

openEuler 操作系统

实验手册

v1.6.1



华为技术有限公司

目录

1 构建实验环境	4
1.1 实验介绍	4
1.1.1 关于本实验	4
1.1.2 实验组网介绍	4
1.1.3 实验设备介绍	6
1.2 构建实验环境	7
1.2.1 创建 VPC.....	7
1.2.2 购买 ECS	9
1.2.3 通过 ssh 登录系统.....	11
2 openEuler 系统环境实验	13
2.1 实验介绍	13
2.1.1 关于本实验	13
2.1.2 实验目的	13
2.2 系统编程环境实验	13
2.2.1 查看系统信息	13
2.2.2 查看编程环境	15
3 iSulad 实验	19
3.1 实验介绍	19
3.1.1 关于本实验	19
3.1.2 实验目的	19
3.1.3 参考资料	19
3.2 安装 isulad.....	19
3.2.1 安装	19
3.2.2 启动并查看版本	19
3.3 容器与镜像管理	20
3.3.1 准备工作	20
3.3.2 运行容器 busybox.....	22

3.3.3 运行容器 openeuler:20.09	22
3.4 使用 isula-build 构建容器镜像.....	25
3.4.1 安装 isula-build	25
3.4.2 构建容器镜像并导入到 isulad.....	27
3.4.3 扩展实验： isula-build 的其他镜像导出方式.....	28
4 智能优化引擎 A-Tune 实验.....	30
4.1 实验介绍	30
4.1.1 关于本实验	30
4.1.2 实验目的	30
4.1.3 参考资料	30
4.2 安装和启动 A-Tune.....	30
4.2.1 安装	30
4.2.2 启动	31
4.3 运行 atune-adm 命令	33
4.3.1 查看版本	33
4.3.2 查询负载类型	33
4.3.3 分析负载类型并自优化	33
4.3.4 系统信息查询	34
4.3.5 离线业务自调优	34
5 内核编程实验.....	36
5.1 实验介绍	36
5.1.1 关于本实验	36
5.1.2 实验目的	36
5.2 内核编程实验	36
5.2.1 内核的编译与安装	36
5.2.2 Hello, world!	39
5.2.3 使用 tasklet 打印 Hello, world!.....	40
6 资源清理	42
6.1 资源清理	42
6.1.1 ECS 关机	42
6.1.2 删除资源	43



7 思考题	45
8 附录	46
8.1 在 x86_64 平台上进行编译内核实验	46
8.2 在本地 PC 与虚拟机之间互相拷贝文件	49
8.2.1 从本地 PC 拷贝文件到远端虚拟机	49
8.2.2 从远端虚拟机拷贝文件到本地 PC	50
8.3 终端软件的使用	50
9 缩略语表	51

1 构建实验环境

1.1 实验介绍

1.1.1 关于本实验

openEuler 是一款通用服务器操作系统，支持 x86 和 ARM 等多种处理器架构，本实验旨在熟悉基于 Kunpeng 架构弹性云服务器 ECS 上 openEuler 操作系统基本系统环境、学习轻量级容器 iSulad 的基本用法以及了解智能调优引擎 A-Tune 的调优过程。

有关 openEuler 的详细介绍和资源可以参考如下网址：

<https://openeuler.org/zh/documentation/>

<https://gitee.com/openeuler/>

1.1.2 实验组网介绍

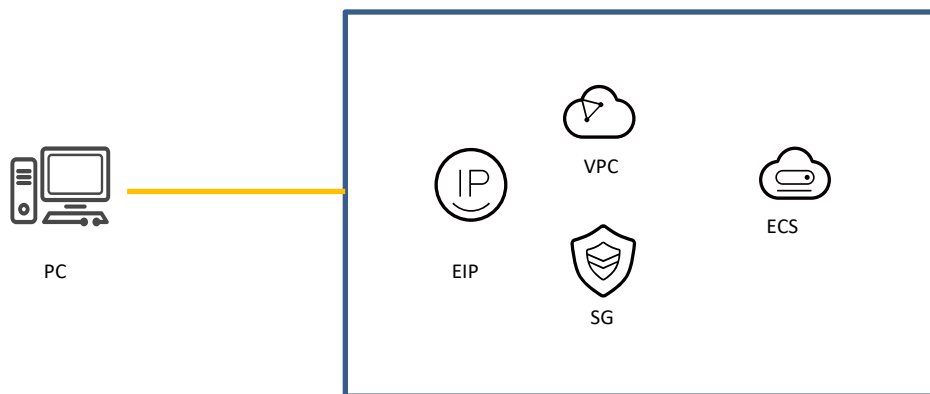


图1-1 openEuler 操作系统实验的云环境

注意，图中缩略语全程如下：

缩略语	英文全称	中文全称
ECS	Elastic Cloud Server	弹性云服务器

EIP	Elastic IP	弹性IP地址
PC	Personal Computer	个人电脑
SG	Security Groups	安全组
VPC	Virtual Private Cloud	虚拟私有云

1.1.3 实验设备介绍

关键配置如下表所示:

表1-1 关键配置

云资源	规格
ECS 鲲鹏计算	2vCPUs 4GB 40GB
EIP 带宽	按流量计费

软件方面, 本实验需要一台终端电脑与弹性云服务器(ECS)链接以输入操作命令或/和传输文件。对于 Windows 10 / macOS / Linux, 我们可以用命令行工具 ssh 和 scp 完成这个过程。

如果在有些 Windows 系统下不能运行 ssh 工具, 也可以使用 Putty 和 WinSCP 工具软件。其中 Putty 工具的推荐下载地址:

<https://hcia.obs.cn-north-4.myhuaweicloud.com/v1.5/putty.exe>

WinSCP 的推荐下载地址:

<https://winscp.net/eng/index.php>

下文若无特殊说明, 均以命令行工具 ssh 和 scp 为例进行讲解。

1.2 构建实验环境

1.2.1 创建 VPC

步骤 1 在浏览器地址栏输入华为云控制台网址 console.huaweicloud.com 并按回车键，这时页面将跳转至登录页。




步骤 2 按要求输入账号密码，进行登录。

注意：在此之前您需要在华为云主页注册华为云账号。

步骤 3 登录成功后会自动进入控制台页面，这时将区域选在 “华北-北京四”。



步骤 4 将鼠标悬停于左侧导航栏  图标处展开服务列表，然后在服务列表中点击 “虚拟私有云 VPC” 项。



步骤 5 点击“虚拟私有云”控制台页面右上角的“创建虚拟私有云”按钮。



步骤 6 在创建虚拟私有云的页面中按照下表内容配置虚拟私有云参数。

参数	配置
区域	华北-北京四
名称	vpc-test
网段	192.168.1.0/24
企业项目	default
默认子网可用区	可用区1
默认子网名称	subnet-test
子网网段	如192.168.1.0/24

步骤 7 配置完成后，点击“立即创建”，创建完成后会自动回到 VPC 控制台。

步骤 8 点击 VPC 控制台左侧导航栏的“访问控制” → “安全组”，进入安全组控制台。




步骤 9 点击右上角的“创建安全组”。



步骤 10 在弹出的对话框中按“通用 Web 服务器”配置安全组参数，然后点击“确定”。



1.2.2 购买 ECS

步骤 1 将鼠标悬停于左侧导航栏  图标处展开服务列表。然后在服务列表中点击“弹性云服务器 ECS”项。



步骤 2 点击弹性云服务器 ECS 控制台页面右上角的“购买弹性云服务器 ECS”按钮进入购买页面。



步骤 3 按照下表内容配置弹性云服务器 ECS 的参数。

参数	配置
计费模式	按需计费
区域	华北-北京四
可用区	可用区1
CPU架构	鲲鹏计算
规格	鲲鹏通用计算增强型 kc1.large.2 2vCPUs 4GB
镜像	公共镜像 openEuler openEuler 20.03 64bit with ARM(40GB)
系统盘	通用型SSD 40GB

