hostfile
   Resolving Dependencies
   --> Running transaction check
   ---> Package mkrepo.noarch 0:0.0.1-1 will be updated
   ---> Package mkrepo.noarch 0:0.0.1-6 will be an update
   --> Finished Dependency Resolution

   Dependencies Resolved

   ================================================================================
    Package             Arch               Version                Repository          Size
   ================================================================================
   Updating:
    mkrepo              noarch             0.0.1-6                base               5.5 k

   Transaction Summary
   ================================================================================
   Upgrade  1 Package
   ```

3. 使用yum获取软件包信息，执行命令如下：

   ```
   yum info aaa
   ```

   示例结果如下：


```
Loaded plugins: fastestmirror, langpacks, priorities
Loading mirror speeds from cached hostfile
Installed Packages
Name        : anaconda-core
Arch        : x86_64
Version     : 19.31.123
Release     : 1.el7.centos.2
Size        : 6.8 M
Repo        : installed
From repo   : anaconda
Summary     : Core of the Anaconda installer
URL         : http://fedoraproject.org/wiki/Anaconda
License     : GPLv2+
Description : The anaconda-core package contains the program which was used to install
            : your system.

Available Packages
Name        : anaconda-core
Arch        : x86_64
Version     : 21.48.22.56
Release     : 5.1.h2
Size        : 1.5 M
Repo        : local
Summary     : Core of the Anaconda installer
URL         : http://fedoraproject.org/wiki/Anaconda
License     : GPLv2+ and MIT
Description : The anaconda-core package contains the program which was used to install
            : your system.
```

4. 列出所有已安装的软件包，执行命令如下：

```
yum list installed
```

执行结果如下：

```
Installed Packages
NetworkManager.x86_64                       1:1.0.6-27                  @anaconda/rawhide
NetworkManager-config-server.x86_64         1:1.0.6-27                  @anaconda/rawhide
NetworkManager-libnm.x86_64                 1:1.0.6-27                  @anaconda/rawhide
NetworkManager-team.x86_64                  1:1.0.6-27                  @anaconda/rawhide
NetworkManager-tui.x86_64                   1:1.0.6-27                  @anaconda/rawhide
acl.x86_64                                  2.2.51-12                   @anaconda/rawhide
aic94xx-firmware.noarch                     30-6                        @anaconda/rawhide
alsa-firmware.noarch                        1.0.28-2                    @anaconda/rawhide
alsa-lib.x86_64                             1.0.28-2                    @anaconda/rawhide
alsa-tools-firmware.x86_64                  1.0.28-2                    @anaconda/rawhide
at.x86_64                                   3.1.13-20                   @anaconda/rawhide
attr.x86_64                                 2.4.46-12                   @anaconda/rawhide
audit.x86_64                                2.4.1-5                     @anaconda/rawhide
audit-libs.x86_64                           2.4.1-5                     @anaconda/rawhide
authconfig.x86_64                           6.2.8-10                    @anaconda/rawhide
...
```

说明

aaa/bbb/ccc为软件包名。

8.3.3 回退删除软件

1. 回退到上一个版本，执行如下命令：

```
yum downgrade aaa
```

2. 回退到具体的版本，执行如下命令：

```
yum downgrade aaa-xx
```

3. 删除软件，执行如下命令：

```
yum remove aaa
```

说明

aaa为软件包名，xx为版本号。

8.4 Yum 安全签名

8.4.1 签名机制说明

GNU Privacy Guard (GnuPG or GPG)是一个GPL许可证的软件加密工具，和PGP比较类似。与许多非对称加密方法一样，GPG和PGP使用一个密钥对——公钥和私钥——来加密数据。公钥可被广泛传播，甚至保存在公共密匙数据库中（如Euler Yum 仓库与本地ISO中）以被其他用户查阅。私钥属于私密信息，不会泄漏给其他人。公匙和私匙相互作用对数据进行加密及解密。被公匙加密的数据只能被私匙解密，被私匙加密的数据也只能被一个公匙解密。这样就可以实现双重认证。当用户想发送关键信息给其他人时，首先用户使用他的私匙来加密信息，然后发送给有公钥的其他人。因为只有使用发送者的公钥才能对接收信息进行解密，这样接收者就能确信信息的确来自某个人。Redhat的RPM软件包签名 就是使用一组 GPG 密钥对来验证软件包的合法性。Euler Yum仓库上与发行版的ISO中的 二进制包，都是经过了Euler 的私钥签名的，用户只需要用公钥进行校验即可。

8.4.2 签名导入

- 用户可以通过从Euler Yum仓库导入签名：
  ```
  rpm --import http://developer.huawei.com/ict/site-euleros/euleros/repo/yum/2.x/os/RPM-GPG-KEY-EulerOS
  ```
- 也可以从ISO中直接导入：
  ```
  rpm --import RPM-GPG-KEY-EulerOS
  ```
- 用户可以通过修改Euler Yum配置文件中的gpgcheck和gpgkey来使能安全签名：
  ```
  [root@localhost ~]# vim /etc/yum.repos.d/euleros.repo
  [Euler]
  name=Euler
  bsaseurl=https://developer.huawei.com/ict/site-euleros/euleros/repo/yum/2.x/os/x86_64/
  enabled=1
  gpgcheck=1
  gpgkey=http://developer.huawei.com/ict/site-euleros/euleros/repo/yum/2.x/os/RPM-GPG-KEY-EulerOS
  ```

8.4.3 签名查询

用户查询系统中存在的公钥

```
[root@Euler]# gpg -k
/root/.gnupg/pubring.gpg
------------------------
pub   4096R/30DB6168 2017-05-24
uid                  EulerOS (EulerOS 2.0) <EulerOS@huawei.com>
sub   4096R/04B315DB 2017-05-24
```

8.4.4 rpm 包签名校验

用户对rpm包进和签名校验：

```
[root@Euler ~]# rpm -K ftp-0.17-54.el6.x86_64.rpm
ftp-0.17-54.el6.x86_64.rpm: rsa sha1 (md5) pgp md5 OK
```

8.5 cloud-init 使用指导

8.5.1 概述

本章节主要介绍cloud-init的背景信息和一些基础概念。

cloudinit是专为云环境中虚拟机的初始化而开发的工具，它从各种数据源读取相关数据并据此对虚拟机进行配置。

常见的数据源包括：云平台的metadata服务、ConfigDrive等，常见的配置包括：设定虚拟机的hostname、hosts文件、设定用户名密码、调整文件系统的大小（注意不是调整分区的大小）等。

8.5.2 工作原理

首先，数据服务器开启HTTP服务，cloud-init会向数据服务器发送请求，确认数据源模块，从中获取元数据（meta data）和用户数据（user data），前者是指虚拟机的必要信息，如主机名、网络地址等；后者是系统或用户需要的数据和文件，如用户组信息、启动脚本等。当cloud-init获取这些信息后，开始使用一些模块对数据进行处理，如新建用户、启动脚本等。

cloud-init会在虚拟机启动的过程中，分4个阶段运行，按照时间顺序分别是：

- cloud_init_local
- cloud_init
- cloud_config
- cloud_final

cloud-init-local阶段主要是运行本地的一些初始化脚本。

cloud-init阶段执行配置文件中名为cloud_init_modules下的所有模块，如果模块列表为空，则什么都不运行。

其他两个阶段类似，分别执行配置文件中名为cloud_config_modules，cloud_final_moudles下的所有模块，如果模块列表为空，则什么都不运行。

另外，模块有多种运行模式，包括per-once、per-instance、per-always，对于模式为per-once的模块，一旦运行完毕会在一个名为sem的目录中创建一个信号文件，从而防止模块的在下次启动时重复运行。

8.5.3 配置 yum 源

该配置非操作系统的配置，是用户注入配置，由用户修改相关信息后，注入数据源（如openstack的userdata），在系统启动时，cloudinit会去数据源读取。

```
#CLOUD-CONFIG
# VIM: SYNTAX=YAML
#
# ADD YUM REPOSITORY CONFIGURATION TO THE SYSTEM
#
# THE FOLLOWING EXAMPLE ADDS THE FILE /ETC/YUM.REPOS.D/EULEROS.REPO
# WHICH CAN THEN SUBSEQUENTLY BE USED BY YUM FOR LATER OPERATIONS.
YUM_REPOS:
    # THE NAME OF THE REPOSITORY
    EULEROS:
       # ANY REPOSITORY CONFIGURATION OPTIONS
       # SEE: MAN YUM.CONF
       #
       # THIS ONE IS REQUIRED!
       BASEURL:HTTP://DEVELOPER.HUAWEI.COM/ICT/SITE-EULEROS/EULEROS/
```



```
REPO/YUM/2.x/OS/X86_64/
        ENABLED:FALSE
        FAILOVERMETHOD:PRIORITY
        GPGCHECK:TRUE
        GPGKEY:HTTP://DEVELOPER.HUAWEI.COM/ICT/SITE-EULEROS/EULEROS/
REPO/YUM/2.x/OS/RPM-GPG-KEY-EULEROS
        NAME:BASE
```

8.5.4 配置系统 user

该配置非操作系统的配置，是用户注入配置，由用户修改相关信息后，注入数据源（如openstack的userdata），在系统启动时，cloudinit会去数据源读取。

```
# Add users to the system. Users are added after groups are added.
users:
  - default
  - name: foobar
    gecos: Foo B. Bar
    primary-group: foobar
    groups: users
    selinux-user: staff_u
    expiredate: 2012-09-01
    ssh-import-id: foobar
    lock_passwd: false
    passwd: $6$j212wezy$7H/1LT4f9/
N3wpgNunhsIqtMj62OKiS3nyNwuizouQc3u7MbYCarYeAHWYPYb2FT.lbioDm2RrkJPb9BZMN1O/
  - name: barfoo
    gecos: Bar B. Foo
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: users, admin
    ssh-import-id: None
    lock_passwd: true
    ssh-authorized-keys:
      - <ssh pub key 1>
      - <ssh pub key 2>
  - name: cloudy
    gecos: Magic Cloud App Daemon User
    inactive: true
    system: true
  - snapuser: joe@joeuser.io

# Valid Values:
#   name: The user's login name
#   gecos: The user name's real name, i.e. "Bob B. Smith"
#   homedir: Optional. Set to the local path you want to use. Defaults to
#            /home/<username>
#   primary-group: define the primary group. Defaults to a new group created
#            named after the user.
#   groups:  Optional. Additional groups to add the user to. Defaults to none
#   selinux-user:  Optional. The SELinux user for the user's login, such as
#            "staff_u". When this is omitted the system will select the default
#            SELinux user.
#   lock_passwd: Defaults to true. Lock the password to disable password login
#   inactive: Create the user as inactive
#   passwd: The hash -- not the password itself -- of the password you want
#            to use for this user. You can generate a safe hash via:
#                mkpasswd --method=SHA-512 --rounds=4096
#            (the above command would create from stdin an SHA-512 password hash
#            with 4096 salt rounds)
#
#            Please note: while the use of a hashed password is better than
#                plain text, the use of this feature is not ideal. Also,
#                using a high number of salting rounds will help, but it should
#                not be relied upon.
#
#                To highlight this risk, running John the Ripper against the
#                example hash above, with a readily available wordlist, revealed
#                the true password in 12 seconds on a i7-2620QM.
```



```
#
#               In other words, this feature is a potential security risk and is
#               provided for your convenience only. If you do not fully trust the
#               medium over which your cloud-config will be transmitted, then you
#               should use SSH authentication only.
#
#               You have thus been warned.
#   no-create-home: When set to true, do not create home directory.
#   no-user-group: When set to true, do not create a group named after the user.
#   no-log-init: When set to true, do not initialize lastlog and faillog database.
#   ssh-import-id: Optional. Import SSH ids
#   ssh-authorized-keys: Optional. [list] Add keys to user's authorized keys file
#   sudo: Defaults to none. Set to the sudo string you want to use, i.e.
#           ALL=(ALL) NOPASSWD:ALL. To add multiple rules, use the following
#           format.
#               sudo:
#                   - ALL=(ALL) NOPASSWD:/bin/mysql
#                   - ALL=(ALL) ALL
#           Note: Please double check your syntax and make sure it is valid.
#               cloud-init does not parse/check the syntax of the sudo
#               directive.
#   system: Create the user as a system user. This means no home directory.
#   snapuser: Create a Snappy (Ubuntu-Core) user via the snap create-user
#             command available on Ubuntu systems.  If the user has an account
#             on the Ubuntu SSO, specifying the email will allow snap to
#             request a username and any public ssh keys and will import
#             these into the system with username specifed by SSO account.
#             If 'username' is not set in SSO, then username will be the
#             shortname before the email domain.
#

# Default user creation:
#
# Unless you define users, you will get a 'ubuntu' user on ubuntu systems with the
# legacy permission (no password sudo, locked user, etc). If however, you want
# to have the 'ubuntu' user in addition to other users, you need to instruct
# cloud-init that you also want the default user. To do this use the following
# syntax:
#    users:
#      - default
#      - bob
#      - ....
#  foobar: ...
#
# users[0] (the first user in users) overrides the user directive.
#
# The 'default' user above references the distro's config:
# system_info:
#   default_user:
#    name: Ubuntu
#    plain_text_passwd: 'ubuntu'
#    home: /home/ubuntu
#    shell: /bin/bash
#    lock_passwd: True
#    gecos: Ubuntu
#    groups: [adm, audio, cdrom, dialout, floppy, video, plugdev, dip, netdev]
```

8.5.5 配置系统 groups

该配置非操作系统的配置，是用户注入配置，由用户修改相关信息后，注入数据源（如openstack的userdata），在系统启动时，cloudinit会去数据源读取。

```
# Add groups to the system
# The following example adds the ubuntu group with members foo and bar and
# the group cloud-users.
groups:
   -ubuntu:[foo,bar]
   -cloud-users
```


8.5.6 配置 ssh key

该配置非操作系统的配置，是用户注入配置，由用户修改相关信息后，注入数据源（如openstack的userdata），在系统启动时，cloudinit会去数据源读取。

```
#cloud-config
# add each entry to ~/.ssh/authorized_keys for the configured user or the
# first user defined in the user definition directive.
ssh_authorized_keys:
   -ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAGEA3FSyQwBI6Z+nCSjUUk8EEAnnkhXlukKoUPND/
RRClWz2s5TCzIkd3Ou5+Cyz71XOXmazM3l5WgeErvtIwQMyT1KjNoMhoJMrJnWqQPOt5Q8zWd9qG7PBl9+eiH5qV7NZ
mykey@host
   -ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEA3I7VUf2l5gSn5uavROsc5HRDpZdQueUq5ozemNSj8T7enqKHOEaFoU2VoPgGEWC9RyzSQVe
yD6s7APMcE82EtmW4skVEgEGSbDc1pvxzxtchBj78hJP6Cf5TCMFSXw
+Fz5rF1dR23QDbN1mkHs7adr8GW4kSWqU7Q7NDwfIrJJtO7Hi42GyXtvEONHbiRPOe8stqUly7MvUoN
+5kfjBM8Qqpfl2+FNhTYWpMfYdPUnE7u536WqzFmsaqJctz3gBxH9Ex7dFtrxR4qiqEr9Qtlu3xGn7Bw07/+i1D+ey3ONkZLN
+LQ714cgj8fRS4Hj29SCmXp5Kt5/82cD/VN3NtHw== smoser@brickies
# Send pre-generated ssh private keys to the server
# If these are present, they will be written to /etc/ssh and
# new random keys will not be generated
# in addition to 'rsa' and 'dsa' as shown below, 'ecdsa' is also supported
ssh_keys:
  rsa_private:|
    -----BEGIN RSA PRIVATE KEY-----
    MIIBxwIBAAJhAKDOYSHy73nUgysO13XsJmd4fHiFyQ+OOR7VVu2iV9Qcon2LZS/x1cydPZ4pQpfjEha6WxZ6o8ci/Ea/w0n
+OHGPwaxlEG2Z9inNtj3pgFrYcRztfECbl j6HCibZbAzYtwIBIwJgO8h
72WjcmvcpZ8OvHSvTwAguO2TkR6mPgHsgSaKy6GJoPUJnaZRWuba/
HXOKGyhz19nPzLpzG5f0fYahlMJAyc13FV7K6kMBPXTRR6FxgHEgLOMPC7cdqAwOVNcPY6A7AjEA1bNaIjOzFN2sfZX0j7OMhQ
   uc4zP7r80zaGc5oy6Wp58hRAncFKEvnEq2CeL3vtuZAjEAwNBHpbNsBYTRPCHM7rZuG/iBtwp8Rxhc9I5wixvzMgi
+HpGLWzUIBS+P/XhekIjPAjA285rVmEP+DR255Ls65QbgYhJmTzIXQ2T9luLvc
mFBC6l35Uc4gTgg4ALsmXLn71MCMGMpSWspEvuGInayTCL+vEjmNBT+FAdOW7D4zCpI43jRS9UO6JVOeSc9CDk2lwiA3wIwCTB/
6uc8Cq85D9YqpM1OFuHjKpnPREPPOyrAspdeOAV+6VKRavstea7+    2DZmSUgE
    -----END RSA PRIVATE KEY-----
  rsa_public:ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAGEAoPRhIfLvedSDKw7XdewmZ3h8eIXJD7TRHtVW7aJX1ByifYtlL/
HVzJ09nilCl+MSFrpbFnqjxyL8Rr/DSf7QcY/BrGUQbZn2Kc22PemAWt    hxHO18QJvWPocKJtlsDNi3 smoser@localhost
 dsa_private:
    -----BEGIN DSA PRIVATE KEY-----
    MIIBuwIBAAKBgQDP2HLu7pTExL89USyMO264RCyWX/
CMLmukxXOJdbm29ax8FBJTpLrO8TIXVY5rPAJm1dTHnpuyJhOvU9G7M8tPUABtzSJh4GVSHlwaCfycwcpLv9TXDgWIpSj
+6EiHCyaRlB1/CBp    9RiaB+1OQcFbm+lapuET+/Au6vSDp9IRtlQIVAIMR8KucvUYbOEI+yv+5LW9u3z/BAoGBAIOq6JP
+JvJmwZFaeCMMVxXUbqiSko/P1lsaLNNBHZ5/8MOUIm8rB2FC6ziidfueJpqTMqeQmSAlEBCwnw
reUnGfRrKoJpyPNENYd15MG6N5J+z81sEcHFeprryZ
+D3Ge9VjPq3Tf3NhKKwCDQO24OaPezbnjPeFm4mHbYxxcZ9GAoGAXmLIFSQgiAPu459rCKxT46tHJtMOQfnNiEnQLbFluefZ/
yiI4DI38UzTC    OXLhUA7ybmZha+D/csj15Y9/BNFuO7unzVhikCQV9DTeXX46pG4s1o23JKC/QaYWNMZ7kTRv
+wWow9MhGiVdML4ZN4XnifuO5krqAybngIy66PMEoQCFEIsKKWv99iziAHOKBMVbxyO3Trz
    -----END DSA PRIVATE KEY-----
 dsa_public:ssh-dss AAAAB3NzaC1kc3MAAACBAM/
Ycu7ulMTEvz1RLIzTbrhELJZf8Iwua6TFfQ11ubb1rHwUElOkus7xMhdVjms8AmbV1Meem7ImE69TObszyO9QAG3NImHgZVIeXB
oJ/JzByku/l    NcOBYilKP7oSIcLJpGUHX8IGn1GJoH7XRBwVub6Vqm4RP78C7q9IOnOhG2VAAAAFQCDEfCrnL1GGzhCPsr/
uS1vbt8/wQAAAIEAjSrok/4m8mbBkVp4IwxXFdRuqJKSj8/WWxos00Ednn/ww5QibysHY
ULrOKJ1+54mmpMyp5CZICUQELCfCt5ScZ9GsqgmnI8OQ1h3Xkwbo3kn7PzWwRwcV6muvJn4PcZ71WM+rdN/
c2EorAINDTbjRo97NueM94WbiYdtjHFxnOYAAACAXmLIFSQgiAPu459rCKxT46tHJtMOQ   fnNiEnQLbFluefZ/
yiI4DI38UzTCOXLhUA7ybmZha+D/csj15Y9/BNFuO7unzVhikCQV9DTeXX46pG4s1o23JKC/QaYWNMZ7kTRv
+wWow9MhGiVdML4ZN4XnifuO5krqAybngIy66PMEoQ= smoser@l   ocalhost
```

8.5.7 运行系统命令

该配置非操作系统的配置，是用户注入配置，由用户修改相关信息后，注入数据源（如openstack的userdata），在系统启动时，cloudinit会去数据源读取。

```
#cloud-config

# run commands
# default: none
# runcmd contains a list of either lists or a string
# each item will be executed in order at rc.local like level with
```



```
# output to the console
# - runcmd only runs during the first boot
# - if the item is a list, the items will be properly executed as if
#   passed to execve(3) (with the first arg as the command).
# - if the item is a string, it will be simply written to the file and
#   will be interpreted by 'sh'
#
# Note, that the list has to be proper yaml, so you have to quote
# any characters yaml would eat (':' can be problematic)
runcmd:
 - [ ls, -l, / ]
 - [ sh, -xc, "echo $(date) ': hello world!'" ]
 - [ sh, -c, echo "=========hello world'=========" ]
 - ls -l /root
 - [ wget, "http://slashdot.org", -O, /tmp/index.html
```

8.5.8 创建文件

该配置非操作系统的配置，是用户注入配置，由用户修改相关信息后，注入数据源（如openstack的userdata），在系统启动时，cloudinit会去数据源读取。

```
#cloud-config
# vim: syntax=yaml
#
# This is the configuration syntax that the write_files module
# will know how to understand. encoding can be given b64 or gzip or (gz+b64).
# The content will be decoded accordingly and then written to the path that is
# provided.
#
# Note: Content strings here are truncated for example purposes.
write_files:
-   encoding: b64
    content: CiMgVGhpcyBmaWxlIGNvbnRyb2xzIHRoZSBzdGF0ZSBvZiBTRUxpbnV4...
    owner: root:root
    path: /etc/sysconfig/selinux
    permissions: '0644'
-   content: |
        # My new /etc/sysconfig/samba file

        SMBDOPTIONS="-D"
    path: /etc/sysconfig/samba
-   content: !!binary |
        f0VMRgIBAQAAAAAAAAAAAAIAPgABAAAAwARAAAAAAABAAAAAAAAAAJAVAAAAAAAAAAAAAEAAOAAI
        AEAAHgAdAAYAAAAFAAAAQAAAAAAAAABAAEAAAAAAAEAAQAAAAAAAwAEAAAAAAADAAQAAAAAAAAgA
        AAAAAAAAwAAAAQAAAAAgAAAAAAAAACQAAAAAAAAAJAAAAAAAAcAAAAAAAABwAAAAAAAAAQAA
        ....
    path: /bin/arch
    permissions: '0555'
-   encoding: gzip
    content: !!binary |
        H4sIAIDb/U8C/1NW1E/KzNMvzuBKTc7IV8hIzcnJVyjPL8pJ4QIA6N+MVxsAAAA=
    path: /usr/bin/hello
    permissions: '0755'
```

8.5.9 动态分区调整

该配置非操作系统的配置，是用户注入配置，由用户修改相关信息后，注入数据源（如openstack的userdata），在系统启动时，cloudinit会去数据源读取。

```
#cloud-config
#
# growpart entry is a dict, if it is not present at all
# in config, then the default is used ({'mode': 'auto', 'devices': ['/']})
#
#  mode:
#    values:
#     * auto: use any option possible (any available)
```



```
#             if none are available, do not warn, but debug.
#     * growpart: use growpart to grow partitions
#             if growpart is not available, this is an error.
#     * off, false
#
# devices:
#   a list of things to resize.
#   items can be filesystem paths or devices (in /dev)
#   examples:
#     devices: [/, /dev/vdb1]
#
# ignore_growroot_disabled:
#   a boolean, default is false.
#   if the file /etc/growroot-disabled exists, then cloud-init will not grow
#   the root partition.  This is to allow a single file to disable both
#   cloud-initramfs-growroot and cloud-init's growroot support.
#
#   true indicates that /etc/growroot-disabled should be ignored
#
growpart:
  mode: auto
  devices: ['/']
  ignore_growroot_disabled: false
```


9 负载均衡配置说明

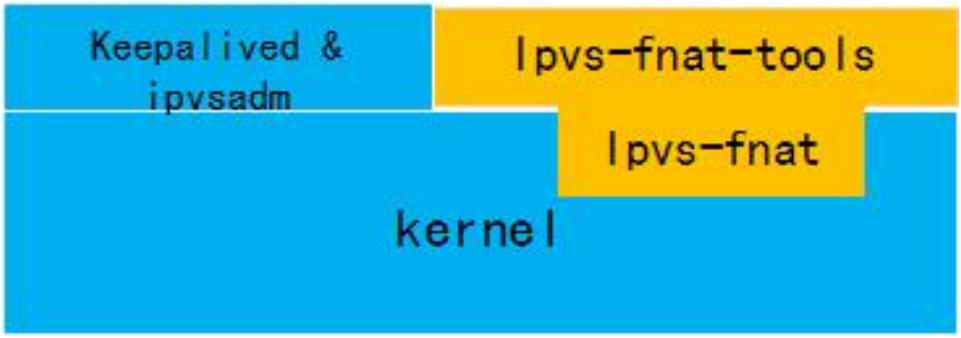
9.1 内核态 IPVS 配置说明

9.1.1 简介

什么是 LVS

LVS (Linux Virtual Server)是一种集群（Cluster）技术，采用IP负载均衡技术和基于内容请求分发技术。调度器具有很好的吞吐率，将请求均衡地转移到不同的服务器上执行，且调度器自动屏蔽掉服务器的故障，从而将一组服务器构成一个高性能的、高可用的虚拟服务器。整个服务器集群的结构对客户是透明的，而且无需修改客户端和服务器端的程序。

LVS 主要组成部分

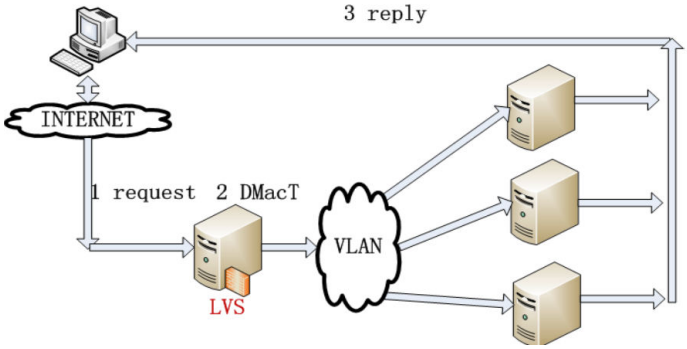


- keepalived&ipvsadm、用户态工具，用于健康检查、定义及管理lvs服务。
- ipvs-fnat-tools、fullnat以及原生内核的ipvs模块切换工具。
- ipvs-fnat，fullnat模式实现模块。

LVS 负载均衡方式

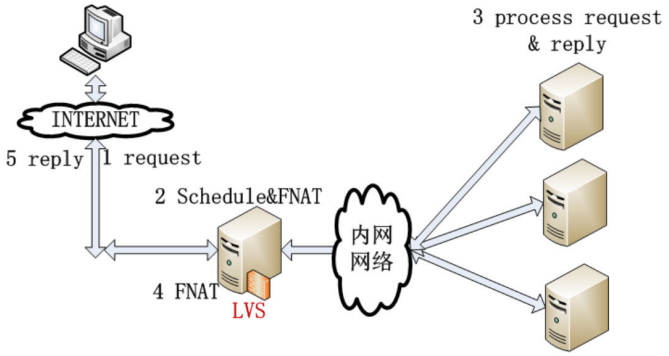
- VS/DR（Virtual Server via Direct Routing）

  VS/DR方式是通过改写请求报文中的MAC地址部分来实现的。Director和RealServer必需在物理上有一个网卡通过不间断的局域网相连。 RealServer上绑定的VIP配置在各自Non-ARP的网络设备上(如lo或tunl),Director的VIP地址对外可见，而RealServer的VIP对外是不可见的。RealServer的地址既可以是内部地址，也可以是真实地址。



- VS/FNAT（Virtual Server via Full NAT）

  VS/FNAT方式是即对报文做DNAT把VIP VPORT修改成RIP RPORT，又做SNAT把CIP CPORT修改成LIP LPORT。把报文转发给RealServer处理。从RealServer回来的报文根据LIP回到LVS，做相反的NAT。后端的网络设备和RealServer不需要做任何配置。



说明

LVS特性只适用于Layer 4分发，不适用Layer 7分发。

9.1.2 配置准备

使用原生内核自带的ipvs模块，不需要进行如下操作。

9.1.2.1 fullnat 模式工具安装

1. 当前已经安装EulerOS。
2. 系统已经设置root密码，并且配置好了IP。

3. 对于EulerOS，可以采用iso方式安装或者配置repo源方式安装。

说明

- 当前支持在物理机和kvm虚拟机上部署。在虚拟机上部署时，当前仅支持virtio-net虚拟网卡、1822网卡PCI直通模式，其他类型网卡的暂不支持。
- 无需使用ipvs-switch fnat去切换模式，安装后自动切换。但重启后ko不会自动加载，需要手动使用ipvs-switch命令切换为fnat模式。

iso 方式安装

步骤1 配置/etc/yum.repos.d/local.repo如下：

```
[iso]
baseurl=file:///iso
gpgcheck=0
enable=1
```

步骤2 挂载iso

```
mount -o loop -t iso9660 /home/elb/EulerOS-V2.0SP5-x86_64-dvd.iso /iso
```

步骤3 安装

```
yum install keepalived
yum install ipvsadm
yum install ipvs-fnat
yum install ipvs-fnat-tools
```

----结束

repo 方式安装

步骤1 配置/etc/yum.repos.d/local.repo如下：

```
[base]
name=EulerOS-2.0SP5 base
baseurl=http://developer.huawei.com/ict/site-euleros/euleros/repo/yum/2.x/os/x86_64/
enabled=1
gpgcheck=1
gpgkey=http://developer.huawei.com/ict/site-euleros/euleros/repo/yum/2.x/os/RPM-GPG-KEY-EulerOS
```

步骤2 安装

```
yum install keepalived
yum install ipvsadm
yum install ipvs-fnat
yum install ipvs-fnat-tools
```

说明

如果需要安装或者升级ipvs-fnat和ipvs-fnat-tools，请确保keepalived服务为关闭状态。

----结束

9.1.3 配置指导

9.1.3.1 配置网络

概述

本节介绍如何进行网络配置，实现网络互通。

包括DR模式，FNAT模式。

注意事项

1. 确保客户端ping通该lvs服务器，lvs服务器能ping通后端。
2. 需重启网络生效配置。
3. 关闭防火墙。
4. 由于ipv6没有ipv4上类似arp_ignore和arp_announce功能，因此在ipv6模式下，如果将VIP配置在lo上，将导致客户端无法获取到LVS的VIP的mac地址，必须将VIP配置在非lo网卡上。

配置步骤

以root权限，在shell命令行环境下操作。

步骤1 编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/sysconfig/network-scripts/ifcfg-xx
```

网络参数需要用户根据实际业务情况进行相应的配置，此处简单举例如下：

```
TYPE=Ethernet
BOOTPROTO="static"
NAME=eth0
UUID=2845c1a2-4dcb-47c6-920e-72387845aaa3
DEVICE=eth0
#ONBOOT=no
IPADDR=192.168.31.247     //配置ip
NETMASK=255.255.0.0       //配置掩码
GATEWAY=192.168.0.1       //配置网关
STARTMODE="auto"
```

步骤2 重启网络服务，使配置生效，执行：

```
systemctl restart network
```

----结束

9.1.3.2 配置 keepalived

概述

本节介绍如何对keepalived进行配置。

注意事项

- 用户修改完配置文件，使用systemctl restart keepalived或者systemctl reload keepalived使配置生效。
- 配置文件必须为常规文件，如果具有可执行权限，会跳过不进行加载。
- 启动keepalived前需要关闭selinux。
- 当使用quorum_up配置vip时，如果配置的virtual server过多，后端上线过程中keepalived会出现卡住的现象。在后端为“cpu：CPU_E5-2650v3@2.30GHz*2；网卡：Intel_82599_10GE；内存：256G，后端监听服务为nginx”的情况下，最多能支持30000个左右的virtual server正常上线。若需配置更多的vip，建议使用“vip_bind_dev lo”配置，该方式与quorum_up的区别：
  a. 当未配置alpha或者配置了check_off时，virtual server下不配置real server，此时quorum_up会配置vip，而vip_bind_dev不会配置vip。
  b. 当停止keepalived服务时，vip_bind_dev会自动删除有real server在线的virtual server对应的vip，而仅配置quorum_up且未配置quorum_down时不会删除vip。

- 在quorum_up大配置场景下，启动keepalived服务，对比vip配置完成的时间，keepalived-1.3.5版本（EulerOS V2R7）比keepalived-1.2.*版本（EuerOS 2.0SP2）更耗时。
- vip_bind_dev、quorum_up、vrrp都可以配置vip，为了避免冲突，建议只使用其中一种方式配置vip。
- 配置各种健康检查方式时，建议在XXX_CHECK中配置bindto字段，显式指定作为健康检查的源ip。否则keepalived可能根据路由选择一个随机无法使用的源ip，导致健康检查失败。
- DNS_CHECK配置域名name字段时，域名name字段的长度不能超过68个字符，否则会导致DNS报文发送失败。

说明

keepalived服务启动后，如果keepalived子进程被kill，由于keepalived主进程无法响应信号，keepalived主进程会再次拉起keepalived子进程。若此时修改keepalived配置信息且配置中删除了一些后端，或者有的后端此时健康检查还没有上线，可能会导致内核与用户态的后端信息不一致，使用ipvsadm -ln查看时可能会残存一些下线的后端信息。

配置步骤

说明

1. keepalived对配置文件的参数不做语法拼写的异常检查，配置参数语法或者拼写由用户保证正确性。
2. keepalived配置文件的修改与reload不能并发执行，否则会出现不可预知的配置残留问题。

以root权限，在shell命令行环境下操作。

步骤1 编辑keepalived配置文件：

```
vi /etc/keepalived/keepalived.conf
```

FNAT的配置示例如下：

```
! Configuration File for keepalived

global_defs {
}

local_address_group laddr_g1 {
     192.168.31.247    #LOCAL ip 用于和后端设备通信
}

virtual_server_group 70621a7fa6e1db9e0afc2dbe3d42f9be {
     192.168.31.200 80 #虚拟ip用于和client通信
}
virtual_server 192.168.31.200 80 {
    delay_loop 6    #服务轮询的时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh
    persistence_timeout 60 #模板时间
    lb_kind FNAT    #LVS集群模式
    protocol TCP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1
    checker_merge   #后端健康检查合并
    alpha           #开启alpha模式
    omega           #开启omega模式
    quorum 1        #服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum配合使用）
    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    real_server 192.168.31.248 80 {     #真实服务ip
        weight 6                    #该实节点权重
        TCP_CHECK {                 #健康检查方式
            connect_port 80         #检查的端口
            connect_timeout 15      #连接超时时间
```

```
            nb_get_retry 5           #重连次数
            delay_before_retry 3     #重连间隔
            bindto 192.168.31.211    #指定健康检查的源IP
        }
    }
}
```

说明

- 编写keepalived.conf配置文件时，“{”前必须加空格，否则日志中会提示无法识别关键字。
- keepalived.conf配置文件中，禁止将键值对形式的配置设置为空。
- persistence_timeout不支持设置为0，否则会报错：persistent_timeout invalid。
- 如果没有配置enable_script_security就会打印：SECURITY VIOLATION - check scripts are being executed but script_security not enabled. enable_script_security在global_defs中配置。
- 对于quorum_up参数，如果没有开启alpha模式，每次启动keepalived服务或者加载keepalived新配置项时，都会触发执行quorum_up里的命令。
- 对于quorum_up和quorum_down参数，如果其配置项中包含”/“，那么配置的命令需要包含绝对路径，如”/usr/sbin/ip“，否则会提示命令找不到。
- quorum_up和quorum_down参数后面支持配置命令或脚本。
- keepalived的一个vitrual_server下，不允许出现两个ip与port相同的real_server配置，否则会导致无法预知的后果。
- 由于quorum_up和quorum_down可能会出现并发的场景，导致命令执行结果不符合预期，所以不建议配置quorum_down。

DR的配置示例如下（部分参数说明可参考FNAT的配置示例）：

```
! Configuration File for keepalived
global_defs {
    router_id LVS_DEVEL_168 #节点标识，唯一即可
}

vrrp_instance LVS_DR_HW {#主备模式下需配置该结构
    state MASTER      #节点的初始状态，但启动后还是通过竞选由优先级来确定
    interface eth1    #虚拟IP绑定的网卡
    virtual_router_id 200 #设置VRID，相同的VRID为一个组，决定多播的MAC地址
    priority 168 #设置本节点的优先级，启动后优先级高的为master
    advert_int 1 #检查间隔，默认为1秒
    virtual_ipaddress { #VIP，它随着state的变化而增加/删除，当state为master的时候就在该节点添加
        192.168.100.2 #当为backup时删除。正常情况下由优先级来决定的，此处可以设置VIP
    }
}
virtual_server 192.168.100.2 5002 {
    delay_loop 6
    lb_algo sh
    lb_kind DR
    protocol TCP
    real_server 192.168.100.10 5002 {
       weight 1
       TCP_CHECK {
            connect_port 5002
            connect_timeout 15
            nb_get_retry 5
            delay_before_retry 3
        }
    }
    real_server 192.168.100.11 5002 {
       weight 2
       TCP_CHECK {
            connect_port 5002
            connect_timeout 15
            nb_get_retry 5
            delay_before_retry 3
        }
    }
}
```

步骤2 启动keepalived，使用命令：

```
systemctl start keepalived
```

步骤3 用ipvsadm -Ln 查看配置有没有下发成功：

```
ipvsadm -Ln
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Established(Sec.) Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConnTCP
     192.168.31.200:80    rr                          persistent 60
  -> 192.168.31.248:80            FullNat 6      0          0
```

如果不成功，进行故障定位参见**9.1.3.5 故障定位**。

步骤4 设置开机启动：

```
systemctl enable keepalived.service
```

----结束

9.1.3.3 设置/etc/sysctl.conf

概述

本节介绍如何设置/etc/sysctl.conf。

注意事项

IPVS DR和IPVS FNAT需要做的配置有差异。

配置步骤

以root权限，在shell命令行环境下操作。

- 在RealSever上配置（DR方式需要做此配置）：

步骤1 配置vip到lo口：

```
ip addr add <vip>/32 dev lo
```

步骤2 修改内核参数：

```
vi /etc/sysctl.conf
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 2
net.ipv4.conf.all.rp_filter = 0
net.ipv4.ip_forward = 1
```

步骤3 使参数生效：

```
sysctl -p
```

----结束

- 在LVS上需要把ip_forward 打开。 DR和FNAT方式都需要做此配置。

步骤1 修改内核参数：

```
vi /etc/sysctl.conf
net.ipv4.ip_forward = 1
```

步骤2 使参数生效：

```
sysctl -p
```

----结束

9.1.3.4 验证配置成功

概述

本节介绍如何验证配置是否成功。

注意事项

如果配置不成功，请参见下章故障定位。

验证操作

以root权限，在shell命令行环境下操作。

步骤1 在Real Server上安装httpd 或者nginx。

说明

安装httpd 或者nginx不在此说明，请用户查阅相关文档。

步骤2 在LVS 启动 keepalived：

```
systemctl start keepalived
```

步骤3 在LVS上，用ipvsadm查询下发表项是否生效：

```
ipvsadm -Ln
Prot LocalAddress:Port Scheduler Established(Sec.) Flags
-> RemoteAddress:Port          Forward Weight ActiveConn InActConnTCP
   192.168.31.200:80    rr                          persistent 60
-> 192.168.31.248:80           FullNat 6      0          0
```

表项有效的标志是有vip表项->RealServer

步骤4 在客户端用浏览器或者curl访问vip的地址：

```
curl 192.168.31.200:80
```

客户端如果是PC机，可以用IE或者Chrome，地址栏输入 http://192.168.31.200:80进行验证。

----结束

9.1.3.5 故障定位

概述

本节介绍常见IPVS故障。

注意事项

无。

常见故障

1. 使用ping命令检查 lvs物理口地址，如果无法ping通，请检查网络配置。
2. 使用ping命令检查VIP，如果无法ping 通，检查vip 是否以VIP/32的方式配置在LVS网卡上。

3. 从用户能ping通VIP，但是客户端浏览器无法访问IPVS配置的业务端口，检查该报文是否在LVS上或者RealServer上 被iptables拦截，可以用iptables -F清除所有规则。
4. 使用ipvsadm -Ln看ipvs表项有没有正确下发。
5. 安装tcpdump软件包，在LVS上抓ipvs的报文，比如tcpdump | grep <cip>。查看报文是否被捕捉到，并转发给RealServer。
6. 在RealServer抓报文，查看是否有来自LVS的报文 tcpdump | grep <cip> 或tcpdump | grep <lip>。
7. 用ipvsadm -Ln 查询ipvs 的流是否建立：
   a. 正常情况， 表项中有->指向Real Server的端口。
   ```
   Prot LocalAddress:Port Scheduler Established(Sec.) Flags
   -> RemoteAddress:Port           Forward Weight ActiveConn InActConnTCP
     192.168.31.200:80    rr                          persistent 60
   -> 192.168.31.248:80           FullNat 6      0          0
   ```
   b. 不正常情况， 表项中没有->指向Real Server的端口，可能的故障是到Real Server 网络不通，或者RealServer 相关服务或者端口没有开启。
   ```
   Prot LocalAddress:Port Scheduler Established(Sec.) Flags
   -> RemoteAddress:Port           Forward Weight ActiveConn InActConnTCP
     192.168.31.200:80    rr                          persistent 60
   ```
8. 在DR模式，如果RealServer能正常收发IPVS报文，但是业务还是不通，检查**9.1.3.3 设置/etc/sysctl.conf**的RealServer系统参数是否配置生效：
   a. cat /proc/sys/net/ipv4/conf/all/arp_ignore 应该等于1
   b. cat /proc/sys/net/ipv4/conf/all/arp_announce 应该等于2
   c. cat /proc/sys/net/ipv4/conf/all/rp_filter 应该等于0
   d. cat /proc/sys/net/ipv4/ip_forward 应该等于1
9. 在DR模式下，需要RealServer ping通客户端确保RealServer 的报文能直接发给客户端。
10. 在NAT模式，确保RealServer回来的报文通过配置默认网关或者策略路由的方式回流到LVS，使报文能在LVS上做SNAT。
11. 在配置vip realserver时，如果一个VIP的端口有多个real server。当其中一个real server连接状态不正常时，会触发keepalive执行quorum动作，如果quorum动作是删除VIP，那会导致该VIP整体不可用。
12. 不能频繁reload keepalived服务：reload间隔时间建议大于配置中（nb_get_retry +1） *（connect_timeout + delay_before_retry）的最大值再加上读取配置的时间，否则会出现部分后端不通时无法下线的现象 。nb_get_retry 、connect_timeout、delay_before_retry参数详见**9.1.3.2 配置keepalived**中的示例代码。
13. 如果没有keepalived_script用户，会打印WARNING - default user 'keepalived_script' for script execution does not exist - please create. 该告警不影响正常使用。
14. /proc/sys/net/ipv4/vs/下面的一些开关：
   a. defence_tcp_drop_for_udp：只配置支持UDP协议，客户端发送的TCP报文会被LVS丢弃；同时在LVS上可以看到/proc/net/ip_vs_ext_stats的defence_tcp_drop_for_udp统计项变大即发生了丢包。开关值可设置为0或1，0是关闭，1是打开，默认值为1。
   b. defence_udp_drop_for_tcp：只配置支持TCP协议，客户端发送的UDP报文会被LVS丢弃；同时在LVS上可以看到/proc/net/ip_vs_ext_stats的defence_udp_drop_for_tcp统计项变大即发生了丢包。开关值可设置为0或1，0是关闭，1是打开，默认值为1。
   c. defence_udp_not_vport_drop：配置了支持UDP协议，客户端发送的UDP报文的目的ip是vip、但是目的端口不是vport，LVS丢弃该数据包，同时在LVS上

可以看到/proc/net/ip_vs_ext_stats的统计项defence_udp_not_vport_drop变大了。开关值可设置为0或1，0是关闭，1是打开，默认值为1。

d. defence_udp_drop：在同一个vip上配置了TCP和UDP两种不同的listener，UDP的listener无法收到报文。开关值可设置为0或1，0是关闭，1是打开，默认值为1。

15. keepalived升级场景下，kill keepalived进程期间，不允许修改（含增删）配置文件，否则会出现配置残留；待keepalived服务重新拉起后，才能进行文件修改操作。

    a. kill -9 杀死keepalived进程，增加1个配置文件，start keepalived服务成功且新增配置生效；

    b. kill -9 杀死keepalived进程，删除1个配置文件，start keepalived服务成功且删除的svc残留，stop keealived服务后，ipvsadm-ln仍可查询到配置文件对应的svc。

16. 使用ipvsadm -ln与cat /proc/net/ip_vs查询命令不能与添加、删除virtual_server的操作并发，否则可能出现查询结果重复或丢失。

9.2 升级指导

9.2.1 ipvs-fnat 升级指导

1. 停止keepalived服务：
```
systemctl stop keepalived
```
2. 确认keepalived服务停止，停止执行ipvsadm命令以及调用该命令的脚本，停止引用ip_vs内核ko模块，在确认ip_vs的内核模块不会再被引用后，升级ipvs-fnat的rpm包：
```
rpm -Uvh ipvs-fnat-1.0.1-85.x86_64.rpm --force
```
3. 确保rpm包升级完毕后，启动keepalived服务：
```
systemctl start keepalived
```

说明

ipvs-fnat模块在升级阶段会断流，请注意使用场景，如不允许断流请先将流量切换到其他LVS节点再进行升级。

9.2.2 keepalived 升级指导

1. 移除并备份keepalived配置文件：
```
mkdir /home/user/keepalived-backup
mv /etc/keepalived/* /home/user/keepalived-backup/
```
2. 强制关闭keepalived checker子进程，停止keepalived服务，然后升级keepalived的rpm包：
```
kill -9 `cat /var/run/checkers.pid`
systemctl stop keepalived
rpm -Uvh --force keepalived-1.3.5-6.h98.x86_64.rpm --nodeps
```
3. 恢复keepalived配置文件，启动keepalived服务：
```
mv /home/user/keepalived-backup/* /etc/keepalived/
systemctl start keepalived
```

说明

使用kill -9 checker子进程之前，要移除/etc/keepalived/文件夹下的配置文件，否则关闭keepalived服务后，内核的配置会被清除，导致断流。

9.3 自研新特性

9.3.1 不断流特性（draining）

概述

为了实现后端服务器的平滑升级，需要在lvs中实现不断流特性即在后端服务器从lvs服务中移除后，在指定时间内，只要后端服务器的网络正常，已经建立连接的数据依旧能够送到后端服务器，等到当前的连接全部结束后即可升级后端服务器。

注意事项

- 只在fullnat模式下支持该特性。
- 从带有此特性的版本开始，keepalived的部分常用特性参数配置修改，无需restart keeplived，只需要reload keepalived即可。

配置步骤

以root权限，在shell命令行环境下操作。

步骤1 编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/keepalived/keepalived.conf
```

步骤2 参数名称：tcp_draining_timeout。tcp_draining_timeout表示关闭通道的超时时间，单位为秒，0表示永不超时。

说明

tcp_draining_timeout的取值范围[0,3600]，整数。但当前参数配置不生效，用户配置任何取值范围内参数，均表示不超时。

----结束

参数使用情况举例：

表示使能tcp_draining，并且相关流都自然老化，没有时间限制。

```
virtual_server 192.168.31.240 80 {
    delay_loop 6
    lb_algo rr
    persistence_timeout 60
    lb_kind FNAT
    protocol TCP
    laddr_group_name laddr_g1
    alpha
    omega
    quorum 1
    hysteresis 0
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    tcp_draining_timeout 0
    real_server 192.168.31.246 80 {
        weight 1
        TCP_CHECK {
            connect_port 80
            connect_timeout 15
            nb_get_retry 5
            delay_before_retry 3
        }
```

```
    }
}
```

9.3.2 一致性 hash 算法

概述

lvs自带的load balance算法中的源地址hash算法在任意一台后端服务器down时，会导致所有后端服务器的连接几乎全部异常，同时流量的均衡性做的不够好。现网集群模式下部署需依赖前端（DNS或者ECMP等价路由）的sh算法才能使用。因此，需要一种新的sh算法保证当后端服务器异常时的影响降到最低，同时还能保证同一客户端访问集群中任意一台lvs服务器的时候，确保分发到同一后端服务器，这种新的sh算法叫做一致性hash算法。

注意事项

- 只在fullnat模式下支持该特性。
- EulerOS2.5版本keepalived配置中新增consh配置项表示一致性hash算法。安装方法请参考**9.1.2.1 fullnat模式工具安装**。

配置步骤

以root权限，在shell命令行环境下操作。

步骤1 编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/keepalived/keepalived.conf
```

步骤2 参数使用情况举例：

```
virtual_server 192.168.31.240 80 {
    delay_loop 6
    lb_algo consh
    persistence_timeout 60
    lb_kind FNAT
    protocol TCP
    laddr_group_name laddr_g1
    alpha
    omega
    quorum 1
    hysteresis 0
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    tcp_draining_timeout 0
    real_server 192.168.31.246 80 {
        weight 1
        TCP_CHECK {
            connect_port 80
            connect_timeout 15
            nb_get_retry 5
            delay_before_retry 3
        }
    }
}
```

----结束

9.3.3 lvs 流量控制

概述

当前elb组网环境需要一种支持租户cps、并发数控制的功能，当租户的连接大于申请连接时，对租户的访问连接进行限制，才能有效的对租户进行隔离。当某个客户受到攻击时可以通过限制并发连接和新建连接，第一时间降低影响，提供安全防护。

注意事项

- 当前支持的粒度为vip+端口。
- 只在fullnat模式下支持该特性。

配置步骤

以root权限，在shell命令行环境下操作。

步骤1 编辑配置文件。在任意路径下，使用如下命令：

```
vi /etc/keepalived/keepalived.conf
```

步骤2 参数设置。

- 新建连接控制
  - flow_control_cps，cps过载控制。0：关闭；＞0：开启。
  - flow_control_interval，流控时间。0：关闭；＞0：开启。只有在此参数开启的情况下，flow_control_cps参数才有效。
- 并发连接控制
  - flow_control_pc，并发连接数限制。0：关闭；＞0 : 开启。

说明

1. 以上参数必须设置一个数字，不能为空。
2. 对于UDP连接，只有在连接超时之后才认为连接断开，超时时间可以使用 ipvsadm --set命令设置。
3. 新建连接控制、并发连接控制偶尔会有一定误差，导致实际允许通过的连接数略大于所设置的阈值，但误差范围不超过cpu核数。

参数使用情况举例：

```
virtual_server 192.168.31.240 80{
    delay_loop 6
    lb_algo rr
    persistence_timeout 60
    lb_kind FNAT
    protocol TCP
    laddr_group_name laddr_g1
    alpha
    omega
    quorum 1
    hysteresis 0
    quorum_up "/usr/sbin/ip addr add 192.168.31.240/32 dev lo;"
    tcp_draining_timeout 60
    flow_control_cps 5
    flow_control_interval 5
    flow_control_pc 1000000
    real_server 192.168.31.246 80{
        weight 1
        TCP_CHECK {
            connect_port 80
```

```
            connect_timeout 15
            nb_get_retry 5
            delay_before_retry 3
        }
    }
}
```

上述流控参数配置说明：

- cps：5秒内最大支持5*5条连接，超过该连接数不能建链，下个5秒周期恢复。
- pc : 同一时刻最多允许存在1000000条连接，当连接数超过该数值时不能新建连接。

----结束

9.3.4 客户端黑白名单

概述

黑白名单可以实现允许或拒绝特定的客户端访问该lvs服务器。黑白名单以virtual server为粒度单位进行配置。如果只配置了黑名单，那么被列入黑名单中的客户端不能访问该virtual server，其他客户端可以正常访问；如果只配置了白名单，那么只有被列入白名单中的客户端才能访问该virtual server，其他客户端都不允许访问。

注意事项

- 只在fullnat模式下支持该特性。
- 黑白名单以一个virtual server为粒度，各个virtual server之间的名单是互相独立的。
- 如果同一个virtual server中同时配置了黑白名单，此时只有白名单生效，黑名单是无效的。
- 如下命令仅用于调测，大规格（超过1k）添加黑白名单后，不允许执行该命令，否则可能有rlock风险。

  ```
  cat /proc/net/ip_vs_*_client
  ```

配置步骤

以root权限，在shell命令行环境下操作。

步骤1 执行echo 1 > /proc/sys/net/ipv4/vs/client_list_switch使能该功能。1表示使能功能，0表示关闭功能，默认为关闭。

步骤2 编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/keepalived/keepalived.conf
```

步骤3 白名单的关键字为client_ipaddr_allow，黑名单的关键字为client_ipaddr_exclude，配置方法如下：

```
关键字 {
     192.168.11.22
     192.168.0.1/24
     192.168.201.18/8
 }
```

参数使用情况举例：

```
virtual_server 192.168.31.240 80 {
    delay_loop 6
    lb_algo rr
    persistence_timeout 60
```



```
    lb_kind FNAT
    protocol TCP
    laddr_group_name laddr_g1
    alpha
    omega
    quorum 1
    hysteresis 0
    client_ipaddr_allow {
        192.168.11.22
        192.168.0.1/24
        193.168.201.18/8
    }
    real_server 192.168.31.246 80 {
        weight 1
        TCP_CHECK {
            connect_port 80
            connect_timeout 15
            nb_get_retry 5
            delay_before_retry 3
        }
    }
}
```

上述名单配置说明：

- 如果要配置黑名单，修改关键字client_ipaddr_allow为client_ipaddr_exclude即可。
- 名单中的IP有两种写法：
  - 192.168.11.22不带掩码长度的绝对IP；
  - 192.168. 0.1/24带有掩码长度的IP，这里的24就是掩码长度。IP和掩码长度之间用"/"隔开。192.168. 0.1/24表示192. 168. 0.0-192. 168. 0.255之间的IP。

    说明

    - ipv4掩码长度的有效范围为1-32，ipv6掩码长度的有效范围为1-128。
    - IP地址和掩码长度之间不能有空格。
    - 名单中的每行必须要有IP地址，且是合法的IP地址。
    - 每个virtual server中的黑名单或者白名单中，最多配置1000项。
    - 关键字和后面的左括号{之间需要有空格隔开，IP项从左括号开始到最近的右括号结束。

----结束

9.3.5 健康检查

9.3.5.1 支持健康检查关闭

概述

当前健康检查采用rst报文作为结束报文，这种非友好结束链接的方式可能导致客户后端低性能设备产生大量日志挂死，同时给过滤报文造成困难。

关闭健康检查特性可以降低keepalived的cpu使用率，提升keepalived能容纳的listener和vip数量。

注意事项

- 只针对Layer 4客户场景使用。
- 只在FULLNAT模式下使用。

配置接口

使用checker_off作为开关，以listener为单位进行配置。

以root权限，在shell命令行环境下操作。

步骤1 编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/keepalived/keepalived.conf
```

步骤2 参数使用情况举例：

参数名称：checker_off

```
virtual_server 192.168.31.200 80 {   #vip地址和端口
    delay_loop 6    #健康检查时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh|consh
    persistence_timeout 60 #LVS持续会话超时时间，单位秒
    lb_kind FNAT    #均衡转发模式，FNAT，DR
    protocol TCP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1   #指定local ip对应的group名称
    alpha           #开启alpha模式，在keepalived启动时，假设所有的RS都是down，等健康检查通过后rs才
上线。
    omega           #开启omega模式，在keepalived终止时，会执行quorum_down指令所定义的脚本。
    checker_off     #关闭健康检查
    quorum 1        #设置服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum 配合使用，保证小于quorum）

    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    real_server 192.168.31.248 80 {     #真实服务ip和端口
        weight 6                    #该实节点权重
        TCP_CHECK {                 #健康检查方式
            connect_port 80         #检查的端口
            connect_timeout 15      #连接超时时间
            nb_get_retry 5          #重连次数
            delay_before_retry 3    #重连间隔
        }
    }
}
```

说明

checker_off与alpha，或者checker_merge同时打开的时候，只有checker_off生效。

----结束

9.3.5.2 支持健康检查合并

概述

对多个VS中ip重复的后端的健康检查进行合并，所有配置了该项的VS中相同ip的后端只会做一次健康检查。存在ip重复项的后端个数最多不能超过1024个。（系统中arp表默认为1024，当存在ip重复项的后端个数太多时，arp表无法刷新，会引起网络异常，可以通过修改arp表项阈值解决）。

注意事项

- 只支持TCP_CHECK健康检查方式。
- 只在FULLNAT模式下使用。

配置步骤

说明

所有配置了checker_merge的conf文件中，alpha的设置必须要保持一致，否则会出现部分vip对应后端上下线失败的现象。

使用checker_merge作为开关，以listener为单位进行配置。

以root权限，在shell命令行环境下操作。

步骤1 编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/keepalived/keepalived.conf
```

步骤2 参数使用情况举例：

参数名称：checker_merge

```
virtual_server 192.168.31.200 80 {   #vip地址和端口
    delay_loop 6    #健康检查时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh|consh
    persistence_timeout 60 #LVS持续会话超时时间，单位秒
    lb_kind FNAT    #均衡转发模式，FNAT，DR
    protocol TCP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1   #指定local ip对应的group名称
    alpha           #开启alpha模式，在keepalived启动时，假设所有的RS都是down，等健康检查通过后rs才上线。
    omega           #开启omega模式，在keepalived终止时，会执行quorum_down指令所定义的脚本。
    checker_merge   #后端健康检查合并
    quorum 1        #设置服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum 配合使用，保证小于quorum）

    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    real_server 192.168.31.248 80 {     #真实服务ip和端口
        weight 6                    #该实节点权重
        TCP_CHECK {                 #健康检查方式
            connect_port 80         #检查的端口
            connect_timeout 15      #连接超时时间
            nb_get_retry 5          #重连次数
            delay_before_retry 3    #重连间隔
        }
    }
}
```

说明

checker_merge与checker_off同时打开的时候，checker_merge不生效。

----结束

9.3.5.3 支持 UDP 健康检查

概述

原有keepalived的UDP健康检查方式是通过调用python脚本处理，python脚本创建socket后会持续等待返回结果，当有大量UDP健康检查事件后会导致系统的cpu升高。基于这个问题，EulerOS新增加一种UDP的健康检查方式，用以解决上述脚本调用导致的系统性能不足的问题。

注意事项

1. 当前系统中有icmp报文限速，会导致unreachable报文回复较慢，从而出现后端不能及时下线的情况，此时需要关闭icmp报文限速。

- EulerOS需要在rs上配置内核参数，在/etc/sysctl.conf中新增net.ipv4.icmp_ratelimit=0。
- Windows需要自行配置解除icmp报文速率限制。

2. 支持的最大后端数与本地可用的udp端口数相同。
3. MISC_CHECK健康检查性能差，会极大消耗keepalived检查子进程的CPU，且MISC_CHECK可靠性差，所以强烈建议使用UDP_CHECK进行健康检查。

配置步骤

以root权限，在shell命令行环境下操作。

步骤1 编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/keepalived/keepalived.conf
```

参数使用情况举例：

```
virtual_server 192.168.31.200 53 {   #vip地址和端口
    delay_loop 6    #健康检查时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh|consh
    persistence_timeout 60 #LVS持续会话超时时间，单位秒
    lb_kind FNAT    #均衡转发模式，FNAT，DR
    protocol UDP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1   #指定local ip对应的group名称
    alpha           #开启alpha模式，在keepalived启动时，假设所有的RS都是down，等健康检查通过后rs才上线。
    omega           #开启omega模式，在keepalived终止时，会执行quorum_down指令所定义的脚本。
    quorum 1        #设置服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum 配合使用，保证小于quorum）

    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    real_server 192.168.31.248 53 {     #真实服务ip和端口
        weight 6                    #该实节点权重
        UDP_CHECK {                 #健康检查方式
            connect_port 53         #检查的端口
            connect_timeout 15      #连接超时时间
            nb_get_retry 5          #重连次数
            delay_before_retry 3    #重连间隔
            ping_check_off          #关闭ping检查，不配置该项默认开启ping检查
        }
    }
}
```

说明

关闭ping检查会影响“real_server”某些场景下的正常下线。

步骤2 开启ping检查时，需要修改内核参数，新增“**net.ipv4.ping_group_range=0 0**”：

```
vi /etc/sysctl.conf
net.ipv4.ping_group_range=0 0
```

步骤3 使参数生效：

```
sysctl -p
```

----结束

9.3.5.4 支持 vxlan 健康检查

概述

当前的源地址透传特性中，增加了数据流量可以支持vxlan的转发，但是keepalived的健康检查功能并不支持vxlan，因此需要配置checker_off，关闭健康检查功能。

本章节介绍keepalived健康检查支持vxlan。

注意事项

vxlan健康检查不支持checker_merge。

配置步骤

健康检查支持vxlan的配置文件举例如下：

```
LVS_VXLAN_PARAM {
        id 400 #lvs的序号
        local_vtepaddr 10.41.11.24 #本地vxlan外层封装地址
        local_bindaddr 192.168.31.100 #必选，健康检查指定的源地址
}
virtual_server 192.168.31.200 80 {   #vip地址和端口
    delay_loop 6    #健康检查时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh|consh
    persistence_timeout 60 #LVS持续会话超时时间，单位秒
    lb_kind FNAT #转发模式
    protocol TCP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1   #指定local ip对应的group名称
    alpha           #开启alpha模式，在keepalived启动时，假设所有的RS都是down，等健康检查通过后rs才
上线。
    omega           #开启omega模式，在keepalived终止时，会执行quorum_down指令所定义的脚本。
    quorum 1        #设置服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum 配合使用，保证小于quorum）

    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
real_server 192.168.31.248 80 {     #真实服务ip和端口
        weight 6                     #该实节点权重
       check_over_vxlan              #健康检查支持vxlan
        mac 52:54:00:04:xx:xx        #rs的mac地址
        vni 100                      #lvs后端L2GW的vni号
        vtepaddr 10.41.11.12 4789    #远端vtep ip和端口
        vxlan_private_flag_disable   #该项配置表示流量携带标准vxlan flag
        dest_lb_kind NATinVXLAN     #member转发模式，支持NATinVXLAN、FNATinVXLAN
        TCP_CHECK {                  #健康检查方式
            connect_port 80          #检查的端口
            connect_timeout 15       #连接超时时间
            nb_get_retry 5           #重连次数
            delay_before_retry 3     #重连间隔
           bindto 192.168.31.100    #可选，vxlan健康检查内层报文使用的本地ip，当配置local_bindaddr
后无需重复配置
        }
    }
}
```

配置项说明

1. **local_bindaddr**

   当支持vxlan健康检查时，必须配置local_bindaddr或者bindto字段，用来确定内层报文使用的源ip地址以及源mac地址，否则可能使用不确定的内部源地址，导致健康检查失败。当local_bindaddr或bindto的源ip和被检查的real_server目的ip属于不同网段时，同时需要关闭此网卡的反向路由过滤功能。使用如下命令关闭指定网卡的反向路由过滤功能：

   ```
   echo 0 > /proc/sys/net/ipv4/conf/ethX/rp_filter
   ```

   对于后端的real_server，同样要注意由于路由反向过滤的问题。

   说明

   - local_bindaddr或bindto指定的ip必须和本地的local_vtepaddr的ip在同一网卡上。
   - 当local_bindaddr与bindto同时配置时，健康检查优先使用内部bindto的ip。

2. **mac**

   针对同一个vtep节点，请注意多个配置文件中同一个ip地址必须配置相同的mac地址。相同vtep中相同ip配置不同的mac地址将导致健康检查功能混乱。

3. **check_over_vxlan**

   - 相较于原先的源地址透传配置文件，增加了check_over_vxlan表示当前的real_server健康检查是否需要跨越vxlan，用户需要配置此字段，健康检查支持vxlan才能生效。
   - 配置了check_over_vxlan后，keepalived健康检查中的fw_mark配置项将失效。

4. **vxlan_private_flag_disable**

   由于私有云和公有云的ELB方案不同，后者先实现的源地址透传中，默认对vxlan的报文头中增加了ELB特有的标签，而私有云不需要此标签，因此本需求提供了vxlan_private_flag_disable选项来控制LVS节点的vxlan报文是否添加特殊标签，默认为开启状态。在私有云场景需要关闭时，应该使用如下配置关闭：

   ```
   vxlan_private_flag_disable   #该项配置表示流量携带标准vxlan flag
   ```

5. **vni**

   vxlan的健康检查无法支持对有相同vni、不同vtep ip地址的多个vtep节点中存在相同ip的real server的健康检查。

6. 由于健康检查的报文需要在外部封装一层vxlan网络报文头，为了防止可能出现的封装vxlan网络报文头后超过MTU。因此对于创建的健康检查socket，其MSS当前被设置为默认减去外层vxlan网络报文头（50字节）的长度。

9.3.5.5 支持健康检查 status_code 配置多种状态码

概述

当前keepalived配置status_code只支持单个健康检查状态码的配置，本特性可以支持多个健康检查状态码的设置。

注意事项

无

配置步骤

健康检查status_code配置多种状态码的配置文件举例如下：

```
virtual_server 192.168.31.200 80 {   #vip地址和端口
    delay_loop 6    #健康检查时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh|consh
    persistence_timeout 60 #LVS持续会话超时时间，单位秒
    lb_kind FNAT #转发模式
    protocol TCP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1   #指定local ip对应的group名称
    alpha           #开启alpha模式，在keepalived启动时，假设所有的RS都是down，等健康检查通过后rs才上线。
    omega           #开启omega模式，在keepalived终止时，会执行quorum_down指令所定义的脚本。
    quorum 1        #设置服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum 配合使用，保证小于quorum）

    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    real_server 192.168.31.248 80 {     #真实服务ip和端口
        weight 6                    #该实节点权重
        HTTP_GET {                  #健康检查方式
            url {
                 path /index.html    #检查的url
```



```
                status_code 200 300-399 402-404   #设置返回码区间
            }
            connect_port 80          #检查的端口
            connect_timeout 15       #连接超时时间
            nb_get_retry 5           #重连次数
            delay_before_retry 3     #重连间隔
        }
    }
}
```

配置项说明

- status_code只有在http和ssl健康检查中才会用到，且http和ssl健康检查只支持一个url。
- status_code如果未配置，默认值为200-299。
- status_code支持单个配置或者区间配置，如配置区间，需用“-”表示连续的区间。
- status_code多个配置项之间以空格隔开。

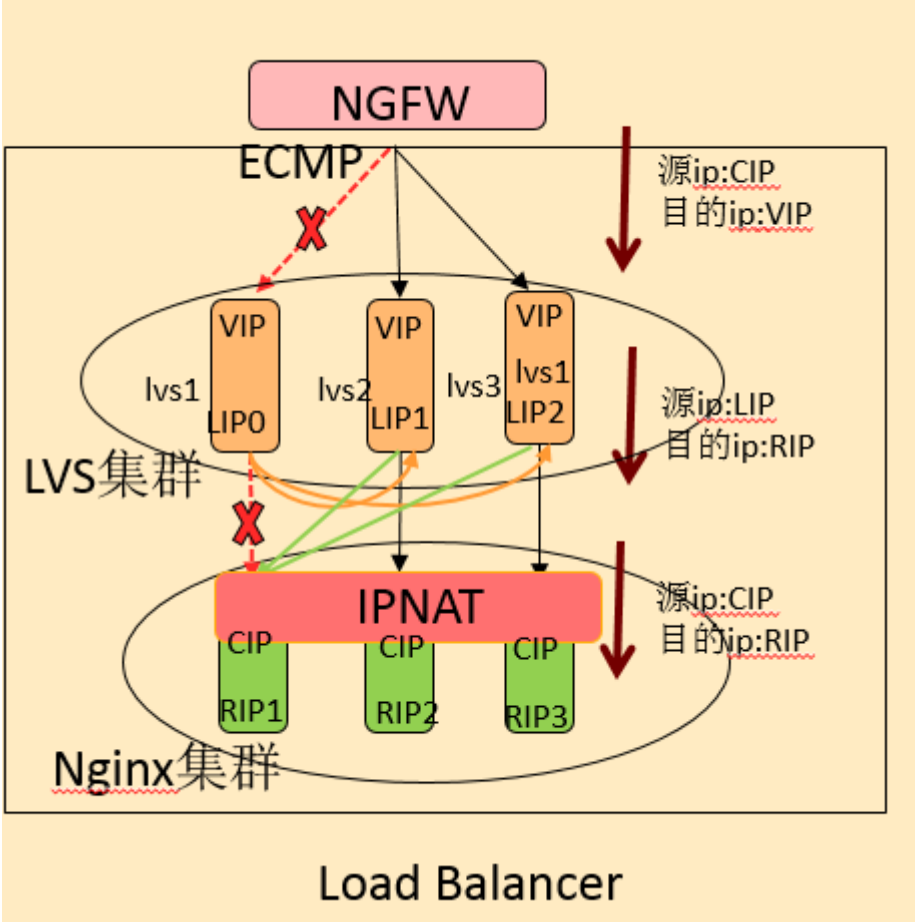
9.3.6 流表同步

概述

LVS作为公有云ELB最前端的组件，其冗余能力、稳定性、平滑升级能力都非常重要，当前LVS无法满足单点故障和升级不影响业务的需求。集群部署中任意一台LVS发生故障、节点升级流量都会中断，最严重的情况可能导致整个集群的流量异常（交换机不带一致性hash）。主备部署虽然能够通过keepalived切换主备，但是已有的流量依旧会中断。流表同步通过LVS节点间同步流表，达到节点升级或者故障客户无感知。

集群部署架构

LVS集群中通过组播将自己的流表信息同步到集群中的其他lvs节点上，保证任意台lvs都有一份全局的流表。当lvs集群中一台lvs需要升级或者故障下线时，防火墙或者vrouter将发给lvs1的数据转发给了其他lvs，由于lvs2和Lvs3也有lvs1的流表信息，所以也能正常转发。同时由于fullnat模式下每台LVS的出口ip（LIP）不一致，为了保证转发到nginx的不会断，需在后端nginx增加IPNAT模块，用于保证和nginx建立连接的ip保持不变。



说明

- 该特性不支持虚拟机场景，物理机场景下用户必须关闭LVS节点网卡GRO、LRO功能。
- 该特性不支持FTP协议。
- 该特性仅支持fullnat模式。

约束限制

1. 同步广播报文中有syncid字段，该字段可以用于同网络部署多个LVS的时候，可以分割广播组。当前仅支持LVS集群配置单个syncid，所有LVS集群节点syncid必须相同。
2. 内核态LVS组播地址224.0.0.81是属于IANA预留的地址，该地址IANA未指定，但是不能保证以后被指定。因此流表同步的组网中不能有其他业务使用该广播地址。

注意

- ipvs-switch切换不能与ipvsadm启动同步进程、ipvsadm -l --daemon操作并发，否则会有造成系统oops的风险。
- 后端节点插入ipnat模块后不允许再插入ip_vs模块，或者执行ipvsadm相关命令，否则会导致业务异常或系统异常。