注意：这里“区域”的配置是和 VPC 的区域配置保持一致的。

步骤 4 配置完成后点击“下一步：网络配置”，进入网络配置，按下表配置网络参数。

参数	配置
网络	vpc-test subnet-test 自动分配IP地址
安全组	sg-test
弹性公网IP	现在购买
线路	全动态BGP
公网带宽	按流量计费
带宽大小	5Mbit/s

步骤 5 配置完成后，点击“下一步：高级配置”，按下表配置 ECS 高级配置参数。

参数	配置
云服务器名称	openEuler (输入符合规则名称)
登录凭证	密码
密码	请输入8位以上包含大小写字母、数字和特殊字符的密码, 如 Euler@123
确认密码	请再次输入密码
云备份	暂不购买
云服务器组	不配置
高级选项	不勾选

步骤 6 配置完成后点击右下角“下一步：确认配置”。勾选同意协议，然后点击：“立即购买”。

步骤 7 在提交任务成功后，点击“返回云服务器列表”，返回 ECS 控制台。

1.2.3 通过 ssh 登录系统

步骤 1 在 ECS 控制台查看 ECS 弹性公网 IP 地址。

名称/ID	监控	可用区	状态	规格/镜像	IP地址	计费模...
openEuler 00ad36b3-1920-47a...	 	可用区1	 运行中	2vCPUs 4GB kc1... openEuler 20.03 64...	119.8.238.1... 192.168.1.1...	按需计费 2020/10/14...

步骤 2 在客户端机器操作系统里的 Console 控制台或 Terminal 终端里运行 ssh 命令：

```
$ ssh root@119.8.238.181
```

(注意：此处的 IP 地址 119.8.238.181 即是刚刚购买的弹性公网 IP 地址。)

在客户端（本地 PC）第一次登录时会有安全性验证的提示：

```
The authenticity of host '119.8.238.181 (119.8.238.181)' can't be established.  
ECDSA key fingerprint is SHA256:RVxC1cSuMmqLtWdMw4n6f/VPsfWLkT/zDMT2q4qWxc0.  
Are you sure you want to continue connecting (yes/no)? yes
```

在这里输入 yes 并按回车键继续：

Warning: Permanently added '119.8.238.181' (ECDSA) to the list of known hosts.

Authorized users only. All activities may be monitored and reported.

root@119.8.238.181's password:

输入密码(注意这里不会有任何回显)并回车，登录后的界面如下所示：

Welcome to Huawei Cloud Service

Last login: Wed Aug 19 20:10:20 2020 from 119.3.119.18

Welcome to Huawei Cloud Service

Last login: Wed Aug 19 20:10:20 2020 from 119.3.119.18

Welcome to 4.19.90-2003.4.0.0036.oe1.aarch64

System information as of time: Wed Aug 19 20:19:56 CST 2020

System load: 0.00
Processes: 118
Memory used: 11.9%
Swap used: 0.0%
Usage On: 12%
IP address: 192.168.1.88
Users online: 1

[root@openeuler ~]#

2 openEuler 系统环境实验

2.1 实验介绍

2.1.1 关于本实验

本实验通过运行 shell 命令查看系统信息以达到了解 openEuler 操作系统的目的。另外运行简单的 C 程序和汇编程序以了解基于 ARMv8-64 的开发环境。

2.1.2 实验目的

- 了解 openEuler 操作系统的基本信息；
- 了解基于 Kunpeng 架构的 openEuler 操作系统开发环境。

2.2 系统编程环境实验

2.2.1 查看系统信息

步骤 1 查看总体架构

```
[root@openeuler ~]# uname -a
Linux openeuler 4.19.90-2003.4.0.0036.oe1.aarch64 #1 SMP Mon Mar 23 19:06:43 UTC 2020 aarch64 aarch64
aarch64 GNU/Linux
```

可以看到 CPU 的架构是 aarch64。

提示：执行 “uname --help” 可查看更多命令。

步骤 2 查看操作系统信息

```
[root@openeuler ~]# cat /etc/os-release
NAME="openEuler"
VERSION="20.03 (LTS)"
ID="openEuler"
VERSION_ID="20.03"
PRETTY_NAME="openEuler 20.03 (LTS)"
ANSI_COLOR="0;31"
```

步骤 3 查看 CPU 信息

```
[root@openeuler ~]# lscpu
Architecture:          aarch64
CPU op-mode(s):      64-bit
Byte Order:           Little Endian
CPU(s):               2
On-line CPU(s) list: 0,1
Thread(s) per core:  1
Core(s) per socket:  2
Socket(s):            1
NUMA node(s):        1
Vendor ID:            HiSilicon
Model:                0
Model name:           Kunpeng-920
Stepping:             0x1
CPU max MHz:         2400.0000
CPU min MHz:         2400.0000
BogoMIPS:            200.00
L1d cache:           128 KiB
L1i cache:           128 KiB
L2 cache:             1 MiB
L3 cache:            32 MiB
NUMA node0 CPU(s):  0,1
.....
```

以下文件是在泰山 2280 物理服务器 openEuler 上运行该命令的结果：

```
# lscpu
Architecture:          aarch64
CPU op-mode(s):      64-bit
Byte Order:           Little Endian
CPU(s):               128
On-line CPU(s) list: 0-127
Thread(s) per core:  1
Core(s) per socket:  64
Socket(s):            2
NUMA node(s):        4
Vendor ID:            HiSilicon
Model:                0
Model name:           Kunpeng-920
Stepping:             0x1
CPU max MHz:         2600.0000
CPU min MHz:         200.0000
BogoMIPS:            200.00
L1d cache:           8 MiB
L1i cache:           8 MiB
L2 cache:            64 MiB
L3 cache:            256 MiB
```

```

NUMA node0 CPU(s):          0-31
NUMA node1 CPU(s):          32-63
NUMA node2 CPU(s):          64-95
NUMA node3 CPU(s):          96-127
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf:         Not affected
Vulnerability Mds:          Not affected
Vulnerability Meltdown:    Not affected
Vulnerability Spec store bypass: Vulnerable
Vulnerability Spectre v1:   Mitigation; __user pointer sanitization
Vulnerability Spectre v2:   Not affected
Vulnerability Tsx async abort: Not affected
Flags:                       fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp cpuid
                               asimdrdm jscvt fcma dcpop asimddp asimdfhm
    
```

这里以其 L1、L2 级缓存大小为例分析一下。我们知道鲲鹏 920 有 L1、L2、L3 三级 cache，其中 L1 的指令 cache (L1I) 和数据 cache (L1D) 大小都是 64KiB，L2 cache 不区分指令或数据，大小为 512KiB，L1 和 L2 两级 cache 由各个 CPU core 独享。鲲鹏 920 芯片共有 128 个 core，故：

L1d cache: 64Kib x 128 = 8192Kib 即 8MiB

L1i cache: 64Kib x 128 = 8192Kib 即 8MiB

L2 cache: 512Kib x 128 = 65536Kib 即 64MiB

扩展知识：在泰山 2280 V2 服务器有 2 个 CPU socket，4 个 NUMA 节点，总核数达到 128 个，L1d/L1i cache 大小分别是 8MiB，L2 cache 大小是 64MiB，主存有 1000G。

2.2.2 查看编程环境

下面以 C 程序为例，比较鲲鹏架构和 x86_64 架构的不同点。

先查看一下 gcc 版本：

```

[root@openeuler ~]# gcc --version
gcc (GCC) 7.3.0
.....
    
```

推荐使用 gcc7.3.0 及以上版本（不低于 4.8.5）。

以下文件包含了将要用到的参考代码：



可以用 scp 命令将其从本地拷贝至 ECS，如：

```
scp code.zip root@139.9.118.183:/home/
```


2.2.2.1 char 数据类型

步骤 1 按以下文件准备源代码

```
code/ch.c
```

步骤 2 在 KUNPENG 平台编译并执行

```
[root@openeuler ~]# gcc ch.c
[root@openeuler ~]# ./a.out
sizeof ch is 1, 1
```

```
char ch = ff, +255, positive
```

```
signed char ch = ff, -1, negative
unsigned char ch = ff, +255, positive
```

可见, (1) 由于计算机中的整数用补码表示的原因, -1 在内存中的数值是 0xff。(2) 在鲲鹏平台上 char 的默认数据类型与 unsigned char 同。

步骤 3 在 x86 平台上进行同样的实验

```
# gcc ch.c
[root@EulerOS test]# ./a.out
sizeof ch is 1, 1
```

```
char ch = ff, -1, negative
```

```
signed char ch = ff, -1, negative
unsigned char ch = ff, +255, positive
```

可见, 在 x86 平台上, char 的默认数据类型与 signed char 同。

其实, C 语言标准并没有规定 char 应该是 unsigned char 还是 signed char:

“The C standards do say that "char" may either be a "signed char" or "unsigned char" and it is up to the compilers implementation or the platform which is followed.”

所以, 在 C 语言中, char 类型是不是相当于 unsigned char 和 signed char **泛型**, 即仅仅代表一个 8bit 位数的集合, 至于这个集合的具体含义, 是 unsigned char 还是 signed char, 那是和 C 编译器与 CPU 指令集都有关的, 可以看出: **信息就是上下文**。

步骤 4 编译时指定 char 的类型

```
[root@openeuler ~]# gcc -fsigned-char test_char.c
```

重新运行看看结果。

步骤 5 在 x86 平台上指定 char 的类型

```
# gcc -funsigned-char test_char.c
```

查看运行结果。

在软件行业有一个灵丹妙药，即“**再加一层**”，gcc 参数-f 算不算是被加上的那一层？

2.2.2.2 ARMv8-64 架构的汇编指令和机器指令

由于 KUNPENG 处理器是基于 ARMV8-64 架构，此小节查看 C 程序编译后的汇编指令并与 X86_64 架构下的汇编指令比较。

步骤 1 按照以下文件准备源代码

```
code/abc.c
```

步骤 2 编译源文件

```
[root@openeuler ~]# gcc abc.c
```

步骤 3 查看程序执行结果是否正确

```
[root@openeuler ~]# ./a.out  
c = 3
```

步骤 4 生成汇编文件

```
[root@openeuler ~]# gcc -S abc.c  
[root@openeuler ~]# ls  
abc.c abc.s a.out
```

查看生成的.s 汇编语言文件，能否找出 C 源文件对应的汇编代码？

在 x86_64 的机器上进行同样的实验，查看两次的汇编代码各有什么特点？

步骤 5 用 objdump 命令观察机器码

```
[root@openeuler ~]# gcc -g abc.c  
[root@openeuler ~]# objdump -S ./a.out
```

请查看与汇编代码对应的机器码。

在 X86_64 机器的 Linux 操作系统上进行同样的实验，观察两者的机器码，你能说出精简指令集和复杂指令集各自的特点吗？

2.2.2.3 C 代码中的汇编语句

步骤 1 按以下文件内容准备源代码

```
code/add.c
```

步骤 2 编译并执行

```
[root@openeuler ~]# gcc -o add add.c  
[root@openeuler ~]# ./add 2 3  
2 + 3 = 5
```

试着将代码中 32 位的寄存器 Wn 换成 64 位的寄存器 Xn 再重新编译运行一遍。

2.2.2.4 纯汇编程序

步骤 1 按以下内容编辑源代码

```
code/hello.s
```

步骤 2 编译、链接

```
[root@openeuler ~]# as -o hello.o hello.s  
[root@openeuler ~]# ld -o hello hello.o  
ld: warning: cannot find entry symbol _start; defaulting to 00000000004000b0
```

此处的警告信息请忽略。

步骤 3 执行程序

```
[root@openeuler ~]# ./hello  
Hello openEuler!
```

分析各个汇编语句的作用，写出 x86 平台同样功能的汇编代码。

3 iSulad 实验

3.1 实验介绍

3.1.1 关于本实验

本实验介绍如何在弹性云服务器上安装 iSulad 及其有关容器生命周期的基本操作。同时，尝试使用 iSula 容器镜像构建工具 isula-build 构建自己的容器镜像。

3.1.2 实验目的

- 学会安装 iSula 容器引擎 isulad；
- 学会如何建立、运行、停止以及删除一个容器；
- 学会用 isula-build 构建自己的容器镜像并运行之。

3.1.3 参考资料

https://openeuler.org/zh/docs/20.03_LTS/docs/Container/container.html

<https://gitee.com/openeuler/iSulad>

<https://gitee.com/openeuler/isula-build>

3.2 安装 isulad

3.2.1 安装

步骤 1 用 yum 命令安装

```
[root@openeuler ~]# yum install -y iSulad
```

3.2.2 启动并查看版本

步骤 1 用 systemctl start 命令启动

```
[root@openeuler ~]# systemctl start isulad
```

步骤 2 查看状态

```
[root@openeuler ~]# systemctl status isulad
● isulad.service - iSulad Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/isulad.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2020-09-14 15:53:43 CST; 4h 35min ago
 Main PID: 1248 (isulad)
    Tasks: 25
   Memory: 88.3M
    CGroup: /system.slice/isulad.service
            └─1248 /usr/bin/isulad
               └─1315 isulad-img .....
```

按 'q' 或 'Q' 键退出信息显示。

步骤 3 查看版本

```
[root@openeuler ~]# isula version
Client:
  Version:      2.0.0
  Git commit:   5bf7c66ad4f7095156e87ca455da016f57c3e60f
  Built:        2020-03-23T21:35:55.821352180+00:00

Server:
  Version:      2.0.0
  Git commit:   5bf7c66ad4f7095156e87ca455da016f57c3e60f
  Built:        2020-03-23T21:35:55.821352180+00:00

OCI config:
  Version:      1.0.1
  Default file: /etc/default/isulad/config.json
```

步骤 4 查看帮助

```
[root@openeuler ~]# isula --help
```

3.3 容器与镜像管理

3.3.1 准备工作

步骤 1 安装 JSON 格式数据处理工具

```
[root@openeuler ~]# yum install -y jq
```

步骤 2 修改 isulad 的配置文件

```
[root@openeuler ~]# mkdir iSula && cd iSula
```

```
[root@openeuler iSula]# cp /etc/isulad/daemon.json /etc/isulad/daemon.json.origin
[root@openeuler iSula]# vi /etc/isulad/daemon.json    #在此处修改
[root@openeuler iSula]# cat /etc/isulad/daemon.json
{
  "group": "isulad",
  "default-runtime": "lcr",
  "graph": "/var/lib/isulad",
  "state": "/var/run/isulad",
  "engine": "lcr",
  "log-level": "ERROR",
  "pidfile": "/var/run/isulad.pid",
  "log-opts": {
    "log-file-mode": "0600",
    "log-path": "/var/lib/isulad",
    "max-file": "1",
    "max-size": "30KB"
  },
  "log-driver": "stdout",
  "hook-spec": "/etc/default/isulad/hooks/default.json",
  "start-timeout": "2m",
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ],
  "registry-mirrors": [
    "hub.oepkgs.net"
  ],
  "insecure-registries": [
  ],
  "pod-sandbox-image": "",
  "image-opt-timeout": "5m",
  "image-server-sock-addr": "unix:///var/run/isulad/isula_image.sock",
  "native.umask": "secure",
  "network-plugin": "",
  "cni-bin-dir": "",
  "cni-conf-dir": "",
  "image-layer-check": false,
  "use-decrypted-key": true,
  "insecure-skip-verify-enforce": false
}
```

在上述文件中，我们设"registry-mirrors"的值为"hub.oepkgs.net"。（hub.oepkgs.net 为 openEuler 社区与中科院软件所共建的、开源免费的容器镜像仓库）

步骤 3 检查配置文件合法性

```
[root@openeuler iSula]# cat /etc/isulad/daemon.json | jq
```

如果配置文件的内容能正确显示出来，即表示其格式合法。

步骤 4 重启 isulad 服务

```
[root@openeuler iSula]# systemctl restart isulad
```

3.3.2 运行容器 busybox

步骤 1 用 isula run 命令直接运行

```
[root@openeuler iSula]# isula run busybox echo "hello world"
Unable to find image 'busybox' locally
Image "busybox" pulling
Image "219ee5171f8006d1462fa76c12b9b01ab672dbc8b283f186841bf2c3ca8e3c93" pulled
hello world
```