环境准备

- 流表同步前需要增大系统socket buffer的值，尤其环境流量较大的情况下，否则同步会出错ip_vs_send_async error。配置举例：

```
sysctl -w net.core.wmem_max=412992000
sysctl -w net.core.rmem_max=412992000
```

- 网卡mtu：lvs节点间同步网卡的mtu值需要保持一致，否则mtu值大的节点往mtu值小的节点进行流表同步会失败。

配置步骤

步骤1 后端节点插入ipnat模块。

ipnat模块用于lvs后端节点，通过解析报文tcpoption字段中的client ip和port，与realserver建立流表，这样realserver节点看到的客户端ip就是client ip，当lvs节点故障或者升级时，数据从其他lvs节点访问realserver，虽然local ip发生变化，但client ip和port与realserver建立的流表没有变化，所以依然能够命中，同时client ip和port没有发生变化，所以realserver认为是一条连接，不会断开。

步骤2 使能关闭同步功能。

```
ipvsadm --start-daemon backup  --mcast-interface ethx --syncid xx
ipvsadm --start-daemon master  --mcast-interface ethx --syncid xx
ipvsadm --stop-daemon backup
ipvsadm --stop-daemon master
```

假设集群中有4台lvs，通过上面配置，lvs节点都向224.0.0.81的8848端口发送组播报文，同时也从224.0.0.81的8848端口接收同步的UDP组播报文。

说明

- lvs节点和后端必须在同一个vlan中。
- 使用ipvs-switch fnat/orig切换模式前，需要先使用ipvsadm --stop-daemon关闭同步功能，否则由于依赖关系会导致部分驱动无法删除。
- syncid取值（0-6特殊用途），ipvsadm启动同步进程时，syncid取值范围为[7-255]。

步骤3 调整同步间隔、组播的端口数

1. 当前流表同步间隔为“3 50”。

```
[root@localhost /]# cat /proc/sys/net/ipv4/vs/sync_threshold
3       50
```

该参数表示一条流中每50个报文的第3个触发同步。该参数需要客户根据网上实际情况修改，无损升级场景建议修改为1 2，每个报文都同步，降低流表无法同步的概率。

内核态LVS设置同步间隔命令：

```
echo "1 2" > /proc/sys/net/ipv4/vs/sync_threshold
```

2. 组播端口数默认为“ 1”。

客户可以通过查看网卡的丢包数或者netstats -s查看UDP的丢包数，来确认是否需要增加该参数，增加socket数据，提升数据吞吐能力。

内核态LVS设置端口数命令：

```
echo "8" > /proc/sys/net/ipv4/vs/sync_port_num
```


说明

- 该参数修改后需要先调用**步骤2**中的--stop-daemon，然后重新--start-daemon才能生效。
- 组播端口数必须是2的幂次，最大值是64（即使该参数配置成100，启动同步进程时实际port数为64）。
- 流表同步间隔参数前者必须小于后者。

----结束

9.3.7 源地址透传

概述

当前TOA方案部署存在以下问题：

1. 客户来说操作复杂，每台服务器逐一部署，影响用户体验。
2. TOA模块支持的操作系统有限，例如无法支持windows系统，无法满足某些客户需求。
3. 客户设置的iptables安全规则可能导致源地址被过滤。
4. 不支持UDP。

接口配置

源地址透传的配置文件举例如下：

```
LVS_VXLAN_PARAM {
        id 400 #lvs的序号
        local_vtepaddr 10.41.11.24 #本地vxlan外层封装地址
}

virtual_server 192.168.31.200 80 {   #vip地址和端口
    delay_loop 6    #健康检查时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh|consh
    persistence_timeout 60 #LVS持续会话超时时间，单位秒
    lb_kind FNAT #转发模式，源地址透传模式下仅支持FNAT
    protocol TCP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1   #指定local ip对应的group名称
    alpha           #开启alpha模式，在keepalived启动时，假设所有的RS都是down，等健康检查通过后rs才
上线。
    omega           #开启omega模式，在keepalived终止时，会执行quorum_down指令所定义的脚本。
    checker_off     #关闭健康检查
    quorum 1        #设置服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum 配合使用，保证小于quorum）

    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    real_server 192.168.31.248 80 {     #真实服务ip和端口
        weight 6                    #该实节点权重
        mac 52:54:00:04:xx:xx       #rs的mac地址
        vni 100                     #lvs后端L2GW的vni号
        vtepaddr 10.41.11.12 4789   #L2GW的vtep ip和端口
        dest_lb_kind NATinVXLAN     #member转发模式，支持NATinVXLAN、FNATinVXLAN
        TCP_CHECK {                 #健康检查方式
            connect_port 80         #检查的端口
            connect_timeout 15      #连接超时时间
            nb_get_retry 5          #重连次数
            delay_before_retry 3    #重连间隔
        }
    }
}
```

1. lvs模式新增NATinVXLAN、FNATinVXLAN两种模式，在原来的nat和fnat模式转发的报文中增加vxlan隧道。由于需要CNA以及L2GW等组网配合，当前只支持公有云ELB场景。
2. NATinVXLAN模式支持kvm虚拟化，不支持xen虚拟化模式，xen虚拟机可沿用原来fnat模式。
3. FNATinVXLAN用于ip target场景。
4. 支持根据不同的member设置不同的转发模式，member中的**dest_lb_kind**参数如果没有配置，则使用**lb_kind**设置的转发模式（**源地址透传场景下lb_kind只支持配置为FNAT模式**）。
5. keepalived健康检查子进程因参数值为空等错误配置导致进程异常退出时，NATinVXLAN模式下可能会出现重复的rs后端配置。
6. NATinVXLAN、FNATinVXLAN模式下不支持ipvsadm设置、删除操作，只支持ipvsadm查询。
7. NATinVXLAN、FNATinVXLAN模式下，相同rip但vni不同的后端不支持chekcer_merge功能。

说明

- member下的**vtepaddr**参数必须配置port。
- 内层报文默认按照1500的mtu来判断报文是否分片，用户可通过/proc/sys/net/ipv4/vs/dev_mtu接口修改。

9.3.8 源地址透传流量支持策略路由

概述

源地址透传场景，当前需要根据l2-gateway的地址配静态路由，发生l2-gateway扩容时，需要重新添加路由，涉及到工程组网变更，变更成本大。

现在需要改为根据外层源IP（在keepalived配置文件中）匹配策略路由，下一跳到默认vtep的网关，这样l2-gateway的扩容对ELB来说无感知。

注意事项

- 只在fullnat模式vxlan组网下支持该特性。
- 从带有此特性的版本开始，keepalived的部分常用特性参数配置修改，无需restart keeplived，只需要reload keepalived即可。

配置步骤

1. 以root权限，在shell命令行环境下操作。在任意路径下，使用如下命令编辑配置文件：
```
vi  /etc/keepalived/keepalived.conf
```
2. 配置如下：
```
LVS_VXLAN_PARAM {
    id 400    ## used by CNA only
    local_vtepaddr 192.168.30.16
    source_sensitive_routing   ##基于源地址（local_vtep）的策略路由选路
}
```
参数名称：source_sensitive_routing。配置该参数表示使能基于源地址的策略路由选路，默认不使能。
3. 可通过命令查看是否使能，1表示使能。


```
[root@localhost ~]# cat /proc/net/ip_vs_vtep
 Local_VtepAddr  LvsID  Source_Sensitive_Routing
 42423353           400       1
```

4. 通过如下命令配置基于源地址的策略路由。

```
ip rule add from 192.168.10.10 table 100
ip route add default dev eth1 table 100
```

9.3.9 增量加载

概述

在当前的配置修改方案中，keepalived如果需要修改配置，则需要修改配置文件，然后reload进程。在配置大量后端的情况下，这种操作需要大量的时间来重新上线，效率低下，缺乏竞争力。新的方案中，在保持原来的reload的方式不变的情况下，提供了通过jsonrpc的方式来增量进行配置。当前支持删除、添加和更新virtual_server三个jsonrpc接口。

使用方法

1. 首先需要安装lvscli包。

```
yum install lvscli
yum install lvscli-devel
```

2. 通过lvscli命令行工具来调用接口（不保证商用）。
   - 删除virtual_server用法如下：

     ```
     lvscli keep del {{address}} {{port}} [protocol]
     ```

     其中address和port分别是要删除的virtual server的地址和端口。protocol是可选字段，可为TCP或UDP，默认是TCP
   - 新增virtual_server的用法如下：

     ```
     lvscli keep add {{conf_file}}
     ```

     conf_file是新增的增量配置文件的路径。
   - 更新virtual_server的用法如下：

     ```
     lvscli keep set {{conf_file}}
     ```

     conf_file与新增接口相同时配置文件的路径。
3. 通过C接口来调用rpc接口。
   - 新增keepalived config配置

     ```
     int lvs_add_service_config(char *svc_config);
     ```
   - 修改keepalived config配置

     ```
     int lvs_update_service_config(char *svc_config);
     ```
   - 删除keepalived config配置

     ```
     int lvs_del_service_config(struct lvs_service *svc);
     ```

   示例代码：

```
#include <lvs/lvs.h>
#include <lvs/lvs_common.h>
struct lvs_service svc; //调用删除接口需要传入的结构体
svc.af=AF_INET; //设置地址family为ipv4
inet_pton(svc.af,"192.168.1.1",&svc.addr); //设置要删除的virtual_server的IP地址
svc.port=htons(1984); //设置要删除virtual_server的端口
svc.protocol=IPPROTO_TCP; //设置协议为TCP
ret = lvs_del_service_config(&svc);//调用接口

char * conf = "virtual_server 192.168.200.100 443 { \
    ... \
    real_server 192.168.201.100 443 { \
       ... \
    }\
```



```
    ...\
}"; //要新增的virtual server配置
ret = lvs_add_service_config(conf); //调用新增vs接口
ret = lvs_update_service_config(conf); //调用更新vs接口
```

说明

1. 本配置文件不能嵌套include; 根配置文件只有local_address_group对象可以使用。
2. 配置文件中的virtual server不支持配置group。
3. 当前仅支持ipv4的地址。
4. 每次调用接口时，仅允许更新单个virtual server的配置文件，当存在多个virtual server需要操作时，必须分多个配置文件多次进行接口调用。
5. 当根配置文件的local_address_group更新时，必须重新reload或者restart keepalived 服务方可生效。
6. 通过add、update、delete操作的vs配置，用户必须保证其对应的conf文件必须同步在磁盘进行更新，否则将引起keepalived在reload或者restart后出现配置不一致的情况。
7. 不支持smtp方式的健康检查。
8. 单个配置的最大字符串长度为 960KB。
9. 如果用户新增的配置配置了quorum_up参数，可能会导致接口性能下降,因为接口调用时可能需要执行用户配置的命令，属用户配置行为导致接口执行时间变长。
10. cpu占用率 100% 时，热加载耗时可能达到秒级。

9.3.10 vlan 多租户区分（Type2 网络）

概述

vlan多租户特性实现了LVS对FusionSphere Type2网络的适配。通过ALB控制设置网络类型，在Type2网络（vlan）下，支持vlan_id对多租户区分，即在vip、port、protocol相同条件下，能够根据vlan_id来区分LB实例；同一个LB实例，根据member所属网络类型创建子接口集合。所有路由网络共用1个子接口，每个纯二层网络创建1个子接口，子接口个数=1+纯二层网络个数。

注意事项

1. 在Type2网络必须在keepalived配置vlan_config字段为1，指定网络类型为vlan网络；默认为非vlan网络。
2. 本特性兼容非Type2网络：在非Type2网络中无需对各配置文件做任何修改。
3. Type2网络中virtual server必须配置vlan_in参数。
4. Type2网络中同一个virtual server下不能配置同"ip port"、不同interface的real server。
5. 不同vlan的realserver服务不支持在同一台机器上启动。
6. keepalived配置文件中，下联面配置错误的vlan id或者接口，会导致流量不正常，或者配置显示异常。

配置步骤

1. 编辑/etc/keepalived/keepalived.conf配置文件,增加LVS_VLAN_PARAM项，其下配置vlan_config字段为1：

```
LVS_VLAN_PARAM {
    vlan_config 1
}
```

2. 编辑/etc/keepalived/vhosts/*.conf配置文件：

- virtual server字段配置，vlan_in字段中设置LB实例的vlan id，quorum_up最终设置上联面ip配置命令：

```
virtual_server 10.1.1.151 23 {
            …
    vlan_in 102
            …
    quorum_up "/usr/sbin/ip addr add 10.1.1.151/32 dev eth1.102;"
            …
}
```

- real_server字段配置，interface字段中设置数据流出接口名称，bindto中绑定健康检查ip：

```
real_server 10.1.1.110 22 {
                    …
        interface eth2.102
                    …
        TCP_CHECK {
                        …
            bindto 10.1.1.11
                        …
        }
    }
```

3. 编辑/etc/keepalived/路径下的配置文件，在每个local ip项上增加出接口名称：

```
local_address_group laddr_g86369d5ae0034abead44311c6f1d2921 {
    10.1.2.238  eth2.103
    10.1.1.170  eth2.102
}
```

9.3.11 lvs 支持 vxlan

概述

IPv6数据流量支持vxlan转发，在物理网络IPv4情况下虚拟网络层支持IPv4、IPv6双栈流量。keepalived健康检查已同步支持vxlan，配置及详细介绍请参考**9.3.5.4 支持vxlan健康检查**。

配置步骤

如下为IPv6 over IPv4 vxlan模式配置文件举例，若需要配置IPv6 over IPv6 vxlan模式，则将local_vtepaddr、vtepaddr两项中地址改成IPv6地址。

```
LVS_VXLAN_PARAM {
        id 400 #lvs的序号
        local_vtepaddr 10.41.11.24 #本地vxlan外层封装地址
}
virtual_server fc00::31:200 80 {   #vip地址和端口
    delay_loop 6    #健康检查时间间隔
    lb_algo rr      #LVS调度算法，支持rr|wrr|lc|wlc|lblc|sh|dh|consh
    persistence_timeout 60 #LVS持续会话超时时间，单位秒
    lb_kind FNAT #转发模式
    protocol TCP    #支持的协议模式是TCP还是UDP
    laddr_group_name laddr_g1   #指定local ip对应的group名称
    alpha           #开启alpha模式，在keepalived启动时，假设所有的RS都是down，等健康检查通过后rs才上线。
    omega           #开启omega模式，在keepalived终止时，会执行quorum_down指令所定义的脚本。
    quorum 1        #设置服务是否有效的阀值
    hysteresis 0    #延迟系数（跟quorum 配合使用，保证小于quorum）

    #高于或低于阀值时会执行以下脚本。
    quorum_up "/usr/sbin/ip addr add fc00::31:200/32 dev lo;"
    real_server fc00::31:248 80 {     #真实服务ip和端口
        weight 6                      #该实节点权重
        check_over_vxlan              #健康检查支持vxlan
        mac 52:54:00:04:xx:xx         #rs的mac地址
```



```
        vni 100                          #lvs后端L2GW的vni号
        vtepaddr 10.41.11.12 4789        #远端vtep ip和端口
        dest_lb_kind NATinVXLAN          #member转发模式，支持NATinVXLAN、FNATinVXLAN
        vxlan_private_flag_disable       #该项配置表示流量携带标准vxlan flag
        TCP_CHECK {                      #健康检查方式
            connect_port 80              #检查的端口
            connect_timeout 15           #连接超时时间
            nb_get_retry 5               #重连次数
            delay_before_retry 3         #重连间隔
            bindto fc00::31:100          #用于vxlan健康检查内层报文使用的本地ip
        }
    }
}
```

1. 由于私有云和公有云的ELB方案不同，后者先实现的源地址透传中，默认对vxlan的报文头中增加了ELB特有的标签，而私有云不需要此标签，因此本需求提供了一个配置项vxlan_private_flag_disable，该配置用来控制LVS节点的vxlan报文是否添加特殊标签，不配置则携带ELB特有标记，在私有云场景需要配置表示不携带私有flag。
2. 同一个LVS只支持配置一种VXLAN模式（IPv4/IPv6），且vtepaddr与local_vtepaddr地址协议族必须一致（均为IPv4或IPv6）。
3. vxlan特性包含上、下联面两部分，两者共用一个local_vtepaddr。上、下联面vxlan可以单独配置使用，单独使用上联面vtep时，只需配置LVS_VXLAN_PARAM信息即可。
4. vxlan组网需要合理配置接口mtu，建议组网中的设备vtep接口mtu配置为1600，vxlan内层接口mtu配置为1500。

9.3.12 支持 svc 粒度 timeout 特性

概述

lvs原生版本已经支持通过ipvsadm或keepalived配置全局timeout参数（tcp、udp协议），本特性支持svc粒度的timeout参数设置。

注意事项

- 只在fullnat模式下支持该特性。
- 从带有此特性的版本开始，keepalived的部分常用特性参数配置修改，无需restart keeplived，只需要reload keepalived即可。

配置步骤

以root权限，在shell命令行环境下操作。编辑配置文件。在任意路径下，使用如下命令：

```
vi  /etc/keepalived/keepalived.conf
```

参数名称：tcp_timeout、tcp_fin_timeout、udp_timeout。分别对应tcp、udp协议对应状态的超时时间，单位为秒，配置成0或者不配置表示使用全局超时时间，可通过如下命令查看全局超时时间，全局默认值分别为：90 3 300。

```
ipvsadm -ln --timeout
```

说明

上述3个timeout取值范围[0, 2147483]，整数。svc粒度timeout参数优先级高于全局参数。

参数举例

配置timeout参数后，当前virtual_server流表按当前参数老化，若配置参数为0或不配置表示按全局配置值老化。

- 方法一：通过/etc/keepalived/keepalived.conf文件配置。

```
virtual_server 192.168.31.240 80 {
    delay_loop 6
    lb_algo rr
    persistence_timeout 60
    lb_kind FNAT
    protocol TCP
    laddr_group_name laddr_g1
    alpha
    omega
    quorum 1
    hysteresis 0
    quorum_up "/usr/sbin/ip addr add 192.168.31.200/32 dev lo;"
    tcp_timeout 100       #TCP ESTABLISH状态超时时间
    tcp_fin_timeout 20    #TCP FIN_WAIT状态超时时间
    udp_timeout 90        #UDP流量超时时间
    real_server 192.168.31.246 80 {
        weight 1
        TCP_CHECK {
            connect_port 80
            connect_timeout 15
            nb_get_retry 5
            delay_before_retry 3
        }
}
}
```

- 方法二：通过ipvsadm命令配置。

  配置TCP timeout参数：

  ipvsadm -A|E -t *service-address* --set *tcp tcpfin*

```
# ipvsadm -A -t 192.168.21.20:80 --set 100 50
```

  配置UDP timeout参数：

  ipvsadm -A|E -u *service-address* --set *udp*

```
# ipvsadm -E -u 192.168.21.21:53 --set 60
```

  查询timeout参数

  ipvsadm -ln －t|u *service-address* --timeout

```
# ipvsadm -ln -u 192.168.21.21:53 --timeout
```

9.4 命令参考

1. ipvsadm
   - 查看lvs的配置的负载均衡的连接配置
     ```
     ipvsadm -ln
     ```
   - 查看激活的连接(注意，由于用户态限制，lnc中倒计时使用cpu的tsc寄存器，溢出周期为100年以上)
     ```
     ipvsadm -lnc
     ```
   - 清空配置
     ```
     ipvsadm -C
     ```
   - 添加vip及端口号
     ```
     ipvsadm -A -t 192.168.31.200:80 -s rr
     ```
   - 添加rs的ip


```
ipvsadm -a -t 192.168.31.200:80 -r 192.168.31.248 -q
```

- 添加lip

```
ipvsadm --add-laddr -z 192.168.31.100 -t 192.168.31.200:80
```

2. iptables
   - iptables规则备份

```
iptables-save > iptables.dump
```

   - 恢复iptables规则

```
iptables-restore iptables.dump
```

3. rpm
   - 安装rpm包

```
rpm -ivh xxx.rpm
```

   - 查询是否安装了rpm包

```
rpm -qa | grep xxx
```

   - 卸载rpm包

```
rpm -e xxx
```

9.5 业务安全性说明

IPVS负载均衡通过在内核hook业务点挂钩子的方式接收报文，根据报文的IP地址和端口信息做负载均衡处理，不解析不收集报文中的用户数据。

9.6 常见问题处理

流表同步失败

1. 使用ifconfig检查流表同步的网卡是否在线。
2. 如果在线，确认是否已经配置为同一网段的ip，可使用ping进行检查。

arp 表满导致的异常现象以及解决方法

现象：

- 流表同步异常，messages中打印IPVS:ip_vs_send_async error -22。
- 导致lvs节点管理ip断连，ping返回：sendmsg:Invalid argument。

解决方法：

释放arp表，或者增加arp表阈值，即可恢复环境。

网卡丢包，通过 ifconfig 和 ethtool 查看网卡看到 rx_missed_errors 一直增加

可以通过修改网卡ringbuffer大小来规避：

```
ethtool -G ethx rx 4096
ethtool -G ethx tx 4096
```

sysctl_lblcr_expiration 配置

/proc/sys/net/ipv4/vs/sysctl_lblcr_expiration中，sysctl_lblcr_expiration参数用来设置lblcr算法超时时间，默认值为86400（单位是秒）。必须配置为非负整数，否则会出现无限超时。

性能问题

1. 使用1822网卡PF直通运行keepalived时，在重启过程中，后端上线的速度相比于用virtio网卡慢很多。可参见**37.28 l2nic_interrupt_switch参数使用说明**。

2. 1822网卡环境下打流导致cpu0的软中断太多，使得系统复位，关闭kernel.timer_migration开关可以避免。

Fullnat 模式下 rr 或 wrr 调度算法出现连接建立不均衡问题

在Fullnat模式下，调度算法为rr或者wrr时，由于Local IP的选择同样为rr调度算法，如果Local IP的数量和Real Server的数量恰好成公约数的关系，将导致Local IP的分配和Real Server呈一一对应的映射关系，Real Server只能轮询分配到指定的Local IP，而无法分配到其他的Local IP，使得仅有指定IP端口被大量使用，导致无法均衡建立连接。因此，在配置rr或者wrr算法时，建议先通过计算Local IP和Real Server的数量，合理配置Local IP的数量，避免出现碰撞的情况。

10 中断均衡 irqbalance

使用irqbalance服务可以在很大程度上均衡各个CPU核上的中断负载，达到充分利用多核资源以提升系统整体性能的效果。用户在使用该服务的同时，也可以配合一些其它内核态或用户态的通用接口来针对业务制定对应的均衡策略。

特性描述

随着硬件芯片制造工艺的不断完善，设备对CPU的计算能力提出了更高的要求。比如Intel 82599 10GE网卡，如果只用一个CPU处理该网卡的数据包收发，由于受限于CPU的处理能力，它的带宽将无法达到10GE。因此现在很多设备都采用了多队列、多中断的设计方式来提升性能，其中网络和存储设备最为普遍。比如Intel 82576网卡，该网卡最多可以支持8个收发队列来均摊数据包的收发压力，使多个CPU有机会并行的处理这些数据包的收发。

因此，如何充分利用CPU资源成为了提升系统性能的关键，irqbalance服务正是为了解决该问题而诞生的。Irqbalance服务通过解析系统最近一段时间内的中断压力分布来对将来的中断压力进行重新布局，使得各个CPU核上的中断负载尽可能均衡，从而提高了系统整体的性能。

约束限制

中断均衡服务是通过改变中断的CPU亲和性来重新分布中断压力的，而某些中断的CPU亲和性不能或者没有必要修改，因此服务在生效时有一定的局限性，下述几类中断不会参与中断均衡：

- 标识为IRQD_NO_BALANCING的中断不能进行中断均衡。
- 标识为IRQD_PER_CPU的中断不能进行中断均衡。
- 特殊中断（/proc/interrupts中中断号包含非数字的中断）不能进行中断均衡。

使用说明

在启动irqbalance服务之前，需要先对服务进行配置。该配置文件中主要包含了一些环境变量，通过修改这些环境变量可以对中断均衡服务的行为产生影响。配置文件的路径以及环境变量如下所述。

```
/etc/sysconfig/irqbalance
```


表 10-1 IRQBALANCE 服务环境变量说明

环境变量	含义	取值
IRQBALANCE_BANNED_CPUS	标识出不参与中断均衡的CPU。	取值为CPU位图的形式，如“00ff”表示0-7号CPU不参加中断均衡，默认不定义。
IRQBALANCE_ONESHOT	只进行一次中断均衡，完成后即退出服务。	该环境变量被赋值时即认为ONESHOT模式生效，默认不生效。
IRQBALANCE_ARGS	增加启动服务时的自定义命令选项。	各命令选项的说明请参阅表2。

中断均衡服务的命令原型以及各选项的说明，以下参数适用于ARM版本。

```
irqbalance [--oneshot | -o][--debug | -d][--foreground | -f][--journal | -j][--hintpolicy= | -h
[exact|subset|ignore]][--powerthresh= | -p <off> | <n>][--banirq= | -i <n>][--
policyscript=<script>][--pid= | -s <file>][--deepestcache= | -c <n>][--interval= | -t <n>][--
verifyhint= | -v <n>]
```

表 10-2 IRQBALANCE 命令选项说明

命令选项	含义	取值
--oneshot | -o	只进行一次中断均衡，完成后即退出服务。	-
--debug | -d	输出调试信息，打开该选项会默认打开foreground选项。	-
--foreground | -f	使中断均衡服务工作在前台，pid不再写入pid选项指定的file文件中。	-

命令选项	含义	取值
--hintpolicy= | -h	指定针对中断亲和偏好的处理策略。	可选策略有且仅有三种： ● exact：在驱动给出中断亲和偏好的情况下，在中断均衡时使用该亲和偏好值作为新的亲和性，但若该值不包含有效CPU时，策略对该中断失效；如果驱动未给出偏好值，则该策略无效； ● subset：在驱动给出中断亲和偏好的情况下，在中断均衡时将亲和偏好作为亲和掩码进一步缩小根据中断均衡算法得出的亲和的范围。若两者交集为空，则对该中断不进行均衡；若两者交集不为空，且包含有效CPU时，策略生效。如果驱动未给出偏好值，则该策略无效； ● ignore：无视驱动给出的中断亲和偏好，完全按照中断均衡算法来确定中断均衡后亲和性。 **说明** 1、默认为subset。 2、大小写不敏感。
--powerthresh= | -p	设置CPU的节能策略。	取值仅可能为两种： ● 设置一个阈值n，当处于“空闲”的CPU数量超过这个阈值时，将一个“空闲”的CPU进入“节能模式”。 ● 设置为“off”，表示中断均衡服务不对CPU进行节能处理。 **说明** 1、默认为off。 2、由于中断均衡服务只观察中断的负载，而CPU上的负载不仅仅是中断的负载，所以这里的“节能模式”并不是让CPU进入idle状态或是被disable掉，而只是一个标记，有该标记的CPU将不被均衡中断的负载直至被取消标记。
--banirq= | -i	标识某个特定的linux中断号不参与中断均衡。	取值为某一个有效的linux中断号。

命令选项	含义	取值
--policyscript= | -l	用户自定义的均衡策略，在收集中断信息时，该策略将被应用于所有的中断。	取值为策略脚本的文件路径。策略脚本在执行时会被传入两个参数，依次为sysfs中该中断所属设备的路径和linux中断号。脚本应输出有效的均衡策略到标准输出。有效的均衡策略为以下几种键值对： ● ban=[true | false]。若value为true则不对该中断进行均衡，若未false则反之。 ● balance_level=[none | package | cache | core]。该策略用于重定义该中断的均衡粒度。有效的均衡等级为“none”、“package”、“cache”和“core”，分别对应CPU拓扑层级中的四种粒度“NUMA”、“CPU Package”、“Cache Domain”以及“CPU Core”。 ● numa_node=n。若n为存在于CPU拓扑中的某个有效的numa节点值，则将该设备的numa节点值设为n。
--pid= | -s	仅在服务运行过程中并且file有效时，服务会将进程号写入file中。	取值为pid文件路径。foreground模式打开后该选项失效。
--deepestcache= | -c	为了减小中断迁移带来的性能损失（主要为处理中断时的cache未命中），该选项可指定中断均衡时应考虑的cache共享范围。	设值为中断均衡时参考的最远端cache层级： ● 0：Level 1 data cache ● 1：Level 1 instruction cache ● n：Level n cache，n≥2
--interval= | -t	指定中断均衡的周期。	设值为执行中断均衡的周期，默认为10s。
--verifyhint= | -v	指定轮询中断亲和偏好的周期	设值为轮询中断亲和偏好的周期，有效值至少为1s，默认为2s。

在EulerOS版本中，irqbalance服务被设置为开机自启动，可通过以下命令查询该服务当前的状态：

```
systemctl status irqbalance
```

默认的启动方式为：

```
/usr/sbin/irqbalance --foreground $IRQBALANCE_ARGS
```

自定义启动选项为：

```
IRQBALANCE_ARGS="--policyscript=/etc/sysconfig/irqbalance.rules --hintpolicy=subset --interval=10"
```


其中“--policyscript”选项指定的是用户自定义的均衡策略脚本，由用户自己编写和维护。

当用户需要定制服务（包括修改环境变量、命令选项以及均衡策略脚本）时，为了使定制内容生效，请按照以下步骤执行：

步骤1 关闭中断均衡服务。

```
systemctl stop irqbalance      # stop service
systemctl status irqbalance    # confirm service stopped
```

步骤2 进行定制，可参考表1中环境变量的说明以及表2中命令选项的说明。

步骤3 开启中断均衡服务。

```
systemctl start irqbalance     # start service
systemctl status irqbalance    # confirm service started
```

----结束

注意

建议按此流程操作使得定制生效。若定制的内容不会被中断均衡服务动态修改，那么也可以直接进行定制，定制完成后重启服务使定制内容生效。

实例分析

- 配置中断使其始终在指定核上处理，即所谓的“绑核”。

  在Linux系统中其实是不存在“绑核”接口的，所谓的“绑核”本质上是通过以下两步来达到“绑定”效果的：

  a. 将中断的亲和性设为某个CPU核；
  b. 保证该中断的亲和性不被修改。

  “绑定”方法主要有两种：调用内核态接口完成或者调用用户态接口完成。这里只介绍后者，欲知前者详情，请查询内核相关文档。

  以“将8号中断绑定至2号CPU”为例：

  a. 停止中断均衡服务：
  ```
  systemctl stop irqbalance
  ```
  b. 在策略脚本中加入对该中断的屏蔽策略，
  ```
  vi /etc/sysconfig/irqbalance.rules
  ```
  或者修改配置文件中的环境变量：
  ```
  IRQBALANCE_ARGS="x --ban=8"  # where x is the original setting
  ```
  说明

  由于linux中断号在每次系统启动后可能会变化，所以策略脚本的方式比修改配置文件的方式更简单、通用。

  c. 修改8号中断的CPU亲和性为CPU2：
  ```
  echo 4 > /proc/irq/8/smp_affinity
  ```
  d. 开启中断均衡服务：
  ```
  systemctl start irqbalance
  ```


> **⚠注意**
>
> 绑核意味着该中断不会被均衡，这可能带来一定的风险，EulerOS不建议绑核。若绑定的核下线了，则该绑定关系会失效。若确实需要绑核，那请在绑定的核下线后重新绑定一个新的核。

- 指定某些核为特殊用途不参与负载均衡。

  以“屏蔽CPU3不参与中断均衡”为例：

  a. 修改配置文件将CPU3配置成不参与中断均衡：
  ```
  IRQBALANCE_BANNED_CPUS=8
  ```
  b. 重启中断均衡服务：
  ```
  systemctl restart irqbalance
  ```

- 配置中断在指定的CPU拓扑层级间进行均衡。

  以“配置6号中断在core级别进行均衡”为例：

  a. 在策略脚本中指定6号中断的均衡等级为“core”级别，正确修改策略脚本后可手动测试修改是否正确：
  ```
  linux:~ # . /etc/sysconfig/irqbalance.rules x 6
  balance_level=core
  linux:~ #
  ```
  b. 重启中断均衡服务：
  ```
  systemctl restart irqbalance
  ```

- 配置特定中断在指定范围的核间进行均衡。

  以“配置5号中断在CPU{1,3,5,7}四个核间均衡”为例：

  a. 在5号中断所属的设备驱动中调用CPU亲和偏好的配置接口设定该中断的亲和偏好为CPU{1,3,5,7}四个核：
  ```
  #include <linux/interrupt.h>
  ...
  /* irq is 5, mask indicates CPU{1,3,5,7} */
  irq_set_affinity_hint(irq, mask);
  ```
  b. 编译驱动，更换OS镜像后启动系统（如果必要的话）。

- irqbalance.rules的示例

  进程会把每个中断的路径和中断号作为参数传给irabalance.rules脚本，会根据脚本的输出设置该中断的ban,balance_level,numa_node这三个属性值，输出格式为ban=***。

  示例：
  ```
  #/bin/bash

  if [ $2 -eq "18" ]; then
      echo "ban=true"
      echo "balance_level=core"
      echo "numa_node=1"
  else
      echo > /dev/null
  fi
  ```
  该示例在中断号为18的时候输出ban,balance_level,numa_node的值，覆盖18号中断这三个值的默认值，当参数为其他中断号时，该脚本不输出任何值，中断的这三个值为默认值。

11 Repo 部署说明

11.1 概述

通过mkrepo工具将EulerOS提供的镜像*EulerOS-V2.0SPx-x86_64-dvd.iso*创建为repo源，并使用nginx进行repo源部署，提供http服务。

说明

EulerOS-V2.0SPx-x86_64-dvd.iso为EulerOS的镜像文件名称，其中SP5简称EulerOS 2.5。

11.2 创建/更新本地 repo 源

使用mkrepo和mount挂载，将EulerOS 2.5的镜像EulerOS-V2.0SPx-x86_64-dvd.iso创建为repo源，并能够对repo源进行更新。

11.2.1 获取 ISO 镜像

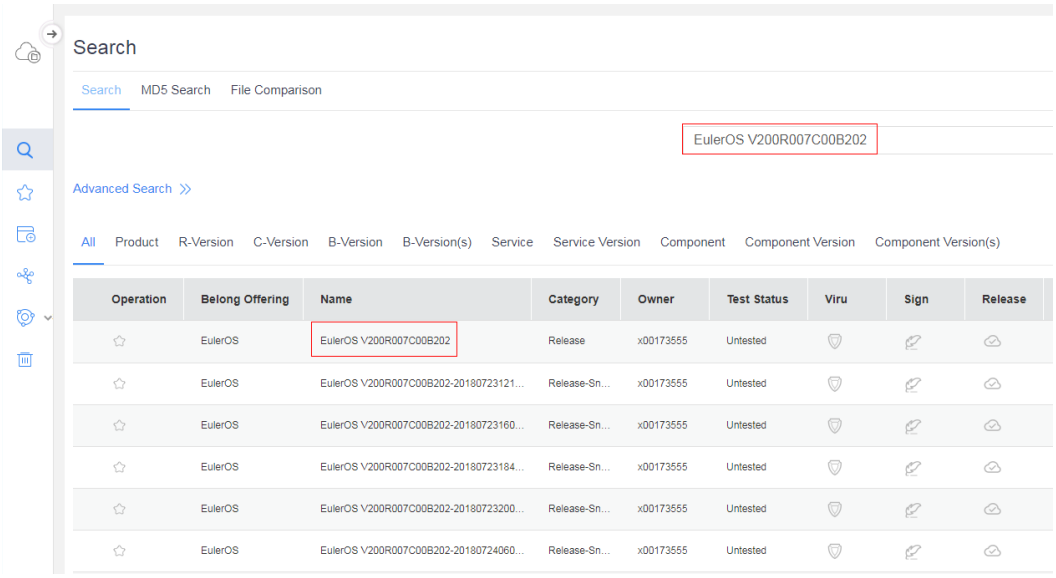
通过CMC上获取EulerOS 2.5版本的ISO。

获取包路径

步骤1 登录CMC，网址为**https://cmc.rnd.huawei.com/cmcversion/**。

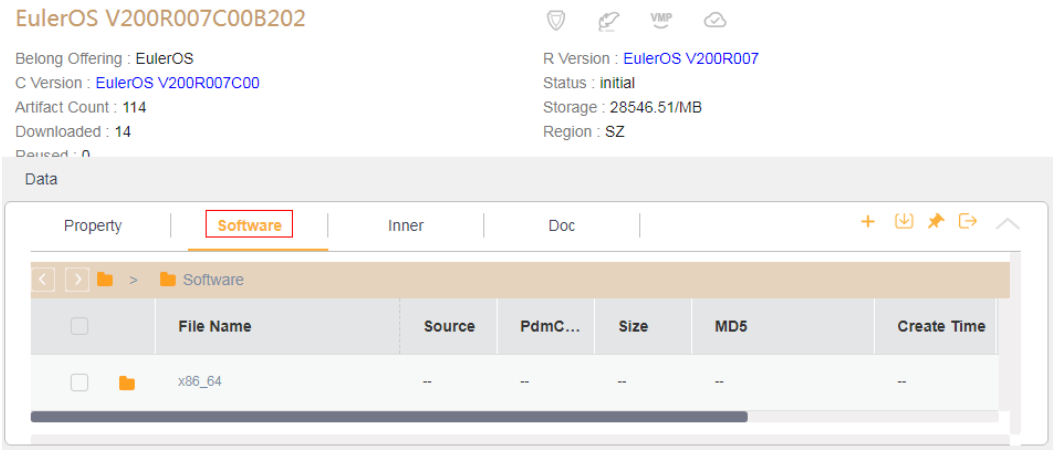
步骤2 查找EulerOS_V200R007C00Bxxx版本，以“EulerOS V200R007C00B202”为例。

图 11-1 从 CMC 上获取包



步骤3 点击发布包名称，在Software相应目录下获取EulerOS_V200R007C00Bxxx.tar.gz。

图 11-2 获取发布包



----结束

11.2.2 挂载 ISO 创建 repo 源

使用mount命令挂载镜像文件。

示例如下：

```
mount /home/Euler/EulerOS-V2.0SP5-x86_64-dvd.iso  /mnt/
```

挂载好的mnt目录如下：

```
.
|—— EFI
|—— images
|—— isolinux
|—— LiveOS
|—— Packages
|—— repodata
|—— ks
```

```
├── TRANS.TBL
└── RPM-GPG-KEY-EulerOS
```

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-EulerOS为EulerOS的签名公钥。

11.2.3 mkrepo 创建 repo 源

使用repo构建工具mkrepo构建repo源，mkrepo集成rsync、createrepo等命令，mkrepo的使用方式如下：

```
mkrepo
Usage: mkrepo -i isoname -d repodirectory
Options:
        -i, --iso        Specify the iso to make
        -d, --dir        Specify the directory to make repo
        --key-dir        GPG key directory
        -h, --help       Display help information.
Example:
        ./mkrepo -i EulerOS.iso -d /home/Euler/
```

其中，-i/--iso表示ISO所在的路径，-d/--dir表示要构建的repo源目录，--key-dir表示EulerOS提供的GPG公钥存放目录，默认存放在repo的父目录，也可指定。

示例如下：

```
mkrepo -i /home/Euler/EulerOS-V2.0SP5-x86_64-dvd.iso -d /srv/repo/os/2.5/base/x86_64 --key-
dir /srv/repo/
```

构建好的repo目录如下：

```
.
├── os
│   └── 2.5
│       └── base
│           └── x86_64
│               ├── Packages
│               └── repodata
└── RPM-GPG-KEY-EulerOS
```

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-EulerOS为EulerOS的签名公钥。

11.2.4 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
  ```
  mkrepo -i /home/Euler/EulerOS-V2.0SP5-x86_64-dvd.iso -d /srv/repo/os/2.5/base/x86_64 --key-
  dir /srv/repo
  ```

  说明

  这里EulerOS-V2.0SP5-x86_64-dvd.iso表示新版本EulerOS的iso文件。

- 在repo源的Packages目录下添加rpm包，然后更新repo源，可通过createrepo命令更新repo源
  ```
  createrepo --update --workers=10 /srv/repo/os/2.5/base/x86_64
  ```
  其中，--update表示更新，--workers表示线程数，可自定义。

对于update升级目录，同样用mkrepo构建repo源，保存到update目录下：

```
mkrepo -i /home/Euler/EulerOS-V2.0SP5-x86_64-dvd.iso -d /srv/repo/os/2.5/update/x86_64
```


11.3 远端存储 repo 源

如果开通了华为对象存储服务（OBS），可将构建好的repo源通过s3fs-fuse工具上传，今后可直接挂载桶使用repo源。

11.3.1 s3fs-fuse 工具

s3fs是一款用于Amazon S3在Linux环境下的命令行工具，并为商业和私人免费使用，只需支付对应的存储服务的费用。它的主要功能是能够把桶（Bucket）挂载到本地目录，进行文件、文件夹的拷入、拷出和删除等操作，即完成对桶内文件、文件夹的上传、下载和删除操作。

s3fs-fuse已经集成到EulerOS-V2.0SP5-x86_64-dvd.iso中，若需要安装可通过配置repo，直接yum install进行安装：

```
yum install -y s3fs-fuse
```

11.3.2 s3fs-fuse 使用说明

使用之前，需要事先在华为企业云服务官方网站（**www.hwclouds.com**）中注册华为云服务账号，并开通华为对象存储服务（OBS），开通方法请参考**《华为对象存储服务（OBS）用户指南》**。

说明

这里以华为企业云为例进行说明，其他云平台的获取方式请参考云平台的说明。

1. 将使用凭证保存到/etc/passwd-uds中：
   ```
   echo 'EDYOURYCMXYYOQJBET5U:AEWNLQVKYHHFW2PX3FUVBRCVR00CNCWO57IIDU4T' > /etc/passwd-uds
   ```
   说明

   EDYOURYCMXYYOQJBET5U:AEWNLQVKYHHFW2PX3FUVBRCVR00CNCWO57IIDU4T为使用凭证示例，请用户根据实际情况输入对应的使用凭证。
2. 创建目录/srv/repo，挂载euleros-repo桶：
   ```
   mkdir repo
   s3fs euleros-repo /srv/repo -o host=http://10.107.193.157:5080,passwd_file=/etc/passwd-uds,use_path_request_style,uid=0,gid=0,nomultipart
   ```
   说明

   10.107.193.157为内部测试使用的服务器，用户需要根据实际环境的IP地址进行配置。
3. 卸载euleros-repo桶：
   ```
   umount /srv/repo
   ```

11.3.3 同步 repo 源

1. 在本地/home/Euler/目录下构建repo源：
   ```
   mkrepo -i /home/Euler/EulerOS-V2.0SP5-x86_64-dvd.iso -d /home/Euler/os/2.5/base/x86_64
   ```
2. 将本地构建的repo同步到挂载的euleros-repo桶中：
   ```
   rsync -a --progress /home/Euler/os /srv/repo
   ```
   说明

   同步速度比较慢，请耐心等待。
3. repo源同步完成之后，可直接挂载euler-repo桶当作repo源使用。

11.3.4 参考文档

华为对象存储服务在这里将不详细介绍，可参考如下文档：

- **华为对象存储服务-第三方客户端用户指南（s3fs）**
- **对象存储服务-客户端操作指南**
- **华为对象存储服务-帮助中心**
- **OBS客户端**

11.4 部署远端 repo 源

安装操作系统EulerOS2.5（EulerOS-V2.0SP5-x86_64-dvd.iso），在EulerOS2.5上通过nginx部署repo源。

11.4.1 nginx 安装与配置

1. nginx已经集成到EulerOS-V2.0SP5-x86_64-dvd.iso中，若需要安装可通过配置repo，直接yum install进行安装：

```
yum install -y nginx
```

2. 安装nginx之后，配置/etc/nginx/nginx.conf

```
user  root;
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         /srv/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;
        }

    }

}
```

11.4.2 启动 nginx 服务

1. 通过systemd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
```

   - 图11-3表示nginx服务启动成功

图 11-3 nginx 服务启动成功

```
[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
```

图 11-4 nginx 服务启动失败

```
[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.
```

如图11-4所示nginx服务创建失败，是由于目录/var/spool/nginx/tmp/client_body创建失败，手动进行创建，类似的问题也这样处理：

```
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
```


11.4.3 repo 源部署

1. 创建nginx配置文件/etc/nginx/nginx.conf中指定的目录/srv/repo：
   ```
   mkdir -p /srv/repo
   ```
2. SELinux设置为宽容模式：
   ```
   setenforce permissive
   ```
   说明

   repo server重启后，需要重新设置。
3. 设置防火墙规则，开启nginx设置的端口（此处为80端口），通过firewall设置端口开启：
   ```
   firewall-cmd --add-port=80/tcp --permanent
   firewall-cmd --reload
   ```
   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，如图11-5：

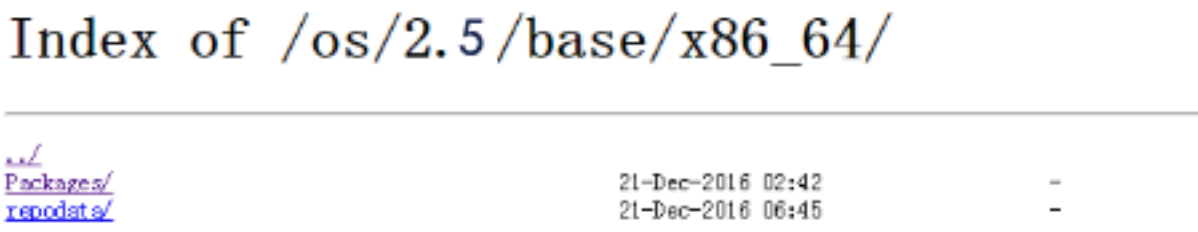
   图 11-5 nginx 部署成功

5. 通过下面几种方式将repo源放入到/srv/repo下：
   - 在/srv/repo下直接创建repo源
     ```
     mkrepo -i /home/Euler/EulerOS-V2.0SP5-x86_64-dvd.iso -d /srv/repo/os/2.5/base/x86_84 --key-dir /srv/repo
     ```
     EulerOS-V2.0SP5-x86_64-dvd.iso存放在/home/Euler目录下。
   - 在/srv/repo下创建repo源的软链接
     ```
     ln -s /home/Euler/os /srv/repo/os
     ```
     /home/Euler/os为已经创建好的repo源，/srv/repo/os将指向/home/Euler/os。
   - 通过s3fs-fuse工具挂载远端存储的repo源桶
     ```
     s3fs euleros-repo /srv/repo/ -o host=http://10.107.193.157:5080,passwd_file=/etc/passwd-uds,use_path_request_style,uid=0,gid=0,nomultipart
     ```
     其中，“euleros-repo”为远端存储的repo源桶，“10.107.193.157”为内部测试IP。

如图11-6所示，repo源部署成功：

图 11-6 repo 部署成功



11.5 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体包，无须繁琐地一次次下载和安装。

11.5.1 repo 配置为 yum 源

构建好的repo可以配置为yum源使用，在/etc/yum.repos.d/目录下创建***.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EulerOS-2.5-Base.repo文件，使用构建的本地repo作为yum源，Euler0S-2.5-Base.repo的内容如下：

  ```
  [base]
  name=base
  baseurl=file:///srv/repo/os/2.5/base/x86_64
  enabled=1
  gpgcheck=1
  gpgkey=file:///srv/repo/RPM-GPG-KEY-EulerOS
  ```

  说明

  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为签名公钥的存放路径。

- 配置http服务器yum源

  在/etc/yum.repos.d目录下创建EulerOS-2.5-Base.repo文件，使用http服务端的repo作为yum源，Euler0S-2.5-Base.repo的内容如下：

  ```
  [base]
  name=base
  baseurl=http://192.168.1.2/os/2.5/base/x86_64
  enabled=1
  gpgcheck=1
  gpgkey=http://192.168.1.2/RPM-GPG-KEY-EulerOS
  ```

  说明

  “192.168.1.2”为示例地址，请用户根据实际情况进行配置。

11.5.2 repo 优先级

当有多个repo源时，可通过在.repo文件的priority参数设置repo的优先级。其中，1为最高优先级，99为最低优先级，如给EulerOS-2.5-Base.repo配置优先级为2：

```
[base]
name=base
baseurl=http://192.168.1.2/os/2.5/base/x86_64
enabled=1
priority=2
gpgcheck=1
gpgkey=http://192.168.1.2/RPM-GPG-KEY-EulerOS
```

说明

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为签名公钥的存放路径。

11.5.3 yum 相关命令

yum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：

```
yum <command> <packages name>
```

常用的命令如下：

- 安装
  ```
  yum install <packages name>
  ```
- 升级
  ```
  yum update <packages name>
  ```
- 回退
  ```
  yum downgrade <packages name>
  ```
- 检查更新
  ```
  yum check-update
  ```
- 卸载
  ```
  yum remove <packages name>
  ```
- 查询
  ```
  yum search <packages name>
  ```
- 本地安装
  ```
  yum localinstall <absolute path to package name>
  ```
- 查看历史记录
  ```
  yum history
  ```
- 清除缓存目录
  ```
  yum clean all
  ```
- 更新缓存
  ```
  yum makecache
  ```


12 update-motd 使用说明

12.1 EulerOS update-motd 安装

12.1.1 简介

update-motd是一款专用于EulerOS环境下的工具，它的主要功能是通过检测系统已安装的软件并与yum源中的软件进行对比，当软件有新的版本出现时，会在用户上线时以登录界面的形式提醒用户有软件需要升级，该更新提醒包括已安装软件包与组更新的提醒。

12.1.2 安装

配置repo源，参考**11 Repo部署说明**。

update-motd已集成到EulerOS系统。

查看update-motd软件是否安装，执行命令：

```
rpm -qa | grep update-motd
```

没有安装update-motd，执行命令：

```
yum install update-motd
```

结果如下：

```
Resolving Dependencies
--> Running transaction check
---> Package update-motd.noarch 0:0.0.1-1 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package              Arch             Version              Repository        Size
================================================================================
Installing:
 update-motd          noarch           0.0.1-1              EulerOS-base      8.1 k
```



```
Transaction Summary
================================================================================
Install  1 Package

Total download size: 8.1 k
Installed size: 4.8 k
Is this ok [y/d/N]: y
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Installing : update-motd-0.0.1-1.noarch                                   1/1
  Verifying  : update-motd-0.0.1-1.noarch                                   1/1

Installed:
  update-motd.noarch 0:0.0.1-1

Complete!
```

结果显示“Complete”表示软件安装成功。

12.1.3 update-motd 使用

安装完成后执行下面命令启用服务：

```
systemctl daemon-reload
systemctl enable update-motd.timer
systemctl start update-motd.timer
```

服务启动后，重新登录时会显示系统信息及需要更新软件数目。

```
Authorized users only. All activities may be monitored and reported.
root@192.168.0.242's password:
Last login: Wed Jan 11 00:50:17 2017 from 192.168.0.1
******************************************************************
*                                                                *
* This system is for the use of authorized users only.  Usage of *
* this system may be monitored and recorded by system personnel. *
*                                                                *
******************************************************************

System load:  0.09
Memory usage: 6.8%  System uptime: 1 days, 2 hours
Usage on /: 37%  Swap usage: 0.0%
IP Addresses: 192.168.0.242
```

说明

示例系统无可更新软件，不显示软件更新数目。

12.1.4 更新检测

- 检测是否有已安装的软件包需要更新，执行命令：

  ```
  yum check-update
  ```

  示例结果如下：

  ```
  Loaded plugins: fastestmirror, langpacks, priorities
  Loading mirror speeds from cached hostfile
  ModemManager.x86_64                     1.1.0-8.git20130913           base
  ModemManager-glib.x86_64                1.1.0-8.git20130913           base
  NetworkManager.x86_64                   1:1.0.6-27                    base
  NetworkManager-adsl.x86_64              1:1.0.6-27                    base
  NetworkManager-glib.x86_64              1:1.0.6-27                    base
  NetworkManager-libnm.x86_64             1:1.0.6-27                    base
  NetworkManager-libreswan.x86_64         1.0.6-3                       base
  NetworkManager-team.x86_64              1:1.0.6-27                    base
  ```

```
NetworkManager-tui.x86_64                1:1.0.6-27                   base
PackageKit.x86_64                        1.0.7-5                      base
PackageKit-command-not-found.x86_64      1.0.7-5                      base
PackageKit-glib.x86_64                   1.0.7-5                      base
PackageKit-gstreamer-plugin.x86_64       1.0.7-5                      base
PackageKit-gtk3-module.x86_64            1.0.7-5                      base
PackageKit-yum.x86_64                    1.0.7-5                      base
SDL.x86_64                               1.2.15-14                    base
abattis-cantarell-fonts.noarch           0.0.16-3                     base
abrt.x86_64                              2.1.11-36                    base
abrt-addon-ccpp.x86_64                   2.1.11-36                    base
```

- 组更新

```
yum group update
```

示例结果如下：

```
Installing for group upgrade "Base":
 kpatch-runtime                 noarch           1.0-2.2.1
source                 5.9 k
Updating:
 euleros-config                 x86_64           1.0-7
source                 5.1 k
 euleros-latest-release         noarch           2.0SP5-1501643560.2.5.RC3.SPC1
source                 4.5 k
 file-roller-nautilus           x86_64           3.14.2-7.h1
source                 29 k
 firefox                        x86_64           52.2.0-1.h1
source                 83 M
 gnome-backgrounds              noarch           3.14.1-1.h1
source                 16 M
 gnome-clocks                   x86_64           3.14.1-1.h1
source                 386 k
 gnome-terminal                 x86_64           3.14.3-3.1.h3
source                 1.1 M
 initial-setup-gui              x86_64           0.3.9.31-1.h2
source                 25 k
```

升级部分或全部软件包，参考**12.2.2 yum相关命令**。

12.2 update 相关命令

12.2.1 update-motd 控制命令

1. 重新加载服务，对所有服务都适用：

```
systemctl daemon-reload
```

2. 使能update-motd.service服务：

```
systemctl enable update-motd.service
```

3. 开启update-motd .service服务：

```
systemctl start update-motd.service
```

4. 查看update-motd .service服务：

```
systemctl status update-motd.service
```

5. 使能update-motd.timer服务：

```
systemctl enable update-motd.timer
```

6. 开启update-motd .timer服务：

```
systemctl start update-motd.timer
```

7. 查看update-motd .timer服务：

```
systemctl status update-motd.timer
```


12.2.2 yum 相关命令

1. 检测需要更新软件包列表：
   ```
   yun check-update
   ```
2. 已安装软件版本列表：
   ```
   yum list installed
   ```
3. 一次升级所有可升级软件新版本：
   ```
   yum update
   ```
4. 升级安装：
   ```
   yum update package
   ```
5. 降级安装：
   ```
   yum downgrade package
   ```
6. 删除软件：
   ```
   yum remove | erase package
   ```
7. 列出已安装软件包信息：
   ```
   yum info package
   ```

   说明

   yum用法：**http://3ms.huawei.com/hi/blog/667685_2098989.html**

13 可信计算

本章介绍可信计算相关的组件功能、使用方法、以及可信计算的约束说明。

13.1 概述

概述

TCG组织把“可信计算”定义为：“可信计算”是指软件和硬件能够按照它们被设计的行为运行。它的可信概念为行为可信，它建立在可信测量、可信报告、可信管理为基础的可信平台概念上。可信平台模块TPM是可信计算平台的信任根，是可信计算的核心模块。通过将密钥和加解密运算引擎集成到TPM模块中，为实现系统安全功能提供了安全的可信基点。

可信计算根据实现目的和应用场景，又可以简单区分为可信启动和安全启动，前者在启动过程中度量所有安全相关的组件，并在完成启动过程以后，由本地进程或第三方的挑战服务器来验证启动过程是否同预计一致，也就是确保启动过程是安全的（这里所说的启动过程包括BIOS、GRUB、OS、关键软件等）；安全启动则是在启动过程中通过本地预存的参考值（或数字签名），直接对度量值进行验证，如果发现不一致，立即终止启动过程。

由此可见，在以前的传统计算机/服务器领域，没有TPM和可信计算，那么设备上的BIOS/UEFI、Grub、OS kernel等都有可能被篡改，通过植入恶意软件，进行进一步的业务数据窃取或设备远程控制，而这种篡改在过去是无法被监控和发现的；可信计算的目的就是要从设备上电开始，逐步建立可信链，通过前一步度量后一步，保证后一步被执行的代码和相应的资源是可信任和可被审计的，帮助管理员了解当前软硬件环境是否为预期的状态，如果出现篡改情况，可及时终止执行或给出告警。

功能描述

在支持可信计算TPM2.0的硬件平台上，EulerOS提供如下功能来支撑可信计算：

1. 在引导模块Grub2中支持TPM2.0，实现对Kernel、Initramfs、启动参数、以及Modules的度量，并将度量值扩展到TPM2.0设备的PCR8和PCR9中；
2. EulerOS Kernel支持IMA度量框架，并且默认采用SHA256算法计算文件的hash值和扩展PCR10，默认不启用度量；
3. 支持业务管理员替换IMA度量策略；
4. EulerOS在x86平台的ISO安装镜像中包含TSS2.0协议栈、Resourcemgr、TPM2.0-Tools、euler-tpm2-tools、euler-tpm2-la组件RPM包，默认不安装，管理员可以在安装界面选择安装。

注意

可信计算Grub2、IMA度量框架对系统及业务文件只做hash运算，不会分析、保存用户敏感数据。

架构组成

可信计算相关功能主要涉及到以下几个部分，如表13-1所示。

表 13-1 可信计算架构

组成	说明
TPM2.0设备	硬件芯片，通过扣卡或者引脚焊接方式装备到服务器主板上；
Grub2	EulerOS引导模块；
TPM2.0驱动	EulerOS默认自带的TPM2.0设备驱动程序；
IMA度量框架	Linux内核中的完整性度量框架，根据度量策略对系统中指定的文件进行hash计算，将hash值保存到IMA日志中，并扩展到TPM2.0设备的PCR10中；
IMA Policy	IMA框架度量策略；
TSS2.0协议栈	对上层应用提供TPM2.0设备访问接口；
Resourcemgr	对TPM2.0设备进行底层访问的封装，向上提供Socket接口给TSS2.0协议栈使用；
TPM2.0-Tools	封装了一套TPM2.0的访问、管理命令，这套命令向下通过Resourcemgr访问TPM2.0设备，管理员可在命令行中操作相关命令，以便对TPM2.0设备进行管理；
euler-tpm2-tools	提供TPM2.0芯片口令初始化功能； 提供一个用于修改TPM2.0芯片口令的命令； 安装IMA默认策略到系统（默认不启用）；
euler-tpm2-la（本地证明）	提供两个命令，一个用于在管理员认可的某个可信状态时创建受保护文件的基准值，另一个命令用于在此之后的任意时间的可信校验；

13.2 约束限制

EulerOS作为通用操作系统平台，安装镜像包含可信计算组件，但由于Linux内核版本众多，以及硬件平台差异较大，对于可信计算的使用作出如下限制：

1. 仅支持TCG的TPM2.0规范；
2. TPM2.0芯片目前只在英飞凌SLB9665完成相关兼容性验证；
3. 仅支持OS安装于物理裸机场景，不支持虚拟机场景；
4. 仅支持UEFI启动，不支持Legacy方式；
5. IMA框架只支持SHA256算法，不兼容SHA1算法（TPM1.2）；
6. IMA框架只支持度量场景使用，不支持appraise场景使用；
7. EulerOS在x86平台ISO安装镜像中默认不安装可信计算相关组件，需要在安装部署界面选择安装；
8. EulerOS默认不启用IMA度量，管理员可通过配置启用；
9. Resourcemgr无法良好的支持多进程（线程），上层应用在调用TSS2.0协议栈接口时请确保已经启动Resourcemgr服务，且不在多进程（线程）中并发调用TSS2.0协议栈；
10. 受限于TCG规范，EulerOS无法提供TPM2.0芯片清除功能，管理员可以通过BIOS对TPM2.0芯片进行清除操作（不同硬件平台存在差异，请依据实际情况进行操作）。

13.3 功能说明

本节介绍TPM芯片初始化、Grub2、IMA、TSS2.0协议栈、TPM2.0-Tools以及本地证明等提供的基本功能和特性。

13.3.1 TPM 芯片

TPM芯片是服务器、单板上一颗独立的安全芯片，一般以扣卡形式存在，可以通过插拔方式更换，也可以不安装，一般不支持热插拔。

TPM芯片实现符合TCG相应的规范，目前主要的供应商有英飞凌、意法半导体、国民技术。

TPM芯片主要提供芯片级的加解密算法、hash扩展寄存器、安全随机数、秘钥保存、隐私数据封存等功能。

EulerOS目前只支持TPM2.0规范。

初始化与 Lockout

TPM2.0芯片有相应的自我保护机制，比如多次授权失败（非Lockout授权，具体失败次数等需要参考芯片厂商手册，不同厂商设置不同），就会进入授权锁定状态，而Lockout授权如果失败1次，就会进入Lockout状态，此时只能等待超时（不同厂商设置时间不同）或者通过硬件清除（BIOS菜单中清除）的方式来解除Lockout。

所以在服务器上安装EulerOS之前时，建议用户确保TPM2.0芯片处于出厂默认状态（可以在BIOS菜单中对TPM芯片进行清除操作来恢复出厂状态，EulerOS在完成安装以后，第一次重启的时候会从软件层面进行一次TPM口令初始化操作，如果TPM芯片在这之

前没有进行清除操作，且TPM留存有旧口令，则口令初始化会失败，并进入Lockout状态）。

英飞凌SLB9665芯片Lockout相关参数设置如下：

表 13-2 英飞凌 SLB9665 芯片 Lockout 相关参数

参数	取值	说明
授权失败最大允许次数	32	在访问TPM需要授权的命令时（非Lockout授权），如果授权失败，计数器就加1，当计数器达到32时，就会进入授权锁定状态，无法再进行授权相关的访问。
计数器恢复间隔时间	7200秒	在持续的7200秒内如果没有授权相关操作，或者没有授权失败操作，计数器减1，直到0为止。
Lockout恢复最大允许授权失败次数	1	可以通过Lockout恢复命令对计数器进行清除，但Lockout授权必须一次正确，否则就会锁住，进入Lockout状态。
Lockout恢复间隔时间	86400秒	进入Lockout状态以后，必须等待86400秒或者在BIOS中进行清除，才能恢复。

⚠注意

在BIOS中对TPM2.0芯片进行清除操作不可恢复，请确保TPM2.0上无有用数据。

默认口令

EulerOS安装完毕后第一次启动时会对TPM2.0芯片进行口令初始化，将Owner、Endorsement、Lockout口令初始化为如下口令：

```
Xhdfhwbc@573
```

这个过程不需要用户参与，初始化之后用户可以修改该口令。修改口令请参考**13.4.2 TPM口令修改**。

注意

如果TPM2.0芯片不是全新的，且留存有旧口令，则默认口令初始化会失败，此时TPM2.0芯片将进入Lockout状态，需要等待“Lockout恢复间隔时间”之后才能自动解锁，或者在BIOS中进行TPM2.0芯片清除动作，完成TPM初始化和解锁。除了Lockout锁定之外，默认口令初始化失败不会影响任何TPM相关的业务功能，但在所有需要TPM口令认证的场景，请使用旧口令进行认证。

TPM2.0芯片默认口令请在第一次系统启动以后立即修改，并定期更新，避免密码泄露后，产生安全风险。

13.3.2 Grub2

Grub2是Linux系统的引导组件，用于引导Linux内核以及加载相应模块，作为可信计算的一个环节，Grub2负责度量Linux内核、Initramfs，以及加载模块的hash值，并将其扩展到TPM2.0的寄存器PCR8和9中。

由于TCG规范中对于TPM2.0没有定义Legacy模式，所以服务器必须以UEFI方式启动，否则Grub2的度量值无法扩展到寄存器中。

EulerOS默认安装支持可信计算的Grub2。

13.3.3 IMA

IMA（Integrity Measurement Architecture）是Linux内核中完整性度量框架，在系统访问文件时对文件进行完整性度量（计算hash值），并将hash值扩展到TPM2.0的PCR10寄存器中。

默认策略

EulerOS选择可信计算组件安装成功以后，默认不启动IMA度量，但在/etc/ima目录下提供了一个默认的IMA策略文件，文件名为ima-default-policy。

```
/etc/ima/ima-default-policy
```

自定义策略

EulerOS在/etc/ima目录下自带默认策略文件，管理员亦可根据需要自定义策略，可以直接修改ima-default-policy文件，然后将该文件命名为ima-policy，也可以重新创建一个ima-policy文件，并将策略内容写入该文件，建议后者。

```
# cd /etc/ima
# vi ima-policy
# ls
ima-default-policy  ima-policy
#
```

策略文件需要符合一定的书写规则：

1. 每行一条策略，文件头尾及策略行之间不允许有空行；
2. 以“#”开头的行为注释行；
3. 每条策略分别由action，func，mask，fsmagic，fsuuid，uid、fowner等关键字定义。

关键字含义及取值如下表：

表 13-3 IMA 策略关键字定义

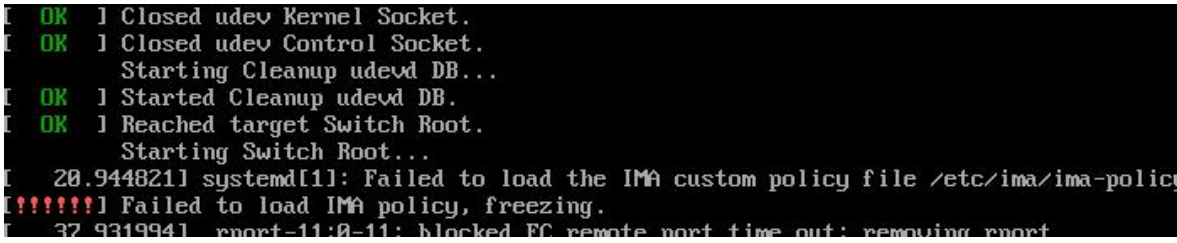
关键字	取值	说明
action	measure，dont_measure，appraise，dont_appraise，audit	表示该条策略具体的动作，也就是符合该策略的文件是否需要被度量或鉴定，一条策略只能选一个action，实际书写策略时不需要出现action字样，比如可以直接书写dont_measure，而不需要写成action=dont_measure。

关键字	取值	说明
func	FILE_CHECK， PATH_CHECK， MODULE_CHECK， FILE_MMAP， MMAP_CHECK， BPRM_CHECK	表示被度量或鉴定的文件类型，一条策略只能选一个func，比如func=FILE_MMAP，具体同mask如何匹配使用，请参考下面maks栏。
mask	MAY_EXEC， MAY_WRITE， MAY_READ， MAY_APPEND	表示文件在做什么操作时将被度量或鉴定，一条策略只能选一个mask，比如mask=MAY_EXEC。 受限于echo、vim等开源机制，MAY_WRITE并不能在文件每次修改后都触发IMA度量，比如vim是通过新建临时文件，再重命名的方式来修改文件内容的，内核并不能发现原文件被修改了，不会触发度量，所以建议不用MAY_WRITE策略，或连同MAY_EXEC，MAY_READ一起使用。 **说明** 并不是所有的func和mask匹配都是有执行效果的，它们的有效匹配关系如下： ● FILE_CHECK只能同MAY_EXEC、MAY_WRITE、MAY_READ匹配使用； ● MODULE_CHECK、MMAP_CHECK、BPRM_CHECK只能同MAY_EXEC匹配使用； ● 匹配关系以外的组合不会产生效果； ● 另外，PATH_CHECK等同于FILE_CHECK，FILE_MMAP等同于MMAP_CHECK，所以这两个func关键字可以忽略，不建议使用。
fsmagic	fsmagic=xxx	xxx是一个16进制魔数，表示文件系统类型，定义在/usr/include/linux/magic.h文件中，默认情况下度量所有文件系统，除非使用dont_measure/dont_appraise来标记不度量/鉴定某文件系统，也就是说对于文件系统的度量/鉴定，采用的是黑名单。一条策略只能选一个文件系统。比如dont_measure fsmagic=0x64626720就表示不度量debugfs文件系统。
fsuuid	fsuuid=xxx	xxx是一个16位的16进制的字符串，表示系统设备uuid，使用逻辑类似于fsmagic。
obj_type	obj_type=xxx	xxx表示文件的类型，一条策略只能选一个类型，比如obj_type=nova_log_t表示nova log类型的文件。
uid	uid=xxx	xxx表示用户id，比如root用户就是0，这里是指哪个用户对文件进行操作，一条策略只能选一个uid。

关键字	取值	说明
fowner	fowner=xxx	xxx表示用户id，比如root用户就是0，fowner表示的是文件的属主是谁，一条策略只能选一个fowner。

注意

- 文件系统的策略，也就是fsmagic和fsuuid标识的策略，采用的是黑名单方式，如果使用dont_measure显式定义出来，就表示该文件系统下面所有的文件都不会被度量，不管是否命中后面其他的策略，如果没有dont_measure显式定义，那么该文件系统下面的文件，如果命中后面的文件相关的策略，就会被度量。
- 文件策略，也就是func，则采用的白名单方式，如果使用measure定义文件策略，则该文件策略命中的文件就会被度量，其他未命中的文件都不会被度量。
- 文件在读取或执行的时候会引起安全问题，写操作只是改变文件内容，本身并不会引起直接的安全问题，所以建议用户在配置mask的时候考虑采用MAY_EXEC和MAY_READ策略，而不只是MAY_WRITE策略。
- 策略文件一定要满足所有的书写规则，否则错误的ima策略配置文件会导致系统启动失败，失败现象如下图所示。

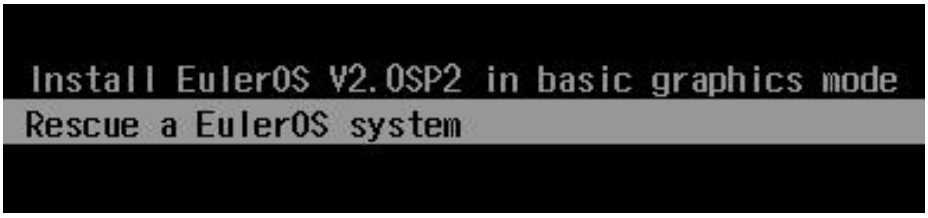


如果因为ima策略文件书写错误而导致上图所示的系统无法启动的问题，解决的方法是：

step1. 登陆iBMC，挂载EulerOS的ISO镜像，如下图所示。

step2. 从光盘启动系统，如下图所示。

step3. 进入光盘系统的急救模式（进入急救模式之后光盘系统会将硬盘系统挂载到/mnt/sysimage目录），首先备份硬盘系统中错误的ima策略文件（cp /mnt/sysimage/etc/ima/ima-policy /mnt/sysimage/etc/ima/ima-error-policy.bak），然后删除硬盘系统中错误的ima策略配置文件（rm /mnt/sysimage/etc/ima/ima-policy），之后重启即可，如下图所示。




```
Rescue

The rescue environment will now attempt to find your Linux installation
mount it under the directory : /mnt/sysimage.  You can then make any ch
required to your system.  Choose '1' to proceed with this step.
You can choose to mount your file systems read-only instead of read-wri
choosing '2'.
If for some reason this process does not work choose '3' to skip direct
shell.