由于这是第一次运行，所以会拉取 busybox 的镜像，然后会运行它的一个实例，并且调用 echo 命令打印 hello world 字符串。（按：由于 oepkgs 目前只有 x86 的 busybox 镜像，在鲲鹏平台上运行这一步会提示错误，请忽略这一步。）

查看其镜像：

```
[root@openeuler iSula]# isula images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
busybox	latest	219ee5171f80	2020-12-04 06:19:53	1.385 MB

以上输出表明这一阶段的安装成功了，下面继续验证其他的一些命令。

3.3.3 运行容器 openeuler:20.09

步骤 1 创建容器并启动之

创建容器：

```
[root@openeuler iSula]# isula create -it openeuler/openeuler:20.09
Unable to find image 'openeuler/openeuler:20.09' locally
Image "openeuler/openeuler:20.09" pulling
Image "8c788f4bfb7290e434b2384340a5f9811db6ed302f9247c5fc095d6ec4fc8f32" pulled
e91e5359be653f534312bc2b4703dcc6c4ca0436ac7819e09e1ff0e75ee1d733
```

由于没有运行容器，所以利用 isula ps 命令无法找到刚刚创建的容器：

```
[root@openeuler iSula]# isula ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

可以使用 isula pa -a 命令可以找到刚刚创建的容器：

```
[root@ecs-cdf3 ~]# isula ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e91e5359be65	openeuler/openeuler:20.09	"/bin/bash"		8 seconds ago		Created

启动容器：

```
[root@openeuler iSula]# isula start e91e5359be65
```

由于已经运行容器，可以利用 `isula ps` 命令查找运行中的容器

```
[root@openeuler iSula]# isula ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
e91e5359be65 openeuler/openeuler:20.09 "/bin/bash" 30 seconds ago Up 4 seconds
e91e5359be653f534312bc2b4703dcc6c4ca0436ac7819e09e1ff0e75ee1d733
```

执行命令输出容器系统版本信息：

```
[root@ecs-cdf3 ~]# isula exec e91e5359be65 cat /etc/os-release
NAME="openEuler"
VERSION="20.09"
ID="openEuler"
VERSION_ID="20.09"
PRETTY_NAME="openEuler 20.09"
ANSI_COLOR="0;31"
```

可以看到我们在 `openeuler 20.03` 系统上成功运行 `openeuler 20.09` 系统容器。

步骤 2 直接运行

```
[root@openeuler iSula]# isula run openeuler/openeuler:20.09 cat /etc/os-release
NAME="openEuler"
VERSION="20.09"
ID="openEuler"
VERSION_ID="20.09"
PRETTY_NAME="openEuler 20.09"
ANSI_COLOR="0;31"
```

同样我们可以看到和步骤一相同的输出。

步骤 3 交互式运行

```
[root@openeuler iSula]# isula run -it openeuler/openeuler:20.09
Welcome to 4.19.90-2003.4.0.0036.oe1.x86_64
```

```
System information as of time: Wed 27 Jan 2021 11:13:26 AM CST
```

```
System load: 0.02
Processes: 5
Memory used: 6.9%
Swap used: 0.0%
Usage On: 9%
Users online: 0
```



```
[root@localhost /]# ls
bin dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
[root@localhost /]# uname -a
Linux localhost 4.19.90-2003.4.0.0036.oe1.aarch64 #1 SMP Mon Mar 23 19:06:43 UTC 2020 aarch64 GNU/Linux
[root@localhost /]# ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
[root@localhost /]# cat /etc/os-release
NAME="openEuler"
VERSION="20.09"
ID="openEuler"
VERSION_ID="20.09"
PRETTY_NAME="openEuler 20.09"
ANSI_COLOR="0;31"
[root@localhost /]# exit
exit
```

以上浅色字表示在容器中运行。

步骤 4 暂停/恢复一个容器

```
[root@openeuler iSula]# isula pause e91e5359be65
e91e5359be65
[root@openeuler iSula]# isula unpause e91e5359be65
e91e5359be65
```

步骤 5 先停止，再删除一个容器

```
[root@openeuler iSula]# isula stop e91e5359be65
e91e5359be65
[root@openeuler iSula]# isula rm e91e5359be65
e91e5359be653f534312bc2b4703dcc6c4ca0436ac7819e09e1ff0e75ee1d733
```

步骤 6 查看正在运行着的容器

```
[root@openeuler iSula]# isula ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
6c1d81467d33 openeuler/openeuler:20.09 "/bin/bash" 37 minutes ago Up 37 minutes
6c1d81467d3367a90dd6e388a16c80411d4ba76316d86b6f56463699306e1394
```

步骤 7 强制删除运行中的容器

```
[root@openeuler iSula]# isula rm -f 6c1d81467d33
6c1d81467d3367a90dd6e388a16c80411d4ba76316d86b6f56463699306e1394
```

步骤 8 查看所有容器（包含运行中的容器）

```
[root@openeuler iSula]# isula ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
bb85ce20525d openeuler/openeuler:20.09 "/bin/bash" 57 seconds ago Up 57 seconds
bb85ce20525db387c81c94e7a0865fccbf868a35ca4ba0dd077c0bba4a9c379a
3139ce089c56 openeuler/openeuler:20.09 "/bin/bash" 4 seconds ago Created
3139ce089c56567ba877f11cba3dbb9dbdc68cf4cdc4ded701451a23a4cc38e
```

步骤 9 先将关联到镜像的容器销毁

```
[root@openeuler iSula]# isula rm -f bb85ce20525d 3139ce089c56
bb85ce20525db387c81c94e7a0865fccbf868a35ca4ba0dd077c0bba4a9c379a
3139ce089c56567ba877f11cba3dbb9dbdc68cf4cdc4ded701451a23a4cc38e
```

步骤 10 然后删除镜像

```
[root@openeuler iSula]# isula rmi openeuler/openeuler:20.09
Image "openeuler/openeuler:20.09" removed
```

如果这一步骤失败，请先将所有容器删除后再运行此命令。

3.4 使用 isula-build 构建容器镜像

目前为止，我们使用的容器镜像都是从 hub.oepkgs.net 下载已经构建好的容器镜像，下面我们尝试使用 isula 提供的容器镜像构建工具 isula-build，构建自己的容器镜像并运行。

3.4.1 安装 isula-build

步骤 1 检查 yum 源

```
[root@openeuler iSula]# tail /etc/yum.repos.d/openEuler_aarch64.repo
[update]
name=update
baseurl=http://repo.openeuler.org/openEuler-20.03-LTS/update/$basearch/
enabled=0
gpgcheck=1
gpgkey=http://repo.openeuler.org/openEuler-20.03-LTS/OS/$basearch/RPM-GPG-KEY-openeuler
```

将文件中[update]节的 enable=0 改为 enable=1 并保存。

```
[root@openeuler ~]# yum repolist | grep update
update update
```

从输出中可以看到 update 赫然纸上。

步骤 2 用 yum 命令安装 isula-build

```
[root@openeuler iSula]# yum install -y isula-build
```

步骤 3 安装 docker-runc

```
[root@openeuler iSula]# yum install -y docker-runc
```

步骤 4 用 systemctl start 命令启动

```
[root@openeuler iSula]# systemctl start isula-build
```

步骤 5 查看版本

```
[root@openeuler iSula]# isula-build version
Client:
  Version:      0.9.2
  Go Version:   go1.13.3
  Git Commit:   a96cf18
  Built:        Thu Aug 25 01:08:45 2020
  OS/Arch:     linux/arm64

Server:
  Version:      0.9.2
  Go Version:   go1.13.3
  Git Commit:   a96cf18
  Built:        Thu Aug 25 01:08:45 2020
  OS/Arch:     linux/arm64
```

步骤 6 修改配置

修改/etc/isula-build/registries.toml，将 hub.oepkgs.net 加入到 isula-build 可搜索的镜像仓库列表里：