 1) Continue

 2) Read-only mount

 3) Skip to shell

 4) Quit (Reboot)

Please make a selection from the above:  1
================================================================================
================================================================================
Rescue Mount

Your system has been mounted under /mnt/sysimage.

If you would like to make your system the root environment, run the com

        chroot /mnt/sysimage
Please press <return> to get a shell.
When finished, please exit from the shell and your system will reboot.
sh-4.2# rm /mnt/sysimage/etc/ima/ima-policy
sh-4.2# reboot_
```

策略文件举例如下：

```
# PROC_SUPER_MAGIC
dont_measure fsmagic=0x9fa0
# SYSFS_MAGIC
```

```
dont_measure fsmagic=0x62656572
# DEBUGFS_MAGIC
dont_measure fsmagic=0x64626720
# TMPFS_MAGIC
dont_measure fsmagic=0x01021994
# RAMFS_MAGIC
dont_measure fsmagic=0x858458f6
# DEVPTS_SUPER_MAGIC
dont_measure fsmagic=0x1cd1
# BINFMTFS_MAGIC
dont_measure fsmagic=0x42494e4d
# SECURITYFS_MAGIC
dont_measure fsmagic=0x73636673
# SELINUX_MAGIC
dont_measure fsmagic=0xf97cff8c
# measure file type define
measure func=FILE_MMAP mask=MAY_EXEC
measure func=BPRM_CHECK mask=MAY_EXEC
measure func=MODULE_CHECK
# measure func=FILE_CHECK mask=MAY_READ uid=0
```

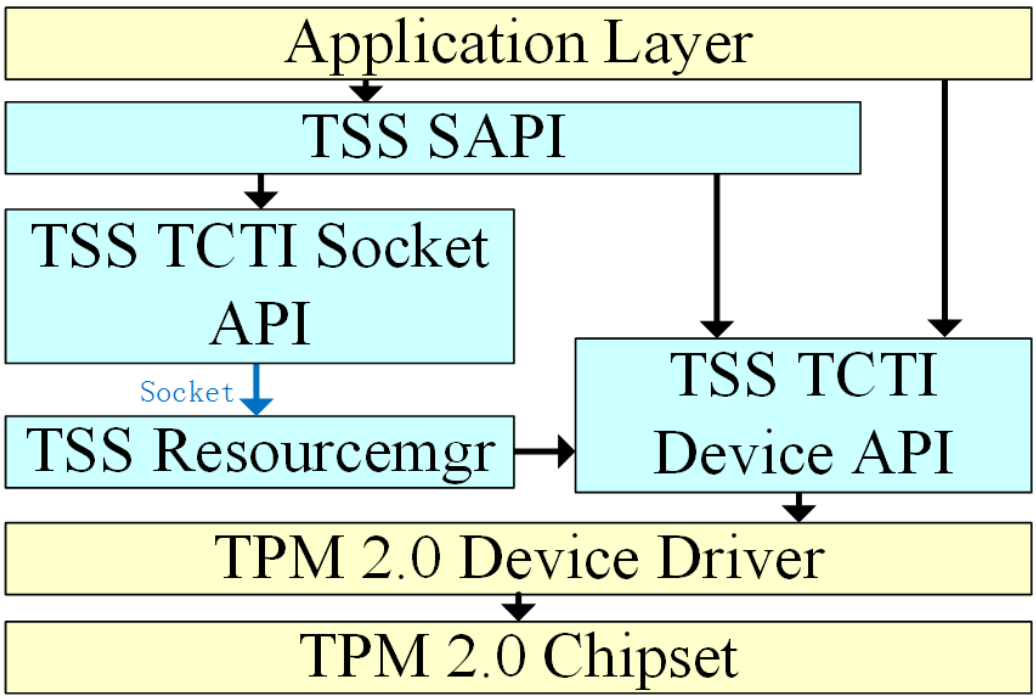
13.3.4 TSS2.0 协议栈

TSS2.0是基于开源社区的一套用于访问TPM2.0芯片的协议栈，上层应用可以通过标准C编程接口进行相关场景的开发。

TSS 架构

TSS2.0协议栈可分为三部分：Resourcemgr服务进程通过对底层TPM2.0设备驱动的封装，向上层应用提供本地Socket访问接口；TCTI（TPM Command Transmission Interface）本身分为两部分：Socket接口和Device接口，为TSS SAPI提供了两种访问TPM芯片的方式；SAPI（TSS system API ）则通过调用TCTI完成对TPM2.0的访问和管理，同时向上提供应用层功能接口。

图 13-1 TSS 协议栈架构图



Resourcemgr

Resourcemgr可认为是一个独立的服务进程，通过调用TCTI的Device API，向上层提供Socket服务。

EulerOS操作系统启动时默认不启动该服务进程，上层应用如果需要通过TSS2.0协议栈操作TPM2.0芯片，并采用本地Socket方式，则必须先启动Resourcemgr，并在使用完成以后关闭该服务。

注意

不建议操作系统启动后就直接启动Resourcemgr，并长期运行。

请避免并发访问Resourcemgr。

Resourcemgr默认安装于/usr/sbin目录下：

```
$ cd /usr/sbin
$ ll
-rwxr-xr-x. 1 root root 136717 Nov 22 02:58 resourcemgr
$
```

可在任何目录下直接启动resourcemgr服务：

```
$ resourcemgr &
[1] 17254
Initializing device TCTI Interface
Initializing Resource Manager
maxActiveSessions = 64
gapMaxValue = 65535
socket created:  0x4
bind to IP address:port:  127.0.0.1:2324
Other CMD server listening to socket:  0x4
socket created:  0x5
bind to IP address:port:  127.0.0.1:2323
TPM CMD server listening to socket:  0x5
Starting SockServer (TPM CMD), socket: 0x5.
Starting SockServer (Other CMD), socket: 0x4.
```

可通过kill命令关闭resourcemgr服务：

```
$ kill -9 17254
[1]+  Killed                  resourcemgr
$
```

13.3.5 TPM2.0-Tools

TPM2.0-Tools是开源社区提供的一套TPM2.0设备的管理工具，提供基本的访问、配置命令，整套工具基于TSS2.0协议栈开发，当前版本共提供33个命令。

注意

在使用这套命令时，需要先启动resourcemgr服务进程，使用完毕以后，关闭resourcemgr进程;可通过resourcemgr >/dev/null&命令开启，通过pkill resourcemgr命令关闭。

所有命令默认安装在/usr/sbin/目录下：

```
$ cd /usr/sbin
$ ls
```



```
tpm2_activatecredential  tpm2_encryptdecrypt  tpm2_getrandom
tpm2_load                tpm2_nvread          tpm2_rc_decode
tpm2_takeownership       tpm2_akparse         tpm2_evictcontrol
tpm2_hash                tpm2_loadexternal    tpm2_nvreadlock
tpm2_readpublic          tpm2_unseal          tpm2_certify
tpm2_getmanufec          tpm2_hmac            tpm2_makecredential
tpm2_nvrelease           tpm2_rsadecrypt      tpm2_verifysignature
tpm2_create              tpm2_getpubak        tpm2_listpcrs
tpm2_nvdefine            tpm2_nvwrite         tpm2_rsaencrypt
tpm2_createprimary       tpm2_getpubek        tpm2_listpersistent
tpm2_nvlist              tpm2_quote           tpm2_sign
$
```

常用命令简要说明如下表：

表 13-4 常用命令简要说明

命令	说明
tpm2_create	创建object
tpm2_createprimary	基于Primary Seed创建Primary Object
tpm2_encryptdecrypt	对称加解密处理命令
tpm2_evictcontrol	用于将object在易失和非易失模式转换
tpm2_getrandom	获取随机数
tpm2_hash	计算hash值
tpm2_hmac	计算hmac值
tpm2_listpcrs	获取PCR的值，数据为明文可读清单
tpm2_listpersistent	获取非易失状态的object清单
tpm2_nvdefine	申请非易失内存空间
tpm2_nvlist	获取非易失内存空间列表
tpm2_nvread	读取非易失内存
tpm2_nvrelease	释放申请的非易失内存空间
tpm2_nvwrite	写非易失内存
tpm2_quote	获取PCR值，数据使用指定的AK算法加密保护
tpm2_rc_decode	获取TPM返回值的含义说明
tpm2_rsadecrypt	RSA解密
tpm2_rsaencrypt	RSA加密
tpm2_sign	签名操作
tpm2_takeownership	设置、获取TPM芯片的属主权限
tpm2_unseal	解封数据
tpm2_verifysignature	校验签名

注意

可以通过tpm2_xxx --help的方式获取所有命令的详细使用方法和举例说明。

13.3.6 本地证明

本地证明是基于TPM2.0的seal/unseal机制的一种证明方式，不依赖于网络上其他的服务器或组件。

管理员可根据需要配置被保护的业务文件列表，然后假定某个时间点服务器是可信的（通常情况下为服务器完成安装部署后，或者是服务器软硬件更新后，运行实际业务前），此时管理员执行指定命令，将服务器当前状态度量值（TPM2.0芯片中指定的寄存器的值）作为基准值进行封存；若干时间段以后，管理员通过命令检查当前状态的度量值同原先封存的是否一致，来完成本地证明：如果一致则表示业务关心的文件未被篡改过，否则表示证明失败，相关文件被篡改。

约束限制

1. 本地证明命令需要root执行权限，如果用来证明启动过程和关键文件完整性的euler-tpm2-la被篡改，则本地证明结果不可信。
2. 本地证明依赖于TPM2.0规范中的封装特性，需要使用到TPM的一些资源，按照功能设计，占用资源如本章节所列。

注意

用户如果在其他产品特性中需要使用TPM2.0的，请避免使用相同的资源，以免引起冲突。

- **Handle：**

  Owner平面：占用Handle资源0x81010019，0x81010020，用于创建本地证明所需要的Key。
- **PCR寄存器：**

  PCR 16：用于文件保护时扩展hash值，有清空、扩展、读取操作。
- **进程资源：**

  本地证明命令依赖于Resourcemgr，在调用命令前必须启动Resourcemgr进程。

13.4 如何使用

本节介绍如何配置、使用IMA、TSS2.0协议栈、TPM2.0-Tools等。

前提条件

- 服务器TPM2.0设备工作正常。
- BIOS中已经启用TPM2.0设备，启动方式为UEFI。
- EulerOS系统已经默认安装到服务器上，且工作正常。

注意

EulerOS的x86平台ISO镜像在安装时请选择安装可信计算，相关组件将被安装，包括TSS2.0协议栈、Resourcemgr、TPM2.0-Tools、本地证明组件。

13.4.1 安装

说明

EulerOS 2.2.RC3.B002，及后续版本支持可信计算。

注意

安装EulerOS之前请确保目标服务器TPM2.0芯片全新，或者TPM2.0芯片的owner，endorsement和lockout口令为空，否则安装EulerOS以后初始化默认口令将失败，并导致TPM2.0芯片进入Lockout锁定状态，需要等待“Lockout恢复间隔时间”之后才能自动解锁，或者在BIOS中进行TPM2.0芯片清除动作，完成TPM初始化和解锁。

如果管理员不希望清除TPM2.0内数据，且TPM2.0芯片的旧口令非空，则可以在安装EulerOS以后继续通过旧口令进行相关业务操作，Lockout状态将在“Lockout恢复间隔时间”之后自动解锁，在此期间除了无法使用Lockout口令进行授权计数器清零操作以外，不影响其他业务操作。

通过 ISO 镜像安装操作系统

适用于新安装服务器，请按照如下步骤操作：

步骤1 准备好EulerOS安装镜像（光盘），准备好待安装的服务器硬件，确保TPM2.0扣卡/芯片已经安装到位；

步骤2 启动服务器，远程登录BMC操作界面（或者通过近端显示器、键盘操作），挂载EulerOS安装镜像（或者通过近端光驱挂载光盘）；

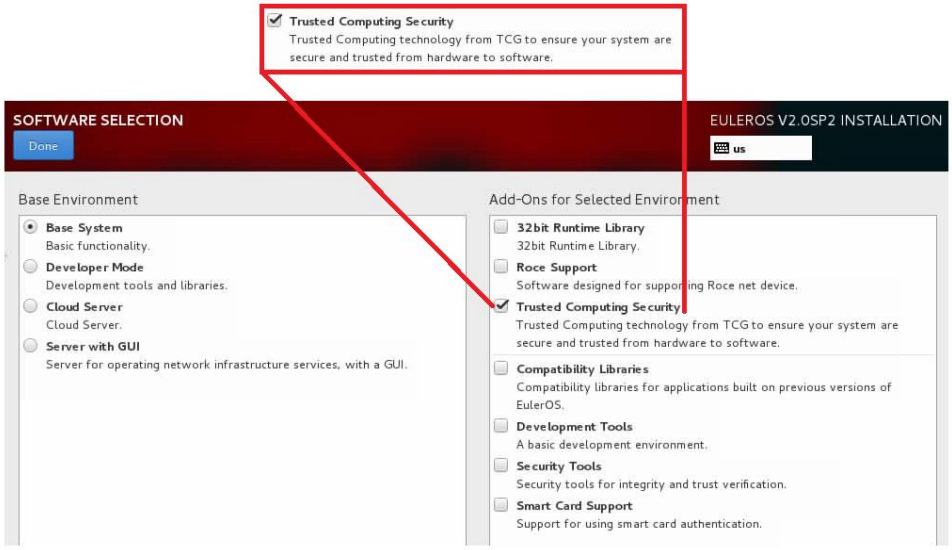
步骤3 进入BIOS操作界面，选择UEFI启动方式；

步骤4 进入BIOS操作界面，将光驱设置为首选启动项；

步骤5 建议在BIOS中选择“TPM清除”选项，否则可能导致TPM2.0芯片口令初始化失败；

步骤6 保存BIOS设置，重启服务器；

步骤7 按提示进入操作系统安装界面，选择安装可信计算组件，其他配置同EulerOS其他版本一致；



步骤8 安装完成后，重启服务器即可使用可信计算提供的相关组件功能。

----结束

通过 yum 安装可信计算组件

适用于已安装EulerOS早期版本的服务器升级可信计算相关组件，请按照如下步骤操作：

步骤1 完成yum源配置，可使用已经支持可信计算的EulerOS镜像作为yum源；

步骤2 在yum源中搜索并安装可信计算相关组件：

- **grub2（如果不关心Linux内核、Initramfs、Modules的可信状态，不度量这些文件，可以不更新）：**

```
# yum search grub2
================= N/S matched: grub2 =================
grub2.x86_64 : Bootloader with support for Linux, Multiboot and more
grub2-efi.x86_64 : GRUB for EFI systems.
grub2-tools.x86_64 : Support tools for GRUB.
...

# yum install grub2-efi
```

- **Linux Kernel（必须更新）：**

```
# yum search kernel
================= N/S matched: kernel =================
kernel.x86_64 : The Linux kernel
...

# yum install kernel
```

- **TSS2.0（如不需要使用TSS协议栈，可以不安装）：**

```
# yum search TPM2.0-TSS
================= N/S matched: TPM2.0-TSS =================
TPM2.0-TSS.x86_64 : TPM (Trusted Platform Module) 2.0 Software Stack (TSS)
...

# yum install TPM2.0-TSS
```

- **TPM2.0-Tools（如不需要使用TPM2.0工具，可以不安装）：**

```
# yum search tpm2.0-tools
================= N/S matched: tpm2.0-tools =================
tpm2.0-tools.x86_64 : TPM (Trusted Platform Module) 2.0 tools based on TPM2.0-TSS
...
```



```
# yum install tpm2.0-tools
```

- **euler-tpm2-tools（如不需要初始化TPM2.0芯片口令和口令配置工具，且不需要IMA默认策略，可以不安装）：**

```
# yum search euler-tpm2-tools
================= N/S matched: euler-tpm2-tools =================
euler-tpm2-tools.x86_64 : Euler TPM (Trusted Platform Module) 2.0 tools based on TPM2.0-TSS
...
# yum install euler-tpm2-tools
```

- **euler-tpm2-la（如不需要本地证明，可以不安装）：**

```
# yum search euler-tpm2-la
================= N/S matched: euler-tpm2-la =================
euler-tpm2-la.x86_64 : Euler Local Attestation component for checking system integrity
...
#yum install euler-tpm2-la
```

说明

1. TSS2.0协议栈和TPM2.0-Tools，会被euler-tpm2-tools和euler-tpm2-la依赖，所以安装后者的时候会自动安装前者。

2. 通过yum的方式安装TPM2.0-TSS，可能提示TSS协议栈的动态库文件找不到（如下图所示），此时需要执行命令ldconfig来刷新系统的库文件环境配置。

```
[root@localhost ~]# resourcemgr > /dev/null &
[1] 935
resourcemgr: error while loading shared libraries: libsapi.so.0: cannot open shared object file: No such file or directory
[1]+  Exit 127                resourcemgr > /dev/null
[root@localhost ~]# ldconfig
[root@localhost ~]# resourcemgr > /dev/null &
[1] 1506
[root@localhost ~]#
```

步骤3 如果安装的是grub2、kernel，请配置相关参数，以便下次重启时使用新的grub2和kernel，配置完毕后重启；如果安装的是euler-tpm2-tools，请重启服务器，并且在BIOS中配置“TPM清除”选项，然后保存配置并退出BIOS设置，完成重启，此时TPM2.0芯片将完成一次口令初始化动作；

步骤4 如果安装的是TSS2.0协议栈、TPM2.0 Tools、以及本地证明组件，则不需要重启服务器，即可使用相关功能。

----结束

13.4.2 TPM 口令修改

EulerOS在安装完成后的第一次启动时，会对TPM2.0芯片的口令进行初始化，之后管理员可以在用户态下对口令进行修改。

注意

TPM2.0芯片默认口令请在第一次系统启动以后立即修改，并定期更新，避免密码泄露后，产生安全风险。

修改口令

root管理员可以通过euler-tpm2-chpwd命令修改TPM2.0芯片的口令，可以一次修改所有三个口令（-a参数），或者分别修改Owner（-o参数）、Endorsement（-e参数）、Lockout（-l参数）。输入命令后管理员可以依据提示输入旧口令和新口令，口令不会在界面上回显，比如：

```
# euler-tpm2-chpwd -a
Changing password for Owner authorization.
Old password:
```

```
New password:
Retype new password:
Changing password for Endorsement authorization.
Old password:
New password:
Retype new password:
Changing password for Lockout authorization.
Old password:
New password:
Retype new password:
```

说明

管理员在设置新口令时必须满足复杂度要求：

1. 口令长度必须大于等于8个字符，且口令长度最大为32个字符。

2. 口令必须包含如下至少三种字符的组合:

一至少一个小写字母。

一至少一个大写字母。

一至少一个数字。

一至少一个特殊字符：`~!@#$%^&*()-_=+\|[{}];:'",<.>/?和空格。

注意

修改Lockout口令时，旧口令必须一次输入正确，如果输入错误，请使用ctrl-c终止命令，然后重新启动命令，并重新输入正确的Lockout旧口令，否则将会导致TPM2.0芯片进入Lockout状态。

13.4.3 启动 IMA 度量

确认 i_version 标记

IMA策略所指定的文件被修改后，IMA度量框架需要通过文件中的i_version标记位来判断文件是否需要重新被度量，而通常情况下在加载（mount）文件系统时并没有启用该标记位，也就是说即使文件被修改了，IMA框架也无法感知，所以启用IMA度量时，请确保加载的文件系统已经启用i_version标记位，对于文件系统，有2种方法启用i_version标记位：

1. 修改/etc/fstab文件，在相应的文件系统加载行第4个参数后面添加“,”和“iversion”，那么系统下次重启时就会自动加载文件系统，并启用i_version标记位，比如：
   ```
   /dev/sdb /mnt/sdb     ext4    defaults,iversion        1 1
   ```
2. 直接通过mount命令加载文件系统，那么在命令行中添加“-o iversion”，比如：
   ```
   mount -t ext4 -o iversion /dev/sdb /mnt/sdb
   ```

说明

EulerOS目前支持ext3、ext4这2种常用的文件系统启用i_version标记位，对于其他类型文件系统，用户可根据相关文件系统标准启闭该标记位。

EulerOS在安装时，如果选择了安装可信计算相关组件（在操作系统安装界面选择软件时，选择了可信计算安装，主要包括两个组件：euler-tpm2-tools，euler-tpm2-la），则已经在/etc/fstab文件中涉及到ext3、ext4文件系统挂载的行添加了iversion字段，所以用户对于系统默认已经加载的这2种文件系统不需要做特别处理；如果需要新加载文件系统，可通过上述方法1或2添加iversion字段。

启动 IMA 度量

EulerOS安装成功以后，默认未启用IMA度量，管理员可以通过以下方式启动IMA度量：

步骤1 以root管理员身份，将/etc/ima/ima-default-policy文件重命名为ima-policy，或者新建一个ima-policy文件

```
# mv /etc/ima/ima-default-policy /etc/ima/ima-policy
```

步骤2 重启服务器

```
# reboot
```

服务器重新启动以后，以及后续服务器每次重新启动，IMA框架都会自动以ima-policy为策略对文件进行度量（ima-policy文件可以被多次修改，服务器重启后生效），管理员可以通过查看度量日志来确定IMA度量是否已经启动：

```
# cd /sys/kernel/security/ima
# cat runtime_measurements_count
400
#
```

如果runtime_measurements_count文件中记录的数据大于1，就表示IMA度量已经启动成功。

管理员亦可通过ascii_runtime_measurements文件查看IMA日志：

```
# cd /sys/kernel/security/ima
# cat ascii_runtime_measurements
10 6ce68515f2bc0d30e8dc9ca8ca9a0e3e3aed3052246b56401aeea09690b1cf0b ima
7ba7d39070b376f74f80ce934ece56af722d19efcc9bdf2513d0eb99993060ac boot_aggregate
10 38ad636c93b72144826138b6bbf336e0f5724ef1587f9cdb24ae47cb5cdc1645 ima
a4f19f4e39d3f24eb099be0935bf6fccd0daa4dce937a335c7f8765ce163bcea /usr/bin/vi
10 623d9882dc81c2f891d2f2d7d99d879984601b574a275612a5a1a019dee083ae ima
e6de2b43a7e46635e5c84a8ceee4c8006951a3164635983a0c57b40429608b79 /etc/virc
...
#
```

每一个被度量的文件都会在ascii_runtime_measurements中形成一条日志，如果一个文件第一次度量以后被修改了，那么第二次访问（可以是读、写或执行操作，具体依赖于策略定义）该文件的时候，会被再次度量，并记录日志；如果文件没有被修改过，则不会再次记录。

IMA在记录日志的同时，会将度量值扩展到PCR10寄存器中，扩展算法如下（其中||表示两个摘要值连接在一起）：

New PCR10 Value = SHA256（Old PCR10 Value || SHA256（被度量文件））

----结束

注意

IMA度量开启以后，IMA日志保存在内核内存中，会随服务器运行时间，被度量文件修改及访问次数的增加而增加，每一条记录消耗大约400字节内存，在正常情况下只有系统启动、业务首次运行时会有大量文件需要被度量，后续新增度量较少，服务器重启以后，该日志文件会被释放，并重新生成。

服务器如果被恶意破坏，并构造某些文件经常被修改和被重新度量，那么可能导致IMA日志容量超出正常范围，比如100万个文件度量记录就会消耗400MB内存空间，服务器系统存在内存被耗尽的可能。

管理员在日常服务器巡检过程中，请关注IMA日志增长情况（可通过查看runtime_measurements_count了解当前日志容量），如果短期内出现大幅增长，或者当前服务器内存容量已接近上限，且由IMA日志导致，则需要排查服务器是否被恶意破坏，在问题排查并解决以后，重启服务器即可恢复IMA日志容量。

13.4.4 基于 TSS2.0 开发

准备

用于应用开发的服务器或PC机需要满足如下条件：

1. 安装有Linux系统，具备基本的gcc、glibc等C语言编程环境；
2. 服务器或PC具备TPM2.0扣卡/芯片，且BIOS支持该扣卡/芯片识别，并已开启；
3. 安装有TSS2.0协议栈；
4. 开发人员具备基本的Linux C程序开发经验；

如何使用 TSS2.0 协议栈

TSS2.0协议栈提供了一套API接口供上层应用调用，从而对TPM芯片进行访问和操作，主要有两类接口：

- TSS TCTI（TPM Command Transmission Interface）

  TCTI封装了两种TPM2.0芯片的访问方式，第一种直接访问/dev/tpm0，通过Device API定义并实现；第二种通过Socket访问Resourcemgr服务进程，进而访问/dev/tpm0设备。TCTI以静态库和动态库方式提供给上层应用集成，默认安装目录为/usr/lib，如下所示：

  ```
  $ cd /usr/lib
  $ ls
  libtcti-device.a  libtcti-device.la
  libtcti-device.so  libtcti-device.so.0  libtcti-device.so.0.0.0
  libtcti-socket.a  libtcti-socket.la
  libtcti-socket.so  libtcti-socket.so.0  libtcti-socket.so.0.0.0
  $
  ```

  上层应用程序在执行的时候必须选择采用Device方式还是采用Socket方式，如果是Device方式，则不需要后台执行Resourcemgr。

  上层应用程序开发所依赖的头文件保存在/user/include/tcti目录下：

  ```
  $ cd /user/include/tcti
  $ ls
  common.h  tcti_device.h  tcti_socket.h
  $
  ```


注意

/dev/tpm0设备只能同时被一个进程打开和访问，如果上层已经启动Resourcemgr服务进程，则无法再采用Device方式访问TPM2.0设备，反之亦然；所以在访问完成以后，建议上层应用尽快关闭Resourcemgr。

- TSS SAPI（TSS system API）

  依据TCG规范，SAPI向上层提供TPM2.0设备访问、管理接口，应用程序通过调用SAPI接口，可以完成Key申请、数据加密、数据封装、PCR读取等操作。SAPI以静态库和动态库方式提供给上层应用集成，默认安装目录为/usr/lib，如下所示：

```
$ cd /usr/lib
$ ls
libsapi.a  libsapi.la  libsapi.so.0  libsapi.so  libsapi.so.0.0.0
$
```

  上层应用程序开发所依赖的头文件保存在/user/include/sapi目录下：

```
$ cd /user/include/sapi
$ ls
implementation.h  sys_api_part3.h  tpm20.h  tpmb.h  tss2_common.h
tss2_sys.h  tss2_tcti.h  tss2_tpm2_types.h
$
```

注意

上层应用程序在创建完TCTI句柄以后，就可以调用SAPI接口对TPM2.0设备进行访问，全部操作完成以后，必须关闭TCTI句柄。

TSS2.0协议栈遵循TCG规范，用户在进行应用开发过程中主要可以参考TCG如下两篇文档：

https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-2-Structures-01.38.pdf

https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-3-Commands-01.38.pdf

13.4.5 本地证明

本地证明的运行逻辑依赖于两个配置文件，管理员可以通过修改配置文件的内容来调整本地证明所保护的范围。

准备

为了使用本地证明，需要满足如下条件：

1. 服务器具备TPM2.0扣卡/芯片，且BIOS支持该扣卡/芯片识别，并已开启；
2. 已经安装支持可信计算的EulerOS操作系统，及所有可信计算组件；
3. 管理员有root权限，且持有TPM2.0的Owner口令；

如何使用本地证明

步骤1 配置相关策略

- 寄存器列表配置

  该配置文件中保存有需要被保护的TPM2.0寄存器的列表，如果在基线建立以后列表中的寄存器的值发生变化，那么在可信证明的时候就会被发现，从而证明某些硬件或软件被篡改了。

  配置文件命名为：la-pcr-list.conf，保存在如下目录：

  ```
  # cd /etc/euler-tpm2.conf.d
  # ls
  la-file-list.conf  la-pcr-list.conf
  ```

  该配置文件包含有默认内容，如下：

  ```
  # cat la-pcr-list.conf
  # PCR list, pcrs which listed below will be protected by local attestation
  # Please add pcr number after -I flag, for example "-I 9", and delete them together
  -I 0 -I 1 -I 2 -I 3 -I 4 -I 5 -I 6 -I 7 -I 8 -I 9
  ```

  管理员可以修改该文件，格式要符合要求，每个需要被保护的寄存器编号跟在“-I”标识后面，删除时也请一起删除，“#”所在行为注释。

  说明

  这个配置文件建议管理员不要修改，除非管理员对TPM2.0的每个寄存器使用场景非常了解，且的确有业务需要。

  TPM2.0总共提供24个寄存器，编号为0~23

  配置文件被修改以后，必须重新做一次基线建立。

- 文件列表配置

  该配置文件中保存有需要被保护的文件的列表，如果在基线建立以后列表中的文件内容发生变化，那么在可信证明的时候就会被发现，从而证明某些软件被篡改了。

  配置文件命名为：la-file-list.conf，保存在如下目录：

  ```
  # cd /etc/euler-tpm2.conf.d
  # ls
  la-file-list.conf  la-pcr-list.conf
  ```

  该配置文件包含有默认内容，管理员可以在配置文件末尾添加需要保护的文件列表，注意必须是全路径的，且只能支持文件，不能支持目录。如下：

  ```
  # cat la-file-list.conf
  # File list, files which listed below will be protected by local attestation
  # One line for one file, must be absolute path, and only support files, folder which is
  listed below will be bypass

  ## These files should be protected by default, PLEASE DO NOT DELETE/MODIFY it.
  /etc/euler-tpm2.conf.d/la-pcr-list.conf
  /etc/euler-tpm2.conf.d/la-file-list.conf
  /etc/euler-tpm2/euler-tpm2-la
  /etc/euler-tpm2/euler-tpm2-pcr16
  /etc/euler-tpm2/euler-tpm2-chhierarchypwd
  /usr/sbin/euler-tpm2-laset
  /usr/sbin/euler-tpm2-lacheck
  /usr/sbin/euler-tpm2-chpwd

  ## Please add new files here, which should be protected by local attestation
  ```

注意

配置文件中默认包含有一些可信计算、本地证明相关的系统文件，请不要从列表中删除，否则可能引起可信计算或本地证明本身的安全风险。

配置文件被修改以后，必须重新做一次基线建立。

步骤2 启动resourcemgr服务进程

本地证明相关操作命令依赖于resourcemgr服务进程，可通过如下命令启动：

```
# resourcemgr >/dev/null&
```

在使用完成后，通过如下命令关闭：

```
# pkill resourcemgr
```

步骤3 建立基线

本地证明是通过将服务器的当前状态同基线状态进行对比，从而确定服务器上的软硬件是否被篡改。

而基线值是通过euler-tpm2-laset命令来创建的，管理员在服务器OS、业务软件等安装完毕以后，并完成配置文件修改，确认服务器当前未被篡改，处于可信状态，然后调用该命令建立本地证明的基线。

建立基线的命令和相关帮助信息如下：

```
# euler-tpm2-laset -h
Usage: euler-tpm2-laset [OPTION]
    -s, --set    set local attestation with all choose pcrs and files
    -c, --clear  clear local attestation which has been set
    -h, --help   show help of local attestation commands
```

该命令有两个参数：

- **-s，--set**

  set参数用于建立本地证明的可信基线，如果之前尚未建立过可信基线，比如首次安装完系统，则可以调用本命令和参数完成基线建立，在调用命令以后，根据提示，输入对应口令，完成操作：

  ```
  # euler-tpm2-laset -s
  Input TPM Owner Authorization password:
  Set new primary key password:
  Re-enter primary key password again:
  Set new seal key password:
  Re-enter seal key password again:
  euler tpm2 la setting......
  # euler-tpm2-laset --set
  PInput TPM Owner Authorization password:
  Set new primary key password:
  Re-enter primary key password again:
  Set new seal key password:
  Re-enter seal key password again:
  euler tpm2 la setting.....
  ```

  一共涉及到三个口令输入：

  Owner口令：TPM2.0初始化后的Owner默认口令或管理员修改后的口令。

  Primary Key口令：TPM2.0创建Owner层主秘钥时使用的授权口令，此处为新设置口令，一旦Primary Key创建成功，下次访问该主秘钥时就需要提供这个口令进行认证授权，该口令无法修改。

  Seal Key口令：本地证明在封存PCR度量数据时使用的授权口令，此处为新设置口令，一旦本地证明基线创建成功，下次做检查时就需要提供这个口令进行认证授权，该口令无法修改。

说明

管理员可以不用关心这几个口令的实现原理，但需要牢记这些口令，并确保口令不被泄露。

管理员在设置新口令时必须满足复杂度要求：

1. 口令长度必须大于等于8个字符，且口令长度最大为32个字符。

2. 口令必须包含如下至少三种字符的组合:

一至少一个小写字母。

一至少一个大写字母。

一至少一个数字。

一至少一个特殊字符：`~!@#$%^&*()-_=+\|[{}];:'",<.>/?和空格。

set操作需要在TPM2.0芯片中申请相关资源，且对指定的PCR寄存器和用户文件进行完整性计算，所以操作比较耗时，具体依赖于需要被保护的文件的数量，操作过程中请耐心等待。

- **-c，--clear**

  clear参数用于清除原先建立的可信基线，包括创建的Primary Key和Seal Key的口令，以及临时文件，clear操作只需要对Owner口令进行认证授权即可：

```
# euler-tpm2-laset -c
Input TPM Owner Authorization password:
euler tpm2 la clearing......
# euler-tpm2-laset --clear
Input TPM Owner Authorization password:
euler tpm2 la clearing......
```

注意

请root管理员保管好Owner口令

步骤4 可信证明

在基线成功建立以后的任何时候，都可以通过euler-tpm2-lacheck命令来检查当前的可信状态，如果受保护的文件有篡改，会给出相应提示。

euler-tpm2-lacheck命令没有参数，调用该命令后根据提示输入口令即可：

```
# euler-tpm2-lacheck
Please input primary key password:
Please input seal key password:
euler tpm2 la checking......
```

做可信证明操作时，由于只是检查可信状态，不需要对Owner进行认证授权，只需要输入Primary Key和Seal Key的口令即可。

注意

口令输入请确保正确，多次失败（英飞凌SLB9665芯片为32次）将会导致TPM进入授权锁定状态。

如果证明成功，则返回如下结果：

```
# euler-tpm2-lacheck
Please input primary key password:
Please input seal key password:
euler tpm2 la checking......
```



```
***************
    Trusted
***************
```

如果证明失败，则返回具体失败原因：

```
# euler-tpm2-lacheck
Please input primary key password:
Please input seal key password:
euler tpm2 la checking......

*****************
    unTrusted
*****************
/usr/sbin/euler-tpm2-chpwd is changed

****************
    Finished
****************
```

说明

本操作需要对指定的PCR寄存器和用户文件进行完整性计算和检查，所以操作比较耗时，具体依赖于需要被保护的文件的数量，且如果文件被篡改过，则需要定位具体文件并给出提示，所以unTrusted情况下会比Trusted情况下更耗时，操作过程中请耐心等待。

----结束

常见错误码列表

表 13-5 常见错误码列表

TCG 错误码	LA 错误码	错误描述	推荐措施
0x921	0x21	TPM授权对象受到DA保护，此时TPM芯片处于Lockout模式，不允许被访问（Lockout计数器超过最大值）	系统管理员需重启机器至BIOS的TPM设备选项中清空（注意此操作会删除TPM芯片所有信息，谨慎操作）；或者等待到TPM芯片Lockout计数器减一（英飞凌芯片计数器周期为2hour，即每2hour计数器减一，计数器最大值为32）
0x98e	0x8e	TPM授权失败（密码错误），该错误会导致计数器加一	系统管理需要确认输入的口令是否正确

TCG 错误码	LA 错误码	错误描述	推荐措施
0x128 0x184 0x12f	0x28 0x84 0x2f	PCR寄存器被更改导致的错误，TPM芯片不支持并发操作，每个PCR寄存器都有一个计数器，PCR授权相关的操作会判断该计数器前后是否一致，不一致则会报错，该错误可能会导致TPM出现NV访问的问题，从而出现authValue或者authPolicy访问失败，或者访问TSS协议栈的context超出范围	系统管理员需明确该问题是由于TPM芯片内部硬件的安全保护机制导致的，故在使用本地证明时推荐管理员控制当前系统没有并发操作TPM。 若可信计算组件报相关的错误码，管理员可以等待TPM并发操作完成之后重新执行命令。

14 热补丁

内核热补丁是一种在不重启操作系统或者插拔内核模块的前提下，修复内核和内核模块中缺陷的一种工具，可以在不中断业务的情况下解决问题。本章介绍内核热补丁的原理、主要功能、使用场景、使用方法和约束限制。

14.1 概述

本节介绍内核热补丁的一些基础概念，让您了解内核热补丁的作用。

背景描述

内核热补丁是用发布的补丁文件修改内核或者内核模块中的缺陷，它可以不影响业务的情况下在线解决大部分内核或者内核模块的问题，增强公司产品竞争力，从而有效提升公司形象及客户满意度。

补丁管理服务具备以下优点：

- 缩短版本发布的时间，提高市场的响应速度。
- 不影响现网的业务，提高客户的满意度。

- 从版本验证改为补丁验证，缩短测试时间。

内核热补丁简介

内核热补丁对编译生成的二进制文件进行操作生成补丁文件，提取缺陷函数的二进制代码，以内核模块形式插入到系统中，检测系统中所有进程是否在调用所修改的函数，通过修改函数的代码段实现函数跳转，进而实现修改内核和模块中函数缺陷。

通过在缺陷函数中插入的钩子，让缺陷函数在被调用时跳转到新函数的地址中，使热补丁生效。

基本概念

- 内核热补丁：linux内核的热补丁，主要是Euler部门用，产品部门可以不关注。
- 模块热补丁：产品模块驱动的热补丁。
- 补丁： 一般一个修改点，做成一个补丁。

工具说明

make_hotpatch：热补丁制作工具。

livepatch：加载补丁、激活补丁、补丁查询、删除补丁、回退补丁工具。

14.2 约束限制

用户在使用内核热补丁功能时，请注意以下约束限制：

- **不支持**
  - 不支持对初始化函数打补丁（初始化函数只执行一次，补丁函数执行不到）。
  - 不支持汇编文件打补丁。
  - 不支持对死循环、不退出函数打补丁（旧函数不退出调用栈，没有机会调用新函数）。
  - 不支持对有前缀notrace修饰的函数打补丁（产品一般不涉及）。
  - 不支持修改数据结构成员（热补丁原理是做函数替换）。
  - 不允许删除函数内部静态局部变量。
  - 不支持新增同名静态局部变量。
  - 不支持对头文件进行修改。
  - 不支持对非C语言编写的代码程序打热补丁。
  - 不允许对NMI中断的处理函数打补丁（stop machine无法stop住NMI中断处理流程，补丁无法保证对该类函数打补丁的一致性和安全性）。
  - 不支持修改全局变量初始值。
  - 不支持修改静态局部变量初始值。
  - 不支持删除函数。
  - 不支持对修改前后内敛情况发生变化的函数打补丁。
  - 不支持对包含以下弱符号的函数打补丁。

    "kallsyms_addresses"

    "kallsyms_num_syms"

"kallsyms_names"

"kallsyms_markers"

"kallsyms_token_table"

"kallsyms_token_index"

- **支持**
  - 支持新增全局数据结构。
  - 支持对内联函数打补丁。
  - 支持对静态函数打补丁。
  - 支持修改多个文件的多个函数。
  - 支持新增全局变量。

  制作热补丁时，用户必须保证编译环境包和基线代码和运行环境中一致，ARM64版本的热补丁工具由于与X86机制不同，不会进行汇编层面的二进制比对。

- **Makefile的约束限制**

  xxx-y目前不支持./xxx.o

```
#ifeq ($(stub),1)^M
ifeq ($(suse11sp1),1)^M
EXTRA_CFLAGS += -DDRV_STUB^M
endif^M
^M
drvmml-y := ./drv_mml.o^M
^M
drvmml-y += ./drv_mmlcommon.o ^M
^M
drvmml-y += ./drv_mmlapi.o ^M
#drvmml-y += ./drv_mmlautotest.o^M
#drvmml-y += ./drv_mmlft.o^M
#ifeq ($(fc),1)^M
drvmml-y += ./drv_mmlfc.o^M
#endif^M
^M
^M
```

  Makefile中不可显示定义CROSS_COMPILE和CC这两个变量，如果有需要在制作热补丁时注释掉。

```
export ARCH=x86_64
PREFIX=
KERNELDIR=/lib/modules/3.10.0-327.22.2.26.x86_64/build
KERNELSOURCEDIR=/lib/modules/3.10.0-327.22.2.26.x86_64/source
KERNELVERSION=3.10.0-327.22.2.26.x86_64
#export CROSS_COMPILE=
#export CC=gcc
export LD=ld
export OBJCOPY=objcopy
export AR=ar
KVM_VERSION=kvm-devel
```

  编译过程中，不可删除中间生成的二进制文件，如果有也请注释掉。


```
default:
        @rm -f sal_version.o
        $(MAKE) -C $(KDIR) M=$(PWD)
        #cat Module.symvers >> $(WORK_DIR)/source/modules/Module.symvers
clean:
        $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
        @rm -f ?odule*.*
```

部分模块在编译时会动态生成头文件，请在制作补丁时保持头文件不变。

```
default:
        @echo \#define DMI_DATE \"`date +%Y/%m/%d" "%H:%M:%S`\" > $(WORK_DIR)/dmi/include/dmi_compile.h
```

部分模块一次编译会生成多个ko，制作热补丁，需要修改Makefile，保证每次只编译出一个ko：

```
-obj-m := kvm.o kvm-intel.o kvm-amd.o
+obj-m := kvm.o

-obj-m := kvm.o kvm-intel.o kvm-amd.o
+obj-m := kvm-intel.o
```

- **c++模块补丁限制**
  - 内核热补丁机制需要支持C++内核模块制作热补丁，包括X86和ARM。当前只承诺支持auto_tiering模块。

    热补丁机制对auto_tiering模块的支持能力保持和C内核模块一致。例如补丁制作限制等。

    说明

    热补丁机制支持C++内核模块制作热补丁当前只支持auto_tiering模块。后续如果产品继续新增了其他C++内核模块，需要重新提需求做更全面的适配以及覆盖测试。
  - C++内核模块热补丁不支持导出函数。

14.3 使用场景

内核热补丁最大的特点是在不重启系统和不中断业务前提下修改内核中的函数，达到动态替换内核函数的目的，最主要的应用场景有两个：

1. 修复内核和模块的缺陷函数。

   内核热补丁能够动态的修复内核和模块的缺陷函数。在开发人员发现问题，或者操作系统发现安全漏洞需要修复时，可以将缺陷函数或者安全补丁制作成内核热补丁打入系统中，通过这种方法，在不需要重启系统或者插拔模块、不中断业务的前提下修复缺陷。
2. 开发过程中的调试和测试手段。

   内核热补丁也适用于在开发过程中进行调试和测试。比如在模块或者内核的开发过程中，如果需要通过在某一个函数中添加打印信息，或者为函数中某一个变量赋予特定的值，可以通过内核热补丁的形式实现，而不需要重新编译内核、安装然后重启的操作。

14.4 准备软硬件环境

本章介绍内核热补丁使用的软硬件环境。

硬件要求

- 编译环境：即热补丁制作环境，能够正常运行64位Linux操作系统，制作补丁需要编译内核和模块，如果有多CPU支持，速度更快。
- 运行环境：即要安装热补丁的环境，必须是X86或ARM64架构。

软件要求

- 编译环境：需要安装Linux环境，并搭建下载与运行环境相匹配的编译环境压缩包Euler_compile_env.tar.gz或Euler_compile_env_cross.tar.gz。
- 在编译环境中做补丁的基线源代码、配置文件、Makefile编译出的二进制要和运行环境中的二进制完全一致。
- 运行环境的EulerOS系统中已经带有livepatch命令。

14.5 管理补丁（运行环境）

本章介绍如何对补丁进行加载、激活、查询、回退等操作。

约束限制

livepatch不支持并发操作。如运行以下命令，只会加载一个补丁并提示"Cannot execute livepatch concurrently"。

```
[root@EulerOS ~]# livepatch -l klp_testmod001.tar.gz & livepatch -l klp_testpfm.tar.gz & livepatch -l klp_testmulti.tar.gz
[1] 17780
[2] 17781
Cannot execute livepatch concurrently
Cannot execute livepatch concurrently
insmod /tmp/hotpatch.17791/klp_testmulti/klp_testmulti.ko
install patch klp_testmulti.tar.gz success
[1]-  Exit 1                  livepatch -l klp_testmod001.tar.gz
[2]+  Exit 1                  livepatch -l klp_testpfm.tar.gz
```

14.5.1 加载补丁

本章介绍热补丁的加载过程。

使用场景

补丁已从编译环境上传到运行环境，在运行环境使用livapatch工具将补丁文件加载到内核中。

注意事项

- 若加载的补丁P2依赖另一个补丁P1，在加载补丁P2之前需先激活补丁P1。
- 若对模块打补丁，在加载此补丁前需先加载此模块。

操作过程

步骤1 假设已将补丁文件（如klp_test.tar.gz）上传到运行环境上，存放在/root/目录下。

步骤2 可在任意目录下执行livepatch命令来加载补丁。可使用相对路径。

```
livepatch -l /root/klp_test.tar.gz
```


显示如下：

```
install patch /root/klp_test.tar.gz success
```

----结束

验证结果

加载补丁成功后，返回补丁加载成功信息。通过“查询补丁信息”查询补丁当前的状态：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

14.5.2 激活补丁

本章介绍热补丁的激活。

使用场景

对已经加载完成的补丁进行操作，设置补丁的状态，使之为激活。

注意事项

激活补丁命令中使用的补丁名必须是通过“查询补丁信息”查询出的补丁名。

操作过程

可在任意目录下执行livepatch命令激活补丁。

步骤1 查询所有补丁文件信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

步骤2 激活补丁：

```
[root@EulerOS ~]# livepatch -a test
active patch klp_test success
```

----结束

验证结果

激活补丁成功后，返回补丁激活成功信息，通过查询命令可以查询到补丁状态信息为Active。

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Active
Changes:
```

```
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

14.5.3 查询补丁

本章介绍如何查询补丁状态。

使用场景

对加载到系统中的补丁进行状态查询。

注意事项

- 如果是查询单个补丁，需要将补丁的id直接跟着-q参数后面。
- 如果是对内联函数打补丁，在补丁查询的时候只会显示调用该内联函数的缺陷函数名称，而不是内联函数本身。

操作过程

步骤1 查询所有补丁文件信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Active
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

步骤2 查询单个补丁文件信息：

```
[root@EulerOS ~]# livepatch -q test
Patch Name: test
Patch State: Active
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
```

步骤3 回退补丁：

```
[root@EulerOS ~]# livepatch -d test
deactive patch klp_test success
[root@EulerOS ~]#
```

----结束

验证结果

回退补丁成功后，返回补丁回退成功信息，通过查询命令可以查询到补丁状态信息发生改变。

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```


调用被打补丁的函数可以进一步验证内核热补丁是否成功去激活，即回退补丁后被打补丁函数是否还原。具体步骤略。

14.5.4 回退补丁

本章介绍补丁的回退。

使用场景

补丁处于激活状态，要使其失效。

注意事项

补丁已加载并处于激活状态。

补丁如果被其他补丁依赖则无法回退，必须先回退依赖补丁，再回退该补丁。

操作过程

可在任意目录下执行livepatch命令回退补丁。

步骤1 查询所有补丁文件信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
        cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

步骤2 回退补丁：

```
[root@EulerOS ~]# livepatch -d test
deactive patch klp_test success
```

----结束

验证结果

回退补丁成功后，返回补丁回退成功信息，通过查询命令可以查询到补丁状态信息发生改变。

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
?cmdline_proc_show
Denpendency: vmlinux
-------------------------------------------------------------
[root@EulerOS ~]#
```

14.5.5 卸载补丁

本章介绍卸载热补丁。

使用场景

在运行环境使用livepatch工具将补丁文件从内核补丁区移除。

注意事项

- 卸载补丁命令中的补丁名必须是通过“查询补丁信息”查询出的补丁名。
- 若有其他已加载的或处于激活状态的补丁依赖于补丁P1，不允许卸载补丁P1。

操作过程

步骤1 查询所有补丁信息：

```
[root@EulerOS ~]# livepatch -q
Patch Name: test
Patch State: Deactive
Changes:
?cmdline_proc_show
Denpendency: vmlinux
------------------------------------------------------------
[root@EulerOS ~]#
```

步骤2 回退补丁：

```
[root@EulerOS ~]# livepatch -d test
deactive patch klp_test success
```

步骤3 卸载补丁：

```
[root@EulerOS ~]# livepatch -r test
remove patch klp_test success
```

----结束

验证结果

卸载补丁成功后，返回补丁卸载成功信息，通过查询命令查询不到补丁的信息（补丁已不在内核补丁区）。

14.6 热补丁制作（编译环境）

14.6.1 制作模块热补丁

内核模块指用户开发的驱动。由于热补丁使用是有一定限制和约束的，所以制作前，请先阅读 **14.2 约束限制** 一节。

前提条件

- EulerOS编译环境中已经安装补丁制作工具和补丁制作目录。
- 修改前的模块需要能够编译通过。
- 修改后的模块需要能够编译通过。

注意事项

- 补丁制作的源码、源码在编译环境中的路径、编译环境等需要和运行环境运行的内核、模块完全一致。
- 如果之前已经做过补丁，现在需要对同一个文件的相同或其他函数再做补丁，需要再将之前的改动代码和本次将要修改的代码都包含在唯一后缀的文件中。

制作步骤

步骤1 部署编译环境（这一步通常由CI工程师搭建一次即可）。

EulerOS提供了对应的编译环境包（X86架构Euler_compile_env.tar.gz，ARM64架构交叉编译环境Euler_compile_env_cross.tar.gz）。用户可从CMC或CI工程师处获取对应的编译环境包等工具，本章节以X86编译环境包为例。

```
tar xf Euler_compile_env.tar.gz
```

步骤2 拷贝或挂载产品代码到编译环境目录下，例如Euler_compile_env/code/。

步骤3 chroot到编译环境中进行工作，开始制作热补丁（以下步骤以testmod模块制作热补丁作为示例，其他模块参照）。

```
chroot Euler_compile_env/ /bin/bash --login
```

步骤4 进入源码目录，拷贝源文件为唯一后缀的文件（这里的后缀即为**步骤5**中的-d参数的值），并修改文件中需要打补丁的函数。

```
cd /code/testmod/
cp testmod_drv.c testmod_drv.c.new
vi testmod_drv.c.new
```

步骤5 进入补丁制作目录，制作热补丁。

```
cd /opt/patch_workspace/
./make_hotpatch -d .new -i test -j 64 -m /code/testmod
```

- -d 后面跟上前面的唯一后缀名。
- -i 后跟补丁ID，可包括字母和数字。
- -j 后跟make编译时的线程数。
- -m 后跟模块源码绝对路径。

步骤6 补丁制作成功后在编译环境/opt/patch_workspace/hotpatch/目录下。

步骤7 将补丁klp_test.tar.gz上传到运行环境上，用livepatch命令加载并激活热补丁。

```
livepatch -l klp_test.tar.gz
livepatch -a test
```

以上两步执行成功后即可验证业务逻辑是否正常，bug是否成功修复。

----结束

特殊处理

- 热补丁编译模块时仅执行make操作，当模块带参数编译时，例如make DEBUG=1。用户需要将DEBUG=1写入一个flag文本，并在制作热补丁时增加--extra_flags参数指定flag文本路径。示例如下：
  ```
  ./make_hotpatch -d .new -i test -j 64 -m /code/testmod --extra_flags /opt/patch_workspace/flags
  ```
- 当模块的Makefile不在源码目录下时，比如Makefile在testmod/build目录，模块源码在testmod/src目录，制作补丁时需要增加-f参数指定Makefile路径。示例如下：
  ```
  ./make_hotpatch -d .new -i test -j 64 -m /code/testmod/src -f /code/testmod/build/Makefile
  ```

14.6.2 制作内核热补丁

由于热补丁使用是有一定限制和约束的，所以制作前，请先阅读“**约束限制**”一节。

前提条件

- EulerOS编译环境中已经安装补丁制作工具和补丁制作目录。
- 修改前的模块需要能够编译通过。
- 修改后的模块需要能够编译通过。

注意事项

- 补丁制作的源码、源码在编译环境中的路径、编译环境等需要和运行环境运行的内核、模块完全一致。
- 如果之前已经做过补丁，现在需要对同一个文件的相同或其他函数再做补丁，需要再将之前的改动代码和本次将要修改的代码都包含在唯一后缀的文件中。

制作步骤

步骤1 部署编译环境（这一步通常由CI工程师搭建一次即可）。

EulerOS提供了对应的编译环境包（X86架构Euler_compile_env.tar.gz，ARM64架构交叉编译环境Euler_compile_env_cross.tar.gz）。用户可从CMC或CI工程师处获取对应的编译环境包等工具，本章节以X86编译环境包为例。

```
tar xf Euler_compile_env.tar.gz
```

步骤2 chroot到编译环境中进行工作，开始制作热补丁（以下步骤以修改内核源码fs/proc/cmdline.c为示例）。

```
chroot Euler_compile_env/ /bin/bash --login
cd /opt/patch_workspace/
./make_hotpatch
```

步骤3 上面这一步执行完后，会在/opt/patch_workspace/目录下生成一个kernel-source的软链接，指向内核源码目录。进入内核源码目录，拷贝源文件为唯一后缀的文件（这里的后缀即为步骤4中的-d参数的值），并修改文件中需要打补丁的函数。

```
cd /opt/patch_workspace/kenrel-source
cd fs/proc/
cp cmdline.c cmdline.c.new
vim cmdline.c.new
```

步骤4 进入补丁制作目录，制作热补丁。

```
cd /opt/patch_workspace/
./make_hotpatch -d .new -i test -j 64
```

- -d 后面跟上前面的唯一后缀名。
- -i 后跟补丁ID，可包括字母和数字。
- -j 后跟make编译时的线程数。

步骤5 补丁制作成功后在编译环境/opt/patch_workspace/hotpatch/目录下。

步骤6 将补丁klp_test.tar.gz上传到运行环境上，用livepatch命令加载并激活热补丁。

```
livepatch -l klp_test.tar.gz
livepatch -a test
```

以上两步执行成功后即可验证业务逻辑是否正常，bug是否成功修复。

----结束

14.7 命令参考

make_hotpatch

- **功能说明**

  通过此命令创建一个热补丁，新创建的补丁将会存放在补丁工作目录的hopatch下。

  说明

  该命令不支持并发，即不能同时执行多个该命令。

- **命令原型**

```
make_hotpatch -d patch_diffext -i patch_id  -j CPU_number -m module_src -f makefile_path --extra_flags flags_path [--debug_info] [-h]
```

- **参数说明**

表 14-1 参数说明

参数名称	描述	可选or必选
-d,--diffext	用来指定修改文件的后缀名，比如“.patch”。	必选
-j,--jobs	指定补丁制作过程中使用的CPU个数，CPU越多速度越快。	可选
-i,--id	指定补丁的id，只能用数字或者字符表示id，并不能超过20个字符。	必选
-m,--modulesrc	用来指定用户模块源码所在的路径，请使用绝对路径，且保证该路径与之前编译模块（编译对应系统运行中的模块）的路径相同。如果是内核或者内核模块补丁，可以省略这个参数。	● 当制作用户模块补丁时，该参数必选。 ● 当制作内核补丁时，该参数不需要。
-f,--makefile	用来指定模块编译时Makefile的路径和文件名，使用绝对路径，且这个Makefile中需要指定obj-m编译的目标 **说明** 该参数只适用于制作模块补丁，并且只有在模块的源码和Makefile文件不在同一目录时必配，通过该参数指定Makefile的路径。	● 当制作用户模块补丁，且模块的源码和Makefile文件不在同一目录时，该参数必选。 ● 当制作内核补丁时，该参数不需要。

参数名称	描述	可选or必选
--extra_flags	用来指定模块编译Makefile中的全局变量，配置文件flags定义了用户模块Makefile引用的全局变量，此参数为可选参数。 **说明** 如果用户模块的Makefile引用了非本Makefile定义的全局变量，在制作用户模块补丁时，可能导致制作补丁失败。使用“--extra_flags”可以将Makefile中非本Makefile定义的全局变量，定义到指定配置文件中，避免上述情况的发生。	● 当制作用户模块补丁时，该参数可选。 ● 当制作内核补丁时，该参数不需要。
--debug_info	制作补丁时打印调试日志以及保存中间临时文件，方便补丁问题分析定位。	可选。正式发布的补丁建议不要添加此参数。
--kallsyms	在模块运行的缺陷环境中执行命令：cat /proc/kallsyms|grep mod，获取到符号表，导出到文件作为本参数的输入。用来指定缺陷环境中的模块符号信息，以保障模块中某些超长符号制作补丁时能成功加载。	● 当制作补丁不存在问题时，可选。 ● 当制作补丁因无法解析的符号而失败时，必选。
--no_stack_check	用来指定该热补丁在激活和回退时不做栈检测	可选。这种方法会导致该热补丁中的函数在补丁激活和回退时不会进行调用栈检查，存在前后一致性风险，即补丁生效后旧函数和新函数可能存在同时运行的场景，请用户充分评估清楚这种场景对模块逻辑的风险，慎重使用，风险由使用者自行承担。
-h,--help	获取帮助信息。	可选。

livepatch

- **功能说明**

  通过此命令管理一个补丁。

- **命令原型**

```
livepatch -l/--load -r/--remove -a/--activate -d/--deactivate <patch>
                -q[patch]/--query[=patch]
                -h/--help -v/--version
```

- **参数说明**

表 14-2 参数说明

参数名称	描述
-l/--load	加载补丁。
-r/--remove	卸载一个补丁。
-a/--activate	激活一个补丁。
-d/--deactivate	回退一个补丁。
-q[patch]/ --query[=patch]	查询所有补丁或者查询指定补丁状态。
-h/--help	帮助信息。
-v/--version	查询livepatch的版本号。

说明

具体的使用实例可以参考**管理补丁**模块。

14.8 常见错误处理

本章介绍在进行内核热补丁操作时遇到的一些常见问题及处理方法。

14.8.1 补丁制作失败，no changed objects found

问题描述

补丁制作完成，没有报错，但没有生成补丁文件，打印出信息no changed objects found.。

问题原因

模块Makefile中显示定义了CC或CROSS_COMPILE。

处理方法

请检查你的Makefile中（包括其include的其他Makefile）是不是显示指定了CC或CROSS_COMPILE这两个变量，如果有，请将其注释后再重新制作热补丁。

图 14-1 某产品公共 Makefile：plat_pub.mak

14.8.2 补丁制作失败，can't find parent xxx for xxx

问题描述

模块热补丁制作失败。

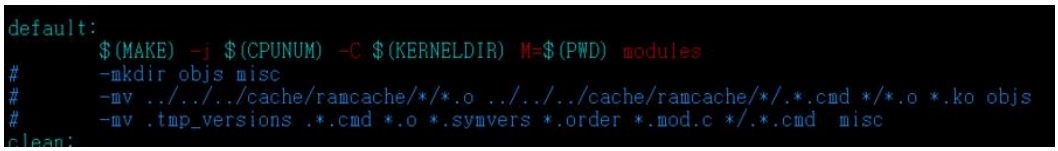
问题原因

模块Makefile问题，编译末尾存在mv或rm操作。

处理方法

请检查你的Makefile，在编译完成后是否有mv或rm操作将生成的二进制移走或删除，如果有，请将其注释后再重新制作热补丁。

图 14-2 某产品模块 Makefile



14.8.3 补丁制作失败，reference to static local variable xxx in xxx was removed

问题描述

补丁制作失败，提示reference to static local variable xxx in xxx was removed。

问题原因

热补丁不支持删除静态局部变量。

处理方法

热补丁不支持删除静态局部变量，请检查你的源码改动，找到被删除的静态局部变量，通过在修改处恢复该静态局部变量进行规避，然后再重新制作热补丁。

示例：找到根据错误提示的函数，这里是handleWriteChunkWriteSuccessEvent，开发在该函数中删了一个内联函数的引用，引用了其他函数，原来的内联函数中存在日志限频打印宏（可能层层内联，需要一层一层往下找）PRINT_LIMINT_PERIOD，这个宏中存在静态局部变量，即报错中提示的ulLast。

图 14-3 示例

```
#define PRINT_LIMIT_PERIOD( level, logid, interval, burst, can) \
do {                                                            \
    /*使用静态变量保存最大配额，减少运算*/                      \
    static OSP_U64 ulMaxToks = (burst) * (interval);            \
    static OSP_U64 ulToks = (burst) * (interval);               \
    static OSP_U32 uiMissed = 0;                                \
    static OSP_U64 ulLast = 0;                                  \
    OSP_U64 ulNow = jiffies;                                    \
                                                                \
```

规避方法

在改动处，增加类似以下语句，注意静态局部变量的初始值要和删除之前的保持一致，如果存在多个静态局部变量，也作相同处理。

```
do{
static u64 ulLast = 0;
if(!jiffies)
printk("%lx\n",ulLast++);
}while(0);
```

14.8.4 补丁制作失败，invalid ancestor xxx for xxx

问题描述

补丁制作失败，提示类似invalid ancestor xxx for xxx。

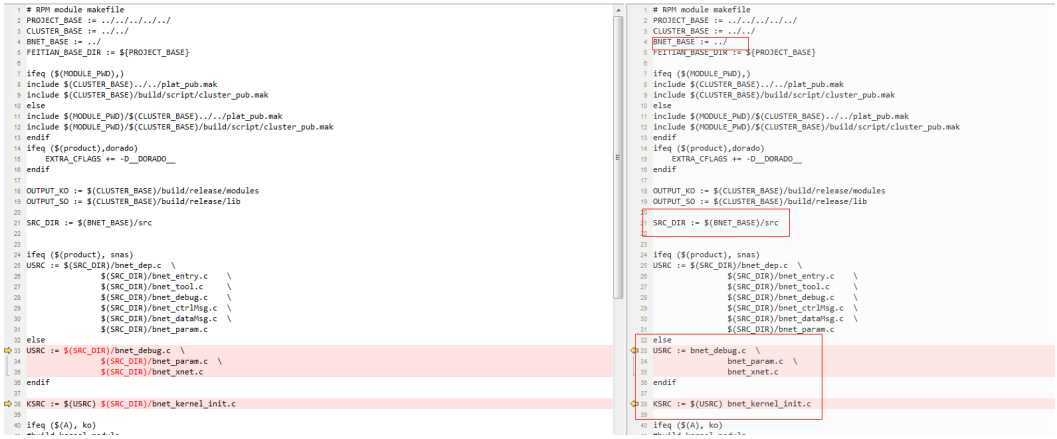
问题原因

模块Makefile问题，使用了相对路径。

处理方法

请修改你的Makefile，在编译中尽量使用绝对路径，避免使用相对路径。示例如图 1：

图 14-4 某产品模块 Makefile



```
# RPM module makefile
PROJECT_BASE := ../../../../../
CLUSTER_BASE := ../../
BNET_BASE := ../
FEITIAN_BASE_DIR := ${PROJECT_BASE}

ifeq ($(MODULE_PWD),)
include $(CLUSTER_BASE)../../plat_pub.mak
include $(CLUSTER_BASE)/build/script/cluster_pub.mak
else
include $(MODULE_PWD)/$(CLUSTER_BASE)../../plat_pub.mak
include $(MODULE_PWD)/$(CLUSTER_BASE)/build/script/cluster_pub.mak
endif
ifeq ($(product),dorado)
    EXTRA_CFLAGS += -D__DORADO__
endif

OUTPUT_KO := $(CLUSTER_BASE)/build/release/modules
OUTPUT_SO := $(CLUSTER_BASE)/build/release/lib

SRC_DIR := $(BNET_BASE)/src


ifeq ($(product), snas)
USRC := $(SRC_DIR)/bnet_dep.c  \
                $(SRC_DIR)/bnet_entry.c    \
                $(SRC_DIR)/bnet_tool.c     \
                $(SRC_DIR)/bnet_debug.c    \
                $(SRC_DIR)/bnet_ctrlMsg.c  \
                $(SRC_DIR)/bnet_dataMsg.c  \
                $(SRC_DIR)/bnet_param.c
else
USRC := $(SRC_DIR)/bnet_debug.c  \
                $(SRC_DIR)/bnet_param.c  \
                $(SRC_DIR)/bnet_xnet.c
endif

KSRC := $(USRC) $(SRC_DIR)/bnet_kernel_init.c

ifeq ($(A), ko)
```

```
# RPM module makefile
PROJECT_BASE := ../../../../../
CLUSTER_BASE := ../../
BNET_BASE := ../
FEITIAN_BASE_DIR := ${PROJECT_BASE}

ifeq ($(MODULE_PWD),)
include $(CLUSTER_BASE)../../plat_pub.mak
include $(CLUSTER_BASE)/build/script/cluster_pub.mak
else
include $(MODULE_PWD)/$(CLUSTER_BASE)../../plat_pub.mak
include $(MODULE_PWD)/$(CLUSTER_BASE)/build/script/cluster_pub.mak
endif
ifeq ($(product),dorado)
    EXTRA_CFLAGS += -D__DORADO__
endif

OUTPUT_KO := $(CLUSTER_BASE)/build/release/modules
OUTPUT_SO := $(CLUSTER_BASE)/build/release/lib

SRC_DIR := $(BNET_BASE)/src


ifeq ($(product), snas)
USRC := $(SRC_DIR)/bnet_dep.c  \
                $(SRC_DIR)/bnet_entry.c    \
                $(SRC_DIR)/bnet_tool.c     \
                $(SRC_DIR)/bnet_debug.c    \
                $(SRC_DIR)/bnet_ctrlMsg.c  \
                $(SRC_DIR)/bnet_dataMsg.c  \
                $(SRC_DIR)/bnet_param.c
else
USRC := bnet_debug.c  \
                bnet_param.c  \
                bnet_xnet.c
endif

KSRC := $(USRC) bnet_kernel_init.c

ifeq ($(A), ko)
```

14.8.5 补丁加载失败，Invalid parameters

问题描述

补丁加载失败，提示Invalid parameters。

问题原因

基线代码正确，运行环境是DEBUG包，但编译时flag文件中缺少DEBUG=1之类的宏，导致编译出的补丁是release版本的热补丁。

处理方法

请确保release版本的热补丁到release包的运行环境验证，debug版本的热补丁到debug包的运行环境验证。

14.8.6 补丁加载失败，Invalid module format

问题描述

补丁加载失败，messages日志“exports duplicate symbol xxx”。

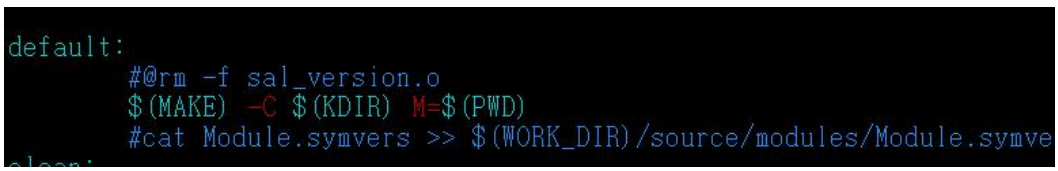
问题原因

模块Makefile问题，每次编译前会删除二进制导致部分符号被热补丁工具识别为新增符号而重复导出。

处理方法

请检查你的Makefile，在MAKE之前是否有删除二进制的操作，有的话请将其注释掉后重新制作热补丁。

图 14-5 示例



```
default:
        #@rm -f sal_version.o
        $(MAKE) -C $(KDIR) M=$(PWD)
        #cat Module.symvers >> $(WORK_DIR)/source/modules/Module.symve
clean:
```

14.8.7 补丁激活失败，编译器优化导致未改动函数被做到补丁中

问题描述

热补丁激活失败，日志显示函数正在被调用，但日志显示的函数并没有修改。

问题原因

编译器优化行为会导致出现没有发生修改的函数在前后两次编译中汇编发生变化，被补丁工具做到热补丁中。

处理方法

在修改的文件中对应的函数xxx后增加以下两句：

```
#include "/usr/share/kpatch/patch/kpatch-macros.h"
KPATCH_IGNORE_FUNCTION(xxx)
```

“xxx”为函数名称。

说明

风险知会：这种方法会导致xxx函数即使发生改动也不做到补丁中，因此必须在确保是由于编译器优化行为导致的函数改动（只是寄存器号发生变化，函数整体汇编逻辑没变）的场景才可使用，否则会导致模块逻辑功能异常。使用时请评估清楚，风险由使用者自行承担。

14.8.8 补丁激活失败，改动的函数频繁调用导致激活失败

问题描述

补丁激活失败，日志显示改动函数正在被调用。

问题原因

改动函数属于频繁调用函数，容易出现激活失败。

处理方法

在修改的文件中对应的函数xxx后增加以下两句：

```
#include "/usr/share/kpatch/patch/kpatch-macros.h"
KPATCH_FORCE_UNSAFE(xxx)
```

“xxx”为函数名称。

说明

风险知会：这种方法会导致xxx函数在补丁激活或回退时不会进行调用栈检查，存在前后一致性风险，即补丁生效后旧函数和新函数可能存在同时运行的场景，请用户充分评估清楚这种场景对模块逻辑的风险，慎重使用，风险由使用者自行承担。

15 批量内核热补丁

15.1 简介

EulerOS内核热补丁应用于EulerOS平台，满足用户场景下一键安装热补丁，免重启生效的需求。热补丁以rpm包方式发布在EulerOS Repo仓库，用户可以通过EulerOS Yum获得和安装热补丁。

15.2 场景&限制

使用场景

- 内核热补丁最大的特点是在不重启系统和不中断业务前提下修改内核中的函数，达到动态替换内核函数的目的。
- 内核热补丁能够动态的修复内核和模块的缺陷函数。在开发人员发现问题，或者操作系统发现安全漏洞需要修复时，可以将缺陷函数或者安全补丁制作成内核热补丁，打入系统中，通过这种方法，在不需要重启系统或者插拔模块、不中断业务的前提下修复缺陷。

约束限制

- 使用EulerOS官方提供的内核热补丁之前，请检查系统不得使用任何自制或者非官方提供的热补丁。否则，可能会引入补丁修改相互覆盖的问题。
- 对于kernel和kernel源码树外的模块，请使用官方提供的版本。内核热补丁与内核版本和模块基线代码状态强相关，重新编译和替换系统中内核模块极有可能造成补丁与模块基线代码状态不一致的问题，给系统带来风险。

15.3 热补丁安装和卸载

15.3.1 安装

热补丁以rpm包方式打包，服务器端应用EulerOS Repo仓库发布，用户场景下使用yum进行热补丁包管理。在安装热补丁之前需要：

1. 配置正确的yum源，即热补丁所在的仓库；
2. 根据需要选择热补丁包，执行yum命令进行安装，即：

```
yum install klp_xxx
```

说明

klp_xxx为热补丁rpm包名称。

yum会自动识别热补丁包依赖关系，自动安装依赖后安装热补丁包。

- 查看热补丁rpm包安装情况，执行命令：

```
rpm -qa | grep klp_
```

执行结果：

```
klp_ext4ioctlv3-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
klp_ext3patch-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
klp_chardevpatch-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
klp_memv1-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
klp_bamemv2-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
klp_ext4dir-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
klp_ememv3-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
```

说明

列出的rpm包均为示例补丁包。

- 通过系统接口查看热补丁激活情况，执行命令：

```
cat /proc/livepatch/state
```

执行结果：

```
Index   Patch                           State
------------------------------------------------
1       klp_memv1                       enabled
2       klp_ext3patch                   enabled
3       klp_ext4dir                     enabled
4       klp_ext4ioctlv3                 enabled
5       klp_bamemv2                     enabled
6       klp_ememv3                      enabled
7       klp_chardevpatch                enabled
------------------------------------------------
```

说明

列出的热补丁均为示例热补丁。

15.3.2 卸载

选择需要卸载的补丁包，执行yum命令进行安全卸载，即：

```
yum remove klp_xxx
```

说明

klp_xxx为热补丁rpm包名称。

热补丁包在安装时会自动检查依赖，并安装依赖的热补丁包。可以执行命令查看热补丁包的依赖：

```
yum deplist klp_xxx
```

如示例：

```
# yum deplist klp_bamemv2-3.10.0-327.49.58.52.x86_64-1.0-1.x86_64
软件包：klp_bamemv2-3.10.0-327.49.58.52.x86_64.x86_64 1.0-1
```

```
  依赖：/bin/sh
不良依赖关系
  依赖：klp_ext4ioctlv3-3.10.0-327.49.58.52.x86_64
  provider: klp_ext4ioctlv3-3.10.0-327.49.58.52.x86_64.x86_64 1.0-1
  依赖：klp_memv1-3.10.0-327.49.58.52.x86_64
  provider: klp_memv1-3.10.0-327.49.58.52.x86_64.x86_64 1.0-1
  依赖：kpatch-runtime
不良依赖关系
```

如果想一同卸载自动安装的依赖的热补丁包，可以执行命令直接卸载依赖包：

```
yum remove klp_ext4ioctlv3-3.10.0-327.49.58.52.x86_64
yum remove klp_memv1-3.10.0-327.49.58.52.x86_64
```

说明

1. 列出的热补丁均为示例热补丁。
2. 对于kbox这类可以卸载的模块，如果有卸载模块的使用场景，并且该模块被安装了热补丁，在卸载模块之前，请使用本章节的方法先卸载它的热补丁。

15.4 热补丁重启恢复

15.4.1 简介

热补丁以内核模块的形式insmod到内核之中，当系统重启之后，被插入的热补丁模块随之失效。 因此需要在开机之时将系统中已经安装的热补丁模块重新加载一遍。

15.4.2 补丁恢复

1. 重启恢复功能已集成到EulerOS系统中。也可以执行命令安装：

   ```
   yum install hotpatchrec
   ```

   安装结果：

   ```
   准备中...                          ################################# [100%]
   正在升级/安装...
      1:hotpatchrec-1.0-0.1              ################################# [100%]
   installing ...Created symlink from /etc/systemd/system/multi-user.target.wants/
   hotpatchrec.service to /usr/lib/systemd/system/hotpatchrec.service.
   ```

2. 查看重启恢复rpm包是否安装，执行命令：

   ```
   rpm -qa | grep hotpatchrec
   ```

   执行结果：

   ```
   # rpm -qa | grep hotpatchrec
   hotpatchrec-1.0-0.1.x86_64
   ```

3. 查看重启恢复的hotpatchrec.service是否已经enabled，执行命令：

   ```
   systemctl is-enabled hotpatchrec.service
   ```

   执行结果：

   ```
   # systemctl is-enabled hotpatchrec.service
   enabled
   ```

4. 每次重启系统之后，hotpatchrec.service会启动恢复进程，进行热补丁恢复，进程在恢复完所有热补丁之后退出。查看hotpatchrec.service状态，执行命令：

   ```
   systemctl status hotpatchrec.service
   ```

   执行结果：

   ```
   [root@LFG1000717738 ~]# systemctl status hotpatchrec.service● hotpatchrec.service - System
   Hotpatch Service. Automatically loads hotpatch when system reboots.   Loaded: loaded
   (/usr/lib/systemd/system/hotpatchrec.service; enabled; vendor preset: disabled)
      Active: active (exited) since 二 2017-06-06 18:07:52 CST; 6min ago
     Process: 948 ExecStart=/usr/bin/hotpatch_recovery (code=exited, status=0/SUCCESS)
   ```



```
  Process: 821 ExecStartPre=/sbin/modprobe hotpatchrec (code=exited, status=0/SUCCESS)
 Main PID: 948 (code=exited, status=0/SUCCESS)
```

5. 查看hotpatchrec.service的日志，执行命令：

```
journalctl _PID=948
```

说明

“948”是执行 systemctl status hotpatchrec.service 命令之后显示的“Main PID”。

找到热补丁恢复结果日志：

```
[  948 2017-06-06 18:09:11.021159] [INFO] [is_finished:689] Hotpatch Recovery Process
completed with 7 succeeded, 0 failed, and 0 missed.
[  948 2017-06-06 18:09:11.021168] [INFO] [dump_result:709] Result Report:
[  948 2017-06-06 18:09:11.021174] [INFO] [dump_result:710] Patch
BuildNo State           Requirements
[  948 2017-06-06 18:09:11.021180] [INFO] [dump_result:711]
--------------------------------------------------------------------------------
[  948 2017-06-06 18:09:11.021188] [INFO] [dump_result:717] klp_memv1
1       Actived
[  948 2017-06-06 18:09:11.021195] [INFO] [dump_result:717] klp_ext3patch
2       Actived
[  948 2017-06-06 18:09:11.021201] [INFO] [dump_result:717] klp_ext4dir
3       Actived
[  948 2017-06-06 18:09:11.021207] [INFO] [dump_result:717] klp_ext4ioctlv3
4       Actived
[  948 2017-06-06 18:09:11.021214] [INFO] [dump_result:717] klp_bamemv2
5       Actived         klp_ext4ioctlv3,klp_memv1
[  948 2017-06-06 18:09:11.021221] [INFO] [dump_result:717] klp_ememv3
6       Actived         klp_bamemv2
[  948 2017-06-06 18:09:11.021226] [INFO] [dump_result:717] klp_chardevpatch
7       Actived
[  948 2017-06-06 18:09:11.021232] [INFO] [dump_result:734]
--------------------------------------------------------------------------------
```

也可以直接通过系统接口查看热补丁恢复情况，执行命令：

```
cat /proc/livepatch/state
```

执行结果：

```
Index   Patch                           State
----------------------------------------------
1       klp_memv1                   enabled
2       klp_ext3patch               enabled
3       klp_ext4dir                 enabled
4       klp_ext4ioctlv3             enabled
5       klp_bamemv2                 enabled
6       klp_ememv3                  enabled
7       klp_chardevpatch            enabled
----------------------------------------------
```

说明

列出的热补丁均为示例热补丁。

16 加入 Windows AD 域

16.1 简介

在大多数环境中， AD域（Active Directory）是网络用户信息的中心，因此将EulerOS系统加入AD域有助于EulerOS用户的集中管理。

本章节介绍如何通过命令行将EulerOS系统加入到Windows AD 域，这将允许我们使用AD域中的用户账户通过SSH登陆到EulerOS系统中。

注意

EulerOS默认使用基于用户名密码本地认证机制，并进行了默认加固。若客户配置使用第三方认证机制，需要确认是否存在口令复杂度校验、防暴力破解、防DOS攻击等机制，建议进行充分测试验证。

16.2 基于 Kerberos

16.2.1 安装需要的软件包

1. 安装realmd ，提供加入域、退出域命令行。执行下面命令：
   ```
   # yum install realmd
   ```
2. 安装sssd，realmd包依赖sssd包来执行加入域的操作。执行下面命令：
   ```
   # yum install sssd
   ```
3. 安装Kerberos，用于认证。执行命令：
   ```
   # yum install krb5-workstation krb5-libs
   ```
4. 安装adcli和samba-common-tools，提供AD和域用户相关命令行。执行命令：
   ```
   # yum install adcli samba-common-tools
   ```


16.2.2 修改配置

1. 配置DNS，确保AD域服务器可以通过域名访问。配置文件为/etc/resolv.conf。
   ```
   # vim /etc/resolv.conf
   ```
   示例如下：
   ```
   search HDOMAIN.LOCAL        #HDOMAIN.LOCAL是域名，请根据实际情况进行配置
   nameserver 192.168.17.12    #192.168.17.12是DNS服务器地址，请根据实际情进行配置
   ```
2. 修改hostname，配置文件是/etc/hostname。
   ```
   #vim /etc/hostname
   ```
   说明

   请根据实际情况修改一个非localhost的名字，确保主机名在域中唯一。
3. 配置时间，确保与AD/DC一致。

   可以安装ntpd和ntpdate这两个包进行设置，与AD/DC保持时间的同步。
4. 修改/etc/ssh/sshd_config，确保如下配置都是yes。
   ```
   # Kerberos options
   KerberosAuthentication yes
   KerberosOrLocalPasswd yes
   KerberosTicketCleanup yes
   KerberosGetAFSToken yes
   KerberosUseKuserok yes
   ```
   执行命令如下命令使得配置生效：
   ```
   systemctl restart sshd
   ```

16.2.3 加入 AD 域

注意事项

1. 确保DNS可以正常工作，主机可以通过域名访问AD域服务器。
2. 确保主机名hostname唯一。
3. 确保主机与AD域服务器时间同步，保证Kerberos认证可以正常工作。

操作步骤

1. 确保能够找到需要加入的域。执行命令：
   ```
   #realm discover HDOMAIN.LOCAL
   ```
   ```
   [root@myeuleros ~]# realm discover HDOMAIN.LOCAL
   hdomain.local
     type: kerberos
     realm-name: HDOMAIN.LOCAL
     domain-name: hdomain.local
     configured: kerberos-member
     server-software: active-directory
     client-software: sssd
     required-package: oddjob
     required-package: oddjob-mkhomedir
     required-package: sssd
     required-package: adcli
     required-package: samba-common
     login-formats: %U
     login-policy: allow-realm-logins
   ```
   说明

   - 检查确认以上图中列出的所有required的包是否都已安装。
   - HDOMAIN.LOCAL：示例域名。
   - 如果该域没有被发现，请确认主机是否可以正常访问域服务器（比如ping HDOMAIN.LOCAL）。
2. 初始化Kerberos。执行命令：

```
#kinit aduser@HDOMAIN.LOCAL
```

```
[root@myeuleros ~]# kinit aduser@HDOMAIN.LOCAL
Password for aduser@HDOMAIN.LOCAL:
[root@myeuleros ~]#
```

没有返回错误，表示成功初始化。

说明

aduser：示例域用户名；

HDOMAIN.LOCAL：示例域名，需要大写。

3. 加入域。执行命令：

```
#realm join --verbose HDOMAIN.LOCAL -U 'aduser@HDOMAIN.LOCAL'
```

```
[root@myeuleros ~]# realm join --verbose HDOMAIN.LOCAL -U 'aduser@HDOMAIN.LOCAL'
 * Resolving: _ldap._tcp.hdomain.local
 * Performing LDAP DSE lookup on: 2002:952:110c::952:110c
 ! Can't contact LDAP server
 * Performing LDAP DSE lookup on: 9.82.17.12
 * Successfully discovered: hdomain.local
Password for aduser@HDOMAIN.LOCAL:
 * Required files: /usr/sbin/oddjobd, /usr/libexec/oddjob/mkhomedir, /usr/sbin/sssd, /usr/bin/net
 * LANG=C LOGNAME=root /usr/bin/net -s /var/cache/realmd/realmd-smb-conf.4YIF4Y -U aduser@HDOMAIN.LOCAL ads join hd
omain.local
Enter aduser@HDOMAIN.LOCAL's password:DNS update failed: NT_STATUS_UNSUCCESSFUL

Using short domain name -- HDOMAIN
Joined 'MYEULEROS' to dns domain 'hdomain.local'
DNS Update for myeuleros.hdomain.local failed: ERROR_DNS_UPDATE_FAILED
 * LANG=C LOGNAME=root /usr/bin/net -s /var/cache/realmd/realmd-smb-conf.4YIF4Y -U aduser@HDOMAIN.LOCAL ads keytab
create
Enter aduser@HDOMAIN.LOCAL's password:
 * /usr/bin/systemctl enable sssd.service
Created symlink from /etc/systemd/system/multi-user.target.wants/sssd.service to /usr/lib/systemd/system/sssd.servi
ce.
 * /usr/bin/systemctl restart sssd.service
 * /usr/bin/sh -c /usr/sbin/authconfig --update --enablesssd --enablesssdauth --enablemkhomedir --nostart && /usr/b
in/systemctl enable oddjobd.service && /usr/bin/systemctl start oddjobd.service
getsebool:  SELinux is disabled
 * Successfully enrolled machine in realm
```

此时日志显示主机已经成功加入了域。

说明

-U 后面的参数和kinit初始化用户名一致。

4. 修改sssd配置文件/etc/sssd/sssd.conf配置项如下：

```
use_fully_qualified_names = False
fallback_homedir = /home/%u
```

```
[sssd]
domains = hdomain.local
config_file_version = 2
services = nss, pam

[domain/hdomain.local]
ad_domain = hdomain.local
krb5_realm = HDOMAIN.LOCAL
realmd_tags = manages-system joined-with-samba
cache_credentials = True
id_provider = ad
krb5_store_password_if_offline = True
default_shell = /bin/bash
ldap_id_mapping = True
use_fully_qualified_names = False
fallback_homedir = /home/%u
access_provider = ad
~
```

执行命令如下命令使得配置生效：

```
systemctl restart sssd
```

5. 修改Kerberos配置文件/etc/krb5.conf，确保默认default_realm指向目标AD域：

```
[libdefaults]
…
# default_realm = EXAMPLE.COM
default_realm = HDOMAIN.LOCAL
…
```



```
[libdefaults]
 dns_lookup_realm = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 rdns = false
# default_realm = EXAMPLE.COM
 default_realm = HDOMAIN.LOCAL
 default_ccache_name = KEYRING:persistent:%{uid}
```

6. 如果需要在登陆时自动为域用户创建home目录，则分别在配置文件/etc/pam.d/system-auth和/etc/pam.d/password-auth 中增加以下内容；否则无需进行以下操作。

```
session     required      pam_oddjob_mkhomedir.so silent skel=/etc/skel umask=0077
```

16.2.4 域用户登陆

1. 成功加入AD域之后，可以执行ssh命令使用域用户登陆：

```
# ssh aduser@myeuleros.hdomain.local
```

说明

aduser：域用户名；

myeuleros.hdomain.local：主机名称。

2. 成功登陆执行命令id，显示如下：

```
# id
uid=1701001116(aduser) gid=1701000513(domain users) groups=1701000513(domain users)
```

16.2.5 退出 AD 域

执行realm leave退出域：

```
# realm leave HDOMAIN.LOCAL
```

说明

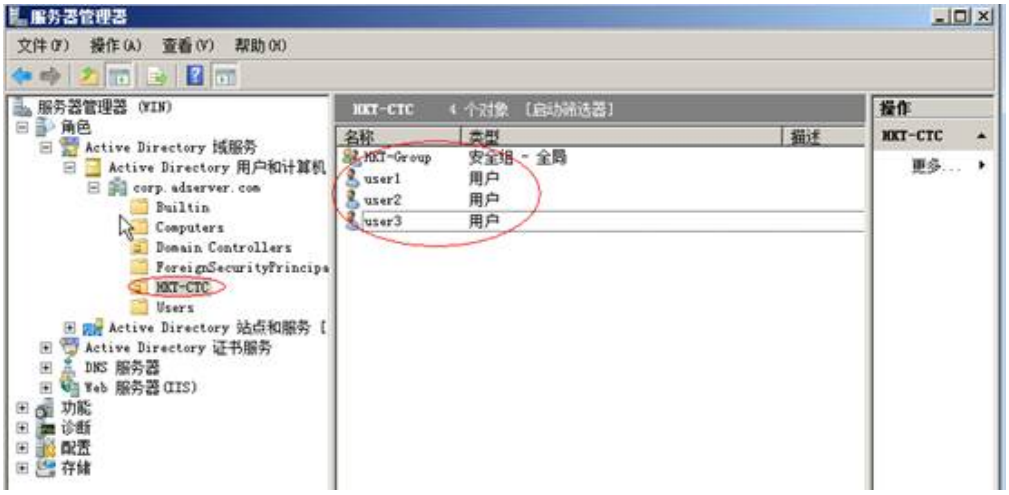
退出域之后，域用户将无法登录。

16.3 基于 openLDAP

16.3.1 Windows 上部署 AD 域服务

1. 详细部署过程参考windows server帮助手册，部署后的信息如下图（示例）。

2. 在AD服务器上创建如下图所示的组及AD域用户（示例）。



16.3.2 EulerOS 上安装软件包

1. 安装sssd，执行下面命令：

```
# yum install sssd
```

2. 安装openldap-clients，提供ldap客户端命令。执行命令：

```
# yum install openldap-clients
```

16.3.3 EulerOS 客户端配置

1. 配置DNS，确保AD域服务器可以通过域名访问。配置文件为/etc/resolv.conf。

```
# vim /etc/resolv.conf
```

示例如下：

```
search CORP.ADSERVER.COM    # HDOMAIN.LOCAL是域名，请根据实际情况进行配置
nameserver 192.168.0.108    # 192.168.0.108是DNS服务器地址，请根据实际情进行配置
```

2. 修改hostname，配置文件是/etc/hostname。

说明

请根据实际情况修改一个非localhost的名字，确保主机名在域中唯一。

示例如下：

```
[root@Euler2 ~]# cat /etc/hostname
Euler2
```

修改主机后重启机器生效，将修改后的主机名添加到DNS主机列表中。

3. 配置AD域名，添加AD域服务器地址及域名到配置文件/etc/hosts，示例如下：

```
[root@Euler2 ~]# vim /etc/hosts
192.168.0.108 corp.adserver.com  #192.168.0.108是AD所在服务器地址，corp.adserver.com是AD对应的域名
```

4. 将本机地址及主机名添加到DNS服务器中。

   使用ping命令测试客户端与AD域服务及DNS服务端的网络是否通畅。

5. 修改配置文件/etc/pam.d/password-auth，增加如下配置，以加粗、斜体表示，因为Euler默认已经集成，所以无需再配置。

```
#%PAM-1.0
# User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        required      pam_faillock.so preauth audit deny=3 even_deny_root unlock_time=300
-auth        sufficient    pam_fprintd.so
auth        sufficient    pam_unix.so nullok try_first_pass
-auth        sufficient    pam_sss.so use_first_pass
auth        [default=die] pam_faillock.so authfail audit deny=3 even_deny_root
unlock_time=300
auth        sufficient    pam_faillock.so authsucc audit deny=3 even_deny_root
unlock_time=300
auth        requisite     pam_succeed_if.so uid >= 1000 quiet_success
auth        required      pam_deny.so

account     required      pam_unix.so
account     required      pam_faillock.so
account     sufficient    pam_localuser.so
account     sufficient    pam_succeed_if.so uid < 1000 quiet
-account     [default=bad success=ok user_unknown=ignore] pam_sss.so
account     required      pam_permit.so

password    requisite     pam_pwquality.so minlen=8 minclass=3 enforce_for_root
try_first_pass local_users_only retry=3 dcredit=0 ucredit=0 lcredit=0 ocredit=0
password    required      pam_pwhistory.so use_authtok remember=5 enforce_for_root
password    sufficient    pam_unix.so sha512 shadow nullok try_first_pass use_authtok
-password    sufficient    pam_sss.so use_authtok
password    required      pam_deny.so

session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
-session     optional      pam_systemd.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet use_uid
session     required      pam_unix.so
-session     optional      pam_sss.so
```

6. 修改/etc/pam.d/sshd，增加如下加粗、斜体部分的配置，确保登录成功后创建用户的home目录。

```
[root@Euler2 ~]# vim /etc/pam.d/sshd
#%PAM-1.0
auth       required     pam_sepermit.so
auth       substack     password-auth
auth       include      postlogin
# Used with polkit to reauthorize users in remote sessions
-auth      optional     pam_reauthorize.so prepare
account    required     pam_nologin.so
account    include      password-auth
password   include      password-auth
# pam_selinux.so close should be the first session rule
session    required     pam_selinux.so close
session    required     pam_loginuid.so
# pam_selinux.so open should only be followed by sessions to be executed in the user context
session    required     pam_selinux.so open env_params
session    required     pam_namespace.so
```



```
session    optional     pam_keyinit.so force revoke
session    include      password-auth
session    include      postlogin
# Used with polkit to reauthorize users in remote sessions
-session   optional     pam_reauthorize.so prepare
# Create user's home directory after successful login
session    required     pam_mkhomedir.so
```

在/etc/pam.d/su中增加同样的配置,确保使用su命令也可以创建home目录。

7. 在/etc/sssd/下创建配置文件sssd.conf，并添加以下内容（根据自己实际环境配置dn、uri）。

```
[domain/corp.adserver.com]
id_provider = ldap
auth_provider = ldap
chpass_provider = ldap
#AD域服务器地址
ldap_uri = ldaps://win.corp.adserver.com:636
cache_credentials = True
ldap_tls_reqcert = allow
ldap_id_use_start_tls = True
ldap_schema = ad

#以下根据实际情况配置
ldap_search_base = DC=corp,DC=adserver,DC=com
ldap_default_bind_dn = CN=user1,OU=HKT-CTC,DC=corp,DC=adserver,DC=com
ldap_default_authtok = huawei@123

ldap_id_mapping = True
default_shell = /bin/bash
fallback_homedir = /home/%u

ldap_user_object_class = user
ldap_user_name = sAMAccountName
ldap_user_home_directory = unixHomeDirectory
ldap_user_principal = userPrincipalName
ldap_group_object_class = group
ldap_group_name = sAMAccountName

[sssd]
services = nss, pam
config_file_version = 2
#定义域名，与domain标签对应
domains = corp.adserver.com

[nss]
homedir_substring = /home

[pam]
```

重启sssd服务：

```
[root@Euler2 ~]# systemctl restart sssd
```

将sssd服务加入到自启动列表中：

```
[root@Euler2 ~]# systemctl enable sssd
```

检查与服务端连接，示例：

```
[root@Euler6 ~]# ldapsearch -x -h win.corp.adserver.com -D "CN=user1,OU=HKT-
CTC,DC=corp,DC=adserver,DC=com" -w "huawei@123" -b " OU=HKT-CTC,DC=corp,DC=adserver,DC=com"
```

返回结果中出现“result: 0 Success”，表示成功。

16.3.4 域用户登录

```
[root@Euler2 ~]# id user2
uid=1185401111(user2) gid=1185400513(Domain Users) groups=1185400513(Domain Users)
[root@Euler2 ~]# ssh user2@localhost

Authorized users only. All activities may be monitored and reported.
```

```
user2@localhost's password:
Creating directory '/home/user2'.
Last login: Sun Jun  3 23:54:57 2018 from 192.168.0.122

Authorized users only. All activities may be monitored and reported.
[user2@Euler2 ~]$
```


17 RAS

17.1 概述

本节介绍RAS的一些基础概念，让您了解RAS的作用。

背景描述

随着信息化技术的广泛应用，大型的数据中心、网络中心，如股票证券交易所，电信机房，银行的数据库中心等应用环境对IT的依赖程度日益加深。针对上述情况，如何确保整个系统尽可能长期可靠的正常运行而不下线，并且具备足够强大的容错机制，已经变得越来越重要，RAS机制已成为不可或缺的一部分。

RAS 简介

RAS是Reliability, Availability and Serviceability的缩写，即可靠性，可用性和可服务性。

RAS设计的核心指导理念就是“最大程度保证客户业务可持续正常运行”。换言之，RAS设计就是“尽量降低宕机的可能性”。高可用的单机系统，必须具有高可靠的底层设计（包括硬件和底层软件）、高容错性、快速修复的能力以及快速服务的能力。

RAS 的重要性

对于关键业务服务器来说，其核心要求就是系统具备提供不中断的持续服务的能力，其原因就在于在不同的应用中，服务器宕机导致业务中断带来的损失不完全相同，业务越关键，宕机所带来的损失越大。随着IT影响的不断快速深入，企业对IT系统的依赖程度日益加深，宕机成本正在变得越来越高。

RAS设计中，最基础的要求是保证器件应用的可靠性，即要求具有“硬件尽量不要出故障”的能力。

器件级的可靠性设计的方法，归根结底是两个基本要求：“使用正确的器件”和“正确的使用器件”。前者对器件选型和引入有较高的要求；后者要求更多从设计上考虑，比如降额应用等。器件级的可靠性质量保证包括供应商物料可靠性管理、产品可靠性设计、生产可靠性筛选三个环节，必须各司其职，相互配合，综合考虑。

17.2 约束限制

本节介绍使用RAS时的一些约束限制，避免用户使用时出现异常。

硬件约束

当前仅支持支持KunLun9016，9032服务器。

软件约束

- 操作系统版本：EulerOS V2.0SP5。
- 内核版本：3.10.0-229.20.1.30.hulk 及以上。

使用限制

- 热插拔会使大页（hugeTLB）的overcommit_memory参数不起作用。

  为了保证内存下线过程中大页迁移不会失败，大页迁移时，OS会跳过overcommit_memory这个判断，即使用户分配的大页不足，OS在下线过程中的内存迁移时也能分配超过阈值的大页。执行过一次节点下线（节点或单独的内存）后，overcommit_memory参数会失效，用户不能通过该接口限制大页的分配。
- 热插拔会破坏内存和node的绑定策略。mbind函数通过参数nodemask限制分配节点。执行过一次节点下线（节点或单独的内存）后，该功能失效。

17.3 热插拔说明

前提条件

- 版本：EulerOS V2.0SP5。
- 内核：3.10.0-229.20.1.30.hulk 及以上。

硬件依赖

支持KunLun9016，9032服务器。

相关配置

- 启动参数配置：配置movable_node参数，开启支持热插拔功能。

  修改“/boot/grub2/grub.cfg”（EFI系统的配置文件在“/boot/efi/EFI/euleros/grub.cfg”），在内核启动选项中增加movable_node numa_zonelist_order=zone。
- 支持内存镜像（address range mirroring），在3.10.0-229.20.1.30.内核版本及其之后的版本已默认支持，早期版本需要添加mirrorable启动参数。

OS 处理热插拔的原则

- 热插拔要保证业务不能中断。即不能因为内存下线触发OOM而导致业务中断，此时热插拔没有意义。
- 热插拔要满足多数典型场景（如oracle场景），确保每种场景下都能热插拔成功。
- 实际应用热插拔时，多数情况是硬件故障或者硬件可能有问题的场景，需要更换硬件。

热插拔须知

- 开启热插拔支持，会将OS内核限定在不可热插拔的（unmovable）node上运行。这样会带来内核跨节点访存的增加，可能导致性能下降。
- 热插拔会破坏内存和node的绑定策略。

  比如，某些应用绑到node1上，如果node1下线了，则绑定关系由bind退化为prefer。即如果该节点内存不足，就会去其他节点分配内存，以防止因为内存下线，而将绑到node1上的进程kill掉。

- 热插拔会使大页（hugeTLB）的overcommit_memory参数不起作用。

  为了保证内存下线过程中大页（HUGETLB）迁移不会失败，大页迁移时OS会跳过overcommit_memory这个判断。即如果用户分配的大页不足，OS会在下线过程中的内存迁移时分配超过阈值的大页，以防止在下线过程中出现因分配不到大页而引起的OOM。

- 热插拔时，系统负载不宜过大。

  以16P为例，内存使用率（memused/（total - mem_tobe_removed））一般应在60%以下。内存使用率高，会使内存迁移时间延长，而且迁移失败概率增大。

- 热插拔过程中，下线有可能会失败。

  应用程序在运行过程中，可能会因锁住内存，或过于频繁访问文件而引起filemap_fault，从而导致下线失败。下线失败不是错误，可以多次尝试重新下线。

  下线失败，可以通过nodectl工具查看失败原因。该工具可以查看哪些进程在使用该节点内存。如查看哪些进程在使用node1的内存，代码如下：

```
[root@localhost nodectl]# nodectl --procs 1
This following process(es) have memory page(s) allocated on node 1
9757,           watch: pages = 119 (476 kb)
169177,         nodectl: pages = 27 (108 kb)
173780,         nodectl: pages = 3 (12 kb)

# 加 -v 参数，可以查看详情
[root@localhost nodectl]# nodectl --procs -v 1
This following process(es) have memory page(s) allocated on node 1
[  3690,        libvirtd] 7f9c7c000000 default anon=287 dirty=286 swapcache=1 active=282
N1=1 N4=1 N6=1 N7=284
[  3690,        libvirtd] 7f9c863d2000 default file=/usr/lib64/libvirt/connection-driver/
libvirt_driver_nodedev.so anon=1 dirty=1 N1=1
[  3690,        libvirtd] 7f9c9943f000 default file=/usr/lib64/libc-2.17.so anon=2 dirty=2
N0=1 N1=1
[  3690,        libvirtd] 7f9c9e3e4000 default heap anon=20 dirty=20 active=19 N0=12 N1=3
N4=2 N7=3
[  9757,           watch] 00e27000 default heap anon=437 dirty=437 N1=106 N2=1 N3=148 N4=8
N6=170 N7=4
[  9757,           watch] 7fc80568b000 default file=/usr/lib64/libc-2.17.so anon=2 dirty=2
N1=1 N3=1
[  9757,           watch] 7fc805abf000 default file=/usr/lib64/libtinfo.so.5.9 anon=1
dirty=1 N1=1
[  9757,           watch] 7ffc546fb000 default stack anon=3 dirty=3 N1=1 N3=2[ 32838,
systemd-udevd] 7f5df58a0000 default heap anon=78 dirty=69 mapmax=5 swapcache=9 active=69
N1=1 N5=24 N6=19 N7=34[ 32838,   systemd-udevd] 7ffd97b72000 default stack anon=14 dirty=14
mapmax=5 N1=2 N4=1 N5=7 N7=4
```


17.4 内存镜像说明

背景

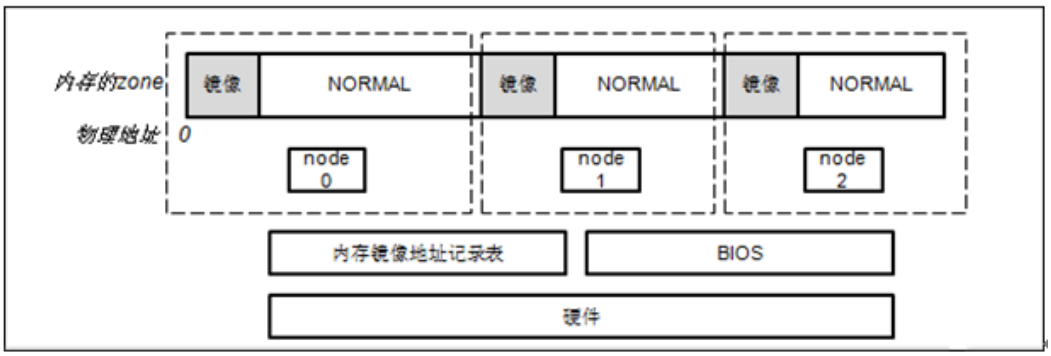
目前大多高端服务器都支持内存镜像特性，其原理是把整个系统的一半内存作为另一半内存的镜像，在主内存数据发生错误时自动从镜像内存中读取，从而提高系统的可靠性。但现有技术的缺点是整个系统有一半的内存OS无法使用，浪费严重。

目前较新的CPU（如Intel Xeon系列）已支持分段内存镜像技术（即部分内存镜像特性），它可以人为指定某块区域做镜像，然后把重要的数据放在镜像内存区，那些不重要的数据所占的内存区域或者是空闲内存区域不做镜像，这样一来就能在保证高可靠性的同时，减少内存浪费。

原理介绍

部分内存镜像特性如图1所示。

图 17-1 部分内存镜像特性示意图



- 如图1所示，一个具有三个node的NUMA系统，每个node都有一段物理地址做内存镜像，OS访问时并无区别，即OS在读写这段内存时和其他区域一样。硬件层面会做镜像处理，如果发生ECC等内存错误，由硬件自动纠正。简单来说就是OS实现了一个能在指定物理地址分配内存的特性。
- 内存镜像硬件支持通过启动时BIOS的RAS内存镜像进行配置。

场景描述

部分内存镜像特性作为一个非常重要的RAS新特性，在不过分减少可用内存的前提下，有效提高了计算机系统的可靠性。该特性可集成到高端服务器中，部署在一些关键业务行业，如银行交易中心、军工科研等涉及高可靠性计算机需求的领域。本特性应用于配置部分内存镜像的计算机系统。主要通过修改操作系统内存分配策略，使操作系统内核数据使用可靠性更高的镜像内存，提高系统整体可靠性，并提供mirror内存地址供查询。

特性功能

1. /proc/iomem_ext接口显示mirror段地址：cat /proc/iomem_ext可显示内存段镜像内存信息，若地址段为镜像内存，则在该地址后会有<Mirrored>标识。

2. 内核代码段、数据段、重要结构数据（struct page等结构数据）使用镜像内存。
3. 内核内存操作使用mirror区（ZONE_NORMAL全镜像内存）：kmalloc、kmem_cache_alloc等内核开辟的内存使用镜像内存。

说明

- 启用“**特性功能1**”，需内核编译选项需开启CONFIG_ACPI_MIRROR_MEMORY。
- 内核已开启“**特性功能2**”，无需额外配置。
- 使能“**特性功能3**”，需在cmdline中加入kernelcore=mirror。

规格限制

1. 计算机系统硬件平台（如cpu）必须支持部分内存镜像特性。
2. 部分内存镜像只能在系统启动时配置，无法在系统起来后动态配置，如改变镜像区域大小。
3. 目前已知支持的服务器：华为9032系列服务器、9008 V5、KunLun V1 E7-V4。
4. 建议预留全部物理内存的1/20以上的容量，作为镜像内存。
5. 建议合理设置内存镜像的大小：不要覆盖整个node的内存大小，node0建议多设置镜像内存。
6. 部分结构数据如struct page是和节点关联的，保存在各个节点管理的内存中，且这些结构数据仅能使用本节点的数据，若需这部分结构数据使用mirror，需要在全部节点中配置mirror区以供该数据使用，建议设置每个节点预留内存镜像的大小为节点内存总量的1/64以上。
7. 要求至少开启EFI中低4G镜像内存配置否则，在**“特性功能3”**开启且存在其他镜像内存的环境下将打印警告：“This configuration results in unmirrored kernel memory.”。
8. **“特性功能3”**与其他kernelcore=xxx配置有冲突，详见Documentation/kernel-parameters.txt中kernelcore部分。

影响

1. 配置了内存镜像，系统内存容量减小了。
2. 镜像内存部分的可靠性相对较高。
3. 硬件如果不支持内存镜像，则无影响，/proc/iomem_ext将与/proc/iomem显示一致，且若指定了kernelcore=mirror，将自动禁用kernelcore=mirror功能。
4. 与热插拔特性兼容。
5. 若硬件上对某节点不配置mirror，则该节点的struct page等结构数据将会用普通内存，并在日志中可查mirror使用失败记录，无其他影响。
6. 开启“**特性功能3**”后，每个节点的ZONE_NORAM区（直接映射区）全部为镜像内存，非镜像内存移入ZONE_MOVABLE区，内核内存仅仅只能使用直接映射区，而用户态内存也可以使用直接映射区，需要保证mirror区足够大（否则将导致内核可使用内存耗尽OOM）。

17.5 内存 patrol scrub error 增强处理

背景

随着内存容量和内存工艺的变化，内存多bit跳变带来的故障率不断上升。内存从DDR3升级到DDR4以后，由于工艺改进，密度更高，能耗更低，也更容易受到干扰，内存故障发生的概率会更高。需要通过系统级的容错策略降低内存多bit故障对系统的影响。

本章节是针对uncorrected patrol scrubbing error（Intel CPU架构下内存多bit故障之一）的增强处理与相关接口的描述。

原理介绍

Uncorrected patrol scrubbing error是在BIOS开启patrol scrub功能后，由CPU的IMC（内存控制器）按照用户设定的频率，周期性内存扫描触发的。扫描周期可以设置，比如：24小时/次。当内存条发生了多bit错误时，如果这部分内存一直没有被CPU“消费”，同时被IMC扫描到了，IMC将此错误记录在Error-Reporting Bank寄存器中，此错误就是uncorrected patrol scrubbing error。

Uncorrected patrol scrubbing error属于当前CPU没有访问（“未消费”）的内存，不会对系统立刻造成危害。但如果不及时处理，等到CPU访问到此内存页后，就会从“未消费”成为“消费”类型，从而会破坏CPU执行上下文，结果必须通过系统复位（panic）来避免错误在系统中扩散和对重要数据的破坏。

对于patrol scrubbing错误，需要CPU、BIOS以及操作系统一起协同工作才能正确处理。大致流程如下：当CPU上的IMC扫描出多bit错误时，CPU会将此错误记录在一组特定的寄存器组中，然后触发中断通知BIOS/OS去读这些寄存器。OS读出这些寄存器的值后，按照Intel CPU的规范解析之。如果解析能识别出是patrol scrubbing error，则OS将对内存页面做隔离处理。比如：如果是空闲内存，隔离后避免后续被分配使用。

上述流程中的关键环节分为两部分：1. CPU记录在寄存器组中的值；2. OS对寄存器值的解析。

实际上，Intel把uncorrected patrol scrubbing error的支持能力归属于高级RAS特性中，只有支持高级RAS的CPU（比如：Intel(R) Xeon(R) Gold 5120T），其记录在寄存器中的值OS才能识别和进一步处理。不支持高级RAS的CPU，其记录的值，OS无法识别并且会复位系统。

针对不支持高级RAS的Intel CPU，EulerOS可以支持uncorrected patrol scrubbing error的识别和处理。

开发流程

因为不同的Intel CPU型号，对于uncorrected patrol scrub error记录在MCE banks中的值不同。开发者在编写业务模块代码时，需要了解清楚CPU实际记录的寄存器信息。

- 如果CPU对于uncorrected patrol scrub error是以UCE（硬件不可纠正错误）形式上报并会触发系统panic，则需要按照**表17-1**配置“mce不要复位系统”规则，避免系统复位。
- 如果需要对uncorrected patrol scrub error做内存隔离或者其它一些增强处理，则需要按照**表17-1**配置“内存隔离”规则。内存隔离时，如果需要做增强处理，可以注册**表17-2**中“fma_register_memory_offline_notifier”接口完成注册，当满足“内存隔离”规则和条件时，系统会通知注册的用户，用户能感知是UCE（硬件不可纠正错误）还是CE（硬件可纠正错误）。

- 如果在系统panic前想感知是mce错误导致并做一些增强处理，则需要按照表17-1配置“mce触发的panic通知用户”规则。还需调用表17-2中“mce_register_panic_notifier_chain”接口完成注册。“mce触发的panic通知用户”规则是配置用户感兴趣的MCE类型，比如：用户只想感知内存类的MCE错误，那么当内存发生多bit错误且导致系统panic时，就会通知用户。

使用说明

用户可以通过添加启动参数，配置“mce不要复位系统”、“内存隔离”、“mce触发panic通知用户”三类规则。比如：

```
/boot/grub2/grub.cfg
```

表 17-1 启动参数说明

参数项	参数说明	取值范围	是否必选
mce_rules	用于设置“mce不要复位系统”、“内存隔离”、“mce触发panic通知用户”三类规则。 配置格式： `mce_rules=not_panic:rule1,rule2,rule3#panic_notify:rule1,rule2,rule3#mem_isolate:rule1,rule2,rule3`	rule表示是U64位大小的16进制形式的字符串。 范围[0x0, 0xFFFFFFFFFFFFFFFF]	选配。

配置示例：

```
mce_rules=panic_notify:0x0#mem_isolate:0xc00000000100080
```

说明

- "not_panic"、"panic_notify"、"mem_isolate"表示3类规则，规则名后跟“:”号，后面是具体规则内容。
- 每一类规则最多支持配置3条子规则，即：rule1,rule2,rule3，用“,”号间隔。
- "not_panic"、"panic_notify"、"mem_isolate"规则之间用“#”号间隔。
- 如果MCE错误直接触发底层硬件发起复位，内核感知不到，则“mce不要复位系统”规则不适用。

表 17-2 接口调用说明

编号	接口	调用场景
1	int mce_register_panic_notifier_chain(struct notifier_block *nb);	系统发生mce panic时并且匹配“mce触发的panic通知用户”的规则，则通知用户。调用此接口完成通知链注册。
2	int mce_unregister_panic_notifier_chain(struct notifier_block *nb);	调用此接口完成“mce触发的panic通知用户”的通知链注销。

编号	接口	调用场景
3	int fma_register_memory_offline_notifier(struct notifier_block *nb)	用户需要感知系统执行内存隔离，调用此接口完成通知链注册。
4	void fma_unregister_memory_offline_notifier(struct notifier_block *nb)	调用此接口完成“内存隔离”的通知链注销。

说明

- mce panic通知链是在NMI上下文，用户必须保证回调函数执行是原子操作，否则会造成CPU hang。
- mce panic通知链不提供防重入和并发，用户根据实际需求决定是否实现防重入和并发。
- 如果用户执行完自己的内存隔离回调函数，不想继续走内核的隔离流程，用户可以在回调函数的返回值中标示”停止通知链执行“的类型。

接口描述

- 通知链struct notifier_block结构体

  结构体定义：

  定义在内核的include/linux/notifier.h头文件中。

```
typedef int (*notifier_fn_t)(struct notifier_block *nb, unsigned long action, void *data);

struct notifier_block {
    notifier_fn_t notifier_call;
    struct notifier_block __rcu *next;
    int priority;
};
```

  成员变量：

成员变量	成员变量说明
notifier_call	函数指针。开发者向通知链上注册的回调函数。请见**notifier_fn_t函数指针**说明。
next	指向通知链中的下一个条目的指针，开发者不用关心。
priority	遍历通知链中条目的优先级。数值越大优先级越高，同一优先级的条目先注册的优先级高。

- 返回值：

  返回值定义在内核include/linux/notifier.h头文件中。

参数项	参数说明
NOTIFY_DONE	表示对相关的事件类型不关心。

参数项	参数说明
NOTIFY_OK	成功执行。
NOTIFY_BAD	执行出错并停止通知链执行。
NOTIFY_STOP	停止通知链执行。
NOTIFY_STOP_MASK	停止执行的掩码。

- 注册通知链

  接口原型：

  ```
  int mce_register_panic_notifier_chain(struct notifier_block *nb);
  ```

  接口功能：

  注册新的条目到通知链上。

  输入参数：

参数项	参数说明
nb	注册到通知链上的新条目的结构体指针。

  notifier_fn_t函数指针在

  函数原型：

  ```
  typedef int (*notifier_fn_t)(struct notifier_block *nb, unsigned long action, void *data);
  ```

  函数功能：

  注册到通知链上的回调函数，由开发者实现。

  输入参数：

参数项	参数说明
nb	通知链条目的结构体指针。
action	为0，暂未使用。
*data	为NULL，暂未使用。

  返回值：

参数项	参数说明
0	执行成功。

参数项	参数说明
-EINVAL	执行失败。失败原因： 1. 入参nb结构体指针为空。 2. 入参nb结构体指针指向的成员变量notifier_call函数指针为空。

- 解注册通知链

  接口原型

  ```
  int mce_unregister_panic_notifier_chain(struct notifier_block *nb);
  ```

  接口功能：

  从通知链上解注册某个条目。

  输入参数：

参数项	参数说明
nb	从通知链上解注册的条目的结构体指针。

  返回值：

参数项	参数说明
0	执行成功。
-EINVAL	执行失败。失败原因： 1. 入参nb结构体指针为空。
-ENOENT	执行失败。失败原因： 1. 通知链中没有找到要解注册的nb。

- 注册通知链

  接口原型：

  ```
  int fma_register_memory_offline_notifier(struct notifier_block *nb)
  ```

  接口功能：

  注册新的条目到通知链上。

  输入参数：

参数项	参数说明
nb	注册到通知链上的新条目的结构体指针。

  notifier_fn_t函数指针在

  函数原型：

  ```
  typedef int (*notifier_fn_t)(struct notifier_block *nb, unsigned long action, void *data);
  ```

函数功能：

注册到通知链上的回调函数，由开发者实现。

输入参数：

参数项	参数说明
nb	通知链条目的结构体指针。
action	mce错误类型。请参见内核include/linux/memory.h头文件中定义的枚举类型fma_mem_offline_notify_types。 如果用户需要感知错误类型时可以参考以下描述： ● FMA_MEM_OFFLINE_NOTIFY_INJECT：内核测试程序注入的mce故障触发的内存隔离； ● FMA_MEM_OFFLINE_NOTIFY_SRAO：patrol scrub error或者L3 cache回写错误触发的内存隔离； ● FMA_MEM_OFFLINE_NOTIFY_PAGE：mcelog针对CE类型的page错误达到阈值触发“hard”方式内存隔离。或者是APEI Generic Hardware Error Source触发的内存隔离； ● FMA_MEM_OFFLINE_NOTIFY_SOFT：mcelog针对CE类型的page错误达到阈值触发“soft”方式内存隔离； ● FMA_MEM_OFFLINE_NOTIFY_SRAR：数据加载或者指令获取出错触发内存隔离。
*data	unsigned long类型pfn地址信息

返回值：

参数项	参数说明
0	执行成功。

- 解注册通知链

  接口原型

  ```
  void fma_unregister_memory_offline_notifier(struct notifier_block *nb)
  ```

  接口功能：

  从通知链上解注册某个条目。

  输入参数：

参数项	参数说明
nb	从通知链上解注册的条目的结构体指针。

17.6 单比特 ECC 错误处理

本节介绍EulerOS系统中降低CMCI中断频率的方法，以及具体的操作流程。

概述

在内存发生单bit ECC错误时，为了方便配置CMCI中断频率，以及转轮询处理后的轮询周期，降低CMCI中断对系统的性能影响， EulerOS提供了调整CMCI中断触发门限和轮询周期配置方案：通过修改启动参数指定CMCI中断触发门限，以及修改轮询周期。

使用说明

用户可以通过修改启动参数指定CMCI中断触发门限和CMCI中断轮询处理的频率。比如：

```
/boot/grub2/grub.cfg
```

相关参数说明如下表所示。

表 17-3 启动参数说明

参数项	参数说明	取值范围	是否必选
cmci_threshold	用于设置CMCI中断触发门限。比如内存单bit ECC错误会触发CMCI中断。	[1,0x7fff]	选配。 该参数默认值是1，代表内存触发1次单bit ECC错误，上报一次CMCI中断。
cmci_poll_interval	用于设置CE错误轮询检测的频率，单位是秒。	[1,300]	选配。 该参数默认值是60，代表CMCI中断转轮询处理模式后，每60s轮询处理一次。
cmci_storm_threshold	用于设置CMCI中断风暴抑制的阈值，单位是个数。	[0,15]	选配。 该参数默认值是15，代表1秒内某个CPU连续收到超过15个CMCI中断就进入风暴抑制状态，同时切换成轮询模式。
cmci_poll_times	用于设置在轮询模式下的必须轮询的次数。单位是个数。	[1,300]	选配。 该参数默认值是300，代表轮询模式中轮询的次数。

这4个参数是选配，如果用户希望调整CMCI中断触发的门限或者轮询周期，启动参数需要增加如下配置：

```
cmci_threshold=xxx
```

或者配置轮询周期：

```
cmci_poll_interval=xxx
```

或者2个参数一起配置：


```
cmci_threshold=xxx cmci_poll_interval=xxx
```


18 Docker

18.1 简介

虚拟化技术使应用在各自的空间内运行相互不影响，为人们提供了极大的便利，同时减少了对实体服务器的需求，降低了成本。目前虚拟化技术包括KVM、XEN、VMWare等，全虚拟化技术或半虚拟化技术存在对主机开销大的缺点，而Linux container技术是通过与主机共用内核，结合内核的cgroup和namespace实现的一种虚拟化技术，极大的减少了对主机资源的占用而且具有较快的启动速度。docker就是一个Linux container引擎技术，实现应用的打包、快速部署等。

Docker的英文本意是码头工人，码头工人的工作就是将商品打包到Container（集装箱）并且搬运Container、装载Container。从docker字面上的解释就可以看出docker是干什么的，对应到Linux中，docker就是将App打包到Container，通过Container实现App在各种平台上的部署，运行。Docker通过Linux Container技术将App变成一个标准化的、可移植的、自管理的组件，实现了应用的build once、run everywhere。Docker技术特点是：应用快速发布、应用部署和扩容简单、更高的应用密度、应用管理更简单。

18.2 安装 Docker

安装

对于EulerOS，docker rpm的安装包在iso或者repo中，可以通过配置yum源为公共仓库地址的repo文件，使用yum来安装。

1. 创建yum源，在/etc/yum.repos.d/目录下配置EulerOS-base.repo文件，添加如下信息：

```
[base]
name=EulerOS-base
```



```
baseurl=http://developer.huawei.com/ict/site-euleros/euleros/
repo/yum/2.x/os/x86_64/
enabled=1
gpgcheck=1
gpgkey=http://developer.huawei.com/ict/site-euleros/euleros/
repo/yum/2.x/os/RPM-GPG-KEY-EulerOS
```

2. 安装

```
yum install -y docker-engine
```

3. 启动

说明

docker默认状态是开启的。

启动时的参数（默认只有daemon），可以通过下面的命令查看：

```
root@euler:~/workspace# ps -eaf|grep docker
root295210 16:55  00:00:00 /usr/bin/docker daemon -H fd://
root334212130 17:11 pts/000:00:00 grep docker
```

卸载

```
yum remove -y docker-engine
```

18.3 配置 Docker

Docker服务可以通过手动输入命令启动，它的配置就是传给docker的命令行参数，也可以通过服务的方式来管理，这时的配置是通过配置文件来完成的。

- 通过命令行启动

```
docker daemon -D #命令行启动，-D打开调试,若要停止，kill掉docker进程
```

- 通过systemd服务启动

```
systemctl start docker   #服务的方式来启动，停止为stop
```

用户可以通过修改 /usr/lib/systemd/system/docker.service 文件来修改docker服务的启动参数。

```
# vim /usr/lib/systemd/system/docker.service[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
After=network.target

[Service]
Type=notify
EnvironmentFile=-/etc/sysconfig/docker
EnvironmentFile=-/etc/sysconfig/docker-storage
EnvironmentFile=-/etc/sysconfig/docker-networkEnvironment=GOTRACEBACK=crash# the default is
not to use systemd for cgroups because the delegate issues still# exists and systemd
currently does not support the cgroup feature set required# for containers run by docker
ExecStart=/usr/bin/docker daemon $OPTIONS \
          $DOCKER_STORAGE_OPTIONS \
          $DOCKER_NETWORK_OPTIONS \
          $ADD_REGISTRY \
          $BLOCK_REGISTRY \
          $INSECURE_REGISTRY
LimitNOFILE=1048576
LimitNPROC=1048576
LimitCORE=infinity
TimeoutStartSec=0# set delegate yes so that systemd does not reset the cgroups of docker
containersDelegate=yes
# kill only the docker process, not all processes in the cgroup
KillMode=process

[Install]
WantedBy=multi-user.target
```


说明

1. 通过“systemctl restart docker”重启服务使配置生效。
2. EnvironmentFile=-/etc/sysconfig/docker，定义环境变量所在的文件。该文件中定义的环境变量可以在/usr/lib/systemd/system/docker.service文件中进行引用。
3. docker服务程序的启动参数需要加到“ExecStart”的后面。

- 配置docker的外网代理

  若要从docker hub上pull镜像，还需要设置代理。

  a. 在/usr/lib/systemd/system/docker.service中添加内容如下：
  ```
  Environment="HTTP_PROXY=http://count:passwd@proxyhk.huawei.com:8080"
  ```
  b. 重启daemon使配置生效
  ```
  systemctl restart docker
  ```

18.4 使用 Docker

docker安装完成后，会自动启动，若是直接使用docker的可执行程序，可以使用docker daemon &启动docker的后台进程。docker后台进程的启动需要root权限。这一章介绍docker中一些简单应用，使读者对docker有一个初步了解和印象，对于命令的使用不作详细解释，对于docker更多的使用，请参考docker --help。

- 下载镜像
  ```
  docker pull ubuntu
  ```
  上面的命令将会从dockerhub（需要连接到外网）上去下载ubuntu镜像

  若是有自己的hub，也可以用下面的方式下载：
  ```
  docker daemon -D  --insecure-registry=rnd-dockerhub.huawei.com
  docker pull rnd-dockerhub.huawei.com/library/ubuntu
  ```
  说明

  上面的rnd-dockerhub.huawei.com是自己的hub地址，library是hub上用户的账号。
- 启动容器
  ```
  docker run -ti ubuntu bash
  ```
- 关于docker的使用，可以使用--help获取当前几乎所有关于docker的使用
  ```
  docker  --help        #显示docker子命令
  docker daemon --help  #显示docker daemon的详细信息
  ```

18.5 Docker 使用限制

18.5.1 docker daemon 配置和使用相关

18.5.1.1 daemon 启动时间

Docker服务由systemd管理，systemd对各个服务的启动时间有限制，如果指定时间内docker服务未能成功启动，则可能有以下原因：

- 如果使用devicemapper，且为第一次启动，docker daemon需要对该设备做mkfs初始化操作，而该操作对IO性能依赖较高，很可能出现超时。该设备只需要初始化一次，后续的daemon启动不再需要重复初始化。
- 如果当前系统资源占用太高，导致系统卡顿，当前系统所有的操作都会变慢也可能会出现docker服务启动超时的情况。

- daemon重启过程中需要遍历并读取docker工作目录下每一个容器的配置文件、容器init层和可写层的配置，如果当前系统存在过多容器（包含created和exited的容器）、并且磁盘读写性能受限，也会出现daemon遍历文件过久导致docker服务启动超时的情况。出现服务启动超时情况，建议对以下两种情况进行排查调整：
  - 容器编排层定期清理不需要的容器，尤其是exited的容器。
  - 结合解决方案的性能要求场景，调整编排层的清理周期和docker服务的启动时间。

18.5.1.2 重启 systemd-journald 后需要重启 docker daemon

Journald通过pipe获取Docker daemon的日志，如果journald服务重启，会导致该pipe被关闭，docker的日志写入操作便会触发SIGPIPE信号，正常情况下该错误信号会导致docker daemon crash， 新版本已经合入patch，忽略该信号，即出现该问题后docker daemon不会crash，但是由于pipe已经丢失，docker daemon的日志也就无法写入。所以建议用户在重启journald服务或者journald异常后主动去重启docker daemon，保证docker日志能够被正常记录。

18.5.1.3 重启 firewalld 服务后需要重启 docker daemon

firewalld服务启动会清空当前系统的iptables规则，所以启动docker过程中重启firewalld可能会导致docker服务插入规则失败进而导致docker服务启动失败。docker服务启动后重启firewalld服务会导致docker的iptables规则被清除，创建带端口映射的容器会失败。

所以，建议在重启或者拉起firewalld之后要重启docker服务，保证docker服务在firewalld之后启动。

18.5.1.4 禁止修改 docker daemon 的私有目录

不允许对docker用的根目录（默认/var/lib/docker，可以用-g参数重新指定）和运行时目录（默认/run/docker）以及这两个目录下的文件或者目录作修改，包括在该目录下删除文件，添加文件，对目录或者文件做软/硬链接，修改文件的属性/权限，修改文件的内容等，如果确实需要做修改，请联系欧拉容器团队评审。

18.5.1.5 使用 devicemapper 注意事项

使用devicemapper必须使用devicemapper+direct-lvm的方式，配置的方法可以参考**https://docs.docker.com/engine/userguide/storagedriver/device-mapper-driver/#configure-direct-lvm-mode-for-production**。

配置devicemapper时，如果系统上没有足够的空间给thinpool做自动扩容，请把自动扩容功能禁止，禁止的方法是把/etc/lvm/profile/docker-thinpool.profile中的

```
activation {
  thin_pool_autoextend_threshold=80
  thin_pool_autoextend_percent=20
}
```

的这两个值都改成100。使用devicemapper时推荐加上--storage-opt dm.use_deferred_deletion=true --storage-opt dm.use_deferred_removal=true

在EulerOS上使用devicemapper时，容器文件系统推荐使用ext4,需要在配置参数中加上--storage-opt dm.fs=ext4

18.5.1.6 有高性能要求的业务的容器场景下建议关闭 seccomp

在做容器网络性能测试时发现 “Docker相对于原生内核namespace性能下降30%”，经分析发现开启Seccomp 后， 系统调用（如：sendto）不会通过system_call_fastpath进行，而走tracesys，带来性能大幅下降。因此，建议在有高性能要求的业务的容器场景下关闭seccomp。

18.5.2 容器管理和镜像管理

18.5.2.1 禁止使用强制删除参数删除容器和镜像

- 禁止使用docker rm －f XXX 删除容器
- 禁止使用docker rmi －f XXX删除镜像

如果使用强制删除，docker rmi和docker rm会忽略过程中的错误，可能导致容器或者镜像关元数据残留。如果使用普通删除，如果删除过程出错，则会删除失败，不会导致元数据残留。

18.5.2.2 load 和 docker rmi 操作

如果同时满足如下两个条件，可能导致并发性问题：

- 某个镜像存在在系统中
- 同时对该镜像进行docker rmi和docker load操作

所以使用时应该避免这种场景（注：所有的镜像创建操作如tag，build，load和rmi并发都有可能会导致类似的错误，应该尽量避免这类操作与rmi的并发。）

18.5.2.3 可能会产生 none 镜像场景

1. none镜像是指，没有tag的最顶层镜像，比如ubuntu的imageID，只有一个tag是ubuntu，如果这个tag没了，但是imageID还在，那么这个imageID就变成了none镜像。
2. Save镜像的过程中因为要把镜像的数据导出来，所以对image进行保护，但是如果这个时候来一个删除操作，可能会untag成功，删除镜像ID失败，（因为被保护了），造成该镜像变成none镜像。
3. 执行docker pull时掉电，或者系统panic，可能出现none镜像，为保证镜像完整性，此时可通过docker rmi 删除镜像后重新拉取。

18.5.2.4 执行 docker save 保存镜像时镜像名为 none

执行docker save保存镜像时，如果指定的名字为镜像ID，则load后的镜像也没有tag，其镜像名为none。

18.5.2.5 docker stop/restart 指定 t 参数且 t<0 时，请确保自己容器的应用会处理 stop 信号

Stop的原理：（Restart会调用Stop流程）

Stop会首先给容器发送Stop 信号（15）

然后等待一定的时间（这个时间就是用户输入的 t）

过了一定时间，如果容器还活着，那么就发送kill信号（9），强杀。

输入参数t的含义：

t<0 : 表示死等，不管多久都等待程序优雅退出，既然用户这么输入了，表示对自己的应用比较放心，认为自己的程序有合理的stop信号的处理机制。

t=0 ： 表示不等，立即发送kill -9 到容器。

t>0 ： 表示等一定的时间，如果容器还未退出，就发送kill -9 到容器。

所以如果用户使用t<0 (比如t=-1)，请确保自己容器的应用会正确处理signal 15. 如果容器忽略了该信号，会导致docker stop一直卡住。

18.5.2.6 使用 sh/bash 等依赖标准输入输出的容器应该使用`-ti`参数，避免出现异常

正常情况：不用`-ti`参数启动sh/bash等进程容器，容器会马上退出

出现这种问题的原因在于，docker会先创建一个匹配用于容器内业务的stdin，在不设置-ti等交互式参数时，docker会在容器启动后关闭该pipe，而业务容器进程sh/bash在检测到stdin被关闭后会直接退出。

异常情况：如果在上述过程中的特定阶段（关闭该pipe之前）强制杀死docker daemon，会导致该pipe的daemon端没有被及时关闭，这样即使不带`-ti`的sh/bash进程也不会退出，导致异常场景，这种容器就需要手动清理。

Daemon重启后会接管原有的容器stream，而不带`-ti`参数的容器可能就无法处理（因为正常情况下这些容器不存在stream需要接管）；真实业务下几乎不存在这种使用方式(不带`-ti`的sh/bash没有任何作用)，为了避免这类问题发生，限制交互类容器应该使用`-ti`参数。

18.5.2.7 不能用 docker restart 重启加了--rm 参数的容器

加了--rm参数的容器在退出时，容器会主动删除，如果重启一个加了--rm的参数的容器，可能会导致一些异常情况，比如启动容器时，同时加了--rm与-ti参数，则不允许对该容器执行restart操作，否则起容器端口可能会概率性卡住无法退出。

18.5.2.8 docker exec 进入容器启动多个进程的注意事项

docker exec 进入容器执行的第一个命令为bash命令时，当退出exec时，要保证在这次exec启动的进程都退出了，再执行exit退出，否则会导致exit退出时，终端可能卡住的情况。如果要在exit退出时，exec中启动的进程仍然在后台保持运行，要在启动进程时加上nohup。

18.5.2.9 启动容器不能单独加-a stdin

启动容器时，不能单独加-a stdin，必须要同时加上-a stdout或者-a stderr，否则会导致终端即使在容器退出后也会卡住。

18.5.2.10 docker rename 和 docker stats <container_name>的使用冲突

如果docker stats <container_name>实时监控容器，当使用docker rename重命名容器后，docker stats中显示的名字将还是原来的名字，不是rename后的名字。

18.5.2.11 build 镜像的同时删除该镜像，有极低概率导致镜像 build 失败

目前的build镜像的过程是通过引用计数来保护的，当build完一个镜像后，紧接着就给该镜像的引用计数加1（holdon操作），一旦holdon操作成功，该镜像就不会被删除了，但是在holdon之前，有极低的概率，还是可以删除成功，导致build镜像失败。

18.5.2.12 docker stop 处于 restarting 状态的容器可能容器不会马上停止

如果一个容器使用了重启规则，当容器处于restarting状态时，docker stop这个容器时有很低的概率会立即返回，容器仍然会在重启规则的作用下再次启动。

18.5.2.13 blkio-weight 参数在支持 blkio 精确控制的内核下不可用

HULK实现了容器内更为精确的blkio控制，该控制需要指定磁盘设备，可以通过docker的--blkio-weight-device参数实现。同时在这种内核下docker不再提供--blkio-weight方式限制容器blkio，使用该参数创建容器将会报错:

docker: Error response from daemon: oci runtime error: container_linux.go:247: starting container process caused

"process_linux.go:398: container init caused \"process_linux.go:369: setting cgroup config for ready process caused

\\\"blkio.weight not supported, use weight_device instead\\\"\""

18.5.2.14 共享 pid namespace 容器，子容器处于 pause 状态会使得父容器 stop 卡住，并影响 docker run 命令执行

使用--pid参数创建共享pid namespace的父子容器，在执行docker stop父容器时，如果子容器中有进程无法退出（比如处于D状态、pause状态），会产生父容器docker stop命令等待的情况，需要手动恢复这些进程，才能正常执行命令。

遇到该问题的时候，请对pause状态的容器使用docker inspect 命令查询 PidMode对应的父容器是否为需要docker stop的容器。如果是该容器，请使用docker unpause将子容器解除pause状态，指令即可继续执行。

一般来说，导致该类问题的可能原因是容器对应的pid namespace由于进程残留导致无法被销毁。如果上述方法无法解决问题，可以采用获取容器内残留进程的方法：

- 获取容器pid namespace id docker inspect --format={{.State.Pid}} CONTAINERID | awk '{print "/proc/"$1"/ns/pid"}' |xargs readlink
- 获取该namespace下的线程 ls -l /proc/*/task/*/ns/pid |grep -F PIDNAMESPACE_ID | awk '{print $9}' |awk -F \/ '{print $5}
- 然后可以通过借助linux工具，确定pid namespace中进程无法退出的原因，解决后容器就可以退出。

18.5.2.15 如果 Docker 操作镜像时系统掉电，可能导致镜像损坏，需要手动恢复

由于Docker在操作镜像（pull/load/rmi/build/combine/commit/import等）时, 镜像数据的操作是异步的、镜像元数据是同步的。所以如果在镜像数据未全部刷到磁盘时掉电。可能导致镜像数据和元数据不一致。对用户的表现是镜像可以看到(有可能是none镜像)，但是无法启动容器，或者启动后的容器有异常。这种情况下应该先使用docker rmi删除该镜像，然后重新进行之前的操作，系统可以恢复。

18.5.2.16 使用--blkio-weight-device 需要磁盘支持 CFQ 调度策略

--blkio-weight-device参数需要磁盘工作于完全公平队列调度（CFQ：Completely Fair Queuing）的策略时才能工作。

通过查看磁盘scheduler文件（/sys/block/<磁盘>/queue/scheduler）可以获知磁盘支持的策略以及当前所采用的策略，如查看sda：

```
# cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

当前sda支持三种调度策略：noop, deadline, cfq，并且正在使用deadline策略。通过echo修改策略为cfq：

```
# echo cfq > /sys/block/sda/queue/scheduler
```

18.5.3 热升级

1. 该功能必须在daemon的启动参数中加上--live-restore，使能该功能后，重启/停止daemon不会使正在运行中的容器自动停止；如果停止daemon时要把所有容器都停止，则需要在停止前使用docker stop $(docker ps -q)。
2. 要使用热升级功能时，用户的自定义参数要放在/etc/docker/daemon.json文件中，比如devicemapper的相关配置

   {

   "storage-opts":[

   "dm.datadev=/dev/mapper/vg--docker-data",

   "dm.metadatadev=/dev/mapper/vg--docker-metadata",

   "dm.fs=ext4",

   "dm.use_deferred_removal=true",

   "dm.use_deferred_deletion=true"

   ]

   }

   如果是放在/etc/sysconfig/docker中，那么在升级的时候会被覆盖的，必须在升级完后重新配置/etc/sysconfig/docker。注意：配置参数放在/etc/docker/daemon.json后，docker daemon的进程的参数是看不到这几个配置参数的。
3. 如果容器的网络是使用的外部插件，那么在热升级过程中插件需要保持运行状态或者插件再次启动后能恢复原来容器的一些网络状态，比如分配给容器的ip地址等。使用docker自带的网络插件无此限制。
4. 在热升级过程中，docker daemon不响应外部API，所有对docker daemon的访问都会返回失败或者卡住，同时docker daemon的进程也会停止，因此如果有心跳机制检测docker daemon的健康状态，在热升级过程中需要停止。
5. 容器的stdio是通过fifo传输的，fifo的buffer大小为1M（每个发行版本可能不一样，Euler OS上为1M），在升级过程中，日志都是缓存在fifo中的，在升级过程中，如果日志缓存超过1M，容器将block住，因此热升级过程中需要保证容器日志不会超出1M，如果明确有容器的日志可能超过1M大小，则可以通过修改/proc/sys/fs/pipe-max-size来增加buffer的大小
6. 在热升级过程中userland-proxy（容器有端口映射时会启动一个userland-proxy）会跟随着daemom一起挂掉, daemon再次启动后会启动一个新的userland-proxy，userland-proxy的作用上是转发主机127.0.0.1的流量到容器内部，因此如果有业务依赖是通过主机的127.0.0.1来访问容器时，会导致短暂的中断。不建议docker开启userland-proxy，存在一些已知的问题，如内存泄露，触发内核bug等，建议在

daemon的启动参数中加上--userland-proxy=false,业务不要通过主机127.0.0.1地址来访问到容器内部。

7. 如果容器设置了重启规则并且在升级过程中容器死了，容器重启规则不会起作用，只有daemon再次启动的时候，重启规则会重新起作用。升级后容器的重启次数会丢失，如果有容器设置了--restart on-failure:N，每次升级后重启次数会重新计算。
8. 用systemd启动docker daemon的时候不能在docker.service文件中加上MountFlags=slave。
9. 禁止同版本直接进行热升级。

18.5.4 加速器特性

1. 测试主体为docker daemon；fakefpga插件只是测试辅助工具，不做为交付件对外交付。如果测试中遇到插件错误，请与开发人员联系修复。
2. “docker accel”子命令仅用于辅助测试，不对外交付，如遇到问题请与开发人员联系修复；
3. fakefpga插件只能运行一个实例，应避免启动多个实例。如果启动多实例仅最后一个实例生效，之前启动的实例无法继续与docker daemon通信，即使最后一个实例被kill；
4. fakefpga插件被kill后，在docker端进行加速器相关操作会有16s左右的等待时间，这是由于docker正在尝试重新连接插件，该行为是docker插件机制的默认行为，并未性能故障。
5. 加速器名字最大长度为256个字符，创建加速器时参数最大个数为128，每个参数最长字符数为1024；获取所有加速器资源时指定的过滤器规则最大字符数为1024个字符。

18.5.5 网络

1. Docker daemon使用--bip参数指定docker0网桥的网段之后，在下一次重启的时候去掉--bip参数，docker0网桥会沿用上一次的—bip配置，即使重启之前已经删除docker0网桥。原因是docker会保存网络配置并在下一次重启的时候默认恢复上一次配置。
2. Docker network create 并发创建网络的时候，可以创建具有相同名字的两个网络。原因是docker network是通过id来区分的，name只是个便于识别的别名而已，不保证唯一性。
3. 在做容器网络性能测试时发现 “Docker相对于原生内核namespace性能下降30%”，经分析发现开启Seccomp 后，
4. （如：sendto）不会通过system_call_fastpath进行，而走tracesys，带来性能大幅下降。因此，建议在有高性能要求的业务的容器场景下关闭seccomp。

18.5.6 容器卷

1. 启动容器时如果通过`-v`参数将主机上的文件挂载到容器中，在主机或容器中使用vi或sed命令修改文件可能会使文件inode发生改变，从而导致主机和容器内的文件不同步。容器中挂载文件时应该尽量避免使用这种文件挂载的方式（或不与vi和sed同时使用），也可以通过挂载文件上层目录来避免该问题。

18.5.7 镜像组合

18.5.7.1 使用--no-parent 参数并发 build 镜像时可能会残留镜像

使用—no-parent参数build镜像时，如果有多个build操作同时进行，并且Dockerfile里FROM的镜像相同，则可能会残留镜像，分为以下两种情况：

1. FROM的镜像不是完整镜像，则有可能会残留FROM的镜像运行时生成的镜像。残留的镜像名类似base_v1.0.0-app_v2.0.0，或者残留<none>镜像。
2. 如果Dockerfile里的前几条指令相同，则有可能会残留<none>镜像。

18.5.8 强杀 docker 服务进程可能导致的后果

docker的调用链很长，强杀daemon可能会导致一些数据状态不一致等问题，本章节列举一些强杀可能导致的问题。

使用 devicemapper 作为 graphdriver 时，强杀可能导致信号量残留

docker在操作dm的过程中会创建信号量，如果在释放信号量前，daemon被强杀，可能导致该信号量无法释放，一次强杀最多泄露一个信号量，泄露概率低。而linux系统有信号量上限限制，EulerOS上默认128，当信号量泄露次数达到128将无法创建新的信号量，进而导致docker daemon启动失败。排查方法如下：

1. 首先查看系统上残留的信号量：
```
$ ipcs
------ Message Queues --------
key        msqid      owner      perms      used-bytes   messages
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x0d4d3358 238977024  root       600        1
0x0d4d0ec9 270172161  root       600        1
0x0d4dc02e 281640962  root       600        1
```
2. 接着用dmsetup查看devicemapper创建的信号量，该信号量集合是上一步中查看到的系统信号 量的子集：
```
$ dmsetup udevcookies
Cookie       Semid      Value      Last semop time          Last change time
```
3. 最后查看内核信号量设置上限，第四个值就是当前系统的信号量使用上限：
```
$ cat /proc/sys/kernel/sem
250     32000   32      128
```

如果**步骤1**中残留的信号量数量与**步骤3**中看到的信号量上限相等，则是达到上限，此时docker daemon无法正常启动。可以使用下述命令增加信号量使用上限值来让docker恢复启动：