```
vi /etc/isula-build/registries.toml
.....
[registries.search]
registries = ["hub.oepkgs.net"]
.....
```

当然，如果有不同的镜像仓库也可以配置不同的镜像仓库。

重启 isula-build 服务：

```
[root@openeuler iSula]# systemctl restart isula-build
```

查看配置：

```
[root@openeuler iSula]# isula-build info -H
General:
  MemTotal:    3.6 GB
  MemFree:     1.33 GB
  SwapTotal:   0 B
  SwapFree:    0 B
```

```
OCI Runtime: runc
DataRoot: /var/lib/isula-build/
RunRoot: /var/run/isula-build/
Builders: 0
Goroutines: 11
Store:
Storage Driver: overlay
Backing Filesystem: extfs
Registry:
Search Registries:
hub.oepkgs.net
Insecure Registries:
```

在 Search Registries 配置项里看到“hub.oepkgs.net”赫然纸上。

至此， isula-build 安装完成。

3.4.2 构建容器镜像并导入到 isulad

步骤 1 创建 Dockerfile

创建 “Dockerfile” 的文件：

```
[root@openeuler iSula]# mkdir -p /home/test/ && cd /home/test/
[root@openeuler test]# vi Dockerfile #在此编辑文件内容
[root@openeuler iSula]# cat Dockerfile
FROM openeuler/openeuler:20.09
COPY hello.sh /usr/bin/
CMD ["sh", "-c", "/usr/bin/hello.sh"]
```

这里我们是在在/home/test 目录下创建 Dockerfile 文件的。

编辑 Dockerfile 中出现的 hello.sh 脚本，它将被加到原有的 openeuler:20.09 镜像中以构建出我们自己的镜像：

```
[root@openeuler test]# vi hello.sh #在此编辑文件
[root@openeuler test]# cat hello.sh
#!/bin/sh
echo "hello, isula-build!"
```

修改文件属性：

```
[root@openeuler test]# chmod +x hello.sh
```

验证：

```
[root@openeuler test]# ls -l
total 8
-rw----- 1 root root 91 Jan 27 15:09 Dockerfile
-rwx----- 1 root root 37 Jan 27 15:10 hello.sh
```

以上即是我们刚刚创建的 2 个文件。

步骤 2 构建容器镜像

用 isula-build 构建我们自己的容器镜像并导入到 isulad，镜像命名为 hello-isula-build:v0.1:

```
[root@openeuler test]# isula-build ctr-img build -f ./Dockerfile -o isulad:hello-isula-build:v0.1
STEP 1: FROM openeuler/openeuler:20.09
STEP 2: COPY hello.sh /usr/bin/
STEP 3: CMD ["sh", "-c", "/usr/bin/hello.sh"]
...
Build success with image id: 093721586033ee24966cea91d8276b1cd1cf07240fd770a702e6d2d77577cb7f
```

步骤 3 查询构建出来的容器镜像

```
[root@openeuler test]# isula-build ctr-img images
[root@openeuler test]# isula-build ctr-img images
-----
REPOSITORY              TAG          IMAGE ID          CREATED           SIZE
-----
hello-isula-build       v0.1        093721586033    2021-01-27 07:09:23    550 MB

hub.oepkgs.net/openeuler/openeuler  20.09      8c788f4bfb72    2020-09-28 04:27:37    550 MB
-----

[root@openeuler test]# isula images
REPOSITORY              TAG          IMAGE ID          CREATED           SIZE
hello-isula-build       v0.1        093721586033    2020-01-27 15:09:23    524.466 MB
```

可以看到， isula-build ctr-img images 可以查询到构建出来的容器镜像，同时， isula images 命令也可以查询到从 isula-build 导入到 isulad 的容器镜像

步骤 4 启动容器

我们使用自己构建出来的容器镜像来启动容器，按照预期，容器进程会打印出 “hello, isula-build!” :

```
[root@openeuler test]# isula run hello-isula-build:v0.1
hello, isula-build!
```

3.4.3 扩展实验： isula-build 的其他镜像导出方式

在前述步骤中构建了自己的容器镜像 hello-isula-build:v0.1，并将容器镜像导出到 isulad 启动容器。 isula-build 除了能将容器镜像导出到 isulad 之外，还可以：

- 导出到远端仓库
- 导出到 docker daemon
- 导出到本地压缩包
- 导出到 isula-build 的本地存储

可以通过 `isula-build` 的命令行帮助，查看这些导出方式：

```
[root@openeuler test]# isula-build ctr-img build --help
Build container images
Usage:
isula-build ctr-img build [FLAGS] PATH
Examples:
isula-build ctr-img build -f Dockerfile .
isula-build ctr-img build -f Dockerfile -o docker-archive:name.tar:image:tag .
isula-build ctr-img build -f Dockerfile -o docker-daemon:image:tag .
isula-build ctr-img build -f Dockerfile -o docker://registry.example.com/repository:tag .
isula-build ctr-img build -f Dockerfile -o isulad:image:tag .
isula-build ctr-img build -f Dockerfile --build-static='build-time=2020-06-30 15:05:05' .
```

可以按照命令行帮助信息中给出的详细示例进行练习。

4 智能优化引擎 A-Tune 实验

4.1 实验介绍

4.1.1 关于本实验

本实验介绍如何在一个虚拟机(或 ECS)上安装并运行 A-Tune 智能优化引擎，以及它的一些基本操作。

4.1.2 实验目的

- 学会在 ECS 上安装 A-Tune 引擎 atuned;
- 了解 A-Tune 引擎的基本操作。

4.1.3 参考资料

<https://gitee.com/openeuler/A-Tune>

https://openeuler.org/zh/docs/20.03_LTS/docs/A-Tune/A-Tune.html

4.2 安装和启动 A-Tune

4.2.1 安装

步骤 1 打开源码自带的 README 文件

用浏览器打开如下链接并参考其源码安装方式:

<https://gitee.com/openeuler/A-Tune/blob/master/README-zh.md>

步骤 2 安装依赖包

安装依赖系统软件包:

```
[root@openeuler ~]# yum install -y golang-bin python3 perf sysstat hwloc-gui
```

安装 python 依赖包:

```
[root@openeuler ~]# yum install -y python3-dict2xml python3-flask-restful python3-pandas python3-scikit-optimize python3-xgboost python3-pyyaml
```

步骤 3 下载 A-Tune 的源代码

```
[root@openeuler ~]# git clone https://gitee.com/openeuler/A-Tune.git
```

该命令会在当前目录创建一个 A-Tune 的目录并将 A-Tune 源代码的 master 分支下载到该目录。

步骤 4 编译及安装

编译:

```
[root@openeuler ~]# cd A-Tune
[root@openeuler A-Tune]# make models
[root@openeuler A-Tune]# make
```

安装:

```
[root@openeuler A-Tune]# make collector-install
[root@openeuler A-Tune]# make install
```

注意: 编译过程会花一定时间, 请耐心等待。

4.2.2 启动

步骤 1 修改 A-Tune 的配置文件

```
[root@openeuler A-Tune]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether fa:16:3e:4a:a6:5c brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.168/24 brd 192.168.1.255 scope global dynamic noprefixroute eth0
        valid_lft 31531667sec preferred_lft 31531667sec
    inet6 fe80::f816:3eff:fe4a:a65c/64 scope link
        valid_lft forever preferred_lft forever
```

从以上命令看到系统的网卡名称是 eth0。

```
[root@openeuler A-Tune]# fdisk -l | grep dev
Disk /dev/vda: 40 GiB, 42949672960 bytes, 83886080 sectors
/dev/vda1    2048 2099199 2097152  1G EFI System
/dev/vda2   2099200 83884031 81784832  39G Linux filesystem
```

从以上命令看到系统磁盘为 vda。

下面修改/etc/atuned/atuned.cnf 中的 network 配置和 disk 配置。

```
[root@openeuler A-Tune]# vim /etc/atuned/atuned.cnf    #在此命令中修改配置
[root@openeuler A-Tune]# cat /etc/atuned/atuned.cnf
.....
[system]
# the disk to be analysis
disk = vda