```
$ echo 250 32000  32  1024 > /proc/sys/kernel/sem
```

也可以手动清理devicemapper残留的信号量（下面是清理一分钟以前申请的dm相关信号量）：

```
$ dmsetup udevcomplete_all 1
This operation will destroy all semaphores older than 1 minutes with keys that have a prefix 3405 (0xd4d).
Do you really want to continue? [y/n]: y
0 semaphores with keys prefixed by 3405 (0xd4d) destroyed. 0 skipped.
```

使用 bridge 模式启动容器的过程中，强杀 daemon 可能导致网卡残留

使用bridge网络模式，当docker创建容器时，会先在host上创建一对veth，然后再把该网卡信息存到数 据库中，如果在创建完成，存到docker的数据库之前，daemon被强杀，

那么该网卡无法被docker跟踪， 下次启动也无法删除（docker本身会清理自己数据库中不用的网卡），从而造成网卡残留。

18.5.9 其他

18.5.9.1 避免使用的选项

1. －privileged 选项会让容器获得所有权限，容器可以做挂载操作和修改/proc, /sys等目录，可能会对host造成影响，普通容器需要避免使用该选项。
2. 共享host的namespace，比如 －pid host/--ipc host/--net host等选项可以让容器跟host共享命名空间，同样会导致容器影响host的结果，需要避免使用。

18.5.9.2 docker cp 拷贝大文件问题

使用docker cp向容器里面拷贝文件时，docker ps和对这个容器的所有操作都将等待docker cp结束后才能进行。

18.5.9.3 使用 overlay2/overlay 时修改镜像中大文件问题

使用overlay2/overlay为graphdriver时，在容器中第一次修改镜像中的文件时，如果该文件的大小大于系统剩余的空间，修改将会失败，因为即使修改很小，也要把这个文件完整的copy up到上层，剩余空间不足导致失败。

18.5.9.4 devicemapper 磁盘限额功能

使用devicemapper的磁盘限额功能时，设置的磁盘空间只能大于base的大小，base大小可以通过docker info查看，如下图所示：

```
$ docker info
Containers: 27
 Running: 0
 Paused: 0
 Stopped: 27
Images: 1
Server Version: 1.11.2
Storage Driver: devicemapper
 Pool Name: docker-253:1-134475223-pool
 Pool Blocksize: 65.54 kB
 Base Device Size: 10.74 GB
 Backing Filesystem: xfs
 Data file: /dev/loop0
 Metadata file: /dev/loop1
 Data Space Used: 3.95 GB
 Data Space Total: 107.4 GB
 Data Space Available: 14.42 GB
 Metadata Space Used: 2.728 MB
 Metadata Space Total: 2.147 GB
 Metadata Space Available: 2.145 GB
```


18.5.9.5 系统掉电注意事项

当系统出现意外掉电（或系统panic，总之是系统无法及时做磁盘刷新的危险操作）情况时，可能导致docker daemon的部分状态尚未刷新到磁盘导致重启后docker daemon状态不正常，可能出现的问题有（包括但不限于）：

- 掉电前创建了容器，重启后docker ps －a看不到；该问题是因为该容器的状态文件没有刷新到磁盘，从而导致重启后daemon无法获取到该容器状态导致。（其他状态如镜像、卷、网络等也可能会有类似问题）
- 掉电前某个文件正处于写入状态，尚未完全写入，重启后daemon重新加载该文件发现文件格式不正常或内容不完整，导致的错误。

18.5.9.6 docker cp 文件权限问题

容器以非root用户运行，当使用docker cp命令复制主机上一个非root权限的文件到容器时，文件在容器中的权限角色会变成root。docker cp与cp命令不同，docker cp会修改复制到容器中文件的uid和gid为root。

18.5.9.7 hook 执行时间问题

使用hook时，执行时间应尽量短。如果hook中的prestart时间过长（超过2分钟），则会导致容器启动超 时失败，如果hook中的poststop时间过长（超过2分钟），也会导致容器异常。目前已知的异常如下：执行docker stop命令停止容器时，2分钟超时执行清理时，由于hook还没执行结束，因此会等待hook执行结束（该过程持有锁），从而导致和该容器相关的操作都会卡住（例如docker ps -a命令），需要等到hook执行结束才能恢复。另外，由于docker stop命令的2分钟超时处理是异步的过程，因此即使docker stop命令返回了成功，容器的状态也依然是up状态，需要等到hook执行完后状态才会修改成exit。

18.5.9.8 容器状态检测与使用

在任何情况下，容器的状态应该以docker inspect或者docker ps查询出的结果为准，上层应用如果依赖容器的状态，则不应该以docker stop是否成功返回为判断标准。

18.5.9.9 避免使用 docker kill 命令

docker kill命令发送相关信号给容器内业务进程，依赖于容器内业务进程对信号处理注册，导致信号执行与预期不符合的情况，应该避免使用docker kill命令。

18.5.9.10 并发执行命令数量问题

docker内部的消息缓冲有一个上限，超过这个上限就会将消息丢弃，因此在并发执行命令时建议不要超过1000条命令，否则有可能会造成docker内部消息丢失，从而造成容器无法启动等严重问题。

18.5.9.11 iptables 默认规则对 docker 的影响

docker使用--icc=false选项时，可以限制容器之间互通，但若os自带某些规则，可以造成限制容器之间互通失效。因此，在容器os中使用docker，如果需要使用-icc=false选项时，建议先在host上清理一下iptables的相关规则。例如：

```
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
...
```



```
0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0
...
0     0 DROP       all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0
...
```

在Chain FORWARD中，DROP上面多出了一条ACCEP icmp的规则，造成加了--icc=false后，容器之间也能ping能，但udp/tcp仍然是不通的。

18.5.9.12 docker run --hook-spec 注意事项

docker run --hook-spec可以指定hook配置文件(.json)来配置容器在3个特殊阶段(prestart、poststart、poststop)来执行自定义的二进制。

需要注意当有多个hook配置文件，要运行多个hook时，用户必须自己手工将多个hook配置文件组合成一个配置文件，使用--hook-spec参数指定此合并后的配置文件，方可生效所有的hook；如果配置多个--hook-spec参数，则只有最后一个生效。

配置举例：

hook1.json内容如下：

```
# cat /var/lib/docker/hooks/hookspec.json
{
"prestart": [
{
"path": "/var/lib/docker/hooks/lxcfs-hook",
"args": ["lxcfs-hook", "--log", "/var/log/lxcfs-hook.log"],
"env": []
}
],
"poststart":[],
"poststop":[]
}
```

hook2.json如下：

```
# cat /etc/docker-tools/hookspec.json
{
"prestart": [
{
"path": "/docker-root/hooks/docker-hooks",
"args": ["docker-hooks", "--state", "prestart"],
"env": []
}
],
"poststart":[],
"poststop":[
{
"path": "/docker-root/hooks/docker-hooks",
"args": ["docker-hooks", "--state", "poststop"],
"env": []
}
]
}
```

手工合并后的json如下：

```
{
"prestart": [
{
"path": "/var/lib/docker/hooks/lxcfs-hook",
"args": ["lxcfs-hook", "--log", "/var/log/lxcfs-hook.log"],
"env": []
},
{
```



```
"path": "/docker-root/hooks/docker-hooks",
"args": ["docker-hooks", "--state", "prestart"],
"env": []
}
],
"poststart":[],
"poststop":[
{
"path": "/docker-root/hooks/docker-hooks",
"args": ["docker-hooks", "--state", "poststop"],
"env": []
}
]
}
{
"prestart": [
{
"path": "/docker-root/hooks/docker-hooks",
"args": ["docker-hooks", "--state", "prestart"],
"env": []
}
],
"poststart":[],
"poststop":[
{
"path": "/docker-root/hooks/docker-hooks",
"args": ["docker-hooks", "--state", "poststop"],
"env": []
}
]
}
```

需要注意的是，用户需要识别多个hook之间的依赖关系，如果有依赖关系，在组合hook配置文件时要根据依赖关系灵活调整顺序。

docker-engine会按照数组顺序依次读取hook配置文件中prestart等action中hook二进制，进行执行动作。

19 EulerOS LiveCD 使用说明

EulerOS LiveCD 是一个从光驱/U盘启动运行小型的EulerOS。LiveCD为内存文件系统，重启后针对系统的修改将被还原。

说明

现有LiveCD没有加入日志切割转储功能，日志文件会不断变大，如果系统需要长时间运行，需自行加入日志的切割转储功能。

19.1 获取 EulerOS LiveCD

访问**https://cmc.rnd.huawei.com/cmcversion/**，找到EulerOS V200R007 版本，获取Software/x86_64/EulerOS_minios_livecd.iso 镜像，同时获取校验文件EulerOS_minios_livecd.iso.sha25，即可下载iso，使用 sha256sum 命令校验完整性。

说明

获取发布包后，需要进行完整性校验，也可以使用PGP数字签名进行验证，详细的验证操作说明请参见“安全资料—发布包完整性校验”章节。

19.2 使用 EulerOS LiveCD

使用光驱挂载 EulerOS_minios_livecd.iso，并在 BISO 中配置从光驱启动，启动完成即可进入EulerOS LiveCD。

- 用户名：root
- 密 码：Huawei@SYS3

  说明

  建议用户首次登录后修改密码，并定期更新，避免密码泄露后，产生安全风险。

EulerOS LiveCD 支持 BIOS 以UEFI 或 legacy 模式启动。

19.3 EulerOS LiveCD 烧录到 U 盘方法

有两种方法可以把EulerOS LiveCD烧录到U盘。

- 方法一：烧录到U盘后，U盘为只读状态。
  a. EulerOS LiveCD 从光驱启动。
  b. 插入U盘（如果服务器自带U盘，则忽略该步骤）。
  c. 找到 U盘盘符，比如： /dev/sdb。
  d. 使用 dd 命令把光驱设备或 iso 烧录到U盘：
  ```
  dd   if=/dev/sr0  of=/dev/sdb
  ```
  或者
  ```
  dd   if=EulerOS_minios_livecd.iso  of=/dev/sdb
  ```
  e. 重启系统，并在 BISO 中设置从U盘启动即可。
- 方法二：烧录到U盘后，U盘可以读写挂载，可以参考**19.5 修改EulerOS LiveCD**直接更新LiveCD。
  a. 假设 /dev/sda 为服务器识别到的U盘，创建分区表（legacy只支持msdos分区，UEFI支持msdos、gpt分区）。
  ```
  [root@localhost ~]# parted  /dev/sda -- mktable msdos
  Warning: The existing disk label  on /dev/sda will be destroyed and all data on this
  disk will be lost. Do you  want to continue?
  Yes/No? yes
  Information: You may need to update  /etc/fstab.
  ```
  b. 添加分区。
  ```
  [root@localhost ~]# parted /dev/sda -- mkpart primary fat32 0% 100%
  Information: You may need to update /etc/fstab.
  ```
  c. 格式化并设置label。
  ```
  [root@localhost ~]# mkfs.fat -n "EULEROS" /dev/sda1
  mkfs.fat 3.0.20 (12 Jun 2013)
  [root@localhost ~]# parted /dev/sda -- set 1 boot on
  Information: You may need to update /etc/fstab.

  [root@localhost ~]# parted /dev/sda -- set 1 lba off
  Information: You may need to update /etc/fstab.
  ```
  说明
  -n 设置分区label，字母只支持大写（小写会被自动转为大写），blkid显示如下：
  ```
  [root@localhost ~]# blkid /dev/sda1
  /dev/sda1: LABEL="EULEROS" UUID="6325-45F0" TYPE="vfat"
  ```
  d. 重写mbr。
  ```
  [root@localhost ~]# dd if=/usr/share/syslinux/mbr.bin of=/dev/sda bs=440 count=1
  1+0 records in
  1+0 records out
  440 bytes (440 B) copied, 0.00201452 s, 218 kB/s
  ```
  e. copy光驱中的文件到 /dev/sda1。
  ```
  [root@localhost ~]# mount /dev/sr0 /mnt/
  mount: /dev/sr0 is write-protected, mounting read-only
  [root@localhost ~]# mount /dev/sda1 /media/
  [root@localhost ~]# cp /mnt/* /media/ -afr
  ```
  f. 修改相关配置文件。
  ```
  [root@localhost ~]# cd /media/
  [root@localhost media]# mv isolinux syslinux
  [root@localhost media]# rm syslinux/isolinux.bin
  ```



```
rm: remove regular file ‘syslinux/isolinux.bin’ ? y
[root@localhost media]# mv syslinux/isolinux.cfg syslinux/syslinux.cfg
[root@localhost media]# syslinux -d /syslinux/ -f /dev/sda1
[root@localhost media]# sed -i "s/LABEL=[^ ]*/LABEL=EULEROS/g" syslinux/syslinux.cfg
[root@localhost media]# sed -i "s/LABEL=[^ ]*/LABEL=EULEROS/g" EFI/BOOT/grub.cfg
```

g. 重启系统，从u盘启动即可。

19.4 裁剪定制 EulerOS LiveCD

裁剪定制一个新的EulerOS LiveCD，步骤如下：

1. 获取 EulerOS 标准 iso（EulerOS-V2.0SP5-x86_64-dvd.iso），获取方法参考**19.1 获取EulerOS LiveCD**。
2. 准备一台EulerOS2.0SP5的环境，并把 EulerOS-V2.0SP5-x86_64-dvd.iso 挂载出来，比如：
   ```
   mount EulerOS-V2.0SP5-x86_64-dvd.iso /media
   ```
3. 安装制作 LiveCD 工具。
   ```
   yum install -y lorax anaconda yum-langpacks libselinux-utils
   ```
   使用yum请先配置 yum repo。
4. 定制一个 kickstart 配置文件，模板如下：
   ```
   # Minimal Disk Image
   #
   # Firewall configuration
   firewall --enabled
   # Use network installation
   url --url="file:///media"

   # Root password
   rootpw --iscrypted
   $6$tmWZYrc3$Eyo4oYekTBYDrUOOkyw2vFWuJaqp4mY3cJro9qBdXl24kiarFENlIkcNYHmEtf/
   AO3cTqFjVitmRZcXajOpUO0

   # Network information
   network  --bootproto=dhcp --onboot=on --activate
   # System keyboard
   keyboard --xlayouts=us --vckeymap=us
   # System language
   lang en_US.UTF-8
   # SELinux configuration
   selinux --enforcing
   # Installation logging level
   logging --level=info
   # Shutdown after installation
   shutdown
   # System timezone
   timezone Asia/Beijing
   # System bootloader configuration
   bootloader --location=mbr
   # Clear the Master Boot Record
   zerombr
   # Partition clearing information
   clearpart --all
   # Disk partitioning information
   part / --fstype="ext4" --size=4000
   part swap --size=1000

   %post

   touch /etc/sysconfig/network

   cat << EOF > /etc/sysconfig/network-scripts/ifcfg-eth0
   TYPE=Ethernet
   BOOTPROTO=dhcp
   ```



```
NAME=eth0
DEVICE=eth0
ONBOOT=yes
EOF

rm -rf /etc/systemd/system/multi-user.target.wants/kbox.service
rm -rf /etc/systemd/system/multi-user.target.wants/kdump.service
rm -rf /usr/lib/systemd/system/kbox.service
rm -rf /usr/lib/systemd/system/kdump.service

%end

%packages --nobase --excludedocs
@core --nodefaults
@x11
xulrunner.i686
libXtst.i686
gtk2.x86_64
java-1.8.0-openjdk
strace
vim-minimal
openssh-server
setup
passwd
findutils
sudo
util-linux
net-tools
iproute
rsyslog
rootfiles
openssh-clients
dhclient
yum-plugin-priorities.noarch
euleros-latest-release
kernel
memtest86+
grub2-efi
grub2-efi-x64-cdboot
grub2
shim
syslinux
smartmontools
pciutils
pciutils-libs
libpciaccess
lsscsi
libnl
lftp
zip
unzip
dosfstools
btrfs-progs
genisoimage
nfs-utils
iscsi-initiator-utils
libiscsi
dhcp
-dracut-config-rescue
-kbox-kmod
-kdump
-dump_mem_tool
%end
```

5. 根据kickstart配置文件制作 EulerOS LiveCD。

```
setenforce 0     //关闭 selinux
livemedia-creator --make-iso --ks=`pwd`/euleros-livecd.ks --no-virt --project EulerOS --
releasever V2.0SP5 --tmp `pwd`/tmp
```


这里需要等待 10-20 分钟。制作完成之后，LiveCD 存放到 `pwd`/tmp/tmp***SdXw5A***/images/boot.iso。

说明

1. 步骤**5**的LiveCD存放路径，“tmpSdXw5A”中“SdXw5A”为随机生成的字符串，每次制作LiveCD目录会有差异。
2. 在制作LiveCD之前，可以通过修改如下默认配置，来修改LiveCD默认启动菜单、内核命令行等。

```
HGH1000041148:/usr/share/lorax/live/config_files/x86 # tree
.
├── boot.msg
├── grub2-efi.cfg
├── grub2-efi-lockdown.cfg
├── grub.conf
└── isolinux.cfg
```

19.5 修改 EulerOS LiveCD

本节主要介绍修改 EulerOS LiveCD，并重新打包的方法。

LiveCD 所有文件如下：

```
HGH1000041148:/data/testuser/livecd/result # tree
.
├── EFI
│   └── BOOT
│       ├── BOOTX64.EFI            #efi 启动文件，不可修改
│       ├── fonts                    #启动过程，显示文字编解码，不可修改
│       │   └── unicode.pf2
│       ├── grub.cfg                 #efi 启动使用的 grub.cfg 配置文件
│       ├── grubx64.efi              #efi 启动文件，不可修改
│       └── MokManager.efi           #efi 启动文件，不可修改
├── images
│   ├── efiboot.img                  #efi 启动文件，不可修改
│   ├── install.img                    #安装使用，不可修改
│   └── pxeboot                        #pxe启动使用
│       ├── initrd.img
│       └── vmlinuz
├── isolinux
│   ├── boot.msg
│   ├── grub.conf
│   ├── initrd.img
│   ├── isolinux.bin
│   ├── isolinux.cfg                     #legacy启动使用 grub 配置文件
│   ├── memtest
│   ├── splash.png
│   ├── vesamenu.c32
│   └── vmlinuz                      #内核文件
└── LiveOS
    └── squashfs.img                 #根文件系统压缩为squashfs格式
```

挂载 squashfs.img 如下：

```
# mkdir dir
# mount squashfs.img dir/
# tree dir/
dir/
└── LiveOS
    └── rootfs.img              #挂载后可读写的根文件系统
```

由于iso/squashfs 都是只读的文件格式，所以必须重新制作iso与 squashfs。

1. 挂载 LiveCD，并把所有文件copy出来。

```
mount  EulerOS_minios_livecd.iso  /mnt
mkdir  iso
cp /mnt/*  iso/  -rf
```

2. 把squashfs.img 挂载出来，并copy所有文件到新目录。

```
mkdir  old  new
mount  iso/LiveOS/squashfs.img  old
cp old/*  new/ -rf
```

3. 把rootfs.img挂载出来。

```
mkdir rootdir
mount new/LiveOS/rootfs.img  rootdir
```

4. 在rootdir目录中，用户根据需要做相应的修改。
5. 修改完成之后重新制作squashfs.img，并覆盖原先的 squashfs.img。

```
umount rootdir
mksquashfs new/ squashfs.img -comp xz
umount   old
cp  squashfs.img  iso/LiveOS/squashfs.img
```

6. 重新压缩 iso。
   - 压缩iso命令如下：

     ```
     mkisofs -o EulerOS_LiveCD_new.iso -b isolinux/isolinux.bin -c isolinux/boot.cat -boot-load-size 4 -boot-info-table -no-emul-boot -eltorito-alt-boot -e images/efiboot.img -no-emul-boot -R -J -V "EulerOS V2.0SP5 x86_64" -T -graft-points isolinux=`pwd`/iso/isolinux images/pxeboot=`pwd`/iso/images/pxeboot LiveOS=`pwd`/iso/LiveOS EFI/BOOT=`pwd`/iso/EFI/BOOT images/efiboot.img=`pwd`/iso/images/efiboot.img
     ```

   - 设置hybrid模式，用来支持烧录U盘引导。

     ```
     isohybrid --uefi EulerOS_LiveCD_new.iso
     ```

   - 添加完整性校验码。

     ```
     implantisomd5  EulerOS_LiveCD_new.iso
     ```

19.6 处理空链接文件

EulerOS LiveCD中存在无指向的空链接文件，可能会被恶意用户利用，影响系统安全性。

实现

步骤1 通过如下命令查找系统中的空链接文件。

```
find dir -type l -follow 2>/dev/null
```

说明

- dir为搜索目录的名称，通常需要关注系统关键目录：/bin、/boot、/usr、/lib64、/lib、/var等。
- EulerOS LiveCD默认发布包中存在的无指向的空链接文件为：/etc/localtime -> ../usr/share/zoneinfo/UTC。

步骤2 如果此类文件为无实际作用，可通过如下命令删除。

```
rm -f fileX
```

说明

fileX即为**步骤1**找出的文件名。

----结束

特殊场景

在EulerOS上可能存在由开源组件装出来的空链接（安装的组件数量不同，空链接文件也不同），这些空链接可能会有用途（有些空链接文件是预制的，其他组件可能会有

依赖），底层OS无法确定处理方式，需要在最终环境上由产品处理，处理方式参考**实现**的指导步骤。

例如，EulerOS遵循开源实现，在支持UEFI和legacy BIOS两种安装模式时，和CentOS保持一致，两种引导场景支持的grub相关包默认都安装，会导致当用户确定选择legacy BIOS模式安装时，形成空链接文件“/etc/grub2-efi.cfg”；当用户确定选择UEFI模式安装时，会形成空链接文件“/etc/grub2.cfg”，需要产品在安装完成后由产品自行处理空链接。

20 bbr 算法说明

20.1 简介

bbr算法是一种TCP拥塞控制算法，主要是根据RTT和带宽的变化来调节窗口，进而控制链路带宽，应对网络可能拥塞的情况，其目的是提升TCP的带宽抢占能力。关键技术有：

1. 依赖数据调整滑动窗口：通过分析RTT、实际带宽调整滑动窗口，忽略丢包率，可以准确区分噪声丢包和拥塞丢包，避免假拥塞。
2. RACK判断重传：取代RTO超时，保证丢包、乱序时数据平滑发送。
3. FQ公平队列：根据发送速率控制报文发送，避免波动。

20.2 约束限制

tcpbbr算法在无延时，无丢包的环境，性能比tcp默认算法cubic稍差，会差5%左右。

20.3 使用指导

步骤1 安装tcp_bbr模块

```
modprobe tcp_bbr
```

步骤2 修改tcp拥塞控制算法为bbr

```
sysctl -w net.ipv4.tcp_congestion_control=bbr
```

步骤3 配套修改内核的FQ功能

- 全局生效。在/etc/sysctl.conf配置文件中添加如下选项，全局默认FQ生效。
  ```
  net.core.default_qdisc=fq
  ```
- 针对特定网卡配置FQ。 如果不想FQ全局默认生效，可以针对特定的网卡进行配置FQ。

  插入fch_fq.ko


```
insmod ./kernel/net/sched/sch_fq.ko
```

打开对应网卡的FQ功能，例如eth2：

```
tc qdisc add dev eth2 root fq
```

查看eth2的fq状态：

```
tc qdisc show dev eth2
```

----结束

21 PVSCSI 说明

概述

物理设备映射（PDM）为虚拟机提供了一种机制来直接访问物理存储子系统（仅限光纤通道或iSCSI）上的LUN，因此虚拟机中的业务能够直接访问存储设备或直接对存储设备下发控制命令。

通过使用物理设备映射，可以让虚拟机识别SCSI磁盘，实现在虚拟机内部下发SCSI命令，交给主机然后透传给存储设备进行处理，最后将应答返回。

物理设备映射，作为一个整体特性，将LUN映射到虚拟机中，真正实现物理设备映射特性完整功能，PVSCSI是其最重要的关键技术。

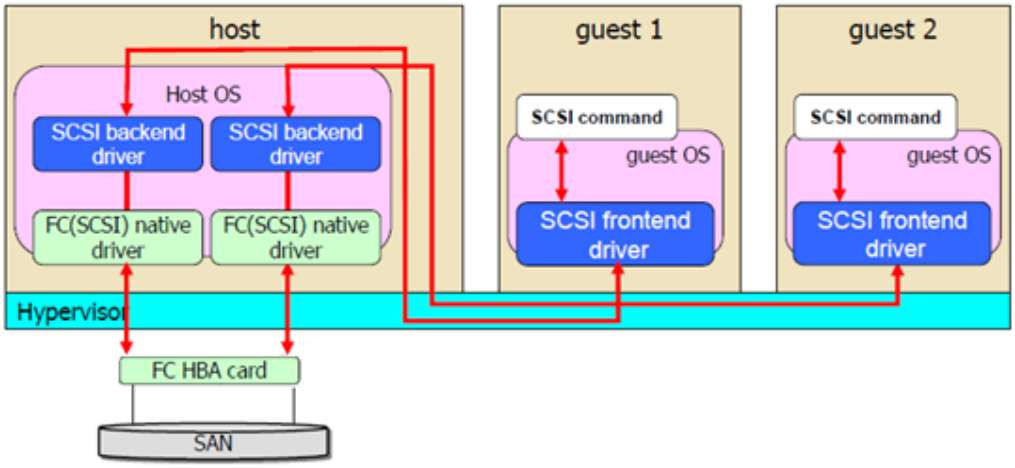
注意事项

- PVSCSI不支持磁盘分区和磁盘上的逻辑卷，只支持整个磁盘或者整个LUN的映射。
- PVSCSI仅支持可以导出SCSI序列号的块设备或RAID设备。
- PVSCSI盘只能被作为数据盘使用。
- 由于PVSCSI呈现给虚拟机的设备id与盘符（如sda、sdb）的对应关系，在虚拟机重启后存在变化的可能，所以在虚拟机中建议指定设备ID来使用。
- 带PVSCSI磁盘的虚拟机内部使用SCSI预留功能后，该功能只会在虚拟机所在的主机生效。虚拟机热迁移到其他主机后，SCSI预留功能并不会随虚拟机迁移而自动适配，会使虚拟机内部访问设置了SCSI预留的磁盘时出现只读等问题。因此不建议对使用SCSI预留功能的PVSCSI虚拟机进行热迁移。

原理

PVSCSI（准虚拟SCSI）技术，可以分为两部分。一部分为SCSI前端驱动，它通过PV Driver安装在虚拟机内部。另一部分为SCSI后端驱动，作为linux内核模块运行在主机系统中。PVSCSI原理流程图如**图21-1**所示。

图 21-1 PVSCSI 原理流程图



应用场景

用户划分好独立的LUN，然后配置好带SCSI磁盘的配置文件，在虚拟机启动或者挂载磁盘的过程中，给虚拟机配置SCSI磁盘，让虚拟机能够使用SCSI磁盘。SCSI磁盘只能配置成虚拟机的数据盘。

22 IO 持久化映射

概述

EulerOS虚拟机在存储IO性能测试中，在单个节点上同时对多个虚拟磁盘（4个及以上）进行IO性能测试，因为虚拟磁盘的IO解映射操作比较耗时导致IO性能较差，用户可通过开启IO持久化映射功能来提升IO性能。IO持久化映射功能主要用于提升单个节点上多虚拟磁盘场景下的IO性能。

约束限制

- 在开启IO环适配情况下，开启IO持久化映射后，虚拟机每个磁盘额外增加大约65M的虚拟机内存消耗，建议增大虚拟机的内存配置，增大内存量的计算方法为磁盘个数*65M，否则会由于内存不足导致虚拟机启动失败或者热插磁盘失败等。
- 只有安装了UVP V100R003C10SPC800B035及以后的xen host版本，才能使用到IO持久化映射功能。
- IO持久化映射功能建议在多虚拟磁盘（1个虚拟机挂载4个及以上的虚拟磁盘）或多虚拟机（4个及以上虚拟机，每个虚拟机挂载1个或多个虚拟磁盘）场景中开启，而且只对读IO有效。否则提升效果不明显，甚至无提升效果。

规格说明

开启IO持久化映射的情况下，IO性能与不开启时相比不能下降，提升比例与IO模型强相关。

设置 IO 持久化映射

虚拟机配置文件中，添加io_persistent='on'配置，示例如下：

```
<disk type='file' device='disk'>
<driver name='tap2' type='vhd' io_persistent='on'/>
<source file='/vims/usr-10G'/>
<target dev='xvda' bus='xen'/>
</disk>
```


23 多队列（Multi-Queue）

EulerOS V2.0SP5内核支持blk-mq。

23.1 使能 scsi-mq

如果需要使能scsi-mq，需要添加scsi_mod.use_blk_mq=y dm_mod.use_blk_mq=y到内核启动参数，提升盘读写性能。

修改方法：

步骤1 打开/boot/grub2/grub.cfg，添加如下红色框中的参数。

```
menuentry 'EulerOS (3.10.0-514.35.4.7.h44.x86_64) 2.0 (SP3)' --unrestricted {
        load_video
        insmod gzio
        insmod part_msdos
        insmod part_gpt
        insmod fat
        echo 'Loading EulerOS (3.10.0-514.35.4.7.h44.x86_64) 2.0 (SP3)'
        linux   /vmlinuz-3.10.0-514.35.4.7.h44.x86_64 root=/dev/cpsVG/rootfs oop
s=panic net.ifnames=0 biosdevname=0 noexec=on 8250.nr_uarts=8 efi=old_map nmi_wa
tchdog=1 intel_iommu=off selinux=0 pci=realloc console=tty0 console=ttyS0,115200
 nohz=off highres=on hpet=enable reserve_kbox_mem=16M crashkernel=334M@48M panic
=3 crash_kexec_post_notifiers audit=0 coredump_filter=0x33f elevator=cfq read_ah
ead_kb=512 nopku LANG=en_US.UTF-8 scsi_mod.use_blk_mq=y dm_mod.use_blk_mq=y
```

步骤2 重启操作系统。

步骤3 查看启动后的内核启动参数，是否包含第一步添加的参数。

```
linux-dTRVTs:/ # cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-3.10.0-514.35.4.7.h44.x86_64 root=/dev/cpsVG/rootfs oops=pan
ic net.ifnames=0 biosdevname=0 noexec=on 8250.nr_uarts=8 efi=old_map nmi_watchdo
g=1 intel_iommu=off selinux=0 pci=realloc console=tty0 console=ttyS0,115200 nohz
=off highres=on hpet=enable reserve_kbox_mem=16M crashkernel=334M@48M panic=3 cr
ash_kexec_post_notifiers audit=0 coredump_filter=0x33f elevator=cfq read_ahead_k
b=512 nopku LANG=en_US.UTF-8 scsi_mod.use_blk_mq=y dm_mod.use_blk_mq=y
```


步骤4 查看参数生效。

```
linux-dTRVTs:/ # cat /sys/module/scsi_mod/parameters/use_blk_mq
Y
```

----结束

说明

EulerOS虚拟机支持virtio-scsi多队列，虚拟机的配置请参考“配置vhost-user-scsi虚拟磁盘”章节，注意队列数必须和虚拟机cpu个数相同，否则影响虚拟机性能。

23.2 配置 vhost-user 虚拟磁盘

vhost-user 简介

随着高性能的存储介质的发展，传统的存储系统无法完全发挥高性能存储介质的性能，或者消耗越来越多的CPU资源。在虚拟化场景下，使用qemu virtio后端访问NVME等高性能存储，虚拟磁盘能达到的性能上限和CPU消耗都不能完全满足使用要求。

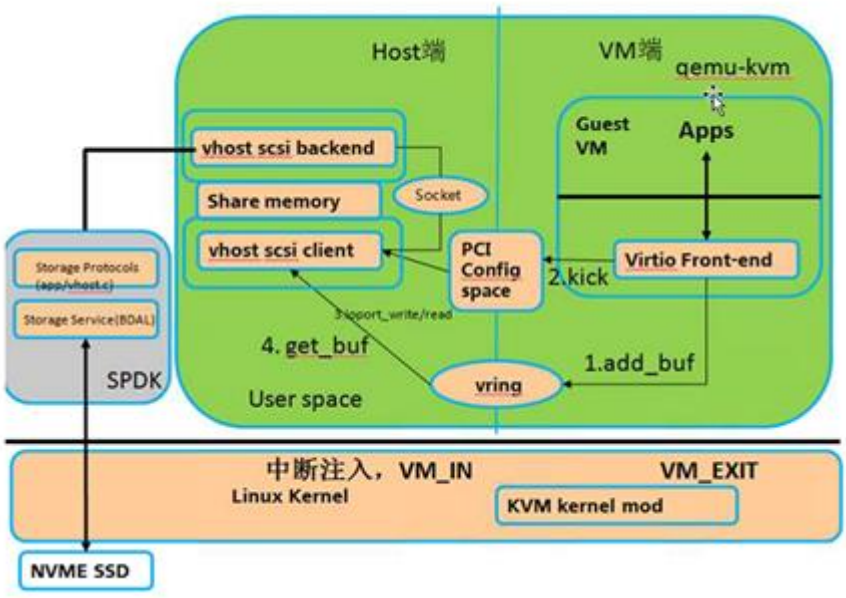
UVP基于SPDK技术，提供vhost-user类型的虚拟磁盘，使用虚拟机virtio-scsi或virtio-blk前端，在后端用户态直接操作底层硬件，相比于Linux传统的IO路径，减少了用户态和内核态的切换、内核VFS、块层以及中断开销，大幅度缩短IO路径，减少CPU消耗，支撑云存储关键业务满足高带宽和低时延的性能要求。

应用场景

公有云小规格高性能存储实例。

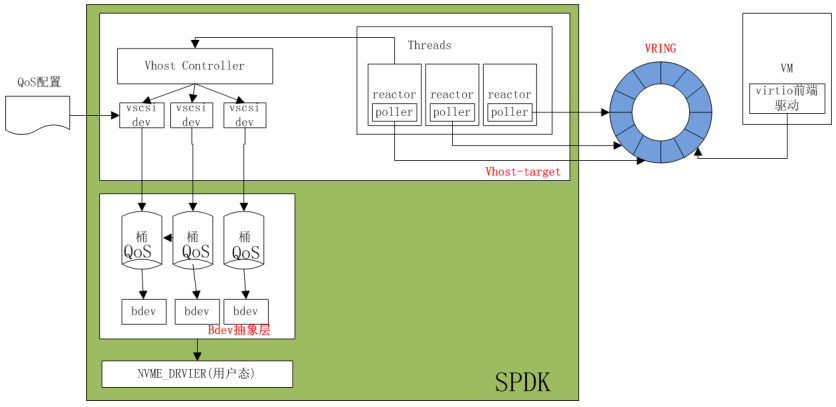
原理

以vhost-user-scsi磁盘为例，vhost-user的原理如下图所示：



- 在Host用户态运行vhost(SPDK)进程，通过用户态驱动高性能地访问NVME硬件。

- 虚拟机启动时，在虚拟机XML中配置virtio-user-scsi控制器，和SPDK进程交互，完成初始化，使得vhost进程能够访问虚拟机内存。
- 虚拟机IO下发时，vhost进程通过轮询方式访问虚拟磁盘共享环内存以及NVME设备内存，完成IO请求转发以及结束。



支持的 Guest 操作系统

EulerOS 2.5（需要在/boot/grub2/grub.cfg中添加scsi_mod.use_blk_mq=1开启设备多队列功能）

请参见《华为客户机操作系统兼容性指南（KVM公有云）》中的客户机操作系统兼容性情况。

23.2.1 vhost 进程管理

环境准备

- 启动vhost进程时需保证系统至少有1G的空闲大页内存，确定方法：

```
linux-jCSKvs:/ # cat /proc/meminfo | grep Huge
AnonHugePages:    729088 kB
HugePages_Total:   10240
HugePages_Free:    10240
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

空闲大页内存大小等于HugePages_Free * Hugepagesize。

申请大页内存的方法有：

- 动态增加系统大页

```
echo N > /proc/sys/vm/nr_hugepages
```

根据Hugepagesize增加合适的nr_hugepages值。

- 通过setGlobalOpts接口配置大页启动参数，在系统启动时即分配足够的大页内存。接口调用方法：

```
libuvp_compute.setGlobalOpts("/etc/xxx.conf", "offline")
```

在conf配置文件中写入如下内容：

```
{
    "kernel.boot": {
        "hugepagesz":"2M",
        "hugepages":"N"
    }
}
```

- vhost进程启动时配置的NVME设备需切换为uio_pci_generic驱动:

  pci_addr：NVME设备的pci地址，可以通过如下命令查询

  ```
  linux-jCSKvs:/ #  lspci -D | grep Non
  0000:03:00.0 Non-Volatile memory controller: Intel Corporation PCIe Data Center SSD (rev 01)
  0000:82:00.0 Non-Volatile memory controller: Intel Corporation PCIe Data Center SSD (rev 01)
  ```

  - 驱动切换：
    ```
    /opt/uvp/spdk/setup.sh config pci_addr
    ```
  - 驱动还原
    ```
    /opt/uvp/spdk/setup.sh reset pci_addr
    ```
  - 查询
    ```
    /opt/uvp/spdk/setup.sh status pci_addr
    ```

命令介绍

- 启动vhost进程
  ```
  /opt/uvp/spdk/vhost_manage.sh start /path/to/vhost.conf /path/to/socketdir
  ```

  vhost.conf：vhost配置文件，具体配置方式见下节**配置介绍**。

  socketdir：vhost-user-scsi或vhost-user-blk磁盘控制器的socket文件存放路径。

  输出：

  0：启动成功。

  其他：启动失败。
- 关闭vhost进程
  ```
  /opt/uvp/spdk/vhost_manage.sh stop /path/to/rpc_socket
  ```

  rpc_socket：vhost进程控制socket文件路径，配置在vhost配置文件的RPC Listen字段。

  输出：

  0：关闭成功。

  其他：关闭失败。
- rpc命令介绍

  vhost进程提供了unix socket方式的rpc命令调用，调用者可以通过配置文件中配置的rpc socket文件地址和vhost进程建立连接，然后使用json格式发送请求。调用rpc命令的示例：uvp_rpc.py

  目前vhost进程提供的rpc功能有：

  - 格式化NVME设备namespace
    ```
    python uvp_rpc.py -s /path/to/rpc_socket nvme_format pci_addr ns_id
    ```

    pci_addr：NVME设备的pci地址，查询方式见NVME设备驱动切换章节。

    ns_id：NVME设备划分的namespace id。

    输出：

    true：格式化命令下发成功

    其他：命令下发失败

    注意

    格式化为危险操作，调用者需要保证指定namespace上已无有效数据。
  - 查询格式化进度
    ```
    python uvp_rpc.py -s /path/to/rpc_socket nvme_format_query pci_addr ns_id
    ```


namespace格式化后可调用，输出：

0：格式化正在进行

1：格式化成功

其他：格式化失败

- 查询NVME设备故障信息

```
python uvp_rpc.py -s /path/to/rpc_socket nvme_identify pci_addr
```

输出NVME设备的健康状态以及错误信息：

```
{
  "Available Spare Space": "OK",
  "Temperature": "OK",
  "Volatile Memory Backup": "OK",
  "Unrecoverable Media Errors": 0,
  "Read Only": "No",
  "Device Reliability": "OK"
}
```

配置介绍

示例：

```
[Global]
  ReactorMask 0x1
  MemNuma 0

[Scsi]
  MaxUnmapLbaCount 0
  MaxUnmapBlockDescriptorCount 0
  OptimalUnmapGranularity 0

[Rpc]
  Enable Yes
  Listen /path/to/vhost.xxx

# NVMe configuration options
[Nvme]
  TransportID "trtype:PCIe traddr:0000:82:00.0" Nvme0
  Timeout 0
  ActionOnTimeout None
  AdminPollRate 100000
  HotplugEnable No

[VhostScsi0]
  Name vhost.0
  Dev 0 Nvme0n1 0:10000:10000:0:0:0

[VhostBlk0]
  Name vhost.1
  Dev Nvme0n2
  Qos 0:10000:10000:0:0:0
```

表 23-1 配置项描述

大项	参数	说明	取值
Global	ReactorMask	vhost进程绑定的cpu掩码	若绑定的cpu为N，则配置为1 << N的16进制表示

大项	参数	说明	取值
Global	MemNuma	vhost进程申请内存绑定的Numa节点	主机Numa节点id
Scsi	MaxUnmapLbaCount	支持的UMAP最大LBA数目	0，不可更改
Scsi	MaxUnmapBlockDescriptorCount	最大UMAP块描述符数目	0，不可更改
Scsi	OptimalUnmapGranularity	最佳UMAP粒度	0，不可更改
Rpc	Enable	是否提供RPC功能	Yes，不可更改
Rpc	Listen	rpc侦听unix socket文件路径	/path/to/vhost.xxx /path/to目录必须存在，vhost启动时会在该目录下生成vhost.xxx文件
Nvme	TransportID	NVME设备描述	"trtype:PCIe traddr:**pci_addr" name** pci_addr：NVME设备pci号 name：名称，不可重复
Nvme	Timeout	超时时长配置	0，不可更改
Nvme	ActionOnTimeout	超时处理策略	None，不可更改
Nvme	AdminPollRate	NVME管理队列处理时间间隔，单位为微秒	100000，不可更改
Nvme	HotplugEnable	是否支持自动识别NVME硬件并热插	No，不可更改
VhostScsi**X**		控制器编号描述	X为控制器编号，0-7，不可重复
VhostScsix	Name	控制器名称，启动后会生成socket文件/path/to/socketdir/Name	不可重复

大项	参数	说明	取值
VhostScsix	Dev	控制器下LUN描述	Dev **lun_id bdev qos** lun_id：LUN编号，配置为0 bdev：块设备描述，配置为[设备名]n[ns_id]，如：nvme0n1，配置名称为nvme0的NVME设备的第一个namespace qos：存储qos描述，不配置则qos不生效。配置方式为total_iops:read_iops:write_iops:total_bytes:read_bytes:write_bytes，其中total和read/write不能同时配置
VhostBlk**X**		磁盘编号描述	X为控制器编号，0-7，不可重复
VhostBlkx	Name	磁盘名称，启动后会生成socket文件/path/to/socketdir/Name	不可重复
VhostBlkx	Dev	磁盘描述	Dev **bdev** bdev：块设备描述，配置为[设备名]n[ns_id]，如：nvme0n1，配置名称为nvme0的NVME设备的第一个namespace
VhostBlkx	Qos	存储qos描述	Qos: **qos** qos：存储qos描述，不配置则qos不生效。配置方式为total_iops:read_iops:write_iops:total_bytes:read_bytes:write_bytes，其中total和read/write不能同时配置

约束限制

- Host上最多允许运行4个vhost进程，每个vhost进程只配置一个NVME设备
- 每个vhost进程只允许配置8个控制器或磁盘（VhostScsi加VhostBlk）
- 不同的vhost进程配置文件不能重复

- 每个vhostscsi控制器或vhostblk磁盘下只能配置1个虚拟磁盘
- 若配置NVME设备未切换为uio_pci_generic驱动，则vhost进程启动失败
- vhost进程启动前，NVME设备必须先划分好namespace
- Qos配置不支持动态修改
- vhost进程运行时，不允许将NVME设备切换回内核驱动
- 不建议多个vhost进程绑定同一个cpu，会降低vhost进程的IO处理能力

23.2.2 配置 vhost-user-scsi 控制器

虚拟机大页配置

虚拟机内存需要配置大页，用于和vhost进程之间共享内存访问：

```
<cpu>
    <numa>
     <cell id='0' cpus='0-3' memory='16777216' memAccess='shared'/>
    </numa>
 </cpu>
 <memoryBacking>
    <hugepages>
      <page size="2048" unit="KiB" nodeset="0"/>
    </hugepages>
 </memoryBacking>
```

cell id：表示虚拟机NUMA节点号

cpus：表示虚拟机vcpu的编号

memory：表示内存大小，单位为KB

memAccess：必须为shared

size：表示使用的大页大小，必须系统支持

unit：大页大小单位

nodeset：表示虚拟机NUMA节点号

vhost-user-scsi 控制器配置

```
<controller type='scsi' index='2' model='vhost-user-scsi'>
      <source path='/home/spdk/vhost.0'/>
      <driver queues='4'/>
</controller>
```

index：scsi控制器编号，和其他的scsi控制器不能重复

model：必须配置为vhost-user-scsi

path：控制器socket路径，必须和指定的vhost进程vhostscsi控制器配置name保持一致

queues：多队列配置，配置为虚拟机vcpu数目。

约束限制

- vhost-user-scsi控制器/虚拟磁盘不支持热插和热拔。
- vhost-user-scsi虚拟磁盘只支持存储Qos上限，不支持虚拟机休眠唤醒、内存热插、虚拟机迁移、存储热迁移、整机迁移、快照、共享卷等其它特性。

- vhost-user-scsi控制器和其他scsi类型控制器不能配置相同index。
- 每个虚拟机最多只能配置两个vhost-user-scsi控制器。
- 一个虚拟机内部vhost-user-scsi和virtio-scsi控制器之和不能超过4个，因此在配置有vhost-user-scsi控制器的情况下，虚拟机无法满足60个磁盘的配置上限。
- 在vhost进程重启(升级/异常恢复)的过程中，虚拟机对vhost-user-scsi磁盘下发的IO请求不会得到处理，此时执行虚拟机关机或者重启会暂时卡住，vhost进程成功重启后恢复。

24 ipvlan

EulerOS支持ipvlan，本文主要介绍ipvlan的配置和使用。

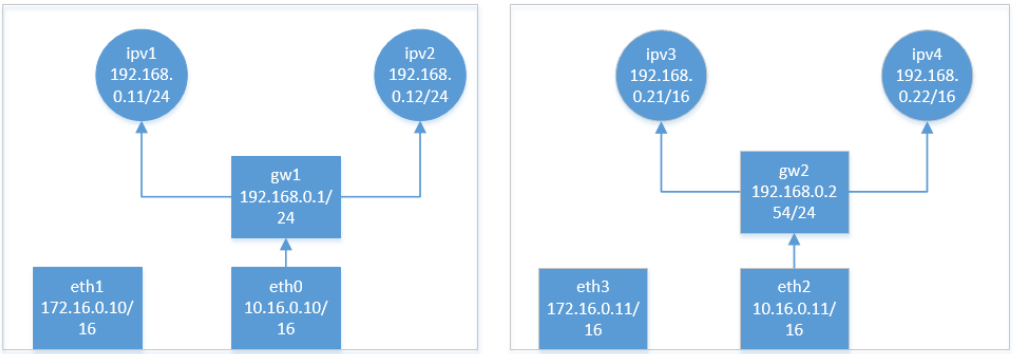
24.1 场景介绍

ipvlan是从一个主机接口虚拟出多个虚拟网络接口，这些虚拟接口都有相同的mac地址，而拥有不同的ip地址。

EulerOS中支持三种ipvlan模式：l2，l3，以及l2e模式。ipvlan各工作模式如**图1 ipvlan工作模式**所示：

- ipv1， ipv2，ipv3，ipv4，gw1，gw2分别为ipvlan接口。eth0，eth1，eth2，eth3分别为物理网口。
- ipv1，ipv2，gw1依附于eth0；ipv3，ipv4，gw2依附于eth2。
- eth0，eth1属于同一节点，eth2，eth3属于另一个节点。

图 24-1 ipvlan 工作模式



以ipv1接口为例说明：

- l2模式：ipv1可与ipv2，ipv3，ipv4通信，与eth0，eth1，eth2，eth3无法通信。
- l3模式：ipv1可与ipv2，eth0，eth2通信，与ipv3，ipv4，eth1，eth3无法通信。
- l2e模式：ipv1可与ipv2，ipv3，ipv4，eth0，eth1，eth2，eth3通信。

  实际可根据需要选择ipvlan工作模式。

24.2 约束限制

1. 由于ipvlan所有接口都有相同的mac地址，在DHCP协议分配ip的时候一般使用mac地址作为机器的标识。这种情况下，客户端动态获取ip的时候需要配置唯一的ClientID字段，并且DHCP server也要正确配置使用该字段作为机器标识，而不是使用mac地址。
2. 父接口只能选择一种模式，依附于它的所有虚拟接口都运行在该模式下，不能混用模式。

24.3 使用指导

ipvlan可通过模块参数“ipvlan_default_mode”控制默认工作模式，默认为l3模式。

ipvlan_default_mode为0，1，2分别对应l2，l3，l2e模式。

配置 ipvlan 工作模式

可通过ip命令配置ipvlan接口的工作模式，通过ip命令指定方式：

```
ip link add ipv1 link eth0 type ipvlan mode l2
ip link add ipv1 link eth0 type ipvlan mode l3
ip link add ipv1 link eth0 type ipvlan mode l2e
```

通过ip命令查看ipvlan接口的工作模式：

```
[root@localhost l2e]# ip -d link show gw
76: gw@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN mode DEFAULT
qlen 1
    link/ether aa:e8:bd:21:xx:xx brd ff:ff:ff:ff:xx:xx promiscuity 0
    ipvlan  mode l2e addrgenmode eui64
```

通过模块参数配置ipvlan接口的默认工作模式：

方法一：

步骤1 在/etc/modprobe.d/目录下创建ipvlan.conf，内容如下：

```
options ipvlan ipvlan_default_mode=2
```

步骤2 用modprobe ipvlan加载ipvlan模块。

```
modprobe ipvlan
```

步骤3 以默认工作模式创建ipvlan接口。

```
ip link add ipv1 eth0 type ipvlan
```

----结束

方法二：

步骤1 用insmod加载ipvlan模块

```
insmod /lib/modules/$(uname -r)/kernel/drivers/net/ipvlan/ipvlan.ko ipvlan_default_mode=2
```


步骤2 以默认工作模式创建ipvlan接口

```
ip link add ipv1 eth0 type ipvlan
```

----结束

环境搭建

两台环境搭建方法相同，以其中一台机器搭建环境为例。

1. 设置eth0，eth1转发，配置nat规则：

```
sysctl net.ipv4.conf.eth0.forwarding=1
iptables -t nat -A POSTROUTING -s 192.168.0.0/255.255.255.0 -o eth0 -j MASQUERADE
sysctl net.ipv4.conf.eth1.forwarding=1
iptables -t nat -A POSTROUTING -s 192.168.0.0/255.255.255.0 -o eth1 -j MASQUERADE
```

2. 配置网关gw：

```
ip link add gw link eth0 type ipvlan
ifconfig gw 192.168.0.1/24
```

3. 配置ipv1，并设置到net1的namespace中：

```
ip netns add net1
ip link add ipv1 link eth0 type ipvlan
ip link set ipv1 netns net1
ip netns exec net1 ip link set ipv1 up
ip netns exec net1 ip link set lo up
ip netns exec net1 ip -4 addr add 192.168.0.11/24 dev ipv1
ip netns exec net1 route add default gw 192.168.0.1
```

4. 配置ipv2，并设置到net2的namespace中：

```
ip netns add net2
ip link add ipv2 link eth0 type ipvlan
ip link set ipv2 netns net2
ip netns exec net2 ip link set ipv2 up
ip netns exec net2 ip link set lo up
ip netns exec net2 ip -4 addr add 192.168.0.12/24 dev ipv2
ip netns exec net2 route add default gw 192.168.0.1
```

常见问题说明

根据**配置ipvlan工作模式**中“方法一”配置ipvlan工作模式后，执行modprobe ipvlan前需确认ipvlan模块是否已加载，如果已加载，需执行以下步骤重新加载ipvlan模块。重新加载后可能会出现ipvlan接口丢失或ipvlan接口ip丢失。

```
[root@localhost l2e]# lsmod | grep ipvlan
ipvlan                 18125  0
[root@localhost l2e]# rmmod ipvlan
[root@localhost l2e]# modprobe ipvlan
```

添加ipvlan时如果同一主接口上存在相同的ip，ipvlan会添加失败，导致网络不通，可以通过如下命令打开日志打印，以便定位问题。

```
ip link add ipv1 link eth0 type ipvlan
ethtool -s ipv1 msglvl 0x20
```

打开后，如果添加ipvlan失败会有如下打印：

```
ipv1: Failed to add IPv4=2.1.1.11 on ipv1 intf.
```

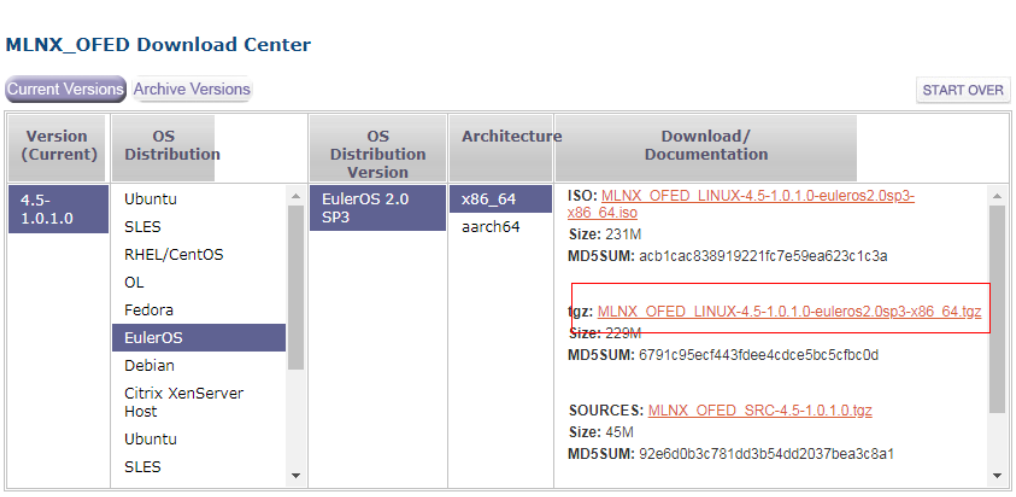
针对ipvlan l2e模式引入了本地队列提升tcp传输性能，提供以下两个sysctl参数用于设置队列长度以及超时时长。

```
net.ipvlan.loop_delay = 10
net.ipvlan.loop_qlen = 131072
```

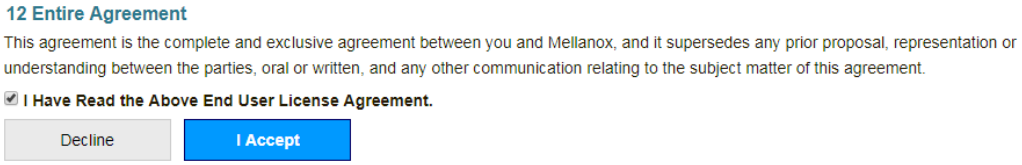

25 mellanox_ofed 编译与安装指导

25.1 下载驱动包

1. 在Mellanox官网下载如下RoCE网卡驱动包，路径为：**http://www.mellanox.com/page/products_dyn?product_family=26&mtag=linux_sw_drivers**。MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64.tgz驱动包的获取路径如图所示：



2. 单击“MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64.tgz”，在新弹出的页面上拖动到页面最下方，勾选"I have Read the Above End User License Agreement"，并选择"I Accept"，开始下载驱动。



25.2 获取源码包

解压下载的驱动包，根据下面的步骤获取源码包，源码包在/root/rpmbuild/SOURCES/目录下。

```
[root@localhost mellanox]# ls
Euler_compile_env  Euler_compile_env.tar.gz  MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64.tgz
[root@localhost mellanox]# tar -xf MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64.tgz
[root@localhost mellanox]# ls -l
总用量 1224808
dr-xr-xr-x. 18 root root       4096 2月  20 18:53 Euler_compile_env
-rw-------.  1 root root 1015842522 2月  18 20:51 Euler_compile_env.tar.gz
drwxr-xr-x.  6 root root       4096 11月 29 13:37 MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64
-rw-------.  1 root root  238345065 2月  20 16:28 MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64.tgz
[root@localhost mellanox]# cd MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64/src/
[root@localhost src]# ls
MLNX_OFED_SRC-4.5-1.0.1.0.tgz
[root@localhost src]# tar -xf MLNX_OFED_SRC-4.5-1.0.1.0.tgz
[root@localhost src]# ls
MLNX_OFED_SRC-4.5-1.0.1.0  MLNX_OFED_SRC-4.5-1.0.1.0.tgz
[root@localhost src]# cd MLNX_OFED_SRC-4.5-1.0.1.0/
RPMS/  SRPMS/
[root@localhost src]# cd MLNX_OFED_SRC-4.5-1.0.1.0/SRPMS/

[root@localhost SRPMS]# rpm -ivh mlnx-ofa_kernel-4.5-OFED.4.5.1.0.1.1.gb4fdfac.src.rpm
正在升级/安装...
   1:mlnx-ofa_kernel-4.5-OFED.4.5.1.0.################################# [100%]
警告：用户builder 不存在 - 使用root
警告：用户builder 不存在 - 使用root
[root@localhost SRPMS]# cd /root/rpmbuild/SOURCES/
[root@localhost SOURCES]# ls
mlnx-ofa_kernel-4.5.tgz
```

25.3 搭建编译环境

获取对应内核版本的编译环境，并解压；然后将上面获取的mlnx-ofa_kernel-4.5.tgz包放到编译环境。

```
[root@localhost mellanox]# ls
Euler_compile_env        MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64
Euler_compile_env.tar.gz  MLNX_OFED_LINUX-4.5-1.0.1.0-euleros2.0sp3-x86_64.tgz
[root@localhost mellanox]# cd Euler_compile_env/
[root@localhost Euler_compile_env]# cp /root/rpmbuild/SOURCES/mlnx-ofa_kernel-4.5.tgz .
[root@localhost Euler_compile_env]# ls
bin   chroot.sh  devlink.ko  home  lib64  mlnx-ofa_kernel-4.5.tgz  mlx_compat.ko  opt   root  sbin  sys  usr
boot  dev        etc         lib   media  mlx4_core.ko             mnt            proc  run   srv   tmp  var
```

25.4 进行编译

1. 进入编译环境，chroot并解压源码包。

```
[root@localhost Euler_compile_env]# chroot .
bash: EulerOS_history: 未找到命令
bash-4.2# export PROMPT_COMMAND=
bash-4.2# tar -xf mlnx-ofa_kernel-4.5.tgz
cbash-4.2# cd mlnx-ofa_kernel-4.5
```


2. 执行如下命令，生成Makefile文件。最终configure完成的界面如下：

```
./configure -j8 --with-core-mod --with-user_mad-mod --with-user_access-mod --with-addr_trans-mod --with-mlx4-mod --with-mlx4_en-mod --with-mlx5-mod --with-mlx5_core-mod --with-iser-mod --with-srp-mod
```

```
checking if t10_pi_ref_tag() exists... checking if struct netdev_bonding_info has prog_attached... checking if tcf_bl
ock_cb_register has fifth parameter... checking if ib_umem_notifier_invalidate_range_start has parameter blockable...
 no
no
no
no
no
checking that generated files are newer than configure... done
configure: creating ./config.status
config.status: creating Makefile
config.status: creating config.h
config.status: executing depfiles commands

CC:           gcc
LD:           ld
CFLAGS:       -g -O2
EXTRA_KCFLAGS: -include /mlnx-ofa_kernel-4.5/compat/config.h

Type 'make' to build kernel modules.
```

3. 根据提示，Makefile已经生成。执行如下命令开始编译：

```
make -j8
```

```
bash-4.2# make -j8
cd ofed_scripts/utils && python setup.py build && cd -
running build
running build_py
creating build
creating build/lib
copying netlink.py -> build/lib
copying dcbnetlink.py -> build/lib
copying genetlink.py -> build/lib
running build_scripts
creating build/scripts-2.7
copying and adjusting mlnx_qos -> build/scripts-2.7
copying and adjusting tc_wrap.py -> build/scripts-2.7
copying and adjusting mlnx_perf -> build/scripts-2.7
copying mlnx_get_vfs.pl -> build/scripts-2.7
copying and adjusting mlnx_qcn -> build/scripts-2.7
copying and adjusting mlnx_dump_parser -> build/scripts-2.7
copying and adjusting mlx_fs_dump -> build/scripts-2.7
changing mode of build/scripts-2.7/mlnx_qos from 600 to 755
changing mode of build/scripts-2.7/tc_wrap.py from 600 to 755
changing mode of build/scripts-2.7/mlnx_perf from 600 to 755
changing mode of build/scripts-2.7/mlnx_qcn from 600 to 755
changing mode of build/scripts-2.7/mlnx_dump_parser from 600 to 755
changing mode of build/scripts-2.7/mlx_fs_dump from 600 to 755
/mlnx-ofa_kernel-4.5
```

4. 出现如下打印，编译完成。


```
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/core/iw_cm.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/core/rdma_cm.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/core/rdma_ucm.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/bnxt_re/bnxt_re.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/cxgb3/iw_cxgb3.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/cxgb4/iw_cxgb4.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/hfi1/hfi1.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/i40iw/i40iw.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/ipath/ib_ipath.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/mlx4/mlx4_ib.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/mlx5/mlx5_ib.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/mthca/ib_mthca.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/nes/iw_nes.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/ocrdma/ocrdma.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/qedr/qedr.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/qib/ib_qib.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/usnic/usnic_verbs.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/hw/vmw_pvrdma/vmw_pvrdma.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/sw/rdmavt/rdmavt.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/ulp/iser/ib_iser.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/ulp/opa_vnic/opa_vnic.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/ulp/srp/ib_srp.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/infiniband/ulp/srpt/ib_srpt.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/net/ethernet/mellanox/mlx4/mlx4_core.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/net/ethernet/mellanox/mlx4/mlx4_en.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/net/ethernet/mellanox/mlx5/core/mlx5_core.ko
  LD [M]  /mlnx-ofa_kernel-4.5/drivers/scsi/scsi_transport_srp.ko
make[1]: 离开目录“/usr/src/kernels/3.10.0-862.14.0.1.h81.eulerosv2r7.x86_64”
bash-4.2#
```

25.5 驱动验证

1. 主要的驱动文件为mlx4_core.ko和mlx_compat.ko。


```
bash-4.2# find . -name *.ko
./drivers/infiniband/core/ib_ucm.ko
./drivers/infiniband/core/rdma_ucm.ko
./drivers/infiniband/core/rdma_cm.ko
./drivers/infiniband/core/iw_cm.ko
./drivers/infiniband/core/ib_umad.ko
./drivers/infiniband/core/ib_uverbs.ko
./drivers/infiniband/core/ib_cm.ko
./drivers/infiniband/core/ib_core.ko
./drivers/infiniband/sw/rdmavt/rdmavt.ko
./drivers/infiniband/ulp/opa_vnic/opa_vnic.ko
./drivers/infiniband/ulp/srpt/ib_srpt.ko
./drivers/infiniband/ulp/iser/ib_iser.ko
./drivers/infiniband/ulp/srp/ib_srp.ko
./drivers/infiniband/hw/mlx4/mlx4_ib.ko
./drivers/infiniband/hw/cxgb3/iw_cxgb3.ko
./drivers/infiniband/hw/qedr/qedr.ko
./drivers/infiniband/hw/hfi1/hfi1.ko
./drivers/infiniband/hw/qib/ib_qib.ko
./drivers/infiniband/hw/cxgb4/iw_cxgb4.ko
./drivers/infiniband/hw/vmw_pvrdma/vmw_pvrdma.ko
./drivers/infiniband/hw/ocrdma/ocrdma.ko
./drivers/infiniband/hw/usnic/usnic_verbs.ko
./drivers/infiniband/hw/mthca/ib_mthca.ko
./drivers/infiniband/hw/ipath/ib_ipath.ko
./drivers/infiniband/hw/bnxt_re/bnxt_re.ko
./drivers/infiniband/hw/mlx5/mlx5_ib.ko
./drivers/infiniband/hw/nes/iw_nes.ko
./drivers/infiniband/hw/i40iw/i40iw.ko
./drivers/scsi/scsi_transport_srp.ko
./drivers/net/ethernet/mellanox/mlx4/mlx4_en.ko
./drivers/net/ethernet/mellanox/mlx4/mlx4_core.ko
./drivers/net/ethernet/mellanox/mlx5/core/mlx5_core.ko
./compat/mlx_compat.ko
```

2. 退出编译环境，将mlx4_core.ko和mlx_compat.ko拷贝出来，进行验证。
   mlx4_core.ko依赖 devlink.ko和mlx_compat.ko，所以需要先加载这两个ko文件。
   devlink.ko是内核自带的ko。验证过程未出现异常则驱动验证成功。

```
[root@localhost Euler_compile_env]# modinfo mlx4_core.ko | grep depends
depends:        devlink,mlx_compat
[root@localhost Euler_compile_env]# modprobe devlink
[root@localhost Euler_compile_env]# lsmod | grep devlink
devlink                42368  0
[root@localhost Euler_compile_env]# insmod mlx_compat.ko
[root@localhost Euler_compile_env]# insmod mlx4_core.ko
[root@localhost Euler_compile_env]# lsmod | grep mlx
mlx4_core             360598  0
mlx_compat             29012  1 mlx4_core
devlink                42368  1 mlx4_core
```


26 QAT 部署手册

26.1 概述

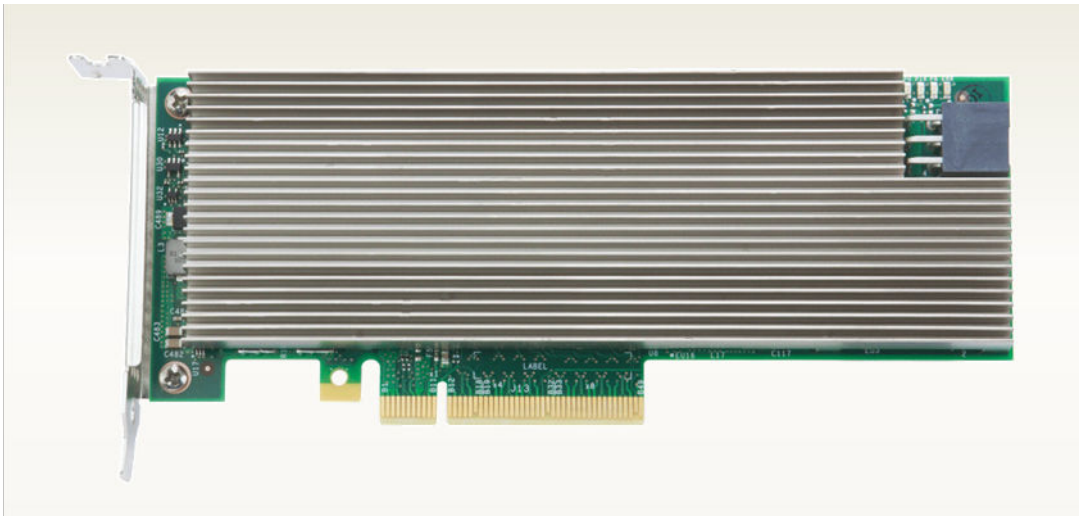
本文主要介绍了Intel QAT软件的安装过程和使用方法。

26.1.1 QAT 概述

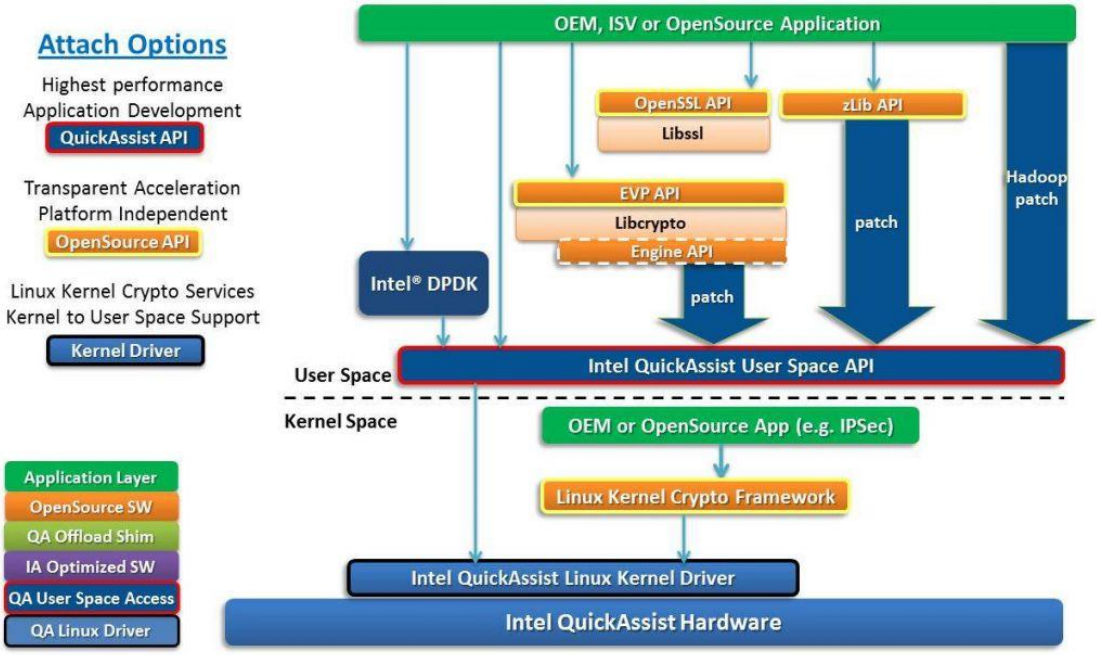
在云计算中，安全的重要性日益提升，安全级别不断提高，需要投入大量的计算资源来处理加解密的复杂计算，采用通用芯片处理加解密计算将让云计算中心中的成本投入不断增大，而采用了专用加解密技术的 QAT（Quick Assist Technology）则为用户解决了这一瓶颈，有助于云计算中心资源优化，提升整体性能。

由于数据压缩都是计算密集型任务，启用压缩通常会极大增加 CPU 的开销。利用英特尔QAT可以在不增加 CPU 开销的情况下对数据进行透明压缩/解压缩，以达到节省磁盘空间和磁盘读写带宽、增加磁盘读写吞吐量（IOPS）的目标。

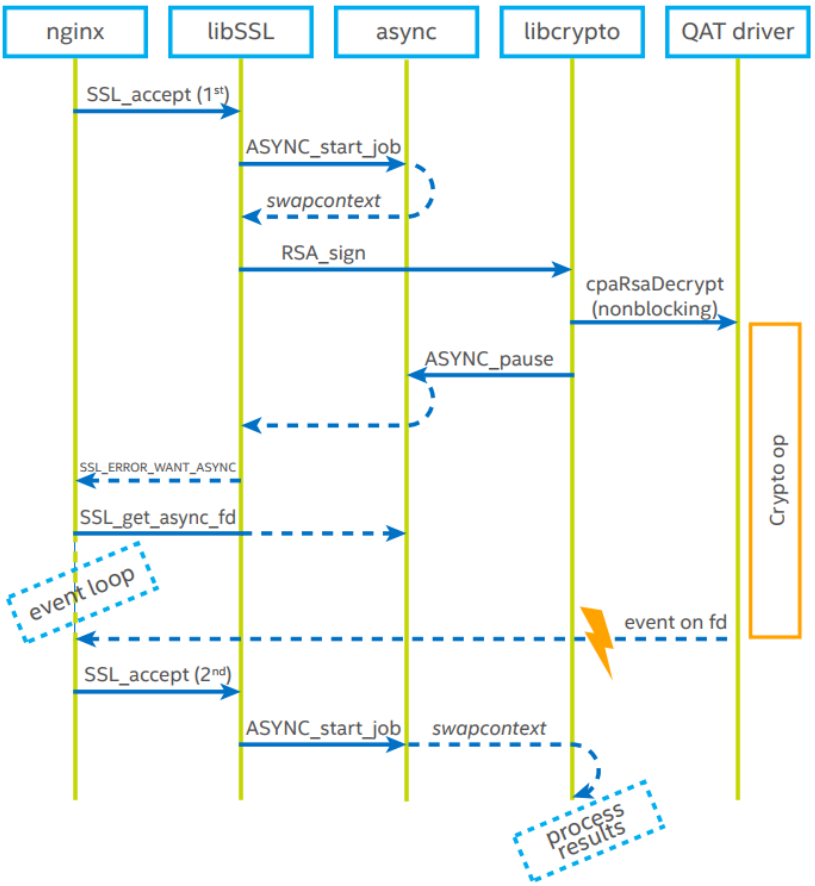
- QAT板卡实物图：



- QAT运行架构图：



- QAT运行时序图：



QAT卡支持物理机运行场景，目前已经测试过的物理机型号为2288V3和2288V5。

26.1.2 运行环境

Intel QAT各型号设备对应的设备ID：

Supported Intel® QuickAssist Technology (QAT) Endpoints and their Device IDs

Intel® QAT Endpoint	Physical Function (PF) Device ID	VF Device ID
8900-8920	0434	0442
8925-8955	0435	0443
Intel® C62x Chipset	37c8	37c9
Intel Atom® C3000 Processor Product Family	19e2	19e3

说明

- 本文档中采用的是895系列，物理功能设备号为435。
- QAT设备故障场景，不能平滑切到CPU执行加解密功能。

26.1.2.1 物理机中运行环境

查询QAT的物理功能设备号命令：

```
lspci -d :435
```

```
[root@localhost ~]# lspci -d :435
8b:00.0 Co-processor: Intel Corporation DH895XCC Series QAT
b1:00.0 Co-processor: Intel Corporation DH895XCC Series QAT
[root@localhost ~]#
[root@localhost ~]# lsmod | grep qa
qat_dh895xcc           17996  0
intel_qat             200991  1 qat_dh895xcc
authenc                17776  1 intel_qat
uio                    19259  1 intel_qat
[root@localhost ~]#
```

26.1.2.2 虚拟机中运行环境

HOST端支持QAT虚拟化功能，需要在/boot/grub2/grub.cfg中添加启动参数：

```
intel_iommu=on
```

如图：

```
linux-rCXvst:/boot/grub2 # pwd
/boot/grub2
linux-rCXvst:/boot/grub2 # ls
fonts  grub.cfg  grubenv  i386-pc  locale  themes
linux-rCXvst:/boot/grub2 #
```

Host端上查询是否存在QAT 895x系列卡的虚拟功能设备号,可以用下述命令：

```
lspci  -d  :443
```

如图：


```
linux-rCXvst:/home/Euler_jn # lspci -d :443
86:01.0 Co-processor: Intel Corporation Device 0443
86:01.1 Co-processor: Intel Corporation Device 0443
86:01.2 Co-processor: Intel Corporation Device 0443
86:01.3 Co-processor: Intel Corporation Device 0443
86:01.4 Co-processor: Intel Corporation Device 0443
86:01.5 Co-processor: Intel Corporation Device 0443
86:01.6 Co-processor: Intel Corporation Device 0443
86:01.7 Co-processor: Intel Corporation Device 0443
86:02.0 Co-processor: Intel Corporation Device 0443
86:02.1 Co-processor: Intel Corporation Device 0443
86:02.2 Co-processor: Intel Corporation Device 0443
86:02.3 Co-processor: Intel Corporation Device 0443
86:02.4 Co-processor: Intel Corporation Device 0443
86:02.5 Co-processor: Intel Corporation Device 0443
86:02.6 Co-processor: Intel Corporation Device 0443
86:02.7 Co-processor: Intel Corporation Device 0443
86:03.0 Co-processor: Intel Corporation Device 0443
86:03.1 Co-processor: Intel Corporation Device 0443
86:03.2 Co-processor: Intel Corporation Device 0443
86:03.3 Co-processor: Intel Corporation Device 0443
86:03.4 Co-processor: Intel Corporation Device 0443
86:03.5 Co-processor: Intel Corporation Device 0443
86:03.6 Co-processor: Intel Corporation Device 0443
86:03.7 Co-processor: Intel Corporation Device 0443
86:04.0 Co-processor: Intel Corporation Device 0443
86:04.1 Co-processor: Intel Corporation Device 0443
86:04.2 Co-processor: Intel Corporation Device 0443
86:04.3 Co-processor: Intel Corporation Device 0443
86:04.4 Co-processor: Intel Corporation Device 0443
86:04.5 Co-processor: Intel Corporation Device 0443
86:04.6 Co-processor: Intel Corporation Device 0443
86:04.7 Co-processor: Intel Corporation Device 0443
linux-rCXvst:/home/Euler_jn #
```

KVM虚拟机创建过程中，xml文件添加配置项：

```
<hostdev mode='subsystem' type='pci' managed='yes'>
      <source>
          <address domain='0x0000' bus='0x86' slot='0x02' function='0x1'/>
      </source>
</hostdev>
```

其中，slot取值范围为0x01-0x04，function取值范围为0x0-0x7，在不同虚拟机中，slot号和function号不能同时相同。

如图：


```
  <graphics type='vnc' port='5902' autoport='yes' listen='0.0.0.0'>
    <listen type='address' address='0.0.0.0'/>
  </graphics>
  <video>
    <model type='cirrus' vram='9216' heads='1'/>
    <alias name='video0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </video>
  <memballoon model='virtio'>
    <alias name='balloon0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
  </memballoon>
  <hostdev mode='subsystem' type='pci' managed='yes'>
    <source>
        <address domain='0x0000' bus='0x86' slot='0x02' function='0x1'/>
    </source>
  </hostdev>
</devices>
<qemu:commandline>
  <qemu:arg value='-device'/>
  <qemu:arg value='isa-ipmi,chardev=mychar,start_time=300,wdt_reboot=300,reboot_force_enable=false'/>
</qemu:commandline>
</domain>
```

26.1.3 安装

安装 QAT 前的准备工作

卸载系统自带的QAT驱动模块。

服务器开机或重启时，系统可能会自动加载几个qat相关的模块（qat_dh895xcc、intel_qat、authenc 等，以实际为主），在安装之前应先卸载系统自带的qat内核模块。查询和操作步骤如下：

步骤1：执行lsmod | grep qa 命令，查看当前加载的qat相关ko模块。

```
[root@localhost ~]# lsmod | grep qa
qat_dh895xcc           13497  0
intel_qat             120091  1 qat_dh895xcc
authenc                17776  1 intel_qat
[root@localhost ~]#
```

步骤2：如果有这几个模块，执行 rmmod qat_dh895xcc 、rmmod intel_qat 、rmmod authenc 命令卸载QAT驱动模块。

步骤3：删除系统中自带的qat驱动目录/lib/modules/$(uname -r)/kernel/drivers/crypto/qat 。

步骤4：执行depmod -a，更新内核模块依赖关系。

QAT 软件的安装

QAT的安装可以通过iso镜像获取安装包：

说明

软件包具体版本号以镜像中实际版本号为准，文档中涉及的所有版本号仅作为说明。

步骤1 首先配置挂载iso镜像：

iso文件：EulerOS-V2.0SP5-x86_64-dvd.iso

挂载iso到/mnt/iso路径下：


```
mkdir /mnt/iso
mount EulerOS-V2.0SP5-x86_64-dvd.iso /mnt/iso/
```

挂载完成后可以进入/mnt/iso看一下是不是成功挂载。

步骤2 挂载成功后获取QAT相关的rpm包：

进入/mnt/iso/Packages/路径，可以看到QAT相关的所有安装包。

- openssl110f-libs-1.1.0f-5.x86_64.rpm
- openssl110f-1.1.0f-5.x86_64.rpm //openssl包
- qat-1.7-L.4.0.1.52.3.x86_64.rpm // QAT driver包
- qat-engine-0.5.28-1.x86_64.rpm // QAT针对openssl的引擎包

步骤3 安装QAT相关的rpm包：

```
rpm -ivh openssl110f-libs-1.1.0f-5.x86_64.rpm
rpm -ivh openssl110f-1.1.0f-5.x86_64.rpm
rpm -ivh qat-1.7-L.4.0.1.52.3.x86_64.rpm --force
rpm -ivh qat-engine-0.5.28-1.x86_64.rpm
```

说明

安装qat-1.7-L.4.0.1.52.3.x86_64.rpm包时需要加--force选项，是为了强制覆盖系统中自带的qat驱动文件。

安装后的文件列表如下（以实际输出为准）：

- openssl110f-libs-1.1.0f-5.x86_64.rpm

```
/etc/pki-1.1/tls
/etc/pki-1.1/tls/certs
/etc/pki-1.1/tls/openssl.cnf
/usr/lib64/engines-1.1
/usr/lib64/engines-1.1/capi.so
/usr/lib64/engines-1.1/padlock.so
/usr/lib64/libcrypto.so.1.1
/usr/lib64/libcrypto.so.1.1.0f
/usr/lib64/libssl.so.1.1
/usr/lib64/libssl.so.1.1.0f
```

- openssl110f-1.1.0f-5.x86_64.rpm

```
/etc/pki-1.1/tls/certs/Makefile
/usr/bin/make-dummy-cert-1.1
/usr/bin/openssl-1.1
/usr/bin/renew-dummy-cert-1.1
```

- qat-1.7-L.4.0.1.52.3.x86_64.rpm

```
/etc/default/qat
/etc/qat
/etc/qat/c3xxx_dev0.conf
/etc/qat/c3xxxvf_dev0.conf.vm
/etc/qat/c6xx_dev0.conf
/etc/qat/c6xx_dev1.conf
/etc/qat/c6xx_dev2.conf
/etc/qat/c6xxvf_dev0.conf.vm
/etc/qat/d15xx_dev0.conf
/etc/qat/d15xxpf_dev0.conf
/etc/qat/d15xxvf_dev0.conf.vm
/etc/qat/dh895xcc_dev0.conf
/etc/qat/dh895xcc_dev1.conf
/etc/qat/dh895xccvf_dev0.conf.vm
/etc/udev/rules.d/00-qat_udev.rules
/lib/firmware/qat_895xcc.bin
/lib/firmware/qat_895xcc_mmp.bin
/lib/firmware/qat_c3xxx.bin
/lib/firmware/qat_c3xxx_mmp.bin
/lib/firmware/qat_c62x.bin
/lib/firmware/qat_c62x_mmp.bin
```



```
/lib/firmware/qat_d15xx.bin
/lib/firmware/qat_d15xx_mmp.bin
/lib/modules/EulerOS/qat
/lib/modules/EulerOS/qat/intel_qat.ko
/lib/modules/EulerOS/qat/qat_c3xxx.ko
/lib/modules/EulerOS/qat/qat_c3xxxvf.ko
/lib/modules/EulerOS/qat/qat_c62x.ko
/lib/modules/EulerOS/qat/qat_c62xvf.ko
/lib/modules/EulerOS/qat/qat_d15xx.ko
/lib/modules/EulerOS/qat/qat_d15xxvf.ko
/lib/modules/EulerOS/qat/qat_dh895xcc.ko
/lib/modules/EulerOS/qat/qat_dh895xccvf.ko
/lib/modules/EulerOS/qat/usdm_drv.ko
/usr/bin/adf_ctl
/usr/bin/qat_service
/usr/bin/qat_service_ctl
/usr/lib/systemd/system/qat.service
/usr/lib64/libqat_s.so
/usr/lib64/libusdm_drv_s.so
```

- qat-engine-0.5.28-1.x86_64.rpm

```
/usr/lib64/engines-1.1/qat.so
```

步骤4 重启系统。

----结束

26.1.4 配置运行

26.1.4.1 QAT1.7 驱动配置文件

- 运行前QAT先配置QAT配置文件，QAT驱动程序根据配置文件进行设置，895系列的QAT配置文件路径为：

  /etc/qat/dh895xcc_dev0.conf

- 如果存在多块QAT板卡，每块板卡的驱动程序配置文件相互独立，配置文件命名如下：

  /etc/qat/dh895xcc_dev0.conf

  /etc/qat/dh895xcc_dev1.conf

  /etc/qat/dh895xcc_dev2.conf

  ……

  QAT1.7驱动配置文件模板如下，请先使用配置模板文件中的内容覆盖具体的QAT配置文件后，然后再执行详细的配置操作：

```
################################################################################
#
# @par
# This file is provided under a dual BSD/GPLv2 license.  When using or
#   redistributing this file, you may do so under either license.
#
#   GPL LICENSE SUMMARY
#
#   Copyright(c) 2007-2017 Intel Corporation. All rights reserved.
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of version 2 of the GNU General Public License as
#   published by the Free Software Foundation.
#
#   This program is distributed in the hope that it will be useful, but
#   WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
#   General Public License for more details.
```

```
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 51 Franklin St - Fifth Floor, Boston, MA 02110-1301 USA.
#   The full GNU General Public License is included in this distribution
#   in the file called LICENSE.GPL.
#
#   Contact Information:
#   Intel Corporation
#
#   BSD LICENSE
#
#   Copyright(c) 2007-2017 Intel Corporation. All rights reserved.
#
#   Redistribution and use in source and binary forms, with or without
#   modification, are permitted provided that the following conditions
#   are met:
#
#     * Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#     * Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in
#       the documentation and/or other materials provided with the
#       distribution.
#     * Neither the name of Intel Corporation nor the names of its
#       contributors may be used to endorse or promote products derived
#       from this software without specific prior written permission.
#
#   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
#   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
#   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
#   SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
#   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
#   THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
#   (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#   OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
#
##############################################################################
#############################################################
#
# This file is the configuration for a single dh895xcc_qa
# device.
#
# Each device has 32 independent banks.
#
# - Each bank can contain up to 2 crypto and/or up to 2 data
#   compression services.
#
# - The interrupt for each can be directed to a
#   specific core.
#
#############################################################

################################################
# General Section
################################################

[GENERAL]
ServicesEnabled = asym

# Use version 2 of the config file
ConfigVersion = 2

# Look Aside Cryptographic Configuration
cyHmacAuthMode = 1
```

```
# Wireless Enable/Disable, valid values: 1,0
WirelessEnabled = 0

# Firmware Location Configuration
Firmware_MofPath = dh895xcc/mof_firmware.bin
Firmware_MmpPath = dh895xcc/mmp_firmware.bin

# Default selection of CY concurrent requests.
CyNumConcurrentSymRequests = 512
CyNumConcurrentAsymRequests = 64

# Default number of DC concurrent requests.
DcNumConcurrentRequests = 512

#Statistics, valid values: 1,0
statsGeneral = 1
statsDc = 1
statsDh = 1
statsDrbg = 1
statsDsa = 1
statsEcc = 1
statsKeyGen = 1
statsLn = 1
statsPrime = 1
statsRsa = 1
statsSym = 1

# Debug feature, if set to 1 it enables additional entries in /proc filesystem
ProcDebug = 1

# Enables or disables Single Root Complex IO Virtualization.
# If this is enabled (1) then SRIOV and VT-d need to be enabled in
# BIOS and there can be no Cy or Dc instances created in PF (Dom0).
# If this is disabled (0) then SRIOV and VT-d needs to be disabled
# in the BIOS and Cy and/or Dc instances can be used in PF (Dom0)
SRIOV_Enabled = 0

############################################################
#
# Logical Instances Section
# A logical instance allows each address domain
# (kernel space and individual user space processes)
# to configure rings (i.e. hardware assisted queues)
# to be used by that address domain and to define the
# behavior of that ring.
#
# The address domains are in the following format
# - For kernel address domains
#       [KERNEL]
# - For user process address domains
#   [xxxxx]
#   Where xxxxx may be any ascii value which uniquely identifies
#   the user mode process.
#   To allow the driver correctly configure the
#   logical instances associated with this user process,
#   the process must call the icp_sal_userStartMultiProcess(...)
#   passing the xxxxx string during process initialisation.
#   When the user space process is finished it must call
#   icp_sal_userStop(...) to free resources.
#   NumProcesses will indicate the maximum number of processes
#   that can call icp_sal_userStartMultiProcess on this instance.
#   Warning: the resources are preallocated: if NumProcesses
#   is too high, the driver will fail to load
#
# Items configurable by a logical instance are:
# - Name of the logical instance
# - The response mode associated wth this logical instance (0
#   for IRQ or 1 for polled).
```



```
# - The core the instance is affinitized to (optional)
#
# The format of the logical instances are:
# - For crypto:
#               Cy<n>Name = "xxxx"
#               Cy<n>IsPolled = 0|1
#               Cy<n>CoreAffinity = 0-7
#
# - For Data Compression
#               Dc<n>Name = "xxxx"
#               Dc<n>IsPolled = 0|1
#               Dc<n>CoreAffinity = 0-7
#
# Where:
#       - n is the number of this logical instance starting at 0.
#       - xxxx may be any ascii value which identifies the logical instance.
#
# Note: for user space processes, a list of values can be specified for
# the core affinity: for example
#             Cy0CoreAffinity = 0,2,4
# These comma-separated lists will allow multiple processes to use
# different accelerators and cores, and will wrap around the numbers
# in the list. In the above example, process 0 will have affinity 0,
# and process 1 will have affinity 2 etc.
#
############################################################

# This flag is to enable SSF features (CNV and BnP)
StorageEnabled = 0

#############################################
# Kernel Instances Section
#############################################
[KERNEL]
NumberCyInstances = 0
NumberDcInstances = 0

#############################################
# User Process Instance Section
#############################################
[SHIM]
NumberCyInstances = 1
NumberDcInstances = 0
NumProcesses = 256
LimitDevAccess = 1

# Crypto - User space
Cy0Name = "UserCY0"
Cy0IsPolled = 1
Cy0CoreAffinity = 0
```

若配置QAT1.7支持**128**个crypto instance，修改模板中如下配置项：

```
[GENERAL]
ServicesEnabled = cy

#############################################
# Kernel Instances Section
#############################################
[KERNEL]  ##对于用户态应用程序，清除kernel instance相关定义
NumberCyInstances = 0
NumberDcInstances = 0

#############################################
# User Process Instance Section
#############################################
 [SHIM]
NumberCyInstances = 1  ##配置每个进程crypto（加解密）instance数量
NumberDcInstances = 0  ##配置每个进程dc（压缩）instance数量
NumProcesses = 128  ##配置最大使用QAT的进程数量
LimitDevAccess = 1  ##配置1表示限制进程只能使用本QAT卡资源
```



```
##按照NumberCyInstances和NumberDcInstances定义每个instance的属性（此处只需1个），有多个##需要
配置多个。
# Crypto - User space
CyOName = "UserCYO"
CyOIsPolled = 1
CyOCoreAffinity = 0  ##配置core的亲和性，如果是polled instance，忽略该配置项。
```

若配置QAT1.7支持**256**个crypto instance，修改模板中如下配置项：

```
[GENERAL]
ServicesEnabled = asym

############################################
# Kernel Instances Section
############################################
[KERNEL]
NumberCyInstances = 0
NumberDcInstances = 0

############################################
# User Process Instance Section
############################################
[SHIM]
NumberCyInstances = 1
NumberDcInstances = 0
NumProcesses = 256
LimitDevAccess = 1

##按照NumberCyInstances和NumberDcInstances定义每个instance的属性（此处只需1个），有多个##需要
配置多个。
# Crypto - User space
CyOName = "UserCYO"
CyOIsPolled = 1
CyOCoreAffinity = 0  ##配置core的亲和性，如果是polled instance，忽略该配置项。
```

- 配置文件配置完成后，在root用户下执行命令 systemctl start qat 启动或者执行 systemctl restart qat 重启QAT服务，并可通过执行命令systemctl status qat查询QAT服务状态。

说明

1. 若配置256个instance，QAT卡仅支持非对称加密（RSA，EC，DH）。
2. 若修改该配置文件，需重启QAT服务方可生效。

26.1.4.2 配置应用程序

应用程序通过openssl-1.1提供的API访问QAT进行加解密运算，这里以配置运行Nginx为例说明。

Nginx使能QAT有以下两种方式，二者选其一即可：

1. 在Nginx的配置文件nginx.conf中配置引擎以及开启异步开关。
2. 在Nginx配置文件中开启异步开关并且对openssl-1.1的配置文件进行修改，指定QAT为工作引擎。

如果选用第一种方式，需要在nginx.conf文件中配置两个配置项（注意如果配置QAT支持256 instances，只能支持非对称加密（RSA，EC，DH），nginx需要一定的针对性修改，否则只能通过第二种方式进行配置）。

```
ssl_engine qat;  //主模块中配置，与worker_processes配置项同级
ssl_asynch on;  //Https对应server模块中配置，与ssl_protocols配置项同级
```

说明

当前，Nginx配置文件中仅能够配置引擎以及开启异步开关，暂不支持default_algorithms等其他openssl相关配置项的配置。

如果选用第二种方式，首先需要配置Nginx配置文件ssl_asynch配置项为on，还需要对openssl-1.1的配置文件进行修改，指定QAT为工作引擎，配置文件路径为/etc/pki-1.1/tls/openssl.cnf。内容修改为：

```
# OpenSSL example configuration file.
# This is mostly being used for generation of certificate requests.
# This definition stops the following lines choking if HOME isn't
# defined.
openssl_conf = openssl_def
[openssl_def]
engines = engine_section
[engine_section]
qat = qat_section
[qat_section]
engine_id = qat
dynamic_path = /usr/lib64/engines-1.1/qat.so
default_algorithms = ALL
```

说明

若QAT驱动配置使用256个instance，则配置项default_algorithms仅支持配置非对称加密（RSA，EC，DH）。

26.1.4.3 配置环境变量

要正确使用QAT服务，还需配置环境变量，为openssl-1.1指定配置文件路径：

```
export LD_LIBRARY_PATH=/usr/lib64:/usr/lib64/engines-1.1
declare -x OPENSSL_CONF="/etc/pki-1.1/tls/openssl.cnf"
```

说明

该环境变量仅在配置环境变量的shell中有效。

配置完成后就可以启动Nginx了。

26.1.5 服务命令

- 启动QAT服务，QAT驱动相关内核模块被插入，执行命令：

  ```
  systemctl start qat
  ```

  如图：

  ```
  [root@localhost ~]# systemctl start qat
  [root@localhost ~]# lsmod | grep qa
  qat_dh895xcc           17996  1
  intel_qat             200991  1 qat_dh895xcc
  authenc                17776  1 intel_qat
  uio                    19259  1 intel_qat
  [root@localhost ~]#
  ```

  说明

  启动QAT服务时，加载usdm_drv内存管理模块，如果模块参数max_mem_reserved不配置，默认情况操作系统总内存的1/128（但最多预留1GB）作为预留内存使用（如128GB的总内存会预留1GB）。模块参数max_mem_reserved可以配置预留内存大小，单位KB。

- 关闭QAT服务，关闭之后QAT相关内核模块被卸载，执行命令：

  ```
  systemctl stop qat
  ```

  如图：

  ```
  [root@localhost ~]# systemctl stop qat
  [root@localhost ~]# lsmod | grep qa
  [root@localhost ~]#
  ```

- 重启QAT服务，执行命令：

  ```
  systemctl restart qat
  ```


- 配置QAT开机自启动，执行命令：
  ```
  systemctl enable qat
  ```
  如图：

  ```
  [root@localhost ~]# systemctl enable qat
  Created symlink from /etc/systemd/system/multi-user.target.wants/qat.service to /usr/lib/systemd/system/qat.service.
  [root@localhost ~]#
  ```

- 配置QAT取消开机自启动，执行命令：
  ```
  systemctl disable qat
  ```
  如图：

  ```
  [root@localhost ~]# systemctl disable qat
  Removed symlink /etc/systemd/system/multi-user.target.wants/qat.service.
  [root@localhost ~]#
  ```

- 查看QAT服务的运行状态，执行命令：
  ```
  systemctl status qat
  ```
  如图：

  ```
  [root@localhost ~]# systemctl status qat
  ● qat.service - start qat service
     Loaded: loaded (/usr/lib/systemd/system/qat.service; enabled; vendor preset: disabled)
     Active: active (exited) since Sat 2018-03-03 00:54:19 EST; 3min 35s ago
       Docs: http://01.org/
   Main PID: 63500 (code=exited, status=0/SUCCESS)

  Mar 03 00:54:19 localhost.localdomain systemd[1]: Started start qat service.
  Mar 03 00:54:19 localhost.localdomain systemd[1]: Starting start qat service...
  Mar 03 00:54:24 localhost.localdomain qat_service_ctl[63500]: Stopping all devices.
  Mar 03 00:54:25 localhost.localdomain qat_service_ctl[63500]: Starting all devices.
  Mar 03 00:54:25 localhost.localdomain qat_service_ctl[63500]: Processing /etc/dh895xcc_dev0.conf
  Mar 03 00:54:27 localhost.localdomain qat_service_ctl[63500]: Processing /etc/dh895xcc_dev1.conf
  Mar 03 00:54:29 localhost.localdomain qat_service_ctl[63500]: Checking status of all devices.
  Mar 03 00:54:29 localhost.localdomain qat_service_ctl[63500]: There is 2 QAT acceleration device(s) in the system:
  Mar 03 00:54:29 localhost.localdomain qat_service_ctl[63500]: qat_dev0 - type: dh895xcc,  inst_id: 0,  node_id: 1,  bsf: 8b:00.0,  #accel: 6 #engines: 12 state: up
  Mar 03 00:54:29 localhost.localdomain qat_service_ctl[63500]: qat_dev1 - type: dh895xcc,  inst_id: 1,  node_id: 1,  bsf: b1:00.0,  #accel: 6 #engines: 12 state: up
  [root@localhost ~]#
  ```

  说明

  QAT服务状态**不等于**QAT卡的运行状态。

- 查看所有QAT卡的状态信息，执行命令：
  ```
  /usr/bin/qat_service_ctl status
  ```
  如图所示，该命令行打印每一张QAT卡的运行状态，up状态表示运行正常：

  ```
  [root@localhost ~]# /usr/bin/qat_service_ctl status
  Checking status of all devices.
  There is 2 QAT acceleration device(s) in the system:
   qat_dev0 - type: dh895xcc,  inst_id: 0,  node_id: 1,  bsf: 8b:00.0,  #accel: 6 #engines: 12 state: up
   qat_dev1 - type: dh895xcc,  inst_id: 1,  node_id: 1,  bsf: b1:00.0,  #accel: 6 #engines: 12 state: up
  [root@localhost ~]#
  ```

26.2 测试验证

26.2.1 Openssl 异步加 QAT 引擎（RSA2048）

```
openssl-1.1 speed -engine qat -elapsed -async_jobs 72 --multi=2 rsa2048
```

```
[root@localhost ~]# openssl-1.1 speed -engine qat -elapsed -async_jobs 72 --multi=2 rsa2048
Forked child 0
Forked child 1
engine "qat" set.
engine "qat" set.
+DTP:2048:private:rsa:10
+DTP:2048:private:rsa:10
+R1:213147:2048:10.00
+R1:206398:2048:10.00
+DTP:2048:public:rsa:10
+DTP:2048:public:rsa:10
+R2:2048931:2048:10.00
+R2:2115928:2048:10.00
Got: +F2:2:2048:0.000048:0.000005 from 0
Got: +F2:2:2048:0.000047:0.000005 from 1
OpenSSL 1.1.0f-fips  25 May 2017
built on: reproducible build, date unspecified
options:bn(64,64) md2(char) rc4(16x,int) des(int) aes(partial) idea(int) blowfish(ptr)
compiler: gcc -DZLIB -DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_NO_STATIC_
SM -DVPAES_ASM -DBSAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DPADLOCK_ASM -DPOLY1305_ASM -DPURIFY
ffer-size=4 -grecord-gcc-switches -m64 -mtune=generic -Wa,--noexecstack
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000024s 0.000003s  42109.9 400000.0
```

26.2.2 Openssl 同步加 QAT 引擎（RSA2048）

```
openssl-1.1 speed -engine qat -elapsed rsa2048
```

```
[root@localhost ~]# openssl-1.1 speed -engine qat -elapsed rsa2048
engine "qat" set.
You have chosen to measure elapsed time instead of user CPU time.
Doing 2048 bit private rsa's for 10s: 22247 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 154450 2048 bit public RSA's in 10.00s
OpenSSL 1.1.0f-fips  25 May 2017
built on: reproducible build, date unspecified
options:bn(64,64) md2(char) rc4(16x,int) des(int) aes(partial) idea(int) blowfish(ptr)
compiler: gcc -DZLIB -DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSSL_N
SM -DVPAES_ASM -DBSAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DPADLOCK_ASM -DPOLY1305_ASM
ffer-size=4 -grecord-gcc-switches -m64 -mtune=generic -Wa,--noexecstack
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000449s 0.000065s   2224.7  15445.0
```

26.2.3 Openssl 软件通过 CPU 运算（RSA2048）

```
openssl-1.1 speed -elapsed rsa2048
```

```
[root@localhost ~]# declare -x OPENSSL_CONF=
[root@localhost ~]# openssl-1.1 speed -elapsed rsa2048
You have chosen to measure elapsed time instead of user CPU time.
Doing 2048 bit private rsa's for 10s: 16358 2048 bit private RSA's in 10.00s
Doing 2048 bit public rsa's for 10s: 563139 2048 bit public RSA's in 10.00s
OpenSSL 1.1.0f-fips  25 May 2017
built on: reproducible build, date unspecified
options:bn(64,64) md2(char) rc4(16x,int) des(int) aes(partial) idea(int) blowfish(p
compiler: gcc -DZLIB -DDSO_DLFCN -DHAVE_DLFCN_H -DNDEBUG -DOPENSSL_THREADS -DOPENSS
SM -DVPAES_ASM -DBSAES_ASM -DGHASH_ASM -DECP_NISTZ256_ASM -DPADLOCK_ASM -DPOLY1305_
ffer-size=4 -grecord-gcc-switches -m64 -mtune=generic -Wa,--noexecstack
                  sign    verify    sign/s verify/s
rsa 2048 bits 0.000611s 0.000018s   1635.8  56313.9
```

26.3 接口

QAT主要解决openssl加解密和压缩等运算效率问题，以独立引擎的方式加入openssl框架，以openssl接口对外呈现，接口主要包括以下部分。

26.3.1 RSA 加解密

```
int RSA_public_encrypt ( int flen, const unsigned char *from,unsigned char *to, RSA *rsa,int
padding ) ;
```



```
int RSA_private_decrypt(int flen, const unsigned char *from,unsigned char *to, RSA *rsa,int
padding);
```

26.3.2 SSL 服务端

```
SSLv3_server_method();
SSL_CTX_new();
SSL_CTX_set_accepted_Cas();
SSL_CTX_use_certificate_file();
SSL_CTX_use_PrivateKey_file();
SSL_CTX_check_private_key();
```

26.4 常见问题分析

- **如何知道当前物理机上是否有QAT卡**

  通过 lspci 命令查询是否有435的物理设备号，如果有输出结果，说明QAT卡已插入设备，如图：

```
[root@localhost ~]# lspci -d :435
8b:00.0 Co-processor: Intel Corporation DH895XCC Series QAT
b1:00.0 Co-processor: Intel Corporation DH895XCC Series QAT
[root@localhost ~]#
[root@localhost ~]# lsmod | grep qa
qat_dh895xcc           17996  0
intel_qat             200991  1 qat_dh895xcc
authenc                17776  1 intel_qat
uio                    19259  1 intel_qat
[root@localhost ~]#
```

- **如何判断QAT驱动是否运行**

  通过 lsmod | grep qa 命令，查看驱动模块是否加载，如图：

```
[root@localhost ~]# lsmod | grep qa
qat_dh895xcc           17996  1
intel_qat             200991  1 qat_dh895xcc
authenc                17776  1 intel_qat
uio                    19259  1 intel_qat
```

- **执行关闭QAT服务后，QAT驱动无法卸载**

  关闭QAT之前，应先将关联QAT的上层服务程序关闭，之后重新执行一遍QAT的开启服务，再关闭QAT就能够成功卸载模块。

- **修改QAT驱动配置文件后，QAT出现异常**

  检查配置文件的 NumProcesses 配置项，该配置项指QAT关联上层应用最多可以使用的进程数。当前硬件最大可配数值为256，应用程序进程数也不能超过这个数值。

- **如何判断QAT是否正在工作**

  在root用户下执行如下操作，能够看到QAT运行过程中的统计信息，如果统计信息发生变化，说明加解密运算已经成功下发QAT卡。统计文件为：

  /sys/kernel/debug/qat_dh895xcc_<PCI设备编号>/fw_counter，以实际QAT设备PCI编号为准，示例如下：


```
[root@localhost ~]# cat /sys/kernel/debug/qat_dh895xcc_8b\:00.0/fw_counters
+-----------------------------------------------+
| FW Statistics for Qat Device                  |
+-----------------------------------------------+
| Firmware Requests [AE  0]:             570708 |
| Firmware Responses[AE  0]:             570708 |
+-----------------------------------------------+
| Firmware Requests [AE  1]:             765944 |
| Firmware Responses[AE  1]:             765944 |
+-----------------------------------------------+
| Firmware Requests [AE  2]:             619938 |
| Firmware Responses[AE  2]:             619938 |
+-----------------------------------------------+
| Firmware Requests [AE  3]:             812597 |
| Firmware Responses[AE  3]:             812597 |
+-----------------------------------------------+
| Firmware Requests [AE  4]:             570968 |
| Firmware Responses[AE  4]:             570968 |
+-----------------------------------------------+
| Firmware Requests [AE  5]:             814706 |
| Firmware Responses[AE  5]:             814706 |
+-----------------------------------------------+
| Firmware Requests [AE  6]:             620665 |
| Firmware Responses[AE  6]:             620665 |
+-----------------------------------------------+
| Firmware Requests [AE  7]:             766526 |
| Firmware Responses[AE  7]:             766526 |
+-----------------------------------------------+
| Firmware Requests [AE  8]:             570776 |
| Firmware Responses[AE  8]:             570776 |
+-----------------------------------------------+
| Firmware Requests [AE  9]:             763905 |
| Firmware Responses[AE  9]:             763905 |
+-----------------------------------------------+
| Firmware Requests [AE 10]:             570823 |
| Firmware Responses[AE 10]:             570823 |
+-----------------------------------------------+
| Firmware Requests [AE 11]:             761282 |
| Firmware Responses[AE 11]:             761282 |
+-----------------------------------------------+
```


27 EulerOS 自研模块升级方式

ixgbevf-2.16.4-2 模块升级问题

问题描述

ixgbevf主要是内核模块，内核模块会放在系统的“/lib/modules/<kernel version>/kernel/”路径下； 由于ixgbevf模块中标识的内核版本与当前系统中的内核版本可能不一致（EulerOS KABI是兼容的，能保证驱动正常安装），导致不能加载正确的驱动版本。

解决方法

EulerOS在其内核目录下，建立链接到“/lib/modules/EulerOS”目录，并修改modprobe搜索路径优先级。当用户用modprobe时优先搜索“/lib/modules/<kernel version>/EulerOS”, 以确保正确加载ixgbevf内核模块。

操作步骤

ixgbevf在3.10.0-514.35.4.7.h35内核版本之前的环境下，进行模块加载时不兼容；需要手动创建“/lib/modules/uname -r/EulerOS”软链接到“/lib/modules/EulerOS”路径下，并且设置“/lib/modules/EulerOS/”目录到“/etc/depmod.d/depmod.conf”的检索目录列表中，此时modprobe ixgbevf模块方可正常加载。

28 SMR 硬盘支持

背景介绍

SMR（Shingled Magnetic Recording）作为下一代被用于增加硬盘单位面积内的存储容量的技术，在将来的一段时间内，将会使得磁盘容量井喷式的爆发增长。SMR硬盘降低了单位面积的存储成本，日后会越来越多的被使用在冷存储的场景中。EulerOS V2.0SP5中增加了对此种类型硬盘的支持。

约束限制

- SMR硬盘要求ZONE内顺序写，EulerOS V2.0SP5不提供顺序保证机制，由业务保证IO的顺序性。
- 对于SMR硬盘不提供文件系统。
- 对于SMR硬盘不支持软RAID。

使用方法

- 在服务器中接入SMR硬盘后，系统启动会自动识别。
- 新增两个ioctl操作SMR硬盘的方式：report zones和reset zones，具体见**接口说明**。
- 读写IO接口与普通硬盘一致，例如使用libaio引擎。

接口说明

新增ioctl调用report zones操作的接口：

```
ioctl(fd, BLKREPORTZONE, struct blk_zone_report *rep)
```

新增ioctl调用reset zones操作的接口：

```
ioctl(fd, BLKRESETZONE, struct blk_zone_range *range)
```

上述上层应用程序开发所用的新增ioctl接口，其参数中的宏定义及结构体声明在/usr/include/linux/blkzoned.h内。

```
$ cd /usr/include/linux/
$ ls | grep blkzoned.h
blkzoned.h
```

libaio的使用可以参考：**https://github.com/dev-lyr/my_linux_coding/wiki/IO(%E4%B8%89)%E5%BC%82%E6%AD%A5IO(libaio)**

使用如下命令查看SMR硬盘顺序写区域的中每个zone含有sector的个数，<sdx>为SMR硬盘对应的磁盘名字，此说明文件对所有用户均为只读文件：

```
$ cat /sys/block/<sdx>/queue/chunk_sectors
```

使用如下命令查看SMR硬盘的类型，<sdx>为SMR硬盘对应的磁盘名字，此说明文件对所有用户均为只读文件：

```
$ cat /sys/block/<sdx>/queue/zoned
```


29 TSX 特性

概述

程序员在编程中往往倾向使用spin_lock锁住大颗粒的临界区，导致性能问题。TSX针对此类问题进行优化，通过硬件指令判别临界区是否冲突，在非冲突的情况下可以显著提高并发性能。

概念

Transactional Synchronization Extensions （TSX）事务同步扩展指令集：是Intel CPU新增的硬件事务内存机制，它将具有并发冲突的临界区组织在事务内存中，提供乐观的多线程并发控制。允许程序员使用细粒度线程锁定，进而提高多线程效率和性能。主要由RTM和HEL技术组成。

Restricted Transactional Memory（RTM）受限事务内存：显式的事务开始、提交和取消（XBEGIN、XEND、XABORT）指令，在XBEGIN处指定fallback路径处理函数，受限的TM实现（即使提交时不冲突，也不保证提交成功，有很多场景会落到fallback路径处理。

Hardware Lock Elision （HLE） 硬件消锁：在原有锁的基础上，增加XAQUIRE和XRELEASE前缀。CPU欺骗应用，告诉应用已经获取锁（实际没有），继续执行。如果在XRELEASE处发现冲突，重新执行XAQUIRE和XRELEASE之间的代码区（但是忽略XAQUIRE和XRELEASE前缀）。

使用场景

后向兼容：不支持TSX指令集的CPU会自动忽略XAQUIRE和XRELEASE前缀。

TSX特性需要硬件支持，可运行在通用X86平台和统一存储X86平台上。对于没有TSX硬件的场景，提供了后向兼容，但性能不会提升。

Dorado 使用了osax封装的用户态接口,不会使用内核态接口；EVR 虚拟机内不使用TSX接口。

使用限制

1. 在临界区假冲突的场景性能提升明显， 如果真冲突比较多，会导致事务abort增加。多读少写的场景更合适。
2. RTM基于L1 cache line检测冲突。操作临界区所需的内存大小会影响性能。

L1缓存容量有限，限制事务不能过大；可以依据具体的cpu，把临界区限制在一定的范围。

当前平台cache line大小是64Byte, 默认允许的临界区最大为8K Byte.

3. 在对应的锁区间避免使用导致事务中断的指令：
   - CPUID：获取CPU的基本信息。例如：boot_cpu_has()
   - PAUSE：循环等待时减少内存乱序并降低耗电量。例如：
     ```
     #define _mm_pause() \
     ({ \
     __asm__ __volatile__("pause" ::: "memory"); \
     })
     ```
   - x87 浮点运算指令。例如：sqrt()
   - MMX多媒体扩展指令集。例如：下列头文件引入的库函数
     ```
     #include <mmintrin.h> //MMX
     #include <xmmintrin.h> //SSE(include mmintrin.h)
     #include <emmintrin.h> //SSE2(include xmmintrin.h)
     #include <pmmintrin.h> //SSE3(include emmintrin.h)
     #include <tmmintrin.h>//SSSE3(include pmmintrin.h)
     #include <smmintrin.h>//SSE4.1(include tmmintrin.h)
     #include <nmmintrin.h>//SSE4.2(include smmintrin.h)
     #include <wmmintrin.h>//AES(include nmmintrin.h)
     #include <immintrin.h>//AVX(include wmmintrin.h)
     #include <intrin.h>//(include immintrin.h)
     ```
   - 释放CPU 例如：yield()
   - 其他性能优化相关参考**intel手册**第12章
4. 避免在事务中进行内存分配和释放；避免页故障导致事务abort。
5. 内置的编译器优化会使高速缓存集中，导致事务冲突可能性变大 。
6. 临界区执行时间应尽量短，避免使用mdelay等函数
7. 使用skylake及以上的intel cpu，支持TSX指令；同时gcc支持4.8以上版本，否则需要显式的使用编译参数才能使能HLE。
8. 为了性能考虑，不在接口内进行入参检测，参数合法性由用户保证。

接口描述

- 初始化TSX自旋锁接口

  接口原型：

  void tsx_spin_lock_init(tsxlock_t * lock);

  接口功能：

  初始化TSX自旋锁

  返回值：无

- 获得TSX自旋锁接口

  接口原型：

  void tsx_spin_lock(tsxlock_t * lock);

  接口功能：

  获取TSX自旋锁

  返回值：无

- 释放TSX自旋锁接口

  接口原型：

  void tsx_spin_unlock(tsxlock_t * lock);

接口功能：

释放TSX自旋锁

返回值：无

使用样例

```
#include "tsx_spin.h"
tsxlock_t lock; //定义spin对象
tsx_spinlock_init(&lock);
tsx_spin_lock(&lock);
    DO_CRITICAL //临界区代码
tsx_spin_unlock(&lock);
```

使用 Perf 工具查看 TSX 性能

基本用例：

```
perf record -e r21d0,tx-start,tx-abort,cycles -a sleep 1
perf report
```

参数说明：

- record 记录数据到文件
- report 根据记录是数据生产报告
- -e 指定event。系统支持的事件可以用perf list查看。其中r21d0是CPU性能计数器 **mem_uops_retired**
- -a 记录整个机器的指令

```
perf stat -T -a sleep 1
```

- -T 统计事务。
- -a 收集整个机器的指令。

30 监听队列哈希桶元素可配

概述

本节主要介绍关于如何配置内核监听队列哈希桶元素个数（默认元素个数为32）。

约束限制

- 桶元素个数上限64*1024。
- 元素个数越多，会影响内存占用。

使用方法

若要配置哈希桶元素个数，在/boot/grub2/grub.cfg对应的内核cmdline中，添加参数“tlhash_entries=xxxx”即可，其中“xxxx”表示用户配置的桶元素个数。

例如：配置tlhash_entries=6000，表示用户配置的桶元素个数为6000。

31 pam_ssh_agent_auth 特性使用方法

pam_ssh_agent_auth用来配置pam认证的时候，可以ssh代理客户端管理的秘钥认证，不需要输入密码。可以实现ssh免密登录，sudo免密执行。使用方法如下：

安装 pam_ssh_agent_auth 软件包

使用如下命令进行安装：

```
rpm -ivh  pam_ssh_agent_auth
```

或

```
yum install pam_ssh_agent_auth
```

使用 ssh-keygen 生成公私钥

使用ssh-keygen生成公私钥：

```
[linyan@localhost ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/linyan/.ssh/id_rsa):
Created directory '/home/linyan/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/linyan/.ssh/id_rsa.
Your public key has been saved in /home/linyan/.ssh/id_rsa.pub.
The key fingerprint is:
2e:05:f2:2f:00:03:58:94:0f:01:d3:72:4b:75:1f:2d linyan@localhost.localdomain
The key's randomart image is:
+--[ RSA 2048]----+
|==++. . ..       |
|oo*  . .E..      |
| +o+. . ..       |
|  .o.o .         |
|    . . S        |
|     . +         |
|      o o        |
|       o         |
|                 |
+-----------------+
```

修改 pam 与 ssh 配置文件

1. 把公钥文件~/.ssh/id_rsa.pub内容写入 ~/.ssh/authorized_keys。
2. 在/etc/ssh/sshd_config中设置“AllowAgentForwarding yes”并重启sshd服务。

3. 在/etc/pam.d/sudo中插入：“auth sufficient pam_ssh_agent_auth.so file=~/.ssh/authorized_keys ”。

   说明

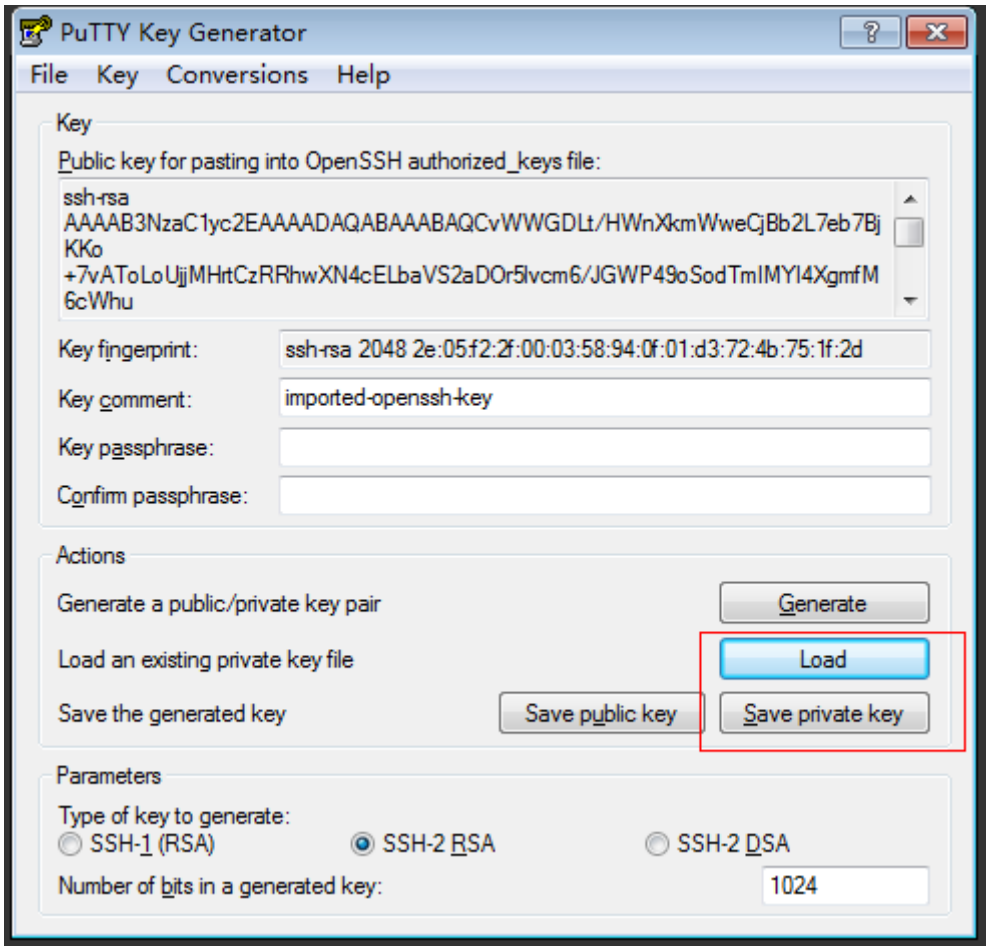
   这行配置要写在其他auth规则之前，否则不生效，建议写到第一行。

使 ssh 客户端配置生效

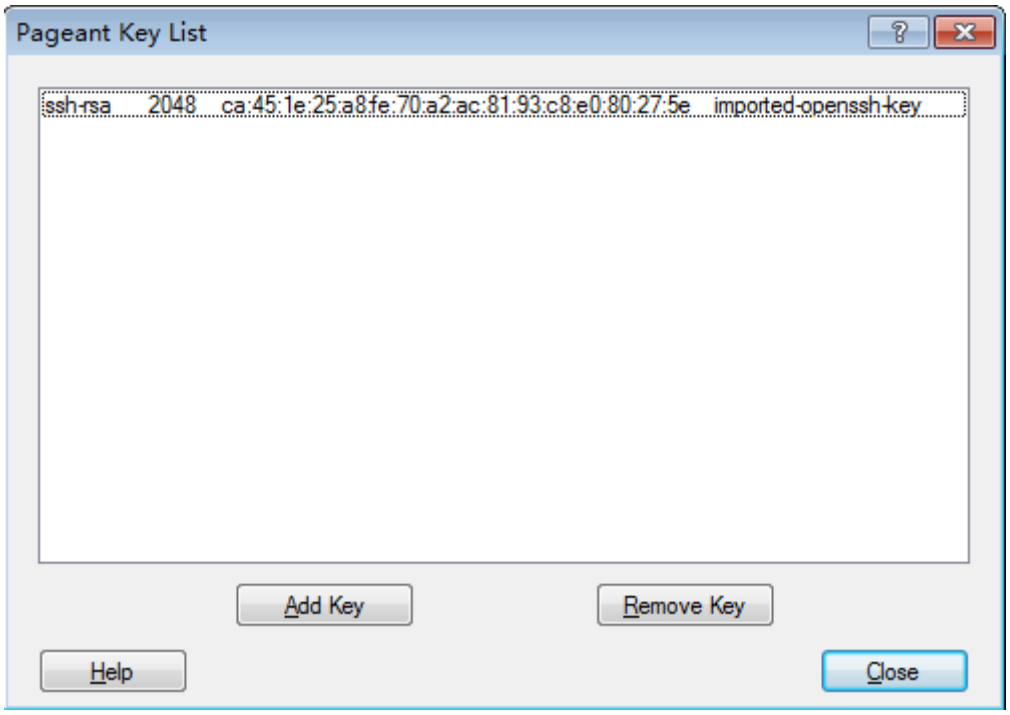
说明

这里以putty软件举例，其他软件类似。

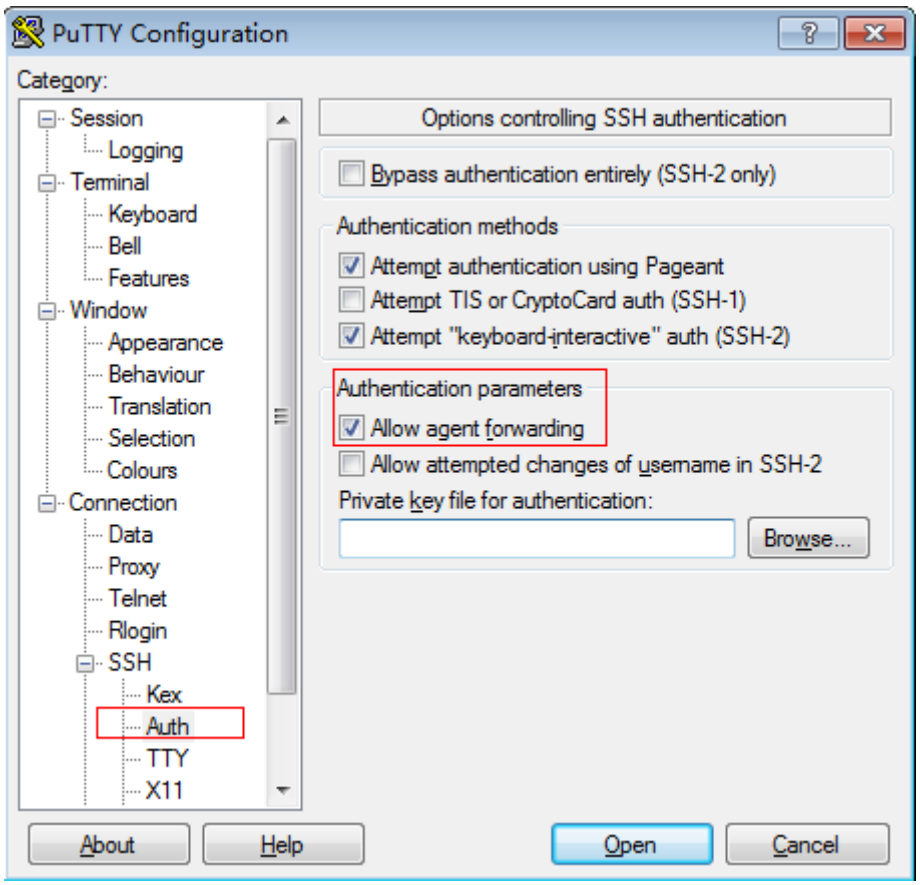
1. 由于~/.ssh/id_rsa 私钥格式 putty无法识别，需要使用puttygen转换一下。id_rsa保存到本地，使用 puttygen load进去，然后点击 Save private key 保存成id.ppk。



2. 在PC上启动ssh-agent（如putty的pageant），添加秘钥。点击Add Key把上个步骤生成的id.ppk导入进去。

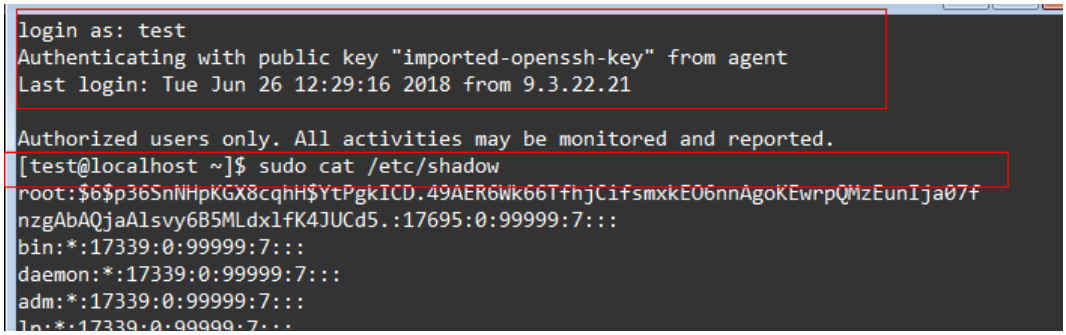


3. 在PC上启动ssh-client上（如putty）使能“Allow agent forwarding”，连接目标系统。



免密登录

设置完成后，使用putty远程登录系统，就可以实现ssh免密登录。在系统中执行命令时，可以使用sudo免密执行。如下图所示：



说明

EulerOS sshd默认配置使用“MACs hmac-sha2-256,hmac-sha2-512”，早期的putty版本不支持这两个加密协议。碰到登录不上的问题，请把/etc/ssh/sshd_config中的这行配置注释或删除，重新启动sshd服务（使用命令systemctl restart sshd），然后再登录。

32 内核特性

32.1 整体介绍

概述

为了保证EulerOS系统的安全性，EulerOS内核中提供了一系列安全机制。通过限制内存的大小、功能实现访问与互动以及将内存空间地址进行随机化来增大入侵者攻击的难度，从而降低数据外泄的风险。下面对一些内核特性做一个简单的描述。

实现

表 32-1 内核特性

特性名称	功能描述
SELinux	提供SELinux内核态功能和用户态libselinux。
yama	可提供ptrace保护特性和软链接、硬连接限制功能。
Audit	audit是linux中的审计系统，可以提供给其它子系统进行审计、日志记录使用。 典型的应用场景：SeLinux，SeLinux必须要使用此功能进行avc审计消息记录。
硬件特性SMAP	禁止内核态访问用户态数据。 硬件支持：x86中引入了SMAP（Supervisor Mode Access Prevention），阻止管理者模式访问。
Modules RO/NX	将新插入的内核模块的代码段设置为只读，数据段设置为不可执行。

特性名称	功能描述
vdso ASLR	vdso地址随机化。
stack ASLR	栈地址随机化。
KASLR	内核地址随机化。
Libs/mmap ASLR	libs/mmap地址随机化。
Stack Protector Strong/Stack Protector	● -fstack-protector只为局部变量中长度超过8-byte（含8-byte）的char数组的函数插入保护代码。 ● -fstack-protector-strong，满足以下三个条件都会插入保护代码： 1. 局部变量的地址作为赋值语句的右值或函数参数。 2. 局部变量包含数组类型的局部变量，不管数组的长度。 3. 带register声明的局部变量。
Write-protect kernel .rodata sections	对内核的 .rodata段的写保护。 ● 功能说明：该功能将内核内存划分为多个逻辑区段，并限定每个区段的页访问权限。将代码标记为只读可执行。将数据区段标记为不可执行，并进一步将其细分为只读区段和读写区段。该功能通过配置选项CONFIG_DEBUG_RODATA启用。 ● 防御范围：通过细分内存使用方式（只读，读写，是否可执行），增加攻击者对内核内存的控制难度。
/dev/mem protection	限制通过/dev/mem设备直接操作内存，开关使能后，只允许通过/dev/mem访问外设映射的内存。
SYN cookies	防范SYN flooding攻击。 SYN Cookie是对TCP服务器端的三次握手做一些修改，专门用来防范SYN Flood攻击的一种手段。它的原理是，在TCP服务器接收到TCP SYN包并返回TCP SYN + ACK包时，不分配一个专门的数据区，而是根据这个SYN包计算出一个cookie值。这个cookie作为将要返回的SYN ACK包的初始序列号。当客户端返回一个ACK包时，根据包头信息计算cookie，与返回的确认序列号（初始序列号 + 1）进行对比，如果相同，则是一个正常连接，然后，分配资源，建立连接。
Module signature verification	对内核模块进行签名校验。
Kernel Page Table Isolation	● 功能说明：让内核和用户态不再共享页表，预防通过缓存预测碰撞方式的缓存侧信道攻击，放置泄露敏感数据 ● 防御举例：防止内核数据泄露到用户态，例如防御2018年CPU芯片漏洞：Meltdown与Spectre。详见**37.12.2 Spectre和Meltdown漏洞补丁开关**。 链接：**https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?h=v4.19-rc7&id=0617052ddde355ee663b2f048e67dd381e5ebd6a**
seccomp-bpf	提供seccomp BPF API对用户态可调用的系统调用进行过滤。

特性名称	功能描述
execveat系统调用	该系统调用是从高版本内核中回合的特性，运作方式与execve相同，不同之处在于，如果第二个参数pathname为相对路径，将被解释为相对于第一个参数dirfd文件描述符引用的目录，而不是相对于调用的当前工作目录。
Disable firmware auto-loading user mode helper 禁用固件自动加载用户模式助手	固件自动化加载助手，在默认kernel路径加载失败时，允许从用户空间加载备用的内核固件。如果该选项被打开，可以作为系统热备份机制，增强系统的稳定性。但是同时也带来了一定的安全风险。
Restrict access to kernel memory 限制对内核空间内存的访问	kernel中的/dev/kmem文件被直接映射到kernel的虚拟内存中，如果攻击者获得了root权限，可以通过此方式直接访问内核虚拟内存，这将会产生灾难性后果。限制对/dev/kmem可以削减攻击者造成的影响。一般情况下，/dev/kmem极少被用到，经常被用于调试，用户空间的应用用不到/dev/kmem，如果需要使用，则必须进行授权使用。 Linux 4.4内核及更新版本，默认已经将此项禁用
DMESG Restrictions 限制对dmesg的访问	dmesg是内核日志文件，当攻击者试图为漏洞开发“随处运行”攻击时，经常会对dmesg输出进行分析。通过将dmesg输出视为敏感信息，攻击者无法使用此信息。
内核CAP项说明	Capabilities机制，是在Linux内核2.2之后引入的。它将root用户的权限细分为不同的领域，可以分别启用或禁用。从而，在实际进行特权操作时，便会检查是否具有该特权操作所对应的capabilities，并以此为依据，决定是否可以执行特权操作。

32.2 CAP_AUDIT_READ

场景介绍

内核增加通过netlink多播审计信息，提供给除auditd之外的用户态处理程序。然而审计信息属于敏感的管理员信息，通过CAP_AUDIT_READ来控制读取权限。

约束限制

内核需要支持该特权CAP_AUDIT_READ。

使用指导

- 使用方法同其他**Capabilities**，更详细介绍可以通过man 7 capabilities查看。
- 容器运行时可以通过--cap-add AUDIT_READ增加特权，若容器运行需要接收内核审计事件，还需要添加--net host，否则会被隔离。
- 读取netlink多播的审计信息，可以参考程序：**http://people.redhat.com/rbriggs/audit-multicast-listen/audit-multicast-listen.c**。

32.3 execveat 系统调用

场景介绍

需要执行相对于某文件描述符的路径中的程序。

约束限制

内核增加对execveat系统调用的支持。当前只能通过系统调用号调用，不能直接调用execveat函数。

使用指导

该系统调用运作方式与execve相同，不同之处在于第二个参数pathname的使用：

1. 如果pathname为相对路径，将被解释为相对于第一个参数dirfd文件描述符引用的目录，而不是相对于调用的当前工作目录。

   如果pathname为相对路径，但dirfd是特殊值AT_FDCWD，则解释为相对于当前工作目录，与execve一样。
2. 如果pathname为绝对路径，则忽略dirfd的值。
3. 如果pathname是空字符串，并且最后一个参数flags置为AT_EMPTY_PATH，则dirfd参数指定可执行文件而不是路径。

33 IPv6 使用差异说明（vs IPv4）

33.1 约束限制

- chrony支持全局地址（global address），不支持链路本地地址（link-local address）。
- Firefox支持通过http/https协议访问全局地址（global address），不支持链路本地地址（link-local address）。

33.2 配置说明

33.2.1 设置接口设备 MTU 值

概述

IPv6场景中会发现整个路由路径中的最小mtu的值作为当前链接的PMTU的值，源端根据PMTU的值确定是否进行分片发送，而在整个路径中的其它设备将不再需要进行分片处理，从而可以降低中间路由设备的负载大小。其中IPv6 PMTU设置的最小值为1280。

设置接口设备的 mtu

如果在配置了IPv6地址的接口上设置mtu的值小于1280（IPv6 PMTU设置的最小值），则会导致该接口的IPv6地址被删除。并且无法再次添加IPv6地址。所以在IPv6场景中，对接口设备的mtu的配置一定要大于等于1280。具体现象如下：

```
[root@localhost ~]# ip addr show ens4
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic ens4
       valid_lft 38663sec preferred_lft 38663sec
```



```
    inet6 2001:222::2/64 scope global
       valid_lft forever preferred_lft forever
[root@localhost ~]# ip link set dev ens4 mtu 1200
[root@localhost ~]# ip addr show ens4
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1200 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic ens4
       valid_lft 38642sec preferred_lft 38642sec
[root@localhost ~]# ip addr add 2001:222::2/64 dev ens4
RTNETLINK answers: No buffer space available
[root@localhost ~]# ip link set dev ens4 mtu 1500
[root@localhost ~]# ip addr show ens4
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic ens4
       valid_lft 38538sec preferred_lft 38538sec
[root@localhost ~]# ip addr add 2001:222::2/64 dev ens4
[root@localhost ~]# ip addr show ens4
3: ens4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 52:54:00:62:xx:xx brd ff:ff:ff:ff:xx:xx
    inet 10.41.125.236/16 brd 10.41.255.255 scope global noprefixroute dynamic ens4
       valid_lft 38531sec preferred_lft 38531sec
    inet6 2001:222::2/64 scope global
       valid_lft forever preferred_lft forever
```

33.2.2 有状态自动配置 IPv6 地址

概述

IPv6与IPv4都可以通过DHCP的方式获得IP地址。IPv6地址有两种配置方式：无状态自动配置和有状态自动配置。

- 无状态自动配置

  不需要DHCP服务进行管理，设备根据网络RA（路由公告）获得网络前缀，或者link-local地址为固定fe80::。而接口ID则根据ifcfg配置IPV6_ADDR_GEN_MODE的具体设置来进行自动获得：

  a. IPv6_ADDR_GEN_MODE="stable-privacy" 则根据设备及网络环境来确定一个随机接口ID。

  b. IPv6_ADDR_GEN_MODE="EUI64" 则根据设备MAC地址来确定接口ID。

- 有状态自动配置：需要DHCP服务器进行管理分配，服从DHCPv6协议来从DHCPv6服务器端租赁IPv6地址。

  在有状态自动配置IPv6地址时，DHCPv6服务端可以通过客户端设置的vendor class将客户端进行分类，不同类别分配不同地址段的IPv6地址。在IPv4场景中，客户端可以直接用dhclient的-V选项来设置vendor-class-identifier，DHCP服务端在配置文件中根据vendor-class-identifier来对客户端进行分类处理。而IPv6场景中，如果使用同样的方法对客户端分类，则分类并不会生效。

  ```
  dhclient -6 <interface> -V <vendor-class-identifier string> <interface>
  ```

  这是由于DHCPv6和DHCP协议存在较大差异，DHCPv6的可选项中使用vendor-class-option替代了DHCP中的vendor-class-identifier。而dhclient的-V选项并不能设置vendor-class-option。

有状态自动配置 IPv6 地址时 dhclient 设置 vendor class 方法

- 在客户端使用配置文件方式添加对vendor class的设置，使用方法如下：

  客户端配置文件（/etc/dhcp/dhclient6.conf），文件位置可以自定义，在使用时需要通过dhclient -cf选项来指定配置文件：

  ```
  option dhcp6.vendor-class code 16 = {integer 32, integer 16, string};
  interface "ens4" {
          send dhcp6.vendor-class <Enterprise-ID number> <vendor class string length> <vendor class string>;
  }
  ```

  说明

  - <Enterprise-ID number>，32位整型数字，企业标识号，企业通过IANA注册。
  - <vendor class string length>，16位整型数字，vendor class字符串长度。
  - <vendor class string>，要设置的vendor class字符串，例如：“HWHW”。

  客户端使用方法：

  ```
  dhclient -6 <interface> -cf /etc/dhcp/dhclient6.conf
  ```

- DHCPv6服务端配置文件（/etc/dhcp/dhcpd6.conf），需要dhcpd -cf选项来指定该配置文件：

  ```
  option dhcp6.vendor-class code 16 = {integer 32, integer 16, string};
  subnet6 2019:4:12:ffff::/64 {
          class "hw" {
                  match if substring ( option dhcp6.vendor-class, 6, 10 ) = "HWHW";
          }
          pool6 {
                  allow members of "hw";
                  range6 2019:4:12:ffff::ff10 2019:4:12:ffff::ff20;
          }
          pool6 {
                  allow unknown clients;
                  range6 2019:4:12:ffff::100 2019:4:12:ffff::120;
          }
  }
  ```

  说明

  substring (option dhcp6.vendor-class, 6, 10) 其中子字符串的开始位置为6，因为前面包含4个字节的<Enterprise-ID number>和2个字节的<string length>。而子字符串的结束位置位：6+<vendor class string length>。这里vendor class string为“HWHW”，字符串的长度为4，所以子字符串的结束位置为6+4=10。用户可以根据实际需要来确定<vendor class string>及相应的<vendor class string length>。

  服务端使用方法：

  ```
  dhcpd -6 <interface> -cf /etc/dhcp/dhcpd6.conf <interface>
  ```

33.2.3 内核支持 socket 相关系统调用

概述

IPv6地址长度扩展到128比特，所以有足够的IPv6地址可供分配使用。同时IPv6头相比IPv4头进行了简化，并增强了IPv6的自动配置功能。IPv6地址分为单播地址，组播地址和任意播地址。常用的单播地址又包含：链路本地地址（link-local address），唯一本地地址（Unique local address）和全局地址（global address）。由于IPv6的全局地址十分充足，唯一本地地址一般不被使用（其前身为站点本地地址（site-local address），已于2004年被废弃）。当前主要使用的单播地址为：链路本地地址（link-local address）和全局地址（global address）。当前内核支持socket系统调用，在使用单播地址的链路本地地址和全局地址时存在差异。

link-local 地址和 global 地址在 socket 调用时的差异

RFC 2553： Basic Socket Interface Extensions for IPv6 定义sockaddr_in6的数据结构如下；

```
struct sockaddr_in6 {
    uint8_t         sin6_len;       /* length of this struct */
    sa_family_t     sin6_family;    /* AF_INET6 */
    in_port_t       sin6_port;      /* transport layer port # */
    uint32_t        sin6_flowinfo;  /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id;  /* set of interfaces for a scope */
};
```

说明

sin6_scope_id： 32位整型，对于链路本地地址（link-local address），对于链路范围的sin6_addr，它可以用来标识指定的接口索引号。如果是站点范围的sin6_addr，则用来作为站点的标识符（站点本地地址已被抛弃）。

在使用link-local地址进行socket通信时，在构造目的地址时，需要制定该地址所对应的接口索引号。一般可以通过if_nametoindex函数将接口名转化为接口索引号。具体方式如下，

```
int port = 1234;
int sk_fd;
int iff_index = 0;
char iff_name[100] = "eth0";
char * ll_addr[100] = "fe80::123:456:789";
struct sockaddr_in6 server_addr;

memset(&server_addr,0,sizeof(structsockaddr_in6));
iff_index=if_nametoindex(iff_name);

server_addr.sin6_family=AF_INET6;
server_addr.sin6_port=htons(port);
server_addr.sin6_scope_id=iff_index;
inet_pton(AF_INET6, ll_addr, &(server_addr.sin6_addr));

sk_fd=socket(AF_INET6, SOCK_STREAM, IPPROTO_TCP);
connect(sk_fd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr_in6));
```

33.2.4 IPv4 的 dhclient 守护进程持久化配置

概述

通过Network服务来管理网络服务时，如果接口ifcfg-<interface-name>配置文件中配置了DHCP方式获得IP地址，则相应地Network服务会拉起dhclient守护进程来通过DHCP协议方式来从DHCP服务器获取IP地址。

dhclient提供了"-1"选项来决定dhclient进程在未获得DHCP服务响应时，是会不断持久化尝试请求地址还是会尝试时间超时后退出。针对IPv4的dhclient守护进程，可以在ifcfg-<interface-name>配置文件中设置PERSISTENT_DHCLIENT来决定是否设置IPv4的dhclient进程的持久化。

约束限制

1. 当dhclient进程在运行中被杀死，network服务无法自动将其拉起，可靠性需要用户自己保障。
2. 配置了持久化选项PERSISTENT_DHCLIENT，需要确保有相应的DHCP服务器。如果在拉起network时无可用DHCP服务器，dhclient进程不断尝试发送请求包但无回应，则会导致network服务卡死直到network服务超时失败。由于network服务在

拉起多个网卡的IPv4 dhclient进程时，是通过串行的方式来拉起的。如果有网卡配置了持久化而DHCP服务器没有准备好，则会导致network服务在给该网卡获取IPv4地址超时卡死，进而导致后续网卡无法获得IPv4/IPv6地址。

以上两种约束限制是特殊的应用场景，需要用户自己进行可靠性保障。

IPv4 DHCP 和 IPv6 DHCPv6 方式获取地址的配置差异

可以通过配置接口ifcfg-<interface-name>参数来分别实现IPv4和IPv6通过DHCP/DHCPv6协议来动态获取IP地址，具体配置说明如下；

```
BOOTPROTO=none|bootp|dhcp
DHCPV6C=yes|no
PERSISTENT_DHCLIENT=yes|no|1|0
```