# the network to be analysis
network = eth0
.....
```

以上命令根据实际情况将 disk 置为 vda，将 network 置为 eth0。

步骤 2 启动服务

加载并启动 atuned 和 atune-engine 服务：

```
[root@openeuler A-Tune]# systemctl daemon-reload
[root@openeuler A-Tune]# systemctl start atuned
[root@openeuler A-Tune]# systemctl start atune-engine
```

查看 atuned 和 atune-engine 服务状态：

```
[root@openeuler A-Tune]# systemctl status atuned
● atuned.service - A-Tune Daemon
   Loaded: loaded (/usr/lib/systemd/system/atuned.service; disabled; vendor preset: disabled)
   Active: active (running) since Thu 2020-07-30 21:44:43 CST; 2min 33s ago
   Main PID: 10275 (atuned)
     Tasks: 13
    Memory: 88.6M
    CGroup: /system.slice/atuned.service
           └─10275 /usr/bin/atuned
             └─10282 python3 /usr/libexec/atuned/analysis/app.py /etc/atuned/atuned.cnf

[root@aarch64 A-Tune]# systemctl status atune-engine
● atune-engine.service - A-Tune AI service
   Loaded: loaded (/usr/lib/systemd/system/atune-engine.service; disabled; vendor preset: disabled)
   Active: active (running) since Sat 2021-01-16 22:04:33 CST; 37s ago
   Main PID: 6025 (python3)
     Tasks: 2
    Memory: 97.7M
    CGroup: /system.slice/atune-engine.service
           └─6025 /usr/bin/python3 /usr/libexec/atuned/analysis/app_engine.py /etc/atuned/engine.cnf
             └─6041 /usr/bin/python3 -c from multiprocessing.semaphore_tracker import main;main(3)
```

注意：有时需要按“q”或“Q”键从查看信息的状态中退出。

4.3 运行 atune-adm 命令

4.3.1 查看版本

步骤 1 查看 atune-adm 版本信息

```
[root@openeuler A-Tune]# atune-adm --version
atune-adm version 0.3(ea0c705)
```

4.3.2 查询负载类型

步骤 1 查询 profile

```
[root@openeuler A-Tune]# atune-adm list
```

查询系统当前支持的 profile，以及 profile 所处的状态。

4.3.3 分析负载类型并自优化

步骤 1 负载类型识别

```
[root@aarch64 A-Tune]# atune-adm analysis --characterization
```

1. Analysis system runtime information: CPU Memory IO and Network...

```
0.0 0.0 0.05 0.0 0.1 0.0 0.0 0.0 0.15 0.0 0.0 0.4 0.0 0.0 0.0 60.0 0.0 10.0 0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
1.0 100.0 1000.0 100.0 100.0 1.0 1.0 0.0 0.0 4.0 1111.0 236.0 0.0 0.0 1.0 0.2 233.0 1.0 188.0 0.1 0.04 0.0
```

.....

2. Current System Workload Characterization is default

采集系统的实时统计数据并进行负载类型识别。

注意：识别出的当前负载类型为 default，由于系统当前并没有什么运行特定任务，所以这是符合实际情况的。

步骤 2 负载类型识别及自动优化

```
[root@aarch64 A-Tune]# atune-adm analysis
```

1. Analysis system runtime information: CPU Memory IO and Network...

```
0.1 0.0 0.1 0.0 0.1 0.0 0.0 0.0 0.3 0.0 0.2 0.4 0.01 0.01 0.0 88.24 68.0 34.0 1.0 1.5 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 1.0 100.0 1000.0 100.0 100.0 1.0 1.0 0.0 0.0 14.0 1114.0 242.0 0.0 0.0 2.0 0.0 241.4 1.0 185.0 0.07 0.03 0.0
```

.....

2. Current System Workload Characterization is default

3. Build the best resource model...

4. Match profile: default-default

```
5. begin to set static profile
[SUGGEST] Bios           please change the BIOS configuration NUMA to Enable
[SUGGEST] Bootloader     Need reboot to make the config change of grub2
effect.
[SUCCESS] memory        memory num is 1, the memory slot is full
[SUGGEST] Kernel         please change the kernel configuration
CONFIG_NUMA_AWARE_SPINLOCKS to y
Completed optimization, please restart application!
```

以上命令使用默认模型进行应用识别，并进行自动优化。

4.3.4 系统信息查询

步骤 1 检查系统当前信息

```
[root@openeuler A-Tune]# atune-adm check
```

检查系统当前的 CPU、BIOS、OS、网卡等信息。

步骤 2 查看 A-Tune 的其他命令

A-Tune 的其他命令使用详见 atune-adm help 信息：

```
[root@aarch64 A-Tune]# atune-adm help
```

4.3.5 离线业务自调优

本实验通过 A-Tune 的离线自调优功能选出最优的压缩算法及其配置。请参考 A-Tune/examples/tuning/compress 目录中的 README 文件。

步骤 1 下载压缩文件样本

```
[root@openeuler A-Tune]# cd ./examples/tuning/compress
```

```
[root@openeuler compress]# wget http://cs.fit.edu/~mmahoney/compression/enwik8.zip
```

此处下载一个名为 enwik8.zip 的压缩文件到指定目录（即当前目录 A-Tune/examples/tuning/compress），该文件作为一个样本压缩了大量的文本内容：

```
[root@aarch64 compress]# ls *.zip
```

```
enwik8.zip
```

步骤 2 调优前的参数配置

```
[root@openeuler compress]# sh prepare.sh enwik8.zip
```

此处运行一个 prepare.sh 脚本，该脚本解压缩 enwik8.zip 文件并进行一些参数设置，例如 compress_client.yaml 文件中 time 的权重为 20，compress_ratio 的权重为 80，表明本次优化目标偏重压缩率。

注意：这个脚本简化了繁杂的人工配置，具体的操作步骤请打开脚本的源代码进行学习。我们同样需要细致研究另一个脚本 compress.py，先备份该文件：

```
[root@openeuler compress]# cp compress.py{,.bak}
```

步骤 3 进行 tuning 以找到最优配置

```
[root@openeuler compress]# atune-adm tuning --project compress --detail compress_client.yaml
Start to benchmark baseline...
  1.Loading its corresponding tuning project: compress
  2.Start to tuning the system.....
Current Tuning Progress.....(1/20)
.....
The final optimization result is: compressLevel=1,compressMethod=gzip
The final evaluation value is: time=1.39,compress_ratio=2.36

Baseline Performance is: (time:4.62,compress_ratio:2.74)

Tuning Finished
```

此命令开启针对本压缩应用的自动化调优，调优结果反映在了参数 `compressLevel` 和 `compressMethod` 的设置上，对比调优前后的 `compress.py` 文件：

```
# diff compress.py compress.py.bak
12,13c12,13
< compressLevel=1
< compressMethod="gzip"
---
> compressLevel=6
> compressMethod="zlib"
```

注意：以上对比结果可能会因为各自具体的运行环境而有所不同。

步骤 4 还原系统配置

```
[root@openeuler compress]# atune-adm tuning --restore --project compress
```

通过此命令恢复到了本次 tuning 调优前的配置，即 `compressLevel` 和 `compressMethod` 的设置。

5 内核编程实验

5.1 实验介绍

5.1.1 关于本实验

本实验在鲲鹏云 ECS 上编译、安装 openEuler 操作系统内核并编写一个简单的内核模块以验证安装是否成功。

5.1.2 实验目的

- 熟悉 openEuler 内核的编译与安装；
- 了解内核模块编程的过程。

5.2 内核编程实验

5.2.1 内核的编译与安装

步骤 1 安装工具，构建开发环境：

```
[root@openEuler ~]# yum group install -y "Development Tools"
[root@openEuler ~]# yum install -y bc
[root@openEuler ~]# yum install -y openssl-devel
```

步骤 2 备份 boot 目录以防后续步骤更新内核失败

```
[root@openEuler ~]# tar czvf original_boot.tgz /boot/
保存当前内核版本信息
[root@openEuler ~]# uname -r > uname_r.log
```

步骤 1 获取内核源代码并解压