- BOOTPROTO： none表示静态配置IPv4地址，bootp|dhcp则会拉起DHCP dhclient来动态获取IPv4地址。
- DHCPV6C： no表示静态配置IPv6地址，yes则会拉起DHCPv6 dhclient来动态获取IPv6地址。
- PERSISTENT_DHCLIENT：no|0表示IPv4的dhclient进程配置为“非持久化”，当dhclient向DHCP服务器发送一次请求报文而无响应，则会间隔一段时间后退出，退出值为2。yes|1则表示IPv4的dhclient进程配置为“持久化”，dhclient会向DHCP服务器反复发送请求报文。**如果没有配置PERSISTENT_DHCLIENT项，则IPv4的dhclient会默认设置为“持久化”**。

  说明

  PERSISTENT_DHCLIENT配置只针对IPv4生效，对IPv6相关dhclient -6进程不生效，IPv6默认不进行持久化配置。

EulerOS 持久化配置策略与开源社区持久化配置策略的差异说明

考虑到EulerOS的具体需求，当前EulerOS持久化配置策略与开源社区持久化策略有一定的差异，具体差异如下表：

表 33-1 EulerOS 持久化配置策略与开源社区持久化配置策略的差异

EulerOS	开源社区方案
未配置PERSISTENT_DHCLIENT时，默认IPv4 dhclient进程都设置为“持久化”。IPv6的dhclient -6进程固定为“非持久化”	未配置PERSISTENT_DHCLIENT时，默认IPv4和IPv6的dhclient进程设置为“非持久化”

- “持久化”定义：不设置“-1”选项就意味着配置了持久化，进程在没有获得地址租约时会按照一定的时间间隔变化进行不断发包尝试。
- 由于EulerOS持久化配置策略是默认进行持久化配置，所以如果用户**配置了BOOTPROTO=dhcp**，就**默认用户知晓当前环境中存在DHCP服务器**。如果配置**BOOTPROTO=dhcp**，就**默认用户知晓当前环境中存在DHCP服务器**且默认没有配置PERSISTENT_DHCLIENT（或设置为持久化PERSISTENT_DHCLIENT=yes|1），而环境中实际并没有DHCP服务器，则此时重启network服务，IPv4 dhclient进程由于没有收到DHCP服务器应答报文而不断重复发送请求报文（以一定的时间间隔）。此时network服务表现为：**network服务卡住一段时间，直到dhclient进程超时休眠（ 默认timeout时间为1分钟，而network服务的总的默认超时时间为5分**

钟，如果有较多网卡没有对应设dhcp服务器，将可能导致network服务超时失败），IPv4的dhclient进程并不退出而是会持久化存在。而如果没有IPv6的DHCPv6服务器，则dhclient -6进程会在超时后退出，并且返回值为1。

- network服务本身无法进行周期性检测dhclient进程在不在，只负责根据ifcfg配置来确定拉起来的dhclient进程是否包含“-1”选项。如果运行中dhclient进程由于其它原因被杀死，network服务不会进行监测及拉起操作。而如果此时进行重启network服务，则会重新进行拉起dhclient进程。

33.2.5 iproute 相关命令配置 IPv4 与 IPv6 时的差异说明

概述

由于IPv4和IPv6是两个不同的协议标准，iproute相关命令在使用方法上存在一定的差异。本章节主要梳理iproute包中用户经常使用到命令在IPv4和IPv6使用方面的差异，从而可以更好地指导用户使用iproute包中相关命令。

IPv6 地址的生命周期

ipv6状态	解释
tentative	临时状态：刚添加地址还处于地址重复检测DAD过程
preferred	首选状态：完成DAD过程，没有收到相应的NA报文，表示该地址没有冲突。
deprecated	弃用状态：地址有一定的使用时限（valid_lft和preferred_lft），preferred_lft到期后地址会变化deprecated状态。 该状态下的地址不能用于创建新的连接，但是原有的连接可以继续使用。
invalid	无效状态：使用时限超过preferred_lft一段时间后仍然没有成功进行租约续约，则valid_lft时间到后地址状态会被设置为invalid，表示该地址不可以再被使用。

其它说明：

- preferred_lft：preferred lifetime，地址为首选状态的寿命，preferred_lft没有到期的地址可以用于正常通信使用，若有多个preferred地址则按照内核具体机制选择地址。
- valid_lft： valid lifetime，地址有效的寿命，在[preferred_lft, valid_lft]时间段内该地址不能被用于新建连接，已经创建的连接继续有效。

ip link 命令

命令：

```
ip link set IFNAME mtu MTU
```

IPv6中PMTU的最小值为1280，如果mtu值设置小于1280则会导致IPv6地址丢失。其它设备无法ping通该ipv6地址。

ip addr 命令

1. 命令：
   ```
   ip [-6] addr add IFADDR dev IFNAME
   ```
   添加IPv6地址可以选择添加-6选项也可以不添加，ip addr命令会根据具体地址类型来判断是ipv4地址还是ipv6地址。

   如果指定“-6”选项，但是IFADDR 是ipv4地址则会有错误返回。

2. 命令：
   ```
   ip [-6] addr add IFADDR  dev IFNAME [home|nodad]
   ```
   [home|nodad] 选项只针对IPv6地址有效。

   - home：将该地址指定为RFC 6275中定义的家庭地址。（这是移动节点从家庭链路获取的地址， 是移动节点的永久地址，如果移动节点保持在相同的归属链路中，则各种实体之间的通信照常进行。）
   - nodad：配置该项（仅限IPv6）添加此地址时不执行重复地址检测DAD（RFC 4862）。如果一台设备上多个接口通过nodad配置了多个相同的ipv6地址，则会按照接口顺序使用该ipv6地址。同一个接口上不能添加一个nodad一个非nodad的相同ipv6地址。因为两个地址时一样的，所以会报“RTNETLINK answers: File exists”。

3. 命令：
   ```
   ip [-6] addr del IFADDR dev IFNAME
   ```
   删除Pv6地址可以选择添加-6选项也可以不添加，ip addr del命令会根据具体地址类型来判断是ipv4地址还是ipv6地址。

4. 命令：
   ```
   ip [-6] addr show dev IFNAME [tentative|-tentative|deprecated|-deprecated|dadfailed|-dadfailed|temporary]
   ```
   - 不指定-6选项，则会同时打印IPv4和IPv6地址。指定-6选项则只打印IPv6地址。
   - [tentative|-tentative|deprecated|-deprecated|dadfailed|-dadfailed|temporary]，这些选项只针对IPv6，可以根据IPv6地址状态对地址进行筛选查看。
     i. tentative：（仅限IPv6）仅列出尚未通过重复地址检测的地址。
     ii. -tentative：（仅限IPv6）仅列出当前未处于重复地址检测过程中的地址。
     iii. deprecated：（仅限IPv6）仅列出已弃用的地址。
     iv. -deprecated：（仅限IPv6）仅列出未弃用的地址。
     v. dadfailed：（仅限IPv6）仅列出重复地址检测失败的地址。
     vi. -dadfailed：（仅限IPv6）仅列出未重复地址检测失败的地址。
     vii. temporary：（仅限IPv6）仅列出临时地址

ip route 命令

1. 命令：
   ```
   ip [-6] route add ROUTE [mtu lock MTU]
   ```
   - -6选项：添加IPv6路由可以选择添加-6选项也可以不添加，ip route命令会根据具体地址类型来判断是IPv4地址还是IPv6地址。
   - mtu lock MTU：锁定路由的MTU值。如果不锁定MTU，则MTU的值则可能在PMTUD过程中被内核改变。如果锁定MTU，则不会尝试PMTUD，所有IPv4包都将不设置DF位发出，IPv6包则会按照MTU进行分段处理。

2. 命令：
   ```
   ip [-6] route del ROUTE
   ```


删除IPv6路由可以选择添加-6选项也可以不添加，ip route命令会根据具体地址类型来判断是IPv4地址还是IPv6地址。

ip rule 命令

1. 命令：
   ```
   ip [-6] rule list
   ```
   -6选项：设置-6选项打印IPv6的策略路由，不设置-6选项打印IPv4的策略路由。所以需要根据具体协议类型来配置-6选项。
2. 命令：
   ```
   ip [-6] rule [add|del] [from|to] ADDR table TABLE pref PREF
   ```
   -6选项：IPv6相关的策略路由表项需要设置-6选项，否则会报错：“Error: Invalid source address.”。相应地，IPv4相关的策略路由表项不可以设置-6选项，否则会报错：“Error: Invalid source address.”。

33.2.6 network 服务配置差异说明

概述

network服务使用ifup/ifdown的逻辑接口定义进行高级网络设置。其参数大多数都是在/etc/sysconfig/network和/etc/sysconfig/network-scripts/ifcfg-<interface-name>两个配置文件设置。前者为全局设置，后者未指定网卡的设置，当两者有冲突时，后者生效。

配置差异说明

其中在/etc/sysconfig/network下的配置差异有：

IPv4	IPv6	含义说明
NA	IPV6FORWARDING=yes|no	ipv6转发，默认不转发。
NA	IPV6_AUTOCONF=yes|no	ipv6转发打开是no，否则是yes。
NA	IPV6_ROUTER=yes|no	ipv6转发打开是yes，否则是no。
NA	IPV6_AUTOTUNNEL=yes|no	指定Tunnel为自动隧道模式，默认是no。
GATEWAY	IPV6_DEFAULTGW=<IPv6 address[%interface]> (optional)	在ipv6中设置默认网关。
NA	IPV6_DEFAULTDEV=<interface> (optional)	指定默认转发的网卡。
NA	IPV6_RADVD_PIDFILE=<pid-file> (optional)	默认ipv6_radvd_pid路径：/var/run/radvd/radvd.pid。
NA	IPV6_RADVD_TRIGGER_ACTION=startstop|reload|restart|SIGHUP (optional)	radvd默认触发动作。

而在/etc/sysconfig/network-scripts/ifcfg-<interface-name>下的差异主要有：

IPv4	IPv6	含义说明
IPADDRn	IPV6ADDR=<IPv6 address>[/<prefix length>]	ip地址。
PREFIXn	NA	网络前缀，网络别名和ppp无效，优先级高于NETMASK。
NETMASKn	NA	子网掩码，仅用于别名和ppp。
GATEWAY	IPV6_DEFAULTGW=<IPv6 address[%interface]> (optional)	默认网关。
MTU	IPV6_MTU=<MTU of link> (optional)	默认MTU。
IPV4_FAILURE_FATAL=yes|no	IPV6_FAILURE_FATAL	默认值是no。若设置为yes，dhclient失败ifup-eth会直接退出。
NA	IPV6_PRIVACY=rfc3041	默认禁用。
NA	IPV6INIT=yes|no	默认开启ipv6。
NA	IPV6FORWARDING=yes|no	默认关闭，已废弃。

33.3 FAQ

33.3.1 iscsi-initiator-utils 不支持登录 fe80 IPV6 地址

问题现象

客户端通过IPV6登录iscsi服务端时，使用如“iscsiadm -m node -p ipv6address -l”的命令格式登录，如果是全局地址（global address），直接替换将命令范例中的“ipv6address”替换为全局地址即可；但如果是链路本地地址（link-local address，fe80开头的IPV6地址）则无法使用，因为iscsi-initiator-utils目前机制还不支持用链路本地地址（link-local address）地址登录iscsi服务端。

原因分析

如果使用格式如“iscsiadm -m node -p fe80::xxxx -l”登录，会登录超时返回，这是因为使用链路本地地址必须指定接口，否则使用iscsi_io_tcp_connect函数调用connect函数会失败，并且产生标准错误码22。

如果使用格式如“iscsiadm -m node -p fe80::xxxx%eth0 -l”登录时，iscsi_addr_match函数会将地址“fe80::xxxx%eth0”与服务端返回的node信息中的地址“fe80::xxxx”对比，对比结果不匹配，导致登录失败。

因此，**iscsi-initiator-utils目前机制还不支持用链路本地地址（link-local address）地址登录iscsi服务端**。

33.3.2 网卡 down 掉之后，ipv6 地址丢失

问题现象

通过ip link down+up网卡或ifconfig down+up网卡命令，将网卡down掉之后再上线，查看网卡上配置的ip地址，发现ipv4地址不丢失，而配置的ipv6地址丢失。通过ifcfg-xxx配置文件下发的ipv6地址也会丢失，将网卡上线之后，通过systemctl restart network，可以重新下发ipv6地址的配置。

原因分析

当前3.10内核中的处理逻辑为如果网卡设置为down状态，会清空所有ipv4及ipv6地址，将网卡重新up之后，ipv4地址自动恢复，网卡上自动配置的ipv6链路本地地址也会恢复，但是其他配置的ipv6地址被丢失，无法自动恢复，需要重新通过ip/ifconfig命令手动添加，或者通过ifcfg-xxx配置文件下发。

33.3.3 bond 口已具有多个 IPV6 地址时，添加或删除 IPV6 地址耗时过久

问题现象

下列方式配置或删除（包括flush）IPV6地址方式，X为动态变化的低16位，并且配置在bond口时，耗时会随已配置的IPV6地址数量成数倍增加。例如由4个物理网卡组成的bond口添加IPV6地址时，单线程添加删除3000 IPV6地址均需大概5分钟，而普通物理网卡耗时在10秒内。

```
ip a add/del 192:168::18:X/64 dev DEVICE
```

原因分析

bond口在添加IPV6地址时，会生成IPV6组播地址，并进行同步到所有的物理网卡上，此耗时会随IPV6数量增加而增加，导致耗时过长。

解决方法

IPV6的组播地址是由IPV6地址的低24位与33-33-ff组合生成，组播地址过多会导致添加删除耗时增加，如果生成的组播地址为少量，耗时不会受此影响。

建议添加IPV6地址时，可保持低24位一致，保持高位变动，单网卡中仅需一个网段的一个地址即可与外部正常通信，此配置更符合常规使用。

33.3.4 Rsyslog 在 IPV4 和 IPV6 混合使用场景中日志传输延迟

问题现象

rsyslog客户端配置文件同时配置ipv4和ipv6地址，且端口配置相同的情况下，服务端收集log时会概率性出现日志打印延迟。

原因分析

延迟是因为rsyslog内部存在缓冲队列机制，默认情况下需要缓冲区队列达到一定数量才会写入文件。

解决方法

可通过配置Direct模式，关闭缓冲队列机制解决该问题。在rsyslog远程传输服务端的/etc/rsyslog.d目录下新增的远程传输配置文件中，最开头增加如下配置：

```
$ActionQueueType Direct
$MainMsgQueueType Direct
```

说明

- Direct模式减少队列大小为1，所以在队列中会保留1条日志到下次日志打印；
- Direct模式会降低服务器端的rsyslog性能。

34 HTTPS 及证书系统说明书

34.1 PKI、CA 与证书

34.1.1 PKI

PKI 就是 Public Key Infrastructure 的缩写，翻译过来就是公开密钥基础设施。它是利用公开密钥技术所构建的，解决网络安全问题的，普遍适用的一种基础设施;是一种遵循既定标准的密钥管理平台，它能够为所有网络应用提供加密和数字签名等密码服务及所必需的密钥和证书管理体系。

PKI既不是一个协议，也不是一个软件，它是一个标准，在这个标准之下发展出的为了实现安全基础服务目的的技术统称为PKI。可以说CA(认证中心)是PKI的核心，而数字证书是PKI的最基本元素，还有如apache等服务器、浏览器等客户端、银行等应用，都是pki的组件。

34.1.2 CA

CA 机构，又称为证书认证中心 (Certificate Authority) ，是一个负责发放和管理数字证书的第三方权威机构，它负责管理PKI结构下的所有用户(包括各种应用程序)的证书，把用户的公钥和用户的其他信息捆绑在一起，在网上验证用户的身份。CA机构的数字签名使得攻击者不能伪造和篡改证书。

认证中心主要有以下5个功能：

1. 证书的颁发：接收、验证用户(包括下级认证中心和最终用户)的数字证书的申请，可以受理或拒绝。
2. 证书的更新：认证中心可以定期更新所有用户的证书，或者根据用户的请求来更新用户的证书。
3. 证书的查询：查询当前用户证书申请处理过程；查询用户证书的颁发信息。

4. 证书的作废：由于用户私钥泄密等原因，需要向认证中心提出证书作废的请求；证书已经过了有效期，认证中心自动将该证书作废。认证中心通过维护证书作废列表 (Certificate Revocation List，CRL) 来完成上述功能。
5. 证书的归档：证书具有一定的有效期，证书过了有效期之后就将作废，但是我们不能将作废的证书简单地丢弃，因为有时我们可能需要验证以前的某个交易过程中产生的数字签名，这时我们就需要查询作废的证书。

34.1.3 证书

数字证书就是互联网通讯中标志通讯各方身份信息的一串数字，提供了一种在Internet上验证通信实体身份的方式，数字证书不是数字身份证，而是身份认证机构盖在数字身份证上的一个章或印（或者说加在数字身份证上的一个签名）。它是由权威机构CA发行的，人们可以在网上用它来识别对方的身份。

SSL公钥证书格式如下：

1. 证书版本号(Version)

   版本号指明X.509证书的格式版本，现在的值可以为：

   1) 0: v1

   2) 1: v2

   3) 2: v3

   也为将来的版本进行了预定义
2. 证书序列号(Serial Number)

   序列号指定由CA分配给证书的唯一的"数字型标识符"。当证书被取消时，实际上是将此证书的序列号放入由CA签发的CRL中，这也是序列号唯一的原因。
3. 签名算法标识符(Signature Algorithm)

   签名算法标识用来指定由CA签发证书时所使用的"签名算法"。算法标识符用来指定CA签发证书时所使用的：

   1) 公开密钥算法

   2) hash算法

   example：sha256WithRSAEncryption

   须向国际知名标准组织（如ISO）注册
4. 签发机构名（Issuer）

   此域用来标识签发证书的CA的X.500 DN（DN-Distinguished Name）名字。包括：

   1) 国家（C）

   2) 省市（ST）

   3) 地区（L）

   4) 组织机构（O）

   5) 单位部门（OU）

   6) 通用名（CN）

   7) 邮箱地址
5. 有效期（Validity）

   指定证书的有效期，包括：

   1) 证书开始生效的日期时间
6. 证书用户名（Subject）

指定证书持有者的X.500唯一名字。包括：

1) 国家（C）

2) 省市（ST）

3) 地区（L）

4) 组织机构（O）

5) 单位部门（OU）

6) 通用名（CN）

7) 邮箱地址

7. 证书持有者公开密钥信息（Subject Public Key Info）

   证书持有者公开密钥信息域包含两个重要信息：

   1) 证书持有者的公开密钥的值

   2) 公开密钥使用的算法标识符。此标识符包含公开密钥算法和hash算法。

8. 扩展项（extension）

   X.509 V3证书是在v2的基础上一标准形式或普通形式增加了扩展项，以使证书能够附带额外信息。标准扩展是指由X.509 V3版本定义的对V2版本增加的具有广泛应用前景的扩展项，任何人都可以向一些权威机构，如ISO，来注册一些其他扩展，如果这些扩展项应用广泛，也许以后会成为标准扩展项。

9. 签发者唯一标识符（Issuer Unique Identifier）

   签发者唯一标识符在第2版加入证书定义中。此域用在当同一个X.500名字用于多个认证机构时，用一比特字符串来唯一标识签发者的X.500名字。可选。

10. 证书持有者唯一标识符（Subject Unique Identifier）

    持有证书者唯一标识符在第2版的标准中加入X.509证书定义。此域用在当同一个X.500名字用于多个证书持有者时，用一比特字符串来唯一标识证书持有者的X.500名字。可选。

11. 签名算法（Signature Algorithm）

    证书签发机构对证书上述内容的签名算法

    example： sha256WithRSAEncryption

12. 签名值（Issuer's Signature）

    证书签发机构对证书上述内容的签名值

34.1.4 根证书

根证书是CA认证中心给自己颁发的证书，是信任链的起始点。安装根证书意味着对这个CA认证中心的信任。

34.2 HTTPS 原理解析

34.2.1 简介

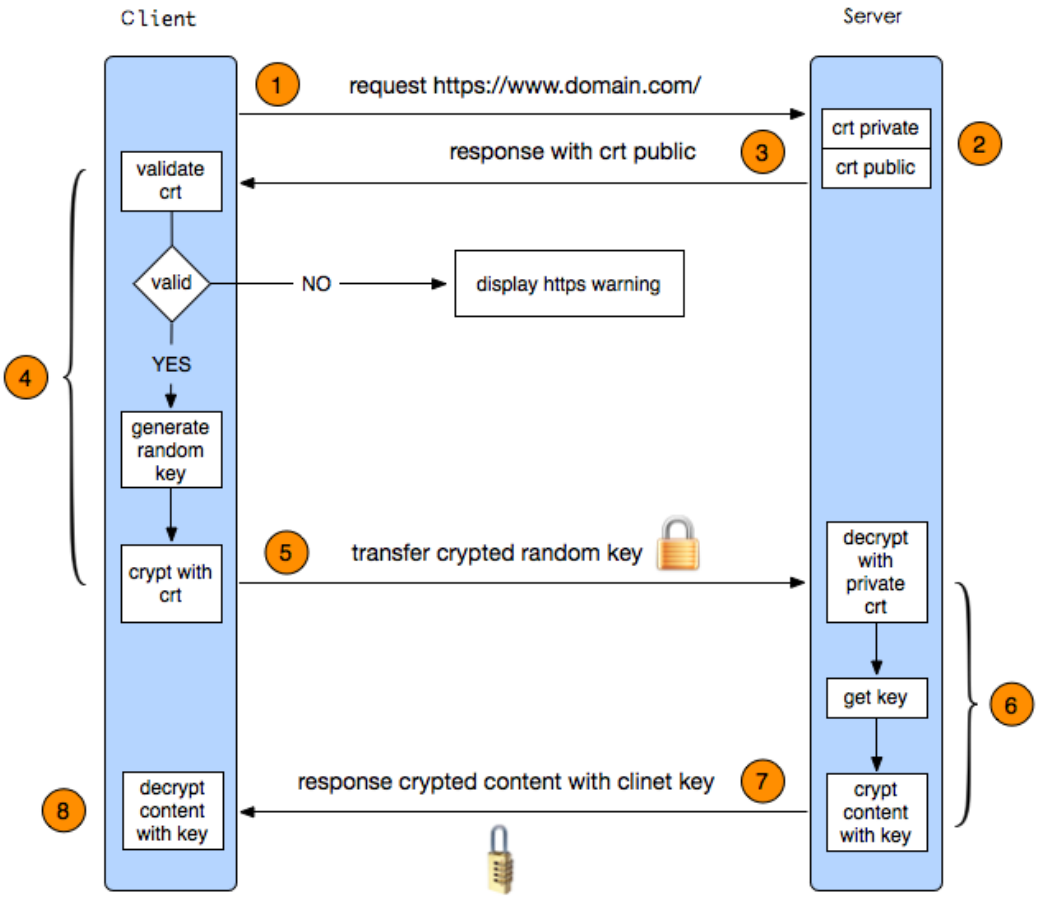
HTTP协议在进行数据传输时采用非加密方式，即采用明文，因此使用HTTP协议传输的可靠性无法保证。为了保证数据能够加密传输，网景公司设计了SSL（Secure Sockets Layer）协议用于对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。SSL目前的版本是3.0，被IETF（Internet Engineering Task Force）定义在RFC 6101中，之后IETF对SSL 3.0进行了升级，于是出现了TLS（Transport Layer Security）1.0，定义

在RFC 2246。实际上HTTPS当前大多使用TLS协议，但是由于SSL出现的时间比较早，并且依旧被现在浏览器所支持，因此SSL依然是HTTPS的代名词，但无论是TLS还是SSL都是上个世纪的产物，SSL最后一个版本是3.0，今后TLS将会继承SSL优良血统继续为传输进行加密服务。目前TLS的版本是1.2，定义在RFC 5246中，暂时未被广泛的使用

34.2.2 HTTPS 验证原理

34.2.2.1 流程图

Https在真正请求数据前，会先与服务有几次握手验证，以证明相互的身份，如图：



34.2.2.2 流程介绍

1. 客户端发起一个https的请求，把自身支持的一系列Cipher Suite（密钥算法套件，简称Cipher）发送给服务端。
2. 服务端接收到客户端所有的Cipher后与自身支持的对比，如果不支持则连接断开，反之则会从中选出一种加密算法和HASH算法以证书的形式返回给客户端，证书中还包含了公钥、颁证机构、网址、失效日期等。
3. 客户端收到服务端响应后，首先验证证书的合法性，包括颁发证书的机构是否合法与是否过期，证书中包含的网站地址是否与正在访问的地址一致等。
4. 如果证书验证通过，或者用户接受了不授信的证书，此时客户端会生成一串随机数，然后用证书中的公钥加密。

5. 客户端用最开始约定好的HASH方式，把握手消息取HASH值，然后用随机数加密“握手消息+握手消息HASH值(签名)”并一起发送给服务端。在这里之所以要取握手消息的HASH值，主要是把握手消息做一个签名，用于验证握手消息在传输过程中没有被篡改过。
6. 服务端拿到客户端传来的密文，用自己的私钥来解密握手消息取出随机数密码，再用随机数密码 解密握手消息与HASH值，并与传过来的HASH值做对比确认是否一致。然后用随机密码加密一段握手消息(握手消息+握手消息的HASH值)给客户端。
7. 客户端用随机数解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前客户端生成的随机密码并利用对称加密算法进行加密。

34.3 基于 OpenSSL 自建 CA 和颁发 SSL 证书

openssl是一个开源程序的套件，这个套件有三个部分组成：一是libcryto，这是一个具有通用功能的加密库，里面实现了众多的加密库；二是libssl，这个是实现ssl机制的，它是用于实现TLS/SSL的功能；三是openssl，是个多功能命令行工具，它可以实现加密解密，甚至还可以当CA来用，可以创建证书、吊销证书。

默认情况ubuntu和CentOS上都已安装好openssl。CentOS 6.x 上有关ssl证书的目录结构：

```
/etc/pki/CA/
newcerts    # 存放CA签署（颁发）过的数字证书（证书备份目录）
private     # 用于存放CA的私钥
crl         # 吊销的证书

/etc/pki/tls/
cert.pem    # 软链接到certs/ca-bundle.crt
certs/      # 该服务器上的证书存放目录，可以房子自己的证书和内置证书
ca-bundle.crt    # 内置信任的证书
private    # 证书密钥存放目录
openssl.cnf    # openssl的CA主配置文件
```

34.3.1 CA 自建证书

CA服务器要给别人颁发证书，首先自己得有一个作为根证书。以下是详细过程。

34.3.1.1 修改 CA 的一些配置文件

在一切工作之前修改好CA的配置文件、序列号、索引等等。

配置文件路径：/etc/pki/tls/openssl.cnf

```
...
[ CA_default ]

dir             = /etc/pki/CA           # Where everything is kept
certs           = $dir/certs            # Where the issued certs are kept
crl_dir         = $dir/crl              # Where the issued crl are kept
database        = $dir/index.txt        # database index file.
#unique_subject = no                    # Set to 'no' to allow creation of
# several ctificates with same subject.
new_certs_dir   = $dir/newcerts         # default place for new certs.

certificate     = $dir/cacert.pem       # The CA certificate
serial          = $dir/serial           # The current serial number
crlnumber       = $dir/crlnumber        # the current crl number
# must be commented out to leave a V1 CRL
crl             = $dir/crl.pem          # The current CRL
```

```
private_key     = $dir/private/cakey.pem # The private key
RANDFILE        = $dir/private/.rand    # private random number file
...
default_days    = 3650                  # how long to certify for
...
# For the CA policy
[ policy_match ]
countryName             = match
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional
...
[ req_distinguished_name ]
countryName                     = Country Name (2 letter code)
countryName_default             = CN
countryName_min                 = 2
countryName_max                 = 2

stateOrProvinceName             = State or Province Name (full name)
stateOrProvinceName_default     = GD
...
[ req_distinguished_name ] 部分主要是颁证时一些默认的值，可以不动
```

在CA目录下创建两个初始文件：

```
# touch index.txt serial
# echo 01 > serial
```

34.3.1.2 生成根密钥

```
# cd /etc/pki/CA/
# openssl genrsa -out private/cakey.pem 2048
```

34.3.1.3 生成根证书

使用req命令生成自签证书：

```
# openssl req -new -x509 -key private/cakey.pem -out cacert.pem
```

以上命令输入后，会提示输入一些内容，因为是私有的，所以可以随便输入（之前修改的openssl.cnf会在这里呈现），但是要记住输入的内容，为了与后面保持一致。上面的自签证书cacert.pem生成在/etc/pki/CA下。

34.3.1.4 为 repo 服务生成 SSL 密钥

以上都是在CA服务器上做的操作，而且只需进行一次，现在转到repo服务器上执行。因为repo提供集群服务，有统一的服务入口，因此只需在入口服务器，即提供负载均衡的服务器上执行以下操作。

```
# cd /etc/nginx/ssl
# openssl genrsa -out nginx.key 2048
```

34.3.1.5 为 nginx 生成证书签署请求

```
# openssl req -new -key nginx.key -out nginx.csr
...
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:GD
Locality Name (eg,  city) []:SZ
Organization Name (eg,  company) [Internet Widgits Pty Ltd]:HUAWEI
Organizational Unit Name (eg,  section) []:EULEROS
Common Name (e.g. server FQDN or YOUR name) []:developer.huawei.com
```



```
Email Address []:jianing1@huawei.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
...
```

同样会提示输入一些内容，Commone Name必须是要授予证书的服务器域名或主机名，challenge password不填。

34.3.1.6 CA 根据请求来签署证书

接下来要把上一步生成的证书请求csr文件，发到CA服务器上，在CA上执行：

```
# openssl ca -in nginx.csr -out nginx.crt
```

上面签发过程其实默认使用了-cert cacert.pem -keyfile cakey.pem，这两个文件就是前两步生成的位于/etc/pki/CA下的根密钥和根证书。将生成的crt证书发回nginx服务器使用。

到此我们已经拥有了建立ssl安全连接所需要的所有文件，并且服务器的crt和key都位于配置的目录下，剩下的是如何使用证书的问题。

34.4 证书的使用

34.4.1 为 EulerOS 系统添加根证书

将之前生成的根证书添加到EulerOS系统的指定配置文件中：

```
# cat /etc/pki/CA/cacert.pem >> /etc/pki/tls/certs/ca-bundle.crt
```

证书的内容如下：

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAwvSag/OaZlT7OHyW3SOPRj3xCmH+uoE8ZZq+7ykPmfXaBxZO
MthspvfsyDqwDrbDKBK7jlOHqon8/8ACRNYDUexTkrZcrJyfkhzSKpsYsOLlOMAw
Eju/dVKUZo3kKmU5MrUfQF+OkVSmTfG+ZW68Yo6BmXtU+4Tb1t6mZmIa1CMQlhFq
bjpMB8Ht7ZaOmijsqywUJ6F/2henpc5jam33xRsMbhVnVb2tyEVGxwtzEBrsHtcD
KR36sfKSFtYM7A8RallkqPsDoa8wTtebSfHeU8gbIsaTm+rmBL+g5FVVbtpe58pK
7eM3qTWQq7i1DVDYJ/LgO8xZf/T7sbOW5+uY4QIDAQABAoIBAHX+NzqfiJZg3r+c
YG6y4+i+epGvJ83XhCd2lhMHxgKHOxKxG1jetjwEF9JD+uSc5tbtJ4kNcyNbar2b
h4QEB3Rg1WAZcVOjY8y6FtVkk3gTi8ZhW3bdpI4skdbUEAE8zHaNOsHqU85P7muE
YZjggbIgz/2FsLSd/7Q3g3y56cTz+Suj0uNkO1Z66EuTk3wcpBxkY641gjhkc36W
3s5htUPWYjZp2Pa+o4iCOj5OmLMCMwUPBHCv1kojipbX4uSPNOQbS7NmrZn212FI
yur68AsH8ng0nIt57CtlTEUNDXIFbOCtRl1TIMHOPCmojpQMQRt9ax7M5mZNK9/O
FCy9gVUCgYEA7IVDGLB1qErXuDKZZB7KCYZzBann7fbVBL3uv+9WfPyjQiZoJGaU
6ic3MOXPM5OAKIQmYhJLC8276K/Wz5u7ViISWy1yhu/TkRA5HOPMxIv3kbxUsUlq
jQkBKOuOMbm7D813iJJnH6vLxz6pl/sRheCxSORi7P+4BOU1puEnDOMCgYEAOwL+
imMZzxxtRG+jgX9aHB6e3ZdijbZtFdvPAQLmqdeuscgeVynx/+9RgzboUCC5rcZl
hS1isoU5TOTdbAZ8TYoZLqdFExHuGQgiBQM8zMIQaMR62bFQmKnYcg6gdw3+1TX/
TTS7Pv+leV7hnGfkyhYpzdaLKhFswVPHzdMIuwsCgYEAyXwYlW61PEFN1T+dhjno
WMbtM1TzG7ea7egy+9SSc/TvyslXDDNLBlGT5ZbPtRshIOK6o+77eTmwqt8oekEr
DIwH4R5NIwAo1bnhPWgdpw2pP8AUOdY9651RIqXiAj47SZH6EyDMsqEx/jC2Xp7l
5Yts2zCbyVNqWtk7pEbHdGsCgYEAr8qAOD17VOXzyd3Ps6SrJ14DL8xcmH7wJhO3
Qrpt+/TpXsa/MBKLv5JasBvgMlDxH2p3TRngbaU7d2SBqutERzzTeeBOVUzMtSHn
ZZGq51KNZRq61f04jdaBsZOp70VDldCkXmedzwAflRuMjUaofgs0zHz99xAJMI5h
mIFFdrOCgYAkBQ5VrYxc3welSuWWcC52moJXbcmqOCg9GrHSXGvqurgtbchf2WeX
zgH+TnvAoakgiL4SgguEB2cMa7bydI55zeht+3PvIhC195ioUb513394As61qmJi
ZdWER8Dv3Hc5XgN98bonUtIUHLQMNnvNOuTATP2MWHkdEJkzzMTi4w==
-----END RSA PRIVATE KEY-----
```

34.4.2 Nginx 配置文件修改

配置文件举例如下：

```
user  root;
worker_processes  auto;    # 建议设置为core-1
error_log  /var/log/nginx/error.log  warn;
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;

upstream test.jianing.com {
server 10.250.244.236:8086;   #集群服务器repo1地址
server 10.250.245.174:8086;   #集群服务器repo2地址
}

server {
listen       80;        # http服务端口号
server_name  localhost;

location / {
root   html;
index  index.html index.htm;
proxy_pass        http://test.jianing.com;
proxy_set_header  X-Real-IP  $remote_addr;
client_max_body_size  100m;
}
}

server {
listen       443;      # https服务端口号
ssl          on;
server_name  developer.huawei.com;   # 与数字证书一致
ssl_certificate     /etc/nginx/ssl/nginx.crt; # 指定证书路径
ssl_certificate_key /etc/nginx/ssl/nginx.key; #指定秘钥路径
ssl_protocols  SSLv3 TLSv1;
ssl_ciphers  ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP;

location / {
proxy_pass        http://test.jianing.com;
}
}

}
```

34.4.3 客户端配置 yum 源

采用EulerOS的客户端，默认添加了我们创建的根证书，因此能够识别由该根证书认证的公共repo数字证书，配置repo源如下：

```
[jn-test]
name=jn-test
baseurl=https://developer.huawei.com/ict/site-euleros/euleros/repo/yum/2.2/os/x86_64/
```



```
enabled=1
gpgcheck=0
```

34.4.4 配置 yum 客户端忽略证书有效性

证书系统的作用主要包括两个部分：

1. 验证证书持有者的合法身份。
2. 利用证书内提供的信息进行密钥协商及加密传输。

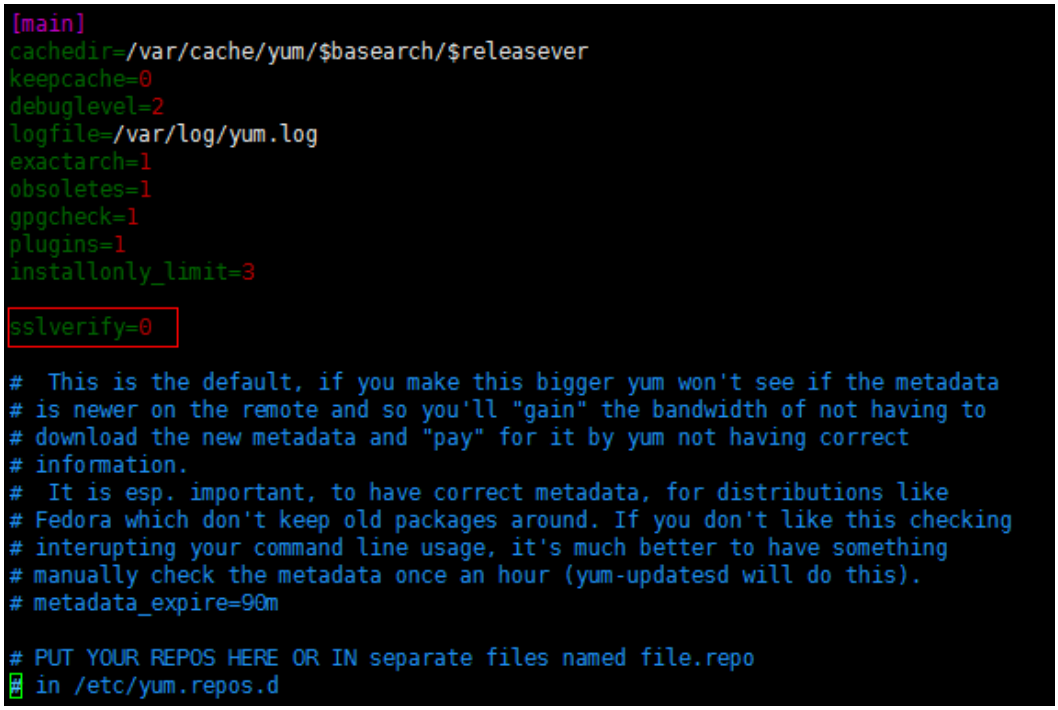
在有些使用场景中，用户不关注证书的合法性，仅需要证书系统的加密传输功能。

以下是对yum工具做相应配置，使其在配置https源时，不对对方的证书做有效性判断。

打开yum配置文件yum.conf：

```
# vim /etc/yum.conf
```

如下图所示，添加配置项“sslverifyfy=0”，保存退出后即可生效。



“sslverify”配置项默认不在配置文件中写明，且默认值为1，需手动添加并赋值0。

35 LVM 使用手册

35.1 简介

逻辑卷LVM管理会根据物理存储生成提取层，以便创建逻辑存储卷。本章节主要介绍配置/etc/lvm/lvm.conf文件时的使用指导。

35.2 使用指导

修改/etc/lvm/lvm.conf中level、log_unless_silent字段，可以配置日志等级。通过设置level值为2~7，可以记录从fatal到debug级别的日志。

- level默认值为3，log_unless_silent默认值为1。level=3时，lvm命令会将fatal、error级别的日志记录到syslog。
- level=2或者level=3时，通过设置字段log_unless_silent=1，还可以记录重要warning日志，例如创建、删除、修改pv/vg/lv。打开log_unless_silent=1会带来syslog日志量的增加，如果不想打印这些warning日志，可通过设置log_unless_silent=0关闭。
- level=0，表示禁止所有的lvm日志记录到syslog。

36 intel-VMD-NVMe 暴力热插拔

应用场景

NVMe SSD缺乏专用的控制点，意味着NVMe SSD的热插拔事件不能够被可靠地支持。同样的原因，NVMe SSD也不能对LED状态灯进行可靠的管理。在硬盘故障时，LED状态灯对数据中心管理员可提供了重要的信息。

所以，Intel VMD（Volume Management Device）特性为NVMe SSD提供了可靠的管理。Intel VMD是集成在Intel Xeon Scalable processor-based server中PCIe根联合体中的硬件。这个新特性让NVMe SSD像SAS/SATA硬盘那样可管理，为NVMe SSD提供了外部控制点，相应的NVMe SSD热插拔事件便由VMD驱动进行处理。

热插拔（Hot-plug）功能是允许用户在不关闭系统，不切断电源的情况下取出和更换设备，从而提高了系统对灾难的及时恢复能力、增强扩展性和灵活性等。支持热插拔的平台能够保证在热插入（Hot Insertion）的时候，自动检测到设备并将其注册到相应驱动，在热拔出的时候，能够自动检测到设备丢失并从驱动移除，整个过程不需要系统重启。 并且在热插拔操作之后，系统和设备能够正常工作。

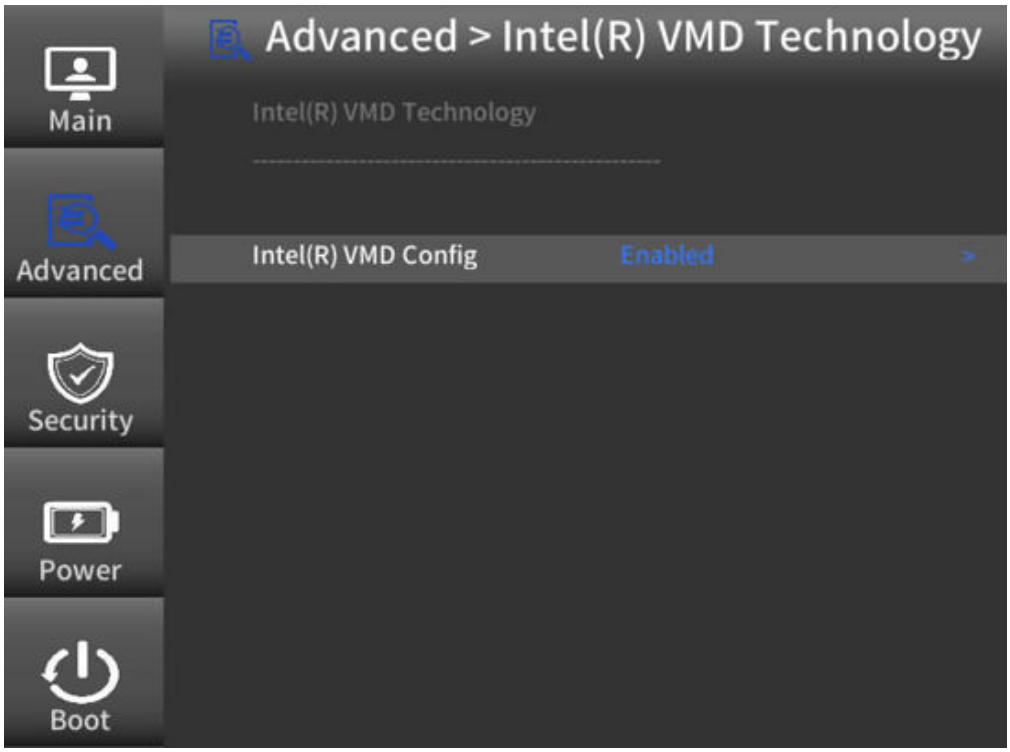
约束限制

- 服务器必须支持Intel VMD特性，且该特性已经在BIOS中开启。
- 服务器必须包括U.2接口的插槽，且相应的插槽必须支持暴力热插拔。
- 用于热插拔的NVMe SSD的接口是U.2接口，且该SSD盘插在对应插槽上。
- 操作系统如安装于NVMe盘上，则该盘不支持热插拔。
- 不支持虚拟化场景。

使用指导

1. BIOS中使能VMD特性

   在UEFI的Devices Manager中，通过Advanced -> Socket Configuration -> IIO Configuration -> Intel(R) VMD Technology菜单进入VMD配置界面，如下图所示将Intel(R) VMD Config配置为Enabled，保存BIOS设置，并重新启动服务器。



2. 对NVMe SSD进行热插拔
   根据实际业务需要，直接对NVMe SSD进行移除、插入操作。

37 FAQ

37.1 查询 EulerOS 版本标识

EulerOS V2.0SP5 版本标识的配置文件是 /etc/euleros-release，此文件中包含了EulerOS版本信息，使用如下命令：

```
[root@localhost ~]# cat /etc/euleros-release
EulerOS release 2.0 (SP5)             #EulerOS版本
```

查询内核版本信息，使用如下命令：


```
[root@localhost ~]# uname -a
Linux localhost.localdomain 3.10.0-327.44.58.19.x86_64 #1 SMP Wed Feb 22 17:25:10 UTC 2017 x86_64
x86_64 x86_64 GNU/Linux
```

说明

EulerOS V2.0SP5系统中存在一个/etc/euleros-lastest文件，该文件是开发人员内部使用的文件，用户不能依赖该文件来获取EulerOS的版本以及内核版本信息。

37.2 EulerOS 支持的文件系统类型

EulerOS版本仅支持EXT3、EXT4文件系统，其余文件系统不支持。

37.3 网络配置约束限制

EulerOS通过 NetworkManager 服务进行网络管理。如果开启了该服务，则必须使用 nmcli 命令或通过修改配置文件来配置网络（如ip、路由等），而不能使用 ip/ifconfig/route 等命令来配置。

说明

在开启NetworkManager 服务的场景下，使用 ip/ifconfig/route 等命令配置网络，则一段时间后配置会被NetworkManager覆盖，导致 ip/ifconfig/route 配置不生效。

查看NetworkManager 服务是否开启：

```
systemctl status NetworkManager
```

说明

nmcli命令使用参考“nmcli --help”或者“man nmcli”。

如果要使用 ip/ifconfig/route 等命令来管理网络，请先关闭 NetworkManager 服务，使用如下命令：

```
systemctl stop NetworkManager
```

说明

如果只是关闭NetworkManager，重启后还是会被其他依赖服务拉起来。所以针对不同业务使用场景，若需要完全屏蔽NetworkManager，仅建议通过卸载该软件包方式解决，具体卸载命令如下：

```
rpm -e --nodeps NetworkManager
```

37.4 glibc 内存管理参数

glibc升级到glibc-2.17-196版本之后，内存管理算法默认采用了PER_THREAD实现。默认情况下，为了减少多线程内存申请时的锁竞争，glibc会为每个线程创建一个分配区arena，部分多线程应用会由于这个变化，导致进程的虚拟内存和物理内存大量增长。可以通过以下环境变量限制arena的个数。

例如，限定arena的数量为1：

```
export MALLOC_ARENA_MAX=1
```

说明

MALLOC_ARENA_MAX不设置的情况下，glibc默认创建的arena最大个数为CPU核数*8。

37.5 如何设置防火墙提高容器网络的性能

EulerOS 的防火墙基于 CentOS 的 firewalld 进行构建，并跟随大多数操作系统，如Windows/Redhat/CentOS/Ubuntu等的策略，默认打开防火墙。

firewalld 打开时，因为会加载一系列iptable/ebtable 的内核模块及默认规则，导致容器网络性能下降 20% 左右（测试环境不同对下降幅度有影响）。相同测试条件下，关闭防火墙性能略优于 SuSE12，打开后劣于 SuSE12。

说明

关闭防火墙时，所有iptable/ebtable 中规则及模块本身都会被删除卸载，即默认的和用户定义的防护规则都将失效。

防护规则及功能

当前 firewalld 支持的防护规则及功能主要有：

1. Zone

   定义安全级别，不同安全级别的接口/连接/地址放入不同的 zone 进行防护，预定义有 drop/block/public/external/dmz/work/home/internal/trusted 9 类。如家庭网络接口可以放入home 甚至 trusted zone，而公共 wifi 则适合 public 级别。

2. Service

   虽然 Service 最终对应的是端口和地址，但 firewalld 将其抽象出来，使得用户可以便捷地使能/关闭对一些服务的防护。例如：

   ```
   firewall-cmd --permanent --zone=public --add-service=http
   ```

3. IPSet

   即 IP 和 MAC 的绑定设置。

   说明

   ipset list命令不支持并发操作。在压力测试场景下，该命令与ipset destroy命令并发执行，可能会导致删除集合失败。

4. ICMP Type

   鉴于 ICMP 在 IP 网络里的特殊性，firewalld 将其提出来进行防护。

5. Direct Interface

   用户可以在 firewalld 里直接定义 iptable/ebtable 的规则。

重要命令

1. 查看防火墙状态

   ```
   [root@localhost firewalld]# systemctl status firewalld
   firewalld.service - firewalld - dynamic firewall daemon
      Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
      Active: active (running) since Wed 2017-03-22 06:17:32 EDT; 1h 25min ago
    Main PID: 28597 (firewalld)
      CGroup: /system.slice/firewalld.service
              └─28597 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
   ```

2. 开启防火墙

   ```
   [root@localhost firewalld]# systemctl start firewalld
   ```

   开启后查看防火墙状态

   ```
   [root@localhost firewalld]# systemctl status firewalld
   firewalld.service - firewalld - dynamic firewall daemon
      Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   ```



```
   Active: active (running) since Wed 2017-03-22 10:36:38 EDT; 2s ago
 Main PID: 1500 (firewalld)
   CGroup: /system.slice/firewalld.service
           └─1500 /usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
```

3. 关闭防火墙

```
[root@localhost firewalld]# systemctl stop firewalld
```

关闭后查看防火墙状态

```
[root@localhost firewalld]# systemctl status firewalld
firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled)
   Active: inactive (dead) since Wed 2017-03-22 07:43:05 EDT; 4min 15s ago
  Process: 28597 ExecStart=/usr/sbin/firewalld --nofork --nopid $FIREWALLD_ARGS
(code=exited, status=0/SUCCESS)
 Main PID: 28597 (code=exited, status=0/SUCC
```

37.6 rpm 安装冲突问题解决方法

现象描述

某些 rpm 包同时被安装的时候会报冲突。以下几种场景会导致冲突：

- 部分 x86_64 与 i686 包冲突；
- 提供了类似功能的包；
- rpm 的 spec 定义不能兼容的包。

解决方法

rpm/yum 安装的时候会提示冲突，用户需要根据提示，删除掉冲突的 rpm 包，如果冲突的只是文档文件，也可以强制安装。

- eg1：

```
Transaction check error:
file /usr/share/doc/dracut-033/dracut.html from install of dracut-033-360.h1.i686 conflicts
with file from package dracut-033-360.h1.x86_64
```

只能选择 x86_64 或 i686 一个包安装，或者强制安装。

```
rpm  -ivh  xxx.rpm --force
```

- eg2：

```
Error: tog-pegasus-libs conflicts with libcmpiCppImpl0-2.0.3-5.x86_64
```

先卸载掉冲突的包：

```
yum remove libcmpiCppImpl0
```

37.7 老版本 kernel 删除方法

现象描述

```
rpm  -Uvh kernel
```

或者

```
yum  update  kernel
```

使用上述命令对 kernel 包升级之后，老版本 kernel 会被保留，以防止新内核有问题无法启动时，可以切换回老版本内核， 是 rpm/yum 对 kernel 包的特殊处理，别的rpm则直接会把老版本删除掉。

解决方法

要删除老版本的kernel，可以在升级kernel成功之后，使用

```
rpm -e kernel-xxx
```

或者

```
yum remove kernel-xxx
```

xxx指定具体的版本号来删除。

37.8 配置虚拟机串口输出文件

背景描述

虚拟机通过串口输出的内容定位到文件中，方便串口日志分析、传递。

配置方法

要实现该功能，需要配置以下两个地方。

1. **修改配置文件grub.cfg中内核启动参数**

   在/etc/grub2/grub.cfg 文件中找到要修改的内核，在该内核的启动参数后增加console=ttyS0，如图37-1。

   图 37-1 增加内核启动参数

   ```
   menuentry 'EulerOS (3.10.0-327.55.58.81.h5.x86_64) 2.0 (SP2)' --class euleros --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option 'gnulinux-3.10.0-327.53.58.73.x86_64-advanced-7b54e8eb-cab0-4c0b-be0d-601703319ba2' {
           load_video
           set gfxpayload=keep
           insmod gzio
           insmod part_msdos
           insmod ext2
           set root='hd0,msdos1'
           if [ x$feature_platform_search_hint = xy ]; then
             search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos1 --hint-efi=hd0,msdos1 --hint-baremetal=ahci0,msdos1 --hint='hd0,msdos1'  ab243b60-2124-4348-b63a-98b86d2dbb4c
           else
             search --no-floppy --fs-uuid --set=root ab243b60-2124-4348-b63a-98b86d2dbb4c
           fi
           linux16 /vmlinuz-3.10.0-327.55.58.81.h5.x86_64 root=/dev/mapper/euleros-root ro crash_kexec_post_notifiers softlockup_panic=1 panic=3 reserve_kbox_mem=16M nmi_watchdog=1 rd.shell=0 crashkernel=auto rd.lvm.lv=euleros/root rd.lvm.lv=euleros/swap rhgb quiet console=ttyS0
   }
   ```

2. **配置Host端**

   Host侧需要在定义虚拟机时配置串口输出文件路径，修改示例如下，需要根据实际情况配置“source path”。

   ```
   <serial type='file'>
   <source path='/sdb/test/Euler2.5_vm/serial.log'/>
   <target port='0'/>
   </serial>
   <console type='file'>
   <source path='/sdb/test/Euler2.5_vm/serial.log'/>
   <target type='serial' port='0'/>
   </console>
   ```

37.9 openldap 默认证书注意事项

openldap在安装时，会提供一份默认的证书和证书数据库密码文件，默认提供的文件安全等级低，建议用户初次使用ldap时，务必使用自己的证书文件，并做好权限控制。

37.10 rootsh 日志防爆特性的约束限制

rootsh日志防爆特性利用logrotate日志来实现日志防爆功能，用日志文件的时间戳来判断是否删除该日志文件以满足日志的空间大小要求。当系统时间发生向前改变时，则存在将新产生的rootsh日志删除，从而导致日志丢失的风险。

说明

处理该场景会引入其他风险，故对此场景不做处理。

37.11 cron 引起/var/spool/postfix/maildrop 目录生成大量小文件

问题现象

cron引起/var/spool/postfix/maildrop目录生成大量小文件。

原因分析

由于EulerOS在执行cron时，会将cron执行脚本中的output和warning信息以邮件的形式发送cron所有者，而由于服务器上的sendmail或postfix没有正常运行，导致邮件发送不成功，全部小文件堆积在了maildrop目录下面，而且没有自动清理转换的机制，经历较长时间后，此目录已堆积了大量的小文件。

解决方法

在crontab的第一行加入“MAILTO=""”便可，这样执行当前用户的cron时，不会发送邮件。

37.12 Intel 漏洞补丁开关

37.12.1 Intel 侧信道攻击 L1TF 漏洞补丁开关

现代微处理器的硬件实现上发现了一个类似于Spectre和Meltdown的漏洞，该漏洞会影响Intel X86系列微处理器（详细影响的处理器列表，请参考Intel官网：https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-00161.html）。非特权攻击者可以利用此漏洞绕过内存安全限制，访问存储在内存中的数据。该漏洞的CVE编号为CVE-2018-3620，在虚拟化方面，该漏洞的CVE编号为CVE-2018-3646。业内称该漏洞为L1TF（L1 Terminal Fault），而安全研究人员则称之为“Foreshadow”和“Foreshadow-NG”。

此漏洞有三个部分的影响， 第一个部分仅影响英特尔“SGX”（Intel Software Guard Extension）安全区域，其可以通过独立于操作系统的微码更新来修补。 另外两个部分需要由操作系统和hypervisors修补。若需要完全修补虚拟化环境中不受信任的来宾虚拟机的潜在攻击风险，则需要系统管理员的特定操作。

EulerOS已经提供内核更新以修复此漏洞，该更新主要包含如下3个部分的改动：

1. 页表反转（Page Table Inversion），这是一个内核中比较小的改动，更新后的内核中默认启用该功能。

2. 在虚拟机之间切换时刷新L1数据高速缓存（Flushing the L1 Data Cache），该选项是可选的。该项加固可以通过更新内核或者更新Intel微码（microcode）来实现。
3. 禁用SMT（SMT Disable），该选项是可选的（注：SMT：Simultaneous multi-threads）。该项加固是通过修改内核来实现的，并且添加了新的控制接口。该改动的另外一种等价实现是在BIOS中禁用Intel超线程（HT）,但这样做比较麻烦，不推荐使用。

这些更新中的每一个部分都独立地针对攻击的不同部分提供保护。

开关说明

- 页表反转（Page Table Inversion）

  攻击者推测性地从页表中的无效（not present PTE访问物理地址，如果该物理地址的内容被高速缓存到L1数据缓存中，则存储器访问成功，造成数据泄漏。而如果没有缓存物理地址的内容，则上述的物理地址访问不会泄漏数据。EulerOS修复这个缺陷是在内核中通过页表反转来实现的：通过设置高阶地址位来更新无效（not present）PTE中的物理地址，使其指向不在内存中或不可缓存的物理地址。 这样，其内容便不能被缓存于L1数据高速缓存中。这样即可防止通过无效（not present）PTE推测造成的数据泄漏。

  EulerOS升级后的内核，通过sysfs文件系统显示该选项的当前状态：

  ```
  # cat /sys/devices/system/cpu/vulnerabilities/l1tf
  Mitigation: PTE Inversion; VMX: SMT vulnerable, L1D conditional cache flushes
  #
  ```

  /sys/devices/system/cpu/vulnerabilities/l1tf 可能的值如下：

  ```
  'Not affected'                当前CPU不受该漏洞影响
  'Mitigation: PTE Inversion'   保护启用
  ```

  若KVM/VMX启用，并且当前CPU受该漏洞影响，则如下信息会被附加在“Mitigation: PTE Inversion”的后面：

  - SMT 状态信息：

  ```
  'VMX: SMT vulnerable'         SMT使能
  'VMX: SMT disabled'           SMT已禁用
  ```

  - L1D 刷新模式：

  ```
  'L1D vulnerable'                    L1D 刷新被禁用
  'L1D conditional cache flushes'     L1D 刷新使能（自动刷新）
  'L1D cache flushes'                 L1D 刷新使能（无条件强制刷新）
  ```

- 刷新L1数据高速缓存（Flushing the L1 Data Cache）

  用户可以通过如下两种方式来实现L1数据高速缓存的刷新（任选一种）：

  a. 更新Intel微码

     若用户已经更新了Intel最新的微码，则Intel X86 CPU会多出一个新的寄存器：IA32_FLUSH_CMD MSR寄存器（IA32_FLUSH_CMD Machine Specific Register），该寄存器被用于刷新L1数据高速缓存。

     用户可以通过执行cat /proc/cpuinfo或者lscpu命令，查看CPU的flag，来确认自己的CPU是否支持该寄存器：

     ```
     # lscpu
     Flags:    … ssbd ibrs ibpb stibp spec_ctrl intel_stibp flush_l1d
     #
     ```

     若未看到该标志，则表示当前环境尚未更新Intel的微码，建议用户更新Intel最新微码，并重启操作系统，以支持CPU从硬件上自动刷新L1数据高速缓存。

  b. 更新EulerOS

     若用户无法更新微码，EulerOS提供的更新可以在软件层面实现L1数据高速缓存的刷新，但是软件层面的刷新会比CPU直接从硬件层面刷新慢。

EulerOS提供了一个启动参数"l1tf="来控制内核刷新L1数据高速缓存动作，可选的值描述如下：

可选值	功能	说明	备注
full	使能所有L1TF漏洞的修补措施，包括：任意虚拟机访问操作后刷新L1数据高速缓存、禁用超线程（SMT）。	指定l1tf=full，用户可以在系统运行过程中动态地通过sysfs接口控制L1数据高速缓存刷新（Host端KVM模块提供的功能，Guest无此功能）、SMT的使能和禁止。	sysfs下SMT控制接口如下： /sys/devices/system/cpu/smt/control （sysfs下提供的SMT控制文件的详细说明见“**• 控制SMT（Control SMT）**”章节描述）
full,force	使能所有L1TF漏洞的修补措施，包括：任意虚拟机访问操作后刷新L1数据高速缓存、禁用超线程（SMT）。	指定l1tf=full,force，此选项不允许用户在系统运行过程中动态地通过sysfs接口控制L1数据高速缓存刷新（Host端KVM模块提供的功能，Guest无此功能）、SMT的使能和禁止。	-
flush	使能有条件的（内核决定）L1数据高速缓存的刷新、不禁用超线程（SMT）。	指定l1tf=flush，用户可以在系统运行过程中动态地通过通过sysfs接口控制L1数据高速缓存刷新（Host端KVM模块提供的功能，Guest无此功能）、SMT的使能和禁止。 在KVM虚拟化场景下，若虚拟机启用了不安全的配置，例如：使能了SMT或者禁用L1数据高速缓存刷新。则hypervisor（KVM）会发出一个告警。	l1tf默认设置为该值

可选值	功能	说明	备注
flush,nosmt	使能有条件的（内核决定）L1数据高速缓存的刷新、禁用超线程（SMT）。	指定l1tf=flush,nosmt，用户可以在系统运行过程中动态地通过通过sysfs接口控制L1数据高速缓存刷新（Host端KVM模块提供的功能，Guest无此功能）、SMT的使能和禁止。 在KVM虚拟化场景下，若虚拟机启用了不安全的配置，例如：使能了SMT或者禁用L1数据高速缓存刷新。则hypervisor（KVM）会发出一个告警。	-
flush,nowarn	类似于flush选项：使能所有L1TF漏洞的修补措施，包括：任意虚拟机访问操作后刷新L1数据高速缓存、禁用超线程（SMT）。禁止hypervisor（KVM）发告警。	指定l1tf=full，用户可以在系统运行过程中动态地通过通过sysfs接口控制L1数据高速缓存刷新（Host端KVM模块提供的功能，Guest无此功能）、SMT的使能和禁止。在KVM虚拟化场景下，若虚拟机启用了不安全的配置，例如：使能了SMT或者禁用L1数据高速缓存刷新。则hypervisor（KVM）也不会发告警。	-
off	禁用hypervisor的修复措施，KVM不会刷新L1数据高速缓存。	-	-

启动参数l1tf的默认值为flush。

备注：Host端，KVM模块提供了一个sysfs接口，可以动态控制L1数据高速缓存刷新动态开启、关闭。sysfs接口如下：

```
/sys/module/kvm_intel/parameters/vmentry_l1d_flush
```

该接口有3个值可供选择设置：

```
always      执行任何访问虚拟机的陷出（VMENTER）操作时刷新L1高速数据缓存。
cond        有条件地刷新L1数据高速缓存，条件为：访问虚拟机的过程中，若陷入（VMEXIT）和陷
```

```
出（VMENTER)之间的代码可能泄露数据时，会在陷出（VMENTER）时刷新L1数据高速缓存。
never       禁用L1数据高速缓存刷新。
```

例如，Host运行过程中，动态禁止L1数据高速缓存刷新，可通过如下命令实现：

```
echo "never" > /sys/module/kvm_intel/parameters/vmentry_l1d_flush
```

- 控制SMT（Control SMT）

  当使用同步多线程（simultaneous multithreading）或超线程技术（HT）时，共享处理器缓存资源上的无关线程可以利用L1TF漏洞从L1数据缓存中读取其他线程的数据。 如果当前的运行环境的是不受信任的共享环境，则需要考虑禁用SMT来防止该漏洞被利用。

  启用或禁用SMT有两种方式（任选一种）：

  a. 从BIOS启用或禁用SMT。

  b. EulerOS提供了一个新的启动参数：nosmt 来控制SMT的启用或禁用。描述如下：

可选值	功能	说明	备注
nosmt	禁用SMT	用户可在系统运行过程中，动态地通过sysfs接口重新开启SMT	sysfs接口如下： /sys/devices/system/cpu/smt/active /sys/devices/system/cpu/smt/control （这2个控制文件的详细说明见下面描述）
nosmt=force	禁用SMT，且指定force选项	指定force选项，禁止用户在系统运行过程中，动态地通过sysfs接口重新开启SMT	同上

EulerOS对控制SMT提供了2个sysfs接口，在允许的情况下，用户可在系统运行期间通过该sysfs接口动态地控制SMT。如下：

```
/sys/devices/system/cpu/smt/active
/sys/devices/system/cpu/smt/control
```

说明：

active是一个只读接口，可执行 cat /sys/devices/system/cpu/smt/active读取其值，可能的值为0或者1。

```
active为0，表示SMT被禁用。EulerOS启动时，设置启动参数nosmt，则active为0。
active为1，表示SMT被启用，且逻辑CPU都online状态。
```

control是一个可读、可写的接口。通过该文件，可查看当前SMT的使能状态：

状态值	是否可写入	说明
on	是	SMT使能，若逻辑CPU是offline的，则使能它们

状态值	是否可写入	说明
off	是	禁用SMT，若逻辑CPU都是online的，则禁用它们
forceoff	是	强行禁用SMT，且无法在系统运行期间动态调整
nosupport	否	当前系统CPU不支持超线程（SMT）

如上表描述，可通过设置control值为on、off、forceoff来使能、禁用SMT（需要注意启动参数"l1tf="的配置值对当前操作的影响）。

例如，Guest运行过程中，要动态禁用SMT，可通过如下命令实现：

```
echo "off" > /sys/devices/system/cpu/smt/control
```

37.12.2 Spectre 和 Meltdown 漏洞补丁开关

对于Spectre和Meltdown漏洞，EulerOS已经提供内核更新以修复这些CVE漏洞。出于安全考虑，在更新的内核中默认启用这些修复，但由于“推测执行”属于处理器的性能优化技术，而此次的安全漏洞修复影响“推测执行”原有逻辑，所以对于内核和微码的更新必定会导致性能下降。然而，部分用户认为他们的系统已经有相当好的保护（比如物理隔离），不会受到此次漏洞影响，所以并不希望由于漏洞修复而导致不必要的性能损失。鉴于此，EulerOS提供了相关的启闭方法。如果用户认为不需要启用这些补丁，可以参考本文进行相应的配置来关闭补丁，以减少性能损失。

开关说明

1. pti（Page Table Isolation）用于在运行用户态时隔离内核页表，这个开关针对CVE-2017-5754（变种3 / Meltdown）。
2. ibrs（Indirect Branch Restricted Speculation）和ibpb（Indirect Branch Prediction Barriers）这两个开关连同微码一起针对CVE-2017-5715（变种2/Spectre）。

   说明

   裸机场景，ibrs和ibpb开关的使能，需要物理机升级微码后才支持，如果物理机没有升级微码，这两个开关默认是关闭的，补丁不生效，且无法通过手工更改内核参数打开这两个开关。

   GuestOS的场景，Guest内这两个开关的使能，除了依赖host升级微码，还依赖于host虚拟化组件将两个功能透传到GuestOS。
3. retp(retpolines)在skylake微架构之前的CPU中取代ibrs，与ibpb连同微码一起针对CVE-2017-5715（变种2/Spectre）；skylake微架构仍然使用ibrs+ibpb方案解决CVE-2017-5715（变种2/Spectre）。
4. CVE-2017-5753（变种1/Spectre）所对应的内核补丁没有开关可以控制。

架构默认开关配置

默认情况下，内核基于检测到的CPU微架构，在启动阶段便自动设置适应于CPU微架构的合适的开关配置。不同Intel CPU微架构默认的开关配置情况：

Architecture	pti	ibrs	retp	ibpb	Description
skylake	1	1	0	1	修复变种#1#2#3
pre-skylake(support mircrocode)	1	0	1	1	修复变种#1#2#3
older Intel CPU with no microcode update available	1	1	0	0	修复变种#1#3

关闭开关

有以下两种方式可以对关闭开关：

1. 永久关闭（系统重启后依然生效）

   这种方法通过在内核启动参数中添加如下参数关闭相关补丁，重启操作系统使能修改：

   ```
   noibrs noibpb nopti
   ```

   说明

   这三个参数可以单独或组合设置，关闭部分补丁，从而获取不同的安全保护和性能影响。

2. 运行时关闭（系统重启后失效，需要重新配置）

   这种方法通过在命令行运行配置如下参数值来关闭相关补丁，配置后立即生效，不需要重启操作系统：

   ```
   # echo 0 > /sys/kernel/debug/x86/pti_enabled
   # echo 0 > /sys/kernel/debug/x86/ibrs_enabled
   # echo 0 > /sys/kernel/debug/x86/retp_enabled
   ```

   说明

   - 这三个配置操作需要root权限，并且需要已经加载debugfs文件系统。在EulerOS中，debugfs文件系统默认加载，如果被卸载，则可以通过如下命令进行手工加载：

     ```
     mount -t debugfs nodev /sys/kernel/debug
     ```

   - ibpb_enabled是只读的配置项，其值由内核设置。

验证修改

可以在命令行下通过如下方法校验开关关闭情况，执行cat命令获取的值如果是0，则表示该开关已经关闭。

```
# cat /sys/kernel/debug/x86/pti_enabled
# cat /sys/kernel/debug/x86/ibpb_enabled
# cat /sys/kernel/debug/x86/ibrs_enabled
# cat /sys/kernel/debug/x86/retp_enabled
```

说明

即使所有开关都已经关闭，有些应用程序也还是可能存在2%左右的性能损失。

spectre_v2 内核启动项参数

spectre_v2内核启动项参数控制消除变种#2漏洞：

- on: 无条件地使能内核消除变种#2漏洞
- off: 无条件地关闭内核消除变种#2漏洞
- auto: 内核检测CPU模型是否存在漏洞

当spectre_v2配置为off，ibrs、retp、ibpb都会关闭。也可以选择具体消除变种#2漏洞的方法：

- retpoline: 该方案取代间接分支
- ibrs: Intel补丁-内核以间接分支受限推断（indirect branch restricted speculation）方式运行；该方式能够保护内核空间免受攻击
- ibrs_always: Intel补丁-内核和用户态都以间接分支受限推断的方式运行；该方式能同时保护内核空间和用户空间免受攻击。

说明

spectre_v2选项没有指定的情况等同于spectre_v2=auto。

检查系统 CPU 状态

我们可以通过命令行检查当前系统CPU是否受Spectre和Meltdown漏洞的影响，执行cat命令获取具体信息：

```
# cat /sys/devices/system/cpu/vulnerabilities/meltdown
# cat /sys/devices/system/cpu/vulnerabilities/spectre_v1
# cat /sys/devices/system/cpu/vulnerabilities/spectre_v2
```

文件名称对应相应的漏洞名称，这些文件的输出反映了当前系统CPU的状态，可能的输出值如下：

```
"Not affected":   CPU不受漏洞的影响
"Vulnerable":     CPU受漏洞影响并且没有对应的消除漏洞的措施
"Mitigation: $M": CPU受漏洞的影响但通过方法$M消除漏洞的影响
```

37.12.3 Speculative Store Bypass 漏洞补丁开关（for x86_64）

说明

针对Speculative Store Bypass漏洞，部分产品（比如：统一存储、Dorado）未合入对应补丁，因此本章节描述的开关配置以及验证方法针对这些产品无效。

在现代微处理器中存在一个安全漏洞，用户需要对 Linux 内核、虚拟化相关的组件以及 microcode 进行更新。一个没有相关权限的攻击者可以绕过最基本的内存安全保护机制，访问到无法访问的内存。这个安全问题已被记录为 CVE-2018-3639，它也被称为“Variant 4（变体 4）” 或 “Speculative Store Bypass”。当前已知这个问题会影响到不同架构的 CPU，包括 AMD、ARM、IBM POWER8、POWER9 和 SystemZ 系列，以及 Intel 处理器。当前版本的EulerOS也受到影响。