```
[root@openEuler ~]# wget https://gitee.com/openeuler/kernel/repository/archive/kernel-4.19.zip
[root@openEuler ~]# unzip kernel-4.19.zip
```

步骤 2 编译内核

```
[root@openEuler ~]# cd kernel
```

在这里，我们按源代码文件 kernel/arch/arm64/configs/openeuler_defconfig 的配置配置内核。

```
[root@openEuler kernel]# make help | grep Image
* Image.gz      - Compressed kernel image (arch/arm64/boot/Image.gz)
  Image         - Uncompressed kernel image (arch/arm64/boot/Image)
```

首先，我们查看了可编译的 Image。

```
[root@openEuler kernel]# make openeuler_defconfig
[root@openEuler kernel]# make -j4 Image modules dtbs
```

这一步是编译内核的 Image、modules 和 dtbs。

步骤 3 安装内核

```
[root@openEuler kernel]# make modules_install
.....
INSTALL sound/soundcore.ko
DEPMOD 4.19.154
[root@openEuler kernel]# make install
/bin/sh ./arch/arm64/boot/install.sh 4.19.154 \
arch/arm64/boot/Image System.map "/boot"
dracut-install: Failed to find module 'xen-blkfront'
dracut: FAILED: /usr/lib/dracut/dracut-install -D /var/tmp/dracut.tlIdPu/initramfs --kernel-dir
/lib/modules/4.19.154/ -m virtio_gpu xen-blkfront xen-netfront virtio_blk virtio_scsi virtio_net virtio_pci virtio_ring
virtio
```

注意：忽略最后一步“make install”时出现的错误。

步骤 4 以 VNC 登录 ECS

<input type="checkbox"/>	名称/ID	监控	可用区	状态	规格/镜像
<input type="checkbox"/>	openEuler 5e4b28a...				openEuler 20.03 64bit with ARM

在控制台“弹性云服务器 ECS”的页面中点击刚刚创建的虚拟机“openEuler”的名字超链接，在新打开的页面中点击“远程登录”按钮：



然后以控制台提供的 VNC 方式登录：

登录Linux弹性云服务器

使用CloudShell登录 **New!**

优势：操作更流畅，命令支持复制粘贴
绑定弹性公网IP的云服务器推荐使用此登录方式。
请确保安全组已放通CloudShell连接实例使用的端口（默认使用22端口）

CloudShell登录

其他方式

1、使用控制台提供的VNC方式登录

立即登录

与以 ssh 登录一样，以 root 身份登录：

```
Authorized users only. All activities may be monitored and reported.
openEuler login: [ 429.483128] systemd-rc-local-generator[2892]: /etc/rc.d/rc.local is not marked executable, skipping.

openEuler login: root
Password:
Last login: Sun Nov 15 14:51:43 from 119.3.119.18

      Welcome to Huawei Cloud Service

Welcome to 4.19.90-2003.4.0.0036.oe1.aarch64

System information as of time: Sun Nov 15 15:51:58 CST 2020

System load:  0.00
Processes:    122
Memory used:  5.2%
Swap used:    0.0%
Usage on:     38%
IP address:   192.168.1.5
Users online: 2

[root@openEuler ~]# _
```

大部分的时间，我们仅将此作为一个监视器使用。

步骤 5 重启系统

在 ssh 终端重启操作系统：

```
[root@openEuler kernel]# reboot
```

步骤 6 登录并验证

在 VNC 窗口中选择以新编译出来的内核启动系统：

```

openEuler (4.19.154) 20.03 (LTS)
openEuler (4.19.90-2003.4.0.0036.oe1.aarch64) 20.03 (LTS)
openEuler (0-rescue-95148c976dae4cd0bfb15a0242465b8a) 20.03 (LTS)
System setup

Use the ▲ and ▼ keys to change the selection.
Press 'e' to edit the selected item, or 'c' for a command prompt.
    
```

在这里新编译出来的内核版本为 4.19.154。您的子版本号可能与此不一样。

步骤 7 登录系统并查看版本

请以 VNC 和/或 ssh 终端登录系统，并在其中查看内核版本：

```
[root@openEuler ~]# uname -r
4.19.154
```

可以看出内核版本已更新。

5.2.2 Hello, world!

步骤 1 编写一个仅打印 “Hello, world!” 内核模块，包括 .c 源文件和 Makefile 文件。在这里我们的示例源文件存放在 tasks_k/1/task3 目录下。

```
[root@openEuler ~]# cd tasks_k/1/task3
[root@openEuler task1]# ls
helloworld.c  Makefile
```



实验中的源文件可以参考以上压缩包中内容（您可以用 scp 命令将其上传到 ECS）。

步骤 2 编译源文件

```
[root@openEuler task3]# make
make -C /root/kernel M=/root/tasks_k/1/task3 modules
make[1]: Entering directory '/root/kernel'
CC [M] /root/tasks_k/1/task3/helloworld.o
```



```
Building modules, stage 2.  
MODPOST 1 modules  
CC      /root/tasks_k/1/task3/helloworld.mod.o  
LD [M]  /root/tasks_k/1/task3/helloworld.ko  
make[1]: Leaving directory '/root/kernel'
```

步骤 3 加载编译完成的内核模块，并查看加载结果。

```
[root@openEuler task3]# insmod helloworld.ko  
[root@openEuler task3]# lsmod | grep helloworld  
helloworld          262144  0
```

步骤 4 卸载内核模块，并查看结果。

```
[root@openEuler task3]# rmmod helloworld  
[root@openEuler task3]# dmesg | tail -n5  
[ 708.247970] helloworld: loading out-of-tree module taints kernel.  
[ 708.248513] helloworld: module verification failed: signature and/or required key missing - tainting kernel  
[ 708.249859] hello_init  
[ 708.250043] Hello, world!  
[ 747.518247] hello_exit
```

您在 VNC 窗口中也会看到同样的结果。

(在这里，请忽略掉最开始安装模块时出现的两行错误提示信息。)

5.2.3 使用 tasklet 打印 Hello, world!

步骤 1 编写一个信号捕捉程序，捕捉终端按键信号。示例文件在 tasks_k/4/task1 目录下。

```
[root@openEuler ~]# cd tasks_k/4/task1  
[root@openEuler task1]# ls  
Makefile  tasklet_intertupt.c
```

步骤 2 编译源文件

```
[root@openEuler task1]# make
```

步骤 3 加载编译完成的内核模块，并查看加载结果。

```
[root@openEuler task1]# insmod tasklet_intertupt.ko  
[root@openEuler task1]# dmesg | tail -n2  
[86929.981168] Start tasklet module...  
[86929.981488] Hello World! tasklet is working...
```

步骤 4 卸载内核模块，并查看结果。

```
[root@openEuler task1]# rmmod tasklet_intertupt  
[root@openEuler task1]# dmesg | tail -n3  
[86929.981168] Start tasklet module...  
[86929.981488] Hello World! tasklet is working...
```

```
[86973.915367] Exit tasklet module...  
[root@openEuler task1]#
```

步骤 5 在虚拟机和 VNC 窗口中退出登录

```
[root@openEuler ~]# exit
```

注意：一般情况下不要 shutdown 虚拟机，若 shutdown 了虚拟机，需要联系实验管理员重启该虚拟机。

步骤 6 关闭 VNC 客户端页面

步骤 7 在 ssh 终端退出登录