CVE-2018-3639（也称为 “Speculative Store Bypass”）提供了一个新的通道（类似于分支预测错误（Branch Misprediction）），攻击者可通过指令预测执行和基于缓存的 side channel，利用这个通道绕过安全保护来访问需要特定权限才可访问的内存。这个问题与 CVE-2017-5753 （也称为 “Spectre v1”）类似，不同之处在于它会利用 Speculative Store Bypass 内存优化机制，Spectre v1 需要利用分支预测错误。

对于Speculative Store Bypass漏洞，EulerOS已经提供内核更新以修复此CVE漏洞。出于安全考虑，在更新的内核中默认启用这些修复，但由于“预测执行”属于处理器的性

能优化技术，启用之后其实就是禁用 MD（Memory Disambiguation）优化功能。这可能会影响到系统的性能。而性能受影响的程度取决于具体的系统负载。对于默认设置，EulerOS采取的基本原则是安全性优先于性能，然而，部分用户认为他们的系统已经有相当好的保护（比如物理隔离），不会受到此次漏洞影响，所以并不希望由于漏洞修复而导致不必要的性能损失。鉴于此，EulerOS提供了相关的启闭方法。如果用户认为不需要启用这些补丁，可以参考本文进行相应的配置来关闭补丁，以减少性能损失。

开关说明

Speculative Store Bypass漏洞修复只提供永久生效（系统重启后依然生效）的设置方法。

此方法通过在内核启动参数中添加参数，请见**表37-1**和**表37-2**，重启操作系统使修改生效。

说明

EulerOS建议用户使用由 CPU 厂商提供的 microcode/firmware 更新，并在相关的内核更新可用时尽快安装内核更新。软件更新可以独立于硬件 microcode 进行，但软件只有在 CPU firmware 已更新后才会起作用。

表 37-1 spec_store_bypass_disable 参数说明

参数值	参数说明
auto	默认值。在启动时使用这个选项，内核会检查处理器是否支持SSB（Speculative Store Bypass）功能，并选择适当的缓解方案。 没有指定spec_store_bypass_disable时等同于spec_store_bypass_disable=auto。实际上，auto缓解方案等同于prctl。
on	启动 Speculative Store Bypass 缓解方案。在所有存储（写）地址都被解析出来前，处理器将不会预测执行加载（读）指令。
off	禁用 Speculative Store Bypass 缓解方案。处理器会使用 MD（Memory Disambiguator）功能，在早期的存储（写）指令前可以预测执行加载（读）指令。
prctl	通过使用 prctl(2) 接口，请见**prctl接口说明**，基于每个线程启用 Speculative Store Bypass 缓解方案。 启动参数不同配置对prctl调用的影响请见**表37-4**。

表 37-2 nospec_store_bypass_disable 参数说明

参数值	含义
nospec_store_bypass_disable	等价于spec_store_bypass_disable=off。禁用 Speculative Store Bypass 安全漏洞的所有缓解方案。

开关配置

如需要永久关闭，则在内核启动参数中添加如下参数：

```
spec_store_bypass_disable=off 或者 nospec_store_bypass_disable
```

检查系统漏洞修复状态

我们可以通过命令行检查当前系统是否受Speculative Store Bypass漏洞的影响，执行cat命令获取具体信息：

```
cat /sys/devices/system/cpu/vulnerabilities/spec_store_bypass
```

可能的输出值如下：

```
"Not affected"：CPU不受漏洞的影响
"Vulnerable"：受CPU漏洞影响，并且没有对应的消除漏洞的措施
"Mitigation: Speculative Store Bypass disabled"：已消除CPU漏洞的影响，使用的是 Speculative Store Bypass 缓解方案
"Mitigation: Speculative Store Bypass disabled via prctl"：已消除CPU漏洞的影响，使用的是prctl Speculative Store Bypass 缓解方案
```

prctl 接口说明

内核通过更新 prctl(2) 接口提供了每个进程的控制选项。

应用程序可以使用它基于每个进程来启用或禁用 Speculative Store Bypass。应用程序可以使用以下方式使用 prctl(2)：

- 获取进程当前 Speculative Store Bypass 状态。返回值说明请见表37-3。

```
prctl(PR_GET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, 0, 0, 0);
```

- 启用 Speculative Store Bypass 功能，没有采用补救方法。

```
prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, PR_SPEC_ENABLE, 0, 0);
```

- 禁用 Speculative Store Bypass 功能，使补救方法有效。

```
prctl(PR_SET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, PR_SPEC_DISABLE, 0, 0);
```

表 37-3 PR_GET_SPECULATION_CTRL 返回值说明

返回值	含义	说明
bit 0	PR_SPEC_PRCTL	如果为1，表示可以通过prctl(PR_SET_SPECULATION_CTRL)设置此进程来消除 Speculative Store Bypass 漏洞的影响。 如果为0，表示不支持prctl(PR_SET_SPECULATION_CTRL)设置。如果调用，系统返回失败。
bit 1	PR_SPEC_ENABLE	如果为1，表示 Speculative Store Bypass 特性是打开的, 即：没有消除漏洞的影响。
bit 2	PR_SPEC_DISABLE	如果为1，表示 Speculative Store Bypass 特性是禁用的, 即：会应用缓解方案，来消除漏洞的影响。

表 37-4 不同的启动参数对 prctl 调用的影响

spec_store_bypass_disable	spec_store_bypass状态	get SSB status	set SSB to enable	set SSB to disable
不设置或者设置auto、prctl	Mitigation: Speculative Store Bypass disabled via prctl	默认状态为： PRCTL : 1 ENABLE : 1 DISABLE: 0	设置成功以及新状态为： PRCTL : 1 ENABLE : 1 DISABLE: 0	设置成功以及新状态为: PRCTL : 1 ENABLE : 0 DISABLE: 1
设置off或者nospec_store_bypass_disable	Vulnerable	默认状态为： PRCTL : 0 ENABLE : 1 DISABLE: 0	设置失败以及状态依旧为： PRCTL : 0 ENABLE : 1 DISABLE: 0	设置失败以及状态依旧为： PRCTL : 0 ENABLE : 1 DISABLE: 0
设置on	Mitigation: Speculative Store Bypass disabled	默认状态为： PRCTL : 0 ENABLE : 0 DISABLE: 1	设置失败以及状态依旧为： PRCTL : 0 ENABLE : 0 DISABLE: 1	设置失败以及状态依旧为： PRCTL : 0 ENABLE : 0 DISABLE: 1

说明

如果prctl(PR_GET_SPECULATION_CTRL, PR_SPEC_STORE_BYPASS, 0, 0, 0)的返回值所有bit都是0，表示CPU不受漏洞的影响。

37.12.4 MDS 漏洞补丁开关

背景介绍

Intel公开了一组新的预测执行侧信道漏洞，命名为：MDS（microarchitectural data sampling）。该组漏洞涉及4个CVE，详细信息如下：

CVE	CVSS Score	CVSS Vector
CVE-2018-12126	6.5	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N

CVE	CVSS Score	CVSS Vector
CVE-2018-12127	6.5	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N
CVE-2018-12130	6.5	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:N/A:N
CVE-2019-11091	3.8	AV:L/AC:L/PR:L/UI:N/S:C/C:L/I:N/A:N

攻击者利用该组漏洞可以获取Host的敏感信息，或者获取同一物理核相邻Guest的敏感信息。Zombie Load几乎影响到2011年以来的所有英特尔芯片和基于Intel芯片的所有操作系统（Windows，Linux，MacOS，Android，Chrome，iOS等）。

攻击场景

- 通用场景

场景	是否受影响	概述
OS Kernel	是	攻击者利用钓鱼、木马注入等方式将恶意程序注入到OS本地，进而在用户态获取Kernel数据、Kernel内存、页表信息等。
用户态进程之间	是	1. 攻击者利用钓鱼、木马注入等方式将恶意程序注入到OS本地，可以获得其它进程的敏感数据； 2. 攻击者利用浏览器基于WebAssembly生成机器码进行攻击，可以突破JavaScript的沙箱环境获取用户其它进程的敏感数据。
虚拟机管理器（VMM）	是	攻击者在VM中运行，可以利用该漏洞获取Host的敏感数据或者同一物理核的其它VM数据。

- 各领域场景

领域	是否受影响	概述
手机	可能	1. 基于Intel芯片的手机会受影响； 2. 非基于Intel芯片的手机暂时不受影响。
公有云	是，风险高	攻击者可以在任意Guest端，获取Host的敏感信息或者同一物理机下相邻Guest的敏感信息。
封闭系统产品	是，风险低	封闭系统产品，从技术角度来讲，都是受影响的。但是由于这类产品，部署场景比较封闭，接入都有相应的权限管理控制，无利用渠道（入侵途径），因此可以认为不受影响，整体风险低。

开关说明

EulerOS已经提供内核更新以修复这些CVE漏洞。出于安全考虑，在更新的内核中默认启用这些修复。但由于“预测执行”属于处理器的性能优化技术，而此次的安全漏洞修复影响“预测执行”原有逻辑，所以在某些业务场景下，对于内核和微码的更新可能会导致性能下降。然而，部分用户认为他们的系统已经有相当好的保护（比如物理隔离），不会受到此次漏洞影响，所以并不希望由于漏洞修复而导致不必要的性能损失。鉴于此，EulerOS提供了相关的启闭方法，如果用户认为不需要启用这些补丁，可以参考本文进行相应的配置来关闭补丁，以减少性能损失。

MDS漏洞修复只提供永久生效（系统重启后依然生效）的设置方法。此方法通过在内核启动参数中添加参数，重启操作系统使修改生效，mds参数说明如下：

表 37-5 mds 参数说明

参数值	参数说明
full	默认值，开启MDS防护。在启动时使用这个选项，内核使能全部可用的MDS缓解方案。
full,nosmt	开启MDS防护并关闭超线程。在启动时使用这个选项，内核使能全部可用的MDS缓解方案，而且关闭SMT（Simultaneous multithreading）。
off	关闭MDS防护。关闭后可能存在安全风险，请谨慎使用该操作。

说明

- 裸机场景，mds开关的使能，需要物理机升级微码后才支持。如果物理机没有升级微码，即使在启动参数配置mds开关，补丁也不生效。
- GuestOS场景，Guest内mds开关的使能，除了依赖host升级微码，还依赖于host虚拟化组件将mds功能透传到GuestOS。

校验修改

可以在命令行下通过如下方法校验开关关闭情况，执行cat命令获取结果：

```
# cat /sys/devices/system/cpu/vulnerabilities/mds
```

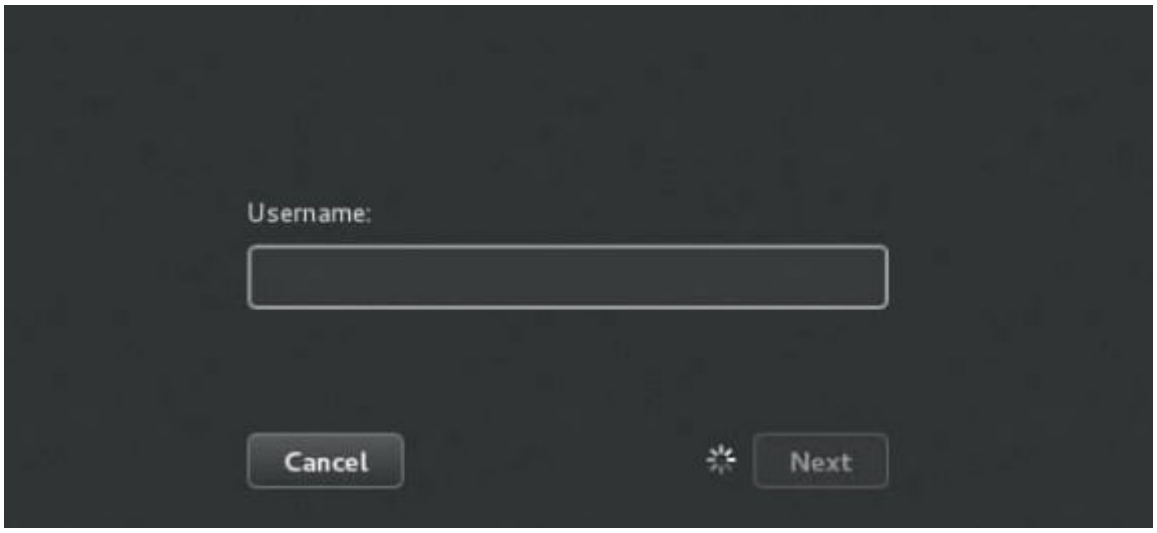
结果值	参数说明
Not affected	该CPU处理器不受MDS漏洞影响。
Vulnerable	该CPU处理器受MDS漏洞影响，而且没有使能MDS缓解方案。
Vulnerable: Clear CPU buffers attempted	该CPU处理器受MDS漏洞影响，但是微码(microcode)没有升级。MDS方案只能尽最大努力减少该漏洞的影响。
Mitigation: CPU buffer clear	该CPU处理器受MDS漏洞影响，而且微码(microcode)也升级成功了，MDS缓解方案全部使能。

37.13 图形界面用户登录卡住现象说明

问题现象

用户使用图形界面时，如果输错密码三次，触发锁定300s，解锁后输入用户名，点击Next按钮，会一直卡住，无法登录成功。或者处于图形登录界面长时间不操作时，同样会出现该现象。如图37-2所示：

图 37-2 用户无法登录



原因分析

该现象是由于GNOME图形登录通信依赖gnome-shell与gdm之间直连的dbus连接，当用户处于登录界面长时间无响应，连接超时会自动断开，断开后继续通信则会一直卡住。点击cancel回到登录主界面重新建立连接则恢复正常。

解决方法

点击cancel回到主界面，重新输入用户名，则可正常登录。

37.14 mail 命令发送邮件失败时不可以使用-S 指定文件的说明

问题现象

当mail客户端执行如下命令提示失败时，不会在指定文件中记录发送失败的邮件信息。

```
echo "hello world" | mail -S DEAD="/root/aaa" -s "test" aaa@huawei.com
```

原因分析

为避免邮件发送失败时，默认产生的/root/dead.letter文件不断增大而占满分区的风险。EulerOS做了加固，对发送失败的邮件默认不记录到文件中，导致在使用mail命令时-S参数不再起作用。

解决方法

如果用户需要记录发送失败的邮件到指定文件，需要修改mail的配置文件，具体方法如下：

```
#vi /etc/mail.rc
```

修改set -S DEAD=""为set -S DEAD="~/$MY_HOPE_DIR"。

说明

$MY_HOPE_DIR是由用户指定的保存路径。

37.15 逻辑卷创建后被自动挂载且挂载点不可用现象说明

问题现象

当使用lvcreate命令创建一个新的逻辑卷后，使用"df -h"查看挂载情况：

```
[root@localhost lvtest]# df -h
Filesystem                Size  Used Avail Use% Mounted on
/dev/mapper/euleros-root   13G  2.5G  9.7G  21% /
devtmpfs                  1.9G     0  1.9G   0% /dev
tmpfs                     1.9G     0  1.9G   0% /dev/shm
tmpfs                     1.9G   41M  1.9G   3% /run
tmpfs                     1.9G     0  1.9G   0% /sys/fs/cgroup
/dev/sda1                 477M   98M  351M  22% /boot
tmpfs                     378M     0  378M   0% /run/user/0
/dev/mapper/vgtest-lvtest  64Z   64Z  958M 100% /home/lvtest
[root@localhost lvtest]#
```

发现新建的逻辑卷被自动挂载了，但逻辑卷的size变得无限大，并且挂载点无法写入。

原因分析

引起该问题需要以下四个要素：

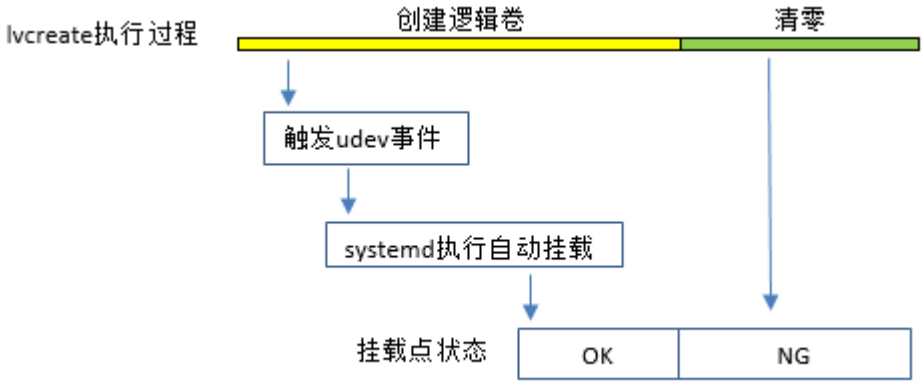
1. 在/etc/fstab中配置了逻辑卷自动挂载项，且已生效。
2. 逻辑卷的文件系统为ext文件系统（fat文件系统已验证不存在此问题，其他文件系统未验证）。
3. udev相关的服务开启：

   systemd-udevd

   systemd-udevd-control.socket

   systemd-udevd-kernel.socket
4. 在同一个卷组中，对名为xxx的逻辑卷执行“创建-删除-再创建”操作（再创建的逻辑卷的大小不能小于前者）。

当删除逻辑卷时，原逻辑卷中文件系统相关的信息仍保留在磁盘上。在磁盘的相同位置新建一个同名同大小的逻辑卷时，该文件系统的信息会被新的逻辑卷复用。lvcreate执行过程如下：



由于lvcreate执行清零的操作在文件系统挂载操作之后，已经挂载的文件系统被清零操作破坏，从而导致了"df -h"显示异常，挂载点不可用。

解决方法

在删除逻辑卷之前，先将逻辑卷中的文件系统系统信息清除。以删除/dev/vgtest/lvtest逻辑卷为例：

```
#dd if=/dev/zero of=/dev/vgtest/lvtest bs=1M count=32
#lvremove /dev/vgtest/lvtest
```

37.16 系统内核大量冲日志导致系统卡住无法登录

现象

在系统内核态大量冲日志并且用户态也大量写入日志情况下，系统会卡住无法进行ssh登录等操作。

原因分析

用户态日志和内核态日志同时打印时，因日志优先级问题，会优先处理内核态日志。当内核态大量冲日志并且内核态写入速率大于journal处理速率时，就会导致journal无法处理用户态日志。当用户态继续使用syslog接口打印日志时，由于journal socket缓冲池/dev/log空间占满，又无法被journal读取，新发送的日志无法获取返回值造成阻塞。进一步导致用户态各个进程执行时打印日志会卡住，从而导致系统无法登录，或进行其他操作。

解决方法

这个问题属于systemd本身性能问题。在该问题解决之前，用户可通修改rsyslog配置文件来避免这种问题。

需要在以下两个地方同时修改rsyslog配置：

- 在/etc/rsyslog.conf文件中将“$OmitLocalLogging on”修改为“$OmitLocalLogging off”。

- 在/etc/rsyslog.d/listen.conf文件中添加“$SystemLogSocketName /dev/log”。

说明

修改完配置后重启rsyslog服务生效。

37.17 防火墙规则不合理和设置 tcp_sack=0 对网络性能的影响

现象

防火墙设置不合理（例如：TCPOPTSTRIP tcp -- anywhere anywhere TCPOPTSTRIP options mss,sack-permitted,sack,timestamp,md5）或设置tcp_sack=0时，在丢包严重场景下会导致系统网络性能大幅下降。

原因分析

在丢包严重的场景下，tcp连接在发生丢包时，SACK可以把接收端报文序列号的空洞反馈给发送端，发送端在进行重传恢复的时候可以避免把所有报文再重新发送一次。

而防火墙“TCPOPTSTRIP tcp -- anywhere anywhere TCPOPTSTRIP options mss,sack-permitted,sack,timestamp,md5”把SACK需要的信息给strip掉了，导致重传的数据信息丢失。与tcp_sack=0本质一样，会导致发送端将所有报文重发，从而导致整体网络性能的下降。

解决方法

- 系统中默认tcp_sack=1，用户不用修改该设置。
- 防火墙中不能添加类似strip报文中SACK信息的规则。

37.18 openLDAP 服务压力规格说明

在客户端使用SSSD的场景下，openLDAP 服务端短时间内收到一个客户端的10000次查询请求，所需CPU资源大约在60%～80%。

当openLDAP 服务端进程可用的CPU资源在20%以下时，SSSD客户端有极大概率会出现请求超时。

由于openLDAP服务端所处的环境存在不确定性，建议将SSSD客户端的ldap_search_timeout配置项设置为10s，以保证在openLDAP服务端所处OS压力过大，其服务进程能使用的CPU资源较小的情况下，SSSD也能正常工作（减小超时的可能性）。

此外，大量的IO读写也会拖慢服务端的响应速度（openLDAP服务需要读取磁盘上的数据库），大量写日志既会产生大量IO也会使得journal服务的CPU使用率极大上升抢占IO和CPU资源，建议服务端的日志级别配置项olcLogLevel不要小于32（数值越小日志量越大）。

37.19 SSSD 开启 enumerate 选项后的性能说明

SSSD的enumerate选项开启后会将服务端的所有数据都缓存到客户端本地，此过程需要持续一段时间才能完成。在同步期间，所有针对用户名和组名的单次请求都会直接从

服务端获取数据而不是查询本地缓存，由于此时SSSD忙于同步数据，所以这样的单次请求会变得非常缓慢。直到数据全部缓存完毕后，单次请求的速度才能恢复正常。根据SSSD官方的建议，enumerate选项不建议开启。

37.20 极端情况下 nslcd 服务 OOM 导致 host 解析失败的现象说明

nslcd在高并发（1000个并发）时内存使用率会持续增加，当系统内存使用率达到一定负荷时，nslcd服务有可能发生OOM进而导致进程终止（panic_on_oom=0时），此后nslcd服务进入失败状态。经过一段时间（默认一小时）后，nscd缓存中的host记录开始过期，此时ping命令在解析域名对应的IP地址时由于缓存已过期且nslcd服务已经失败，导致无法从本地缓存和openLDAP服务端获取任何信息，最终提示“unknown host”。建议根据系统内存大小合理限制并发数量，避免内存耗尽。

37.21 高并发场景下 nscd 服务缓存 negative entries 导致 host 解析失败的问题说明

在高并发的场景下，nslcd客户端需要向openLDAP服务端查询大量信息，openLDAP服务端由于不能及时响应，导致nslcd无法查询到host记录并将negative entries写入缓存以减轻服务端压力。在negative entries过期之前（默认20秒），ping命令等解析操作都将立即返回unknown host的结果（持续20秒）。如果需要重新获取positive entries则需要等到negative entries自然过期或者手动清除nscd的host缓存以便缓存重建。我们建议合理安排客户端的并发数量，保证openLDAP服务端可以稳定地响应请求。

37.22 Intel VMD-暴力拔 NVMe SSD 盘操作指南

场景描述

VMD作为最新Intel CPU中集成的新硬件，能够有效管理NVMe SSD的指示灯状态、热插拔功能。暴力热插拔指的是不通知系统的情况下，直接将盘从系统中拔出，此时盘上可能有I/O负载。本节内容用于说明暴力拔盘的正常处理流程，本节所说暴力拔出的盘均是已经挂载到系统中的NVMe SSD盘。

操作步骤

假设我们待拔盘的设备名称在系统中称为：/dev/nvme0n1p1，挂载点：/mnt/nvme0

1. 暴力拔出NVMe SSD
2. 从系统中卸载设备：umount /dev/nvme0n1p1

说明

如果此处不卸载设备，执行df命令会发现/dev/nvme0n1p1设备仍然挂载在/mnt/nvme0挂载点，该设备名称会被占用。

37.23 使用 authconfig 命令修改 PAM 配置失败

问题现象

使用authconfig命令，针对PAM配置进行修改，修改失败 。

原因分析

EulerOS针对PAM的配置，默认进行了安全加固，为了防止加固配置被覆盖，用户通过authconfig命令自动配置无法直接生效。

解决方法

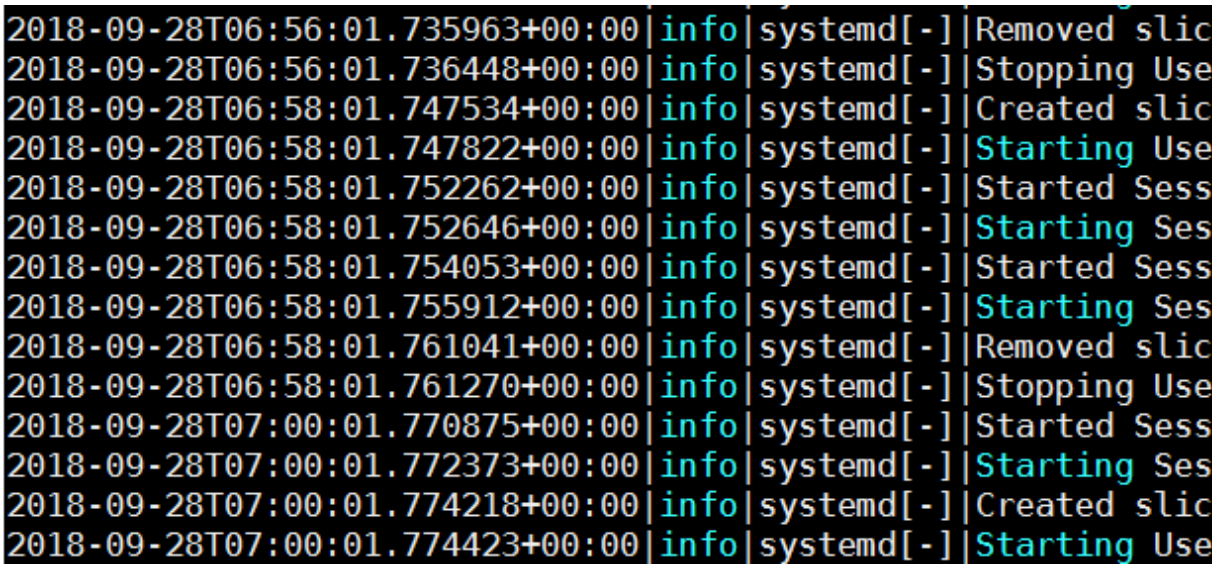
如果用户要在现有PAM配置下进行变更，有以下两种方法。

- 方法一：使用authconfig命令进行配置，命令执行之后，需要将软连接/etc/pam.d/system-auth和/etc/pam.d/password-auth指向/etc/pam.d/system-auth-ac和/etc/pam.d/password-auth-ac文件。
- 方法二：手动配置/etc/pam.d/system-auth-local和/etc/pam.d/password-auth-local文件。

37.24 systemd 进程频繁打印“Starting Session XX of user USERNAME”日志

问题现象

在没做任何操作情况下/var/log/messages不断打印“Starting Session XX of user USERNAME”等一些列日志的信息，这些日志会造成冲日志嫌疑。



原因分析

cron定时服务在执行时，定时任务中配置的特定用户通过systemd拉起会话，并在会话中执行相应的命令或者脚本。systemd拉起会话的过程会打印此类日志，频繁执行定时任务会导致systemd不断打印会话连接的日志到messages文件中。

```
Euler:~ # cat /etc/cron.d/sysstat
# Run system activity accounting tool every 10 minutes
*/10 * * * * root /usr/lib64/sa/sa1 1 1
```

解决方法

systemd打印会话日志属于正常信息，一般情况下日志量很少，只有在cron服务比较频繁的时候才会大量打印此类日志。要过滤messages中大量的会话日志，可以通过在rsyslog服务中添加配置进行操作，具体命令如下：

```
echo 'if $programname == "systemd" and ($msg contains "Starting Session" or $msg contains "Started Session" or $msg contains "Created slice User Slice of"  or $msg contains "Starting User Slice of" or $msg contains "Removed slice User Slice of" or $msg contains "Stopping User Slice of") then stop' >/etc/rsyslog.d/ignore-systemd-session-slice.conf
```

重启rsyslog服务后配置生效：

```
systemctl restart rsyslog
```

37.25 通过虚拟机 img 镜像转换为 qcow2 镜像，复制虚拟机后启动会增加 1 个 ipv6 地址

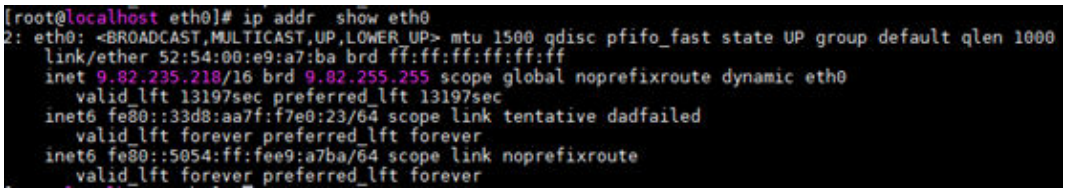
问题现象

通过虚拟机img镜像转换为qcow2镜像，复制虚拟机后启动，复制后的虚拟机会比原虚拟机多了1个ipv6地址，并且复制后的虚拟机中有与原虚拟机一样的IPv6地址。

原因分析

Centos 7.5默认对网络接口ifcfg-****配置文件中将IPv6生成模式设置为stable-privacy模式：IPV6_ADDR_GEN_MODE="stable-privacy"。同样对齐了开源Centos 7.5的EulerOS V2R7同样也会将网络接口ifcfg-****配置文件中将IPv6生成模式设置为stable-privacy模式。stable-privacy模式会根据网络信息和伪随机值来生成具有隐私保障的IPv6地址（RFC7217：RID = F(Prefix, Net_Iface, Network_ID, DAD_Counter, secret_key)），相比于根据MAC地址来生成IPv6地址的EUI64模式，stable-privacy模式生成的IPv6地址更加安全且能保障用户隐私。其中创建stable-privacy模式的IPv6地址依赖于伪随机值，而该伪随机值只被该主机知晓，并且会被一个密码安全加密算法进行加密，一起被加密的还有网络连接的特定值（包括：用以标识网卡的名称、网络地址、其它网络相关的特殊值）。所以当进行虚拟机拷贝时，主机上的伪随机值和网络相关的值都没有改变，因此会产生与被复制的虚拟机相同的IPv6地址。该复制的虚拟机使用该IPv6地址前需要经过重复地址检测DAD，而重复地址检测DAD后发现该IPv6地址已经被使用，所以该IPv6地址的状态被设置为“Tentative”状态。重复地址检测DAD没有发现冲突的IPv6地址的状态才会被设置为“PreferedAddr”状态，只有被设置了“PreferedAddr”状态的IPv6地址才能被使用。所以当DAD发生该IPv6地址重复后，该IPv6地址尽管还没删除，但是不会被使用。此时主机会根据随机值、地址重复检测的次数和网络相关值来重新生成IPv6地址。所以最终产生多个IPv6地址，而其中重复的IPv6地址的状态为“Tentative”状态而不被使用。

如下图，由于fe80::33d8:aa7f:f7e0:23/64被其它虚拟机使用了，所以进行重复地址检测DAD后会发现该地址重复，将该IPv6地址的状态设置为"Tentative"而不被使用。



总结：网络接口ifcfg-****配置文件中将IPv6生成模式设置为stable-privacy模式：IPV6_ADDR_GEN_MODE="stable-privacy"后，复制虚拟机必然会产生多个IPv6地址，尽管有多个IPv6地址，但是被设置为“Tentative”状态的IPv6地址并不会被使用，所以这种现象是在配置IPv6生成模式为stable-privacy模式后的正常现象。

解决方法

如果网络接口ifcfg-****配置文件中将IPv6生成模式设置为stable-privacy模式，则复制虚拟机后必然会出现多个IPv6地址的现象。可以将IPV6_ADDR_GEN_MODE="stable-privacy"配置注释掉来选择默认的EUI64模式生成IPv6地址来规避该现象。

37.26 如何解决配置 2G 内存虚拟机热插内存到 128G 后重启卡住的问题

问题现象

UVP KVM的企业虚拟化场景，使用EulerOS（64位）作为虚拟机操作系统，虚拟机配置2G内存启动后，分别依次热插2G、4G、8G、16G、32G、64G内存且依次上线热插的内存，直到虚拟机总内存为128G时重启虚拟机，虚拟机卡死，VNC或者串口日志有如下消息输出：

```
page      allocation failure
```

原因分析

该问题是操作系统内核自身机制。热插内存后，虚拟机自动上线内存或者用户手动上线内存，需要初始化页表空间，这些页表空间由虚拟机配置内存中的可用内存提供。当虚拟机可用内存不足时，虚拟机在上线热插的内存时会卡死。

在虚拟机重启过程中，第一阶段初始化虚拟机配置的初始内存，第二阶段上线热插的设备内存。在第二阶段中，当前虚拟机配置初始内存的可用内存不足2G（内核或其他用户进程会占用掉部分内存），无法为128G热插的设备内存的上线提供所需的页表空间，导致虚拟机卡死。

解决方法

为虚拟机配置更多的内存解决该问题，建议虚拟机配置初始内存的可用内存与热插的内存的比例至少为1：32。

以虚拟机名为“*EulerOS_64*”为例，具体步骤如下：

步骤1 登录宿主机执行如下命令，强制关闭虚拟机：

```
virsh destroy EulerOS_64
```


步骤2 执行如下命令，修改虚拟机内存为4G或更多（建议虚拟机配置初始内存的可用内存与热插的内存的比例至少为1：32，即要热插128G内存虚拟机至少需要配置4G内存）：

```
virsh edit EulerOS_64
```

将以下加粗的数值分别修改为如下数值或者更大：

```
<memory      unit='KiB'>4194304</memory>
<currentMemory      unit='KiB'>4194304</currentMemory>
```

步骤3 执行如下命令，启动虚拟机：

```
virsh start EulerOS_64
```

----结束

37.27 scp 传送大文件速度逐渐变为 0，传送缓慢问题的解决方法

问题现象

当出现这三个条件共存的情况下会出现scp速度变为0问题。

1. 设置防火墙规则过滤掉sack-permitted、sack 报文头信息：
   ```
   [root@EulerOS-BaseTemplate ~]# iptables -L -t mangle
   ......
   Chain INPUT (policy ACCEPT)
   target prot opt source destination
   TCPOPTSTRIP tcp -- anywhere anywhere TCPOPTSTRIP options mss,sack-
   permitted,sack,timestamp,md5
   ......
   ```
2. 设置 tcp_sack = 1
   ```
   [root@EulerOS-BaseTemplate ~]# sysctl -a | grep tcp_sack
   net.ipv4.tcp_sack = 1
   ```
3. 在链路上发生丢包。

原因分析

- net.ipv4.tcp_sack与防火墙规则冲突导致。
- tcp_sack 设置快速重传，但是防火墙规则把sack需要的信息给strip掉了，导致重传的数据失效，只能慢速传输，速度到最低。
- 在不发生丢包的情况下，没有触发重传场景，则不复现该问题。

解决方法

有两种修改方法，解决冲突问题：

- 打开sack功能，把防火墙过滤sack-permitted,sack规则去掉，在丢包场景下tcp性能较好。
- 关闭sack功能，设置net.ipv4.tcp_sack = 0，在丢包乱序场景下tcp性能较差。

37.28 l2nic_interrupt_switch 参数使用说明

问题现象

使用1822网卡PF直通运行keepalived时，在重启过程中，后端上线的速度相比于用virtio网卡慢很多。

原因分析

1822网卡硬件寄存器被hostos接管，当guestos的网卡中断到来时，也要写那个寄存器，存在资源抢占问题，在寄存器的实现中，有添加硬件锁，导致软中断cpu冲高，性能下降。内核态驱动hinic新增模块加载参数：l2nic_interrupt_switch，参数的默认值是true，针对PF直通VM场景，基于性能因素，可以将l2nic_interrupt_switch设置成false，提升性能。

解决方法

步骤1 在/etc/modprobe.d/ 目录下部署一个配置文件，如 hinic.conf 文件，配置文件内容如下：

```
# This file is intended for users to select the various module options for hinic

options hinic l2nic_interrupt_switch=N
```

步骤2 部署配置后，重启虚拟机即可生效。

----结束

37.29 使用 ping -I 指定网卡，ping 本机 IP 导致无法 ping 通

问题现象

ping -I 指定网卡，然后去ping这个网卡配置的IP地址，结果无法ping通。

例如：

```
[root@localhost ~]# ip add show ens3
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
link/ether 52:54:00:ae:**:** brd ff:ff:ff:ff:**:**
inet 10.0.0.1/16 scope global dynamic ens3
valid_lft 41228sec preferred_lft 41228sec
[root@localhost ~]# ping -I ens3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) from 10.0.0.1 ens3: 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 5999ms
```

原因分析

使用 ping -I 指定网卡时，内核会设置sk->sk_bound_dev_if为指定网卡的ifindex。然后去ping这个网卡配置的IP地址时，由于是本机网卡的IP地址，会直接通过环回口收到这个报文，对应的skb->dev的ifindex为环回口的ifindex。两个ifindex不一致，使得查找raw socket时，找不到对应的sock，导致报文被丢弃。

解决方法

ping本机网卡的IP地址时，不需要使用 -I 指定网卡。

例如：

```
[root@localhost ~]# ip add show ens3
2: ens3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
link/ether 52:54:00:ae:**:** brd ff:ff:ff:ff:**:**
inet 10.0.0.1/16 scope global dynamic ens3
valid_lft 41228sec preferred_lft 41228sec
[root@localhost ~]# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.357 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.217 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.153 ms
^C
--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.153/0.242/0.357/0.086 ms
```

37.30 执行 ping 命令时，向前更改系统时间，导致短时间网络不可达

执行ping命令，ping某个ip地址时，如果同时向前修改本端机器的系统时间，会导致当前时间超过arp表项的超时时间，从而触发arp表项的异步删除机制，可能导致arp表项丢失，短时间内网络不可达，ping命令输出错误信息ping: sendmsg: Network is unreachable，网络不可达持续几秒钟左右，在相应arp表项重新建立之后，ping命令继续正常执行。

37.31 bond 使用说明

37.31.1 注意事项

EulerOS支持商用的bond模式包括bond 1、bond 3、bond 4、bond 6。

37.31.2 配置 bond 网卡为 mode0，查询 ipv6 地址时会有 tenative dad_failed 后缀

问题现象

配置bond网卡，模式为mode 0，绑定多个子网卡，再配置ipv6地址，使用ip addr查询ipv6地址时，会有tenative dad_failed后缀，ipv6地址不可用。

例如：

bond0配置两个子网卡，eth2和eth3：

```
[root@server ~]# cat /proc/net/bonding/bond1
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 100
Up Delay (ms): 0
```

```
Down Delay (ms): 0

Slave Interface: eth2
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:9c:**:**
Slave queue ID: 0

Slave Interface: eth3
MII Status: up
Speed: Unknown
Duplex: Unknown
Link Failure Count: 0
Permanent HW addr: 52:54:00:b7:**:**
Slave queue ID: 0
```

查看配置的ipv6地址：

```
[root@server ~]# ip addr show bond1
16: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
link/ether 52:54:00:9c:**:** brd ff:ff:ff:ff:**:**
inet6 2001:111:1::**:**/64 scope global tentative dadfailed
valid_lft forever preferred_lft forever
inet6 fe80::f840:1da0:**:**/64 scope link noprefixroute
valid_lft forever preferred_lft forever
```

原因分析

配置bond网卡，模式为mode 0，绑定多个子网卡，再配置ipv6地址时，会发送dad地址冲突检测的ipv6组播报文，内核协议栈会拷贝组播报文给每个子网卡，这样其他的子网卡也会收到dad组播报文，并认为配置的ipv6地址冲突了，并标记为tenative dad_failed。

解决方法

配置bond网卡，模式为mode 0时不配置ipv6地址，或者设置bond网卡的accept_dad字段为0。

例如：

```
[root@server ~]# sysctl net.ipv6.conf.bond1.accept_dad=0
net.ipv6.conf.bond1.accept_dad = 0
```

重新配置ipv6地址可以看到tentative dadfailed已经消失了：

```
[root@server ~]# ip add show bond1
16: bond1: <BROADCAST,MULTICAST,MASTER,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
qlen 1000
link/ether 52:54:00:9c:**:** brd ff:ff:ff:ff:**:**
inet6 2001:111:1::**:**/64 scope global noprefixroute
valid_lft forever preferred_lft forever
inet6 fe80::f840:1da0:**:**/64 scope link noprefixroute
valid_lft forever preferred_lft forever
```

37.32 nfs 的客户端上 mount 相关进程卡住

问题现象

使用nfs时，mount默认采用hard模式挂载。在mount成功以后，服务端发生重启。但在服务端重启完成之后，客户端在访问之前挂载的目录时，仍然卡住。

原因分析

服务端重启之前配置了多个ip地址，客户端采用hard模式挂载了多个ip的共享目录。在服务端重启完成之后，部分ip没有恢复配置。导致客户端在访问原先挂载的共享目录时，访问失败，nfs request会一直重试，导致客户端mount相关进程卡住。

解决方法

在使用hard模式挂载时，如果客户端出现mount相关进程卡住，需要排查服务端的网络、nfs服务、rpcbind服务等是否已经恢复正常。在服务端恢复正常以后，客户端进程即可自动恢复。 另外，建议在使用hard模式时增加intr选项来硬挂载目录，这时当有某个进程进入了重试循环，则允许用户使用键盘将其中断。

说明

1. mount命令在挂载时默认采用hard模式。
2. 在nfs客户端采用hard模式挂载时，可以保证客户端和服务端数据的一致性，不会出现静默数据错误。但当服务端出现异常的时候，客户端会一直向服务端发出请求，直到服务端恢复正常，进而导致客户端进程一直阻塞。
3. 在nfs客户端采用soft模式挂载时，可以通过timeo和retry参数配置超时时间。服务端出现异常时，客户端会向服务器端重发请求。当超过配置的超时时间时，则返回错误，不会一直阻塞。但在soft模式下可能会出现静默数据错误。

37.33 ifconfig 设置 IP，掩码不正确

问题现象

ifconfig设置IP，IP会被替换，设置掩码，掩码会错乱，原始IP情况如下：

```
3: br0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 8a:75:38:a5:3f:b2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.3/13 scope global br0
       valid_lft forever preferred_lft forever
    inet 192.168.4.3/19 scope global br0
       valid_lft forever preferred_lft forever
```

执行以下命令后：

```
ifconfig br0 192.168.7.7 netmask 255.255.255.255
```

```
3: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN qlen 1000
    link/ether 8a:75:38:a5:3f:b2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.7.7/24 brd 192.168.7.255 scope global br0
       valid_lft forever preferred_lft forever
    inet 192.168.3.3/32 brd 192.175.255.255 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::8875:38ff:fea5:3fb2/64 scope link
       valid_lft forever preferred_lft forever
```

原因分析

ifconfig设置IP和添加mask分两步进行的，先设置IP再设置mask。此命令是设置IP而非添加IP，会将已有的IP进行替换。内核当中interface的IP地址由链表来保存，每次执行命令会替换掉链表头的项，并将其加到链表尾部。

上图当中，添加IP时候链表头部的IP是192.168.4.3，所以192.168.4.3的IP被替换掉了，修改掩码时链表头部的IP是192.168.3.3，所以它的掩码被修改为32。

此系内核机制就是如此，并非bug。

解决方法

1. 建议使用ip addr命令来添加IP和设置掩码。
2. 用ifconfig添加IP时，应该在IP前面加上add。如：
   ```
   ifconfig lo add 192.168.100.100
   ```
3. 在interface上有超过1个IP时，建议不要使用ifconfig修改掩码。

37.34 IPMI

37.34.1 简介

IPMI（Intelligent Platform Management Interface），即智能平台管理接口，是一种基于message的平台管理接口标准。虚拟机Watchdog特性模拟了一个满足IPMI标准的虚拟Watchdog，为虚拟机提供了一种心跳检测机制，主要用于监视VM内部系统的健康状况。

37.34.2 约束限制

XEN平台虚拟机不支持IPMI Watchdog功能。

37.34.3 ipmitool 使用安全说明

问题现象

执行如下命令，history命令中会记录明文密码，存在密码泄露风险。

```
ipmitool -I lanplus -H 192.168.10.13 -U test_name -P password
```

原因分析

对于涉及bash命令行明文密码的非安全操作方式，会导致明文密码被记录到history命令中，存在密码泄露风险。

解决方法

1. 请及时清理history以防密码泄露，清理命令如下：
   ```
   history -d 行号
   ```
2. 建议使用交互式输入，不在命令行中输入密码，命令如下：
   ```
   #ipmitool -I lanplus -H 192.168.10.13 -U test_name
   Password:
   ```

37.34.4 ipmi 相关模块加载说明

问题现象

在UVP_2.3版本的主机上进行EulerOS V2R7版本虚拟机安装，ipmi相关ko默认未加载；而UVP_2.5版本的主机上进行EulerOS V2R7版本虚拟机安装，ipmi相关ko默认加载。

原因分析

EulerOS V2R7内核删除了ipmi_si的trydefaults参数，需要使用ports=0xca2来进行配置。因为UVP_2.5版本提供ipmi设备的模拟，还提供dmi呈现，而UVP_2.3版本仅提供ipmi设备的模拟，未提供dmi呈现。ipmi驱动扫描dmi中是否存在ipmi信息，不存在就不会加载。

37.34.5 ipmi_si 模块加载卸载卡住说明

问题现象

- 现象一：在ipmitool执行命令处理超时，如执行“ipmitool sel clear”命令清除日志返回超时，然后重启ipmi服务，lsmod命令查看ipmi_si模块状态，会发现used值显示为-1的现象。

```
[root@localhost ipmi]# lsmod | grep ipmi
ipmi_devintf           20480  0
ipmi_si                73728  -1
ipmi_msghandler        69632  2 ipmi_devintf,ipmi_si
[root@localhost ipmi]# ps aux | grep ipmi
root       30537  0.0  0.0 213032   848 pts/2    S+   15:06   0:00 grep --color=auto ipmi
root      123899  0.2  0.0   4020   948 ?        D    14:17   0:05 modprobe -q -r ipmi_si
[root@localhost ipmi]#
```

- 现象二：频繁操作ipmi服务重启，执行“modprobe ipmi_si”命令卡住现象。

```
Euler:~ # lsmod | grep ipmi
ipmi_si                57540  1
ipmi_devintf           17603  0
ipmi_msghandler        46607  2 ipmi_devintf,ipmi_si
Euler:~ # ps axu | grep ipmi
root      20840  0.0  0.0 112708   972 pts/2    S+   18:35   0:00 grep --color=auto ipmi
root      49044  0.0  0.0  14824   820 ?        D    18:03   0:00 modprobe ipmi_si
Euler:~ #
```

原因分析

- 现象一：出现上述现象，是由于卸载ipmi_si卡住，执行“modprobe -q -r ipmi_si”命令耗时较长。

  ipmi模块在处理超时命令的过程中，内部计时器处理未完成的情况下，将该模块重启，因为数据处理不为空，进入了内部循环处理逻辑中，计时器递减值被设置为 10us，导致计时延长，需要等待计时结束，才会去处理数据，退出循环逻辑，将ipmi_si模块卸载掉。

- 现象二：“modprobe ipmi_si”卡住的原因是频繁执行ipmi服务重启操作，会出现获取device id等错误，导致注册模块失败现象，进入了错误处理逻辑，该逻辑处理同上述现象一原因说明。

解决方法

卸载命令具体的卡住时间需要根据ipmi_si重启时，计时器的运行时间来计算，时间计算为一次计时器减10，1s约执行110次，初始计数值为5000000（单位为us）理论上最大卡顿值约为75分钟，问题场景下，一般会卡住20~30分钟。在卡住情况下，需要等待计时结束后，再重启ipmi服务，功能才恢复正常。

建议不要频繁执行ipmi模块卸载和加载操作，出现命令执行超时时，不要立即执行重启服务操作。如需要重启，建议在命令超时5分钟后进行重启，确保内部计时器运行结束，避免出现卸载ipmi_si卡顿时间较长现象。

37.35 NetworkManager

37.35.1 NetworkManager 内存占用说明

问题现象

使用ip命令批量创建大量虚拟网卡设备并删除，NetworkManager的内存占用不回落，再次创建同等数量的虚拟网络设备不上涨。

原因分析

NetworkManager大量使用g_ptr_array，g_array以及ghash等glib2动态库的数据结构来管理设备信息。当大量地并发执行创建、删除、修改设备时会产生内存碎片，这种内存占用不是内存泄露，是NetworkManager设计上的缺陷。

对于不同的虚拟网卡，以1000个数量为例，通过批量添加，配置ip，路由，设置网卡信息等操作，内存占用情况如下：

虚拟网卡类型	数量	内存占用
veth	1000	192M
vlan	1000	128M
dummy	1000	192M
macvlan	1000	128M
bridge	1000	192M
bond	1000	192M

解决方法

可以根据系统实际网卡情况来监控NetworkManager的内存占用。如果所有虚拟网卡被删除而NetworkManager内存不回落，可通过重启NetworkManager服务释放内存占用。

重启服务命令：

```
systemctl restart NetworkManager
```

37.35.2 禁用 NetworkManager 造成 network-online.target 服务无法启动

问题现象

禁用NetworkManager服务，导致network-online.target服务无法启动。

原因分析

系统启动的依赖顺序如下：

1. NetworkManager.sevice。
2. NetworkManager-wait-online.sevice。

3. network-online.target。

所以如果NetworkManager.sevice服务被禁用，则会导致network-online.target服务无法启动。

解决方法

步骤1 使用如下命令，开启NetworkManager服务：

```
systemctl enable NetworkManager
```

步骤2 使用如下命令，重新开启network-online.target服务：

```
systemctl start network-online.target
```

----结束

37.35.3 重新加载网卡驱动和网卡芯片故障重置场景下网卡配置失败问题

问题现象

在重新加载网卡驱动和网卡芯片故障重置场景下，网卡没有根据ifcfg-**配置文件进行配置。

问题现象

network脚本不支持网卡热插拔，而NetworkManager支持网卡热插拔。如果不使用NetworkManager，则在重新加载网卡驱动和网卡芯片故障重置场景下，如果不重启，网卡不会根据ifcfg-**配置文件进行配置。

解决办法

方法1：如果只使用network服务管理网卡，在上述重新加载网卡驱动和网卡芯片故障重置场景下，需要用户重启network服务，来使更改的网卡配置生效。

方法2：安装并开启NetworkManager，使用NetworkManager对网卡进行管理。

37.36 gssproxy 认证后生成临时文件 krb5cc_%{uid}影响同 uid 用户认证

问题现象

用户通过gssproxy认证后会在/tmp目录下生成临时文件krb5cc_%{uid}，这个临时文件是根据uid生成的。如果删除了此用户和其keytab文件，但是没有删除此临时文件，则下次创建了同uid的用户，gssproxy认证会失败。

原因分析

当临时文件krb5cc_%{uid}存在时，无法重新生成同名文件，其内容又与新的用户不匹配，导致认证失败。

解决办法

- 手动删除。使用kerberos提供的kdestroy命令手动删除此临时文件，或者在/etc/login.defs中修改如下内容进行删除。
  ```
  USERDEL_CMD /usr/sbin/userdel_local
  ```
- 自动删除。在/etc/krb5.conf中进行如下修改，把临时文件存入内存，进程结束自动删除。
  ```
  default_ccache_name = MEMORY:%{uid}
  ```

37.37 华为安全函数库说明

当前安全函数库版本Huawei Secure C V100R001C01SPC006B002。

- 用户态：提供安全函数动态库和对应的开发包。

  动态库所属rpm包名为libsecurec， EulerOS默认安装，文件路径/lib64/libsecurec.so；开发包rpm包名为libsecurec-devel，仅在Euler_compile_env.tar.gz默认安装，编译引用头文件 #include <securec.h>；未提供Euler_compile_env.tar.gz的，需要产品自行安装该rpm包。
- 内核态：提供安全函数ko和对应的开发包

  ko所属rpm包名为ksecurec，EulerOS默认安装并加载， 文件路径/lib/modules/EulerOS/securec/ksecurec.ko；开发包rpm包名为ksecurec-devel，仅在Euler_compile_env.tar.gz默认安装，编译引用头文件 #include <linux/securec.h>；未提供Euler_compile_env.tar.gz的，需要产品自行安装该rpm包。

  内核态安全函数当前版本只支持7个函数：memcpy_s、memmove_s、memset_s、strcat_s、strcpy_s、strncat_s、strncpy_s。
- 责任主体：安全函数库由安全能力中心提供，EulerOS只负责集成。
- 安全函数库资料：**http://3ms.huawei.com/km/groups/2034125/blogs/details/5761845**。

37.38 lv 中包含两种扇区规格的磁盘时，用 mount 命令挂载，导致系统重启

问题现象

- 硬件条件：物理机上有两种扇区规格的盘，一种盘的扇区大小为512字节，一种盘的扇区大小为4096字节；且需要用到lvm将两种盘混用来做逻辑分区。
- 软件操作步骤：vgcreate命令创建vg，vg中包含两种规格的盘；lvcreate命令创建条带化的lv；mkfs命令对lv使用ext4文件系统格式化。

此时，使用mount命令挂载lv到指定目录，会触发系统重启现象。

原因分析

当lv中包含两种扇区规格的磁盘时，创建的lv扇区大小是4096字节。第一次调用mount命令挂载时，内核会初始化ext4系统的inode table，内核函数sd_setup_write_same_cmnd会对比lv设备的扇区大小（4096字节）与物理磁盘上的扇区（512字节）大小，对比结果不一致就调用BUG_ON让系统重启。

解决方法

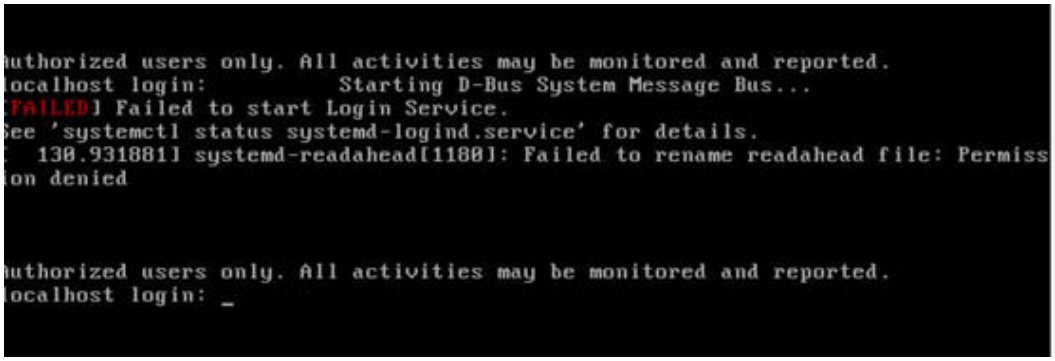
- 避免出现两种扇区规格的磁盘进行混用创建lv。
- 当需要两种规格的磁盘混用创建lv的时候，调用mkfs指令时，添加选项“-E lazy_itable_init=0”，可规避该问题。这种规避方式带来的影响是mkfs命令会等初始化inode table结束才返回，lv越大，耗时越长（实测10T lv耗时2分钟左右）。

37.39 SELinux 重新配置为 enforcing 重启登录失败

问题现象

EulerOS默认情况下SELinux为使能状态，用户因为某种原因把SELinux置为disable状态，后面又在/etc/selinux/config配置里面设置SELINUX=enforcing，重启后登录输入正确用户名和密码也会失败。

图 37-3 登录失败



重启修改添加启动参数selinux=0关闭SELinux后，能重新登录。在/var/log/message或者/var/log/audit/audit.log日志里能搜索到关键字unlabeled_t记录。

原因分析

SELinux在enforcing模式，所有文件都会有个扩展属性存储标记，设置成disable后，创建或重新生成的文件是没有标记签的，因为没有SELinux策略在运行。但重新配置为enforcing 之后，是不会自动给所有文件打标记，执行时遇到访问没有标签的文件操作时就会拒绝。

解决方法

使各文件的标记都能和SELinux策略配置的一致起来。在/etc/selinux/config中，当disable配置为enforcing时，同时需要在根目录下使用“touch /.autorelabel”命令创建文件，重启时会给所有文件打标记。正确标记是SELinux运行的基础。

37.40 新增网卡保序、自动创建/dev/disk/by-id 链接的 UDEV 规则

问题现象

之前需要手动创建udev规则，将指定Mac地址的网卡与指定网口名进行绑定。

当前系统中默认合入了3条规则，分别针对的是网卡x、VF，将其名称固化为ethX、ethX.vfX的格式：

- 55-persistent-net-generator.rules
- 56-net-sriov-names.rules
- 61-euleros-persistent-storage.rules（自动创建/dev/disk/by-id链接）

原因分析

系统启动时，55-persistent-net-generator.rules规则分别会生成高优先级的50-persistent-net.rules，里面会记录新识别网卡的mac和对应的网卡名称，从而达到下次再重启时网卡的名称不变化。

解决方法

当制作模板镜像时，如果在安装的过程存在一张网卡，会生成50-persistent-net.rules，并占用了eth0的网卡名称，当使用此模板镜像实例化时，新环境中的网卡名称不会从eth0开始命名，是从后面的某一个序号递增eth1。要解决此问题有两种方式：

1. 将上面的网卡保序相关的3条udev规则删除，不使用网卡保序的功能。
2. 将生成的50-persistent-net.rules删除，不残留制作模板时的网卡信息。

37.41 krb5 常见问题

37.41.1 krb5 授权失败

问题现象

在krb5客户端执行kadmin等命令时，出现类似于如下错误提示（其中nfs/admin可以是其他票据名称）

```
[root@client01 ETS]# kadmin
Authenticating as principal nfs/admin@EXAMPLE.COM with password.
kadmin: Client 'nfs/admin@EXAMPLE.COM' not found in Kerberos database while initializing kadmin
interface
```

原因分析

krb5服务端数据库中没有nfs/admin（或其它）对应的票据信息，导致认证授权失败。

解决方法

在krb5服务端通过如下命令（其中“nfsnfs”为该票据设置的口令，“nfsnfs\nnfsnfs”表示输入两遍口令，中间用“\n"分隔），添加nfs/admin（或其它）票据信息，即可解决本问题。

```
echo -e 'nfsnfs\nnfsnfs'|kadmin.local -q 'addprinc nfs/admin'
```

37.41.2 krb5 普通用户获取票据后对挂载的共享文件写操作失败问题

问题现象

在krb5客户端普通用户获取票据后对挂载的共享文件进行写操作，第一次操作成功，反复执行后操作失败，显示Permission denied。

```
[root@client01 ETS]# echo -e "rootroot"| kadmin -q "ktadd -k /var/lib/gssproxy/clients/1000.keytab
glance@EXAMPLE.COM"
Authenticating as principal nfs/admin@EXAMPLE.COM with password.
Password for nfs/admin@EXAMPLE.COM:
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type aes256-cts-hmac-sha1-96 added
to keytab WRFILE:/var/lib/gssproxy/clients/1000.keytab.
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type aes128-cts-hmac-sha1-96 added
to keytab WRFILE:/var/lib/gssproxy/clients/1000.keytab.
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type des3-cbc-sha1 added to keytab
WRFILE:/var/lib/gssproxy/clients/1000.keytab.
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type arcfour-hmac added to keytab
WRFILE:/var/lib/gssproxy/clients/1000.keytab.
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type camellia256-cts-cmac added to
keytab WRFILE:/var/lib/gssproxy/clients/1000.keytab.
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type camellia128-cts-cmac added to
keytab WRFILE:/var/lib/gssproxy/clients/1000.keytab.
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type des-hmac-sha1 added to keytab
WRFILE:/var/lib/gssproxy/clients/1000.keytab.
Entry for principal glance@EXAMPLE.COM with kvno 3, encryption type des-cbc-md5 added to keytab
WRFILE:/var/lib/gssproxy/clients/1000.keytab.
[root@client01 ETS]# su glance -c "echo testEL_krb5_FUN.003 > /client_testEL_krb5_FUN.003/
test.glance"
bash: /client_testEL_krb5_FUN.003/test.glance: Permission denied
```

原因分析

如果出现第一次获取票据后写操作成功而第二次失败的现象，大多数情况下失败是由前面操作后环境没有恢复，存在大量文件缓存导致后续的认证无法成功。比如：krb5 1.16版本新增接口服务sssd-kcm.socket，在进行写操作时sssd-kcm.socket会产生缓存，进而导致下一次操作失败；另外，gssproxy认证产生的秘钥表也会导致下一次操作失败。

解决方法

在普通用户获取票据后对挂载的共享文件写操作执行完，删除/tmp/下的认证缓存，删除/var/lib/gssproxy/clients/下的普通用户认证秘钥表，使用kdestroy命令清除掉sssd-kcm留下的缓存。操作示例如下：

```
rm -rf /tmp/krb5cc_*
rm -rf /var/lib/gssproxy/clients/*keytab
su username -c "kdestroy"
```


37.42 强制下电导致文件系统报错

问题现象

将系统以强制下电的非正常方式关闭，会概率性出现文件系统异常或者数据异常。

原因分析

强制断电是属于高危行为，写数据的过程中，异常掉电且硬件没有掉电保护会导致写入数据不一致，导致存储的数据再次被访问的时候会出现问题。

解决方法

需要硬件有防掉电机制来保护数据一致性，对于损坏不是很严重的可以用fsck指令进行恢复。

1. 基于安全的原则，先将出错的分区卸挂载（/dev/sda1为示例）。
   ```
   umount /dev/sda1
   ```
2. 查看文件系统的类型。
   ```
   fdisk -l
   ```
3. 使用fsck指令进行恢复。
   ```
   fsck  -t  文件系统类型  /dev/sda1
   ```

37.43 基于 ipvlan l2e 模式组网，tcp 本地传输过慢

问题现象

基于ipvlan l2e模式的组网，tcp报文本地传输速率过慢。

原因分析

基于ipvlan l2e模式的组网，在网卡未开启GSO和TSO的情况下，ipvlan l2e模式缓存小包的特性成为性能瓶颈，导致tcp报文传输速率大幅下降。

解决方法

调整ipvlan的参数：net.ipvlan.loop_qlen。

- 对于网卡开启GSO和TSO的情况下，net.ipvlan.loop_qlen的默认值是65536*2=131072，为两个大包的大小，无需调整。
- 对于未开启GSO和TSO的情况下，需要调整net.ipvlan.loop_qlen为一个MTU，比如1496。修改方法：
  ```
  sysctl net.ipvlan.loop_qlen=1496
  ```


37.44 并发掉线/上线磁盘，磁盘大小变为 0，变成不可用状态

问题现象

在添加/删除逻辑卷且并发磁盘掉线/上线操作时，有极低概率会发生某个磁盘无法使用的问题，此时使用lsblk -a查看块设备会发现SIZE列的值变为0或没有显示。

原因分析

磁盘对应的内核数据nr_sects在此情况下会被更改为0，从而导致lsblk -a命令显示的结果是该磁盘SIZE列的值变为0或没有显示，其他对磁盘操作的程序也会由于同样的原因而失败。

解决方法

当该问题发生时，推荐使用如下命令，重新扫描磁盘以恢复磁盘：

```
echo 1 > /sys/block/<磁盘名>/device/rescan
```

37.45 多队列场景下的报文乱序问题

问题现象

在虚拟机配置网卡多队列且进程没有绑核场景下，并且HostOS采用dpdk方案，可能会导致报文乱序问题。

原因分析

在虚拟机配置网卡多队列且进程没有绑核场景下，默认单条流会分配到多个cpu进行处理。

解决方法

卸载并重新安装virtio_net.ko, 并在重新安装时配置using_XPS参数为1。

```
rmmod virtio_net
modprobe virtio_net using_XPS=1
```

37.46 nfs-utils 包使用限制

场景介绍

系统是内存文件系统的场景，此时系统的根目录在内存中。

使用限制

nfs服务器端在/etc/exports文件中指定客户端的ipv6地址，服务器端的挂载点务必要挂载在磁盘上，同时将该挂载点设置为nfs的主文件系统，设置方法为在exports文件中配置fsid=0参数。此后客户端使用mount命令挂载的方法为：


```
$ mount [服务端ipv6地址%客户端出口网卡]:/ 客户端挂载目录
```

37.47 单 cpu 软中断太多并且无法避免，从而导致软狗复位应如何处理

问题现象

存储阵列上，某些业务配置会使得单个cpu上的中断和软中断任务太多，可能导致该cpu上的喂狗线程无法得到调度，从而软狗复位。

原因分析

该现象是由于中断和软中断处理都是中断上下文，优先级高于内核线程，如果cpu一直处于中断上下文，线程是无法得到调度的。

解决方法

- 在grub.cfg文件中配置启动参数“softirq_delay=1”，表示在软中断执行时间达到一个阈值时将软中断任务交由ksoftirqd内核线程处理，避免一直处于中断上下文。
- 配置“softirq_delay=0”表示关闭此功能，但有可能出现上述问题，请用户根据实际情况进行配置。

  说明

  EulerOS V2.0SP5通用版本默认支持此功能，不需要用户配置。需要时可以通过配置softirq_delay=0来关闭此功能。

37.48 TCP 连接发送缓冲区设置较小时连接断开

问题现象

业务进程通过setsockopt将发送缓冲区sndbuf设置为较小的值（如4K），那么sndbuf太低就很容易达到内核安全检测机制的水线值，返回NOMEM预警，会出现报文发送延迟的情况。如果产品的用户态有机制主动监测这个延时，延时达到一定值后reset掉这个连接，从而出现连接断开的情况。

原因分析

内核的这个安全检测机制是针对CVE-2019-11478的修复，CVE原理如下：攻击者可以利用这个 Linux TCP SACK的内核安全漏洞，发送一组经过精心设计的SACK数据包来使TCP重新发送队列被分段。攻击者再利用分段的队列，使系统在处理使用相同TCP连接所接收到的后续SACK数据包时消耗大量资源。这将导致CPU花费大量时间来重新构建数据列，从而最终出现DoS问题。修复方案中就是对sndbuf使用情况作出提前预警，防止出现资源被消耗光，导致系统不可用。**https://access.redhat.com/zh_CN/security/vulnerabilities/4233431**。

修复方案是检测发送队列上的数据字节大于“发送缓冲区设置的值+一定常量”，就返回“-ENOMEM”。因此，在发送缓冲区设置的较小并且发送队列上的数据字节较大的情况下，就会满足内核检测条件。

解决方法

发送缓冲区的值，用内核的默认值16K对于大多数场景是没有问题的，在小于这个值的情况下，可能出现的影响包括但不限于：性能降低，连接异常断掉，连接延时加大。因此建议产品用户态如果需要设置SO_SNDBUF，至少改成16K以上。

37.49 rpm 、yum、dnf 等命令被强制 kill 后出现挂死问题

问题描述

rpm 在正常运行期间如果被强制kill（使用kill -9或 killall -9命令），可能导致当前其他正在执行或后续执行rpm、yum、dnf命令挂死。

原因分析

rpm使用了libdb的接口来维护数据，libdb模块为了支持多进程互斥，使用了共享锁机制；如果rpm执行命令期间被强制退出可能导致libdb的接口加共享锁后未解除，导致其他进程不能正常加锁从而挂死。因为yum、dnf依赖rpm接口，因此也会出现类似问题。

解决方法

1. 杀死所有的rpm、yum、dnf进程。
```
killall -9 rpm
killall -9 yum
killall -9 dnf
```
2. 删除/var/lib/rpm/__db.* 文件。
```
rm -rf /var/lib/rpm/__db.*
```
3. 再次执行rpm、yum、dnf等命令即可正常。

37.50 术语

- **account**

  指允许个人连接到系统的登录名称、个人目录、密码以及shell的组合。

- **alias**

  别名。在shell中为了能在执行命令时将某一字符串替换成另一个的一种机制。在提示符中键入alias可了解当前所定义的全部别名。

- **ARP**

  Address Resolution Protocol（地址解析协议）。该网际网络协议用于将网际网络地址动态地对应到局域网络的硬件地址上。

- **batch**

  批处理。将工作按顺序送到处理器，处理器一个接一个执行直到最后一个完成并准备好接受另一组处理清单的一种处理模式。

- **boot**

  引导。即发生在按下计算机的电源开关，机器开始检测接口设备的状态，并把操作系统加载到内存中的整个过程。

- **bootdisk**

  引导盘。包含来自硬盘（有时也可从其本身）加载操作系统的必要程序代码的可开机软磁盘。

- **BSD**

  Berkeley Software Distribution（伯克利软件发行套件）。一套由美国伯克利大学信息相关科系所发展的Unix分支。

- **buffer**

  缓冲区。指内存中固定容量一个小区域，其中的内容可以加载区域模式文件，系统分区表，以及执行中的进程等等。所有缓冲区的连贯性都是由缓冲区内存来维护的。

- **buffer cache**

  缓冲区存取。这是操作系统核心中甚为重要的一部份，负责让所有的缓冲区保持在最新的状态，在必要时可以缩小内存空间，清除不需要的缓冲区。

- **CHAP**

  Challenge-Handshake Authentication Protocol（询问交互式身份验证协议）：ISP验证其客户端所采用的通信协议。它与PAP的不同处在于：进行最初的判别后，每隔固定的时间周期它将会重新再验证一次。

- **client**

  客户端。是指能够短暂地连接到其他程序或计算机上并对其下达命令或要求信息的一个程序或一部计算机。它是服务器/客户端系统组件的一部分。

- **client/server system**

  服务器/客户端系统。由一个server（服务器端）与一个或多个client（客户端）所组成的系统架构或通信协议。

- **compilation**

  编译。指把人们读得懂的以某种程序语言（例如C语言）书写的程序源代码转换成机器可读的二进制文件的一种过程。

- **completion**

  自动补齐。只要系统内有能与之配合对象，shell将自动把一个不完全的子字符串，延展扩大成一个已存在的文件名、用户名。

- **compression**

  压缩。这是一种在通信连接的传送过程中缩小文件或减少字符数目的方法。压缩程序通常包含有compress，zip，gzip及bzip2。

- **console**

  控制台。也就是人们一般使用并称为终端的概念。它们是连接到一部巨型中央计算机的使用者操作的机器。对PC而言，实际的终端就是指键盘与屏幕。

- **cookies**

  由远程web服务器写入到本地硬盘的临时文件。它让服务器可以在使用者再次连上网站的时候可以知道其个人偏好。

- **DHCP**

  Dynamic Host Configuration Protocol（动态主机配置协议）。一种以局域网络机器为设计基础，能从DHCP服务器动态取得IP地址的通信协议。

- **DMA**

  Direct Memory Access。一种运用在PC架构上的技术，它允许接口设备可以从主存储器存取或读写资料而无须通过CPU联系。

- **DNS**

  Domain Name System（网络域名系统）。用来负责分配名称/地址的机制。它可以将机器名称对应到IP地址。同样DNS也允许反向搜寻，也就是说可以从IP地址得知其机器名称。

- **DPMS**

  Display Power Management System（显示器电源管理系统）。用于所有现今生产的显示器以管理其电源使之能够延长使用年限的协议。

- **editor**

  编辑器。一般而言是指编辑文本文件所使用的程序（也就是文字编辑器）。最为人所熟知的GNU/Linux编辑器有Emacs以及VIM。

- **email**

  电子邮件。是处于相同网络里的人们互相传送电子信息的一种方式。与定期邮件相同，email需要收件人以及寄件人地址以便正确地传送信息。

- **NVRAM**

  非易失性随机访问存储器(Non-Volatile Random Access Memory)，是指断电后仍能保持数据的一种RAM。