```
[root@openEuler ~]# exit
```

6 资源清理

6.1 资源清理

6.1.1 ECS 关机

步骤 1 实验完成后，回到 ECS 控制台，勾选 openEuler 虚拟机，先进行关机。

弹性云服务器 ?

诚邀您参加弹性云服务器使用体验调研，您宝贵的意见和建议是我们持续提升产品体验

开机 关机 重置密码 更多 ▾

默认按照名称搜索

<input checked="" type="checkbox"/>	名称/ID	监控	可用区	状态
<input checked="" type="checkbox"/>	openEuler 00ad36b3-1920-47a...		可用区1	运行中

在弹出的对话框中点击“是”按钮：

关机

确定要对以下云服务器进行关机操作吗？

1、按需/竞价计费（竞价计费模式）实例关机后，基础资源（vCPU、内存、镜像）不再计费，绑定的云硬盘（包括系统盘、数据盘）、弹性公网IP、带宽等资源按各自产品的计费方法（“包年/包月”或“按需计费”）进行收费。云服务器再次开机时，可能由于基础资源不足无法正常开机。请耐心等待，稍后重试。

2、包含本地盘（如磁盘增强型、GPU加速型等）和包含FPGA卡的按需/竞价计费实例，以及竞价计费的共享模式实例，关机后仍然计费。如需停止计费，请删除实例。

名称	状态	备注
openEuler	运行中	--

强制关机

强制关机会导致云服务器中未保存的数据丢失，请谨慎操作。

是 否

6.1.2 删除资源

步骤 1 待关机完成后点击“更多” → “删除”



在弹出的对话框中勾选“释放云服务器绑定的弹性公网 IP 地址”和“删除云服务器挂载的数据盘”，然后点击“是”，删除 ECS。



步骤 2 您可以在控制台点击“更多 | 资源 | 我的资源”菜单项，检查资源是否全部删除



注意：(1) 虚拟私有云 VPC 和安全组可以不删除，以留下次使用。(2) 若在除“华北-北京四”之外区域（如“亚太-香港”）购买了 ECS 和 EIP，请切换到那个区域查看。

7 思考题

在 x64 架构的虚拟机上安装 openEuler 操作系统运行上述实验，结果有何不同？

参考答案：和架构相关的实验，如系统环境实验中汇编语言实验会有所不同，其他不依赖于 CPU 架构的实验，如 iSulad 的实验，结果无有不同。

8 附录

8.1 在 x86_64 平台上进行编译内核实验

除了鲲鹏平台，openEuler 操作系统也支持 x86_64、RISC-V 等架构。兹列出在 x86_64 平台上的 openEuler 编译内核的步骤以供参考。

步骤 1 先查看当前的系统信息

```
# uname -m
x86_64
# uname -r
4.19.90-2003.4.0.0036.oe1.x86_64
```

注意当前所用内核版本为 4.19.XX-XXXX.X.X.XXXX.XXX.x86_64。

```
# cat /etc/os-release
NAME="openEuler"
VERSION="20.03 (LTS)"
ID="openEuler"
VERSION_ID="20.03"
PRETTY_NAME="openEuler 20.03 (LTS)"
ANSI_COLOR="0;31"
```

```
# lscpu | grep '^CPU(s):'
CPU(s): 4
```

注意本例中 CPU 是 4 核，您的系统可能与此不同。

步骤 2 准备编译内核的软件环境

```
# yum group install -y "Development Tools"
# yum install -y bc
# yum install -y openssl-devel
# yum install -y elfutils-libelf-devel
```

步骤 3 获取内核源代码

```
# cd ~
# wget https://gitee.com/openeuler/kernel/repository/archive/kernel-4.19.zip
```

```
# unzip kernel-4.19.zip
# ls
kernel-4.19.zip  kernel-kernel-4.19
# mv kernel-kernel-4.19/ kernel/
```

步骤 4 编译安装内核

```
# cd kernel

# make help | grep Image
* bzImage      - Compressed kernel image (arch/x86/boot/bzImage)

# make openeuler_defconfig
arch/x86/configs/openeuler_defconfig:2534:warning: override: reassigning to symbol HINIC
#
# configuration written to .config
#

# make -j4 bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

# make -j4 modules

# make modules_install
DEPMOD  4.19.208

# make install
sh ./arch/x86/boot/install.sh 4.19.208 arch/x86/boot/bzImage \
    System.map "/boot"
dracut-install: ERROR: installing 'virtio_pci'
dracut: FAILED:  /usr/lib/dracut/dracut-install -D /var/tmp/dracut.R4CHaO/initramfs --kerneldir
/lib/modules/4.19.208/ -m virtio_gpu xen-blkfront xen-netfront virtio_blk virtio_scsi virtio_net virtio_pci virtio_ring
virtio
```

注意：这里请忽略以上 dracut FAILED 错误。

以上 make 命令后的-j4 选项是以 4 个线程同时运行（因为在本例中是 4 核 CPU，所以简单用了 4 线程，您应该根据自己实际情况设置该线程数）。

步骤 5 重启系统

```
# reboot
```

步骤 6 重启系统并以选中新内核的 GRUB 启动菜单，以新内核引导系统，登录系统后检查内核版本

```
# uname -r
4.19.208
```


步骤 7 验证

这时可以编写一个 “Hello, world!” 的内核模块来验证可否在该内核上进行内核模块编程实验 (请参见源文件 hello_world.c 及其 Makefile)。

源文件 hello_world.c:

```
/* Hello from Kernel! */

#include <linux/module.h>

MODULE_LICENSE("GPL");

static char* guy = "Kernel";
module_param(guy, charp, 0644);
MODULE_PARM_DESC(guy, "char* param\n");

static int year = 2021;
module_param(year, int, 0644);
MODULE_PARM_DESC(year, "int param\n");

void hello_world(char* guy, int year)
{
    printk(KERN_ALERT "Hello, %s, %d!\n", guy, year);
}

int __init hello_init(void)
{
    printk(KERN_ALERT "Init module.\n");
    hello_world(guy, year);

    return 0;
}

void __exit hello_exit(void)
{
    printk(KERN_ALERT "Exit module.\n");
}

module_init(hello_init);
module_exit(hello_exit);
```

Makefile 文件:

```
# Build module hello_world

ifneq ($(KERNELRELEASE),)
    obj-m := hello_world.o
```

```
else
    KERNELDIR ?= /usr/lib/modules/$(shell uname -r)/build
    PWD := $(shell pwd)
default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
endif

.PHONY:clean
clean:
    -rm *.mod.c *.o *.order *.symvers *.ko
```

命令行:

make

```
insmod hello_world.ko guy="Dinu" year=2013
```

```
lsmod | grep hello
```

```
hello_world          262144  0
```

```
rmmod hello_world
```

```
dmesg | tail -n3
```

```
[ 6228.122161] Init module.
```

```
[ 6228.122433] Hello, Dinu, 2013!
```

```
[ 6254.880206] Exit module.
```

make clean

注意以上实验输出的进程号以及时间戳可能和您的实际情况不一致。

8.2 在本地 PC 与虚拟机之间互相拷贝文件

在实验的过程中，我们往往需要在本地 PC 与虚拟机之间互相拷贝文件，我们可以用 `scp` 命令完成该任务。

8.2.1 从本地 PC 拷贝文件到远端虚拟机

在本地 PC 的命令行终端中执行以下命令：

```
scp file1 root@119.3.185.3:~/
```

```
Authorized users only. All activities may be monitored and reported.
```

```
root@119.3.185.3's password:
```

```
file1 100% 0 0.0KB/s 00:00
```

以上 `scp` 命令将本地 PC 当前目录下的 `file1` 文件以 `root` 身份（当然输入密码的时候就要输入 `root` 用户的密码）拷贝至远端 IP 地址为 119.3.185.3 虚拟机的 `/root` 目录。

如果带“-r”选项则可连文件夹及其中的文件、文件夹一并拷贝，如：

```
scp -r folder1 root@119.3.185.3:~/
```

该命令将本地 PC 当前目录下的 *folder1* 文件夹及其中的文件、子文件夹以 root 身份拷贝至远端 IP 地址为 119.3.185.3 虚拟机的 /root 目录。

注意以上 scp 命中 IP 地址之后有一个冒号（以红色字体标出）。

8.2.2 从远端虚拟机拷贝文件到本地 PC

在本地 PC 的命令行终端中执行以下命令：

```
scp root@119.3.185.3:~/file2 ./
```

以上 scp 命令将远端 IP 地址为 119.3.185.3 虚拟机的 /root 目录下的 *file2* 文件拷贝至本地 PC 当前目录。

如果带“-r”选项则可连文件夹及其中的文件、文件夹一并拷贝，如：

```
scp -r root@119.3.185.3:~/folder2 ./
```

该命令以 root 身份将 IP 地址为 119.3.185.3 远端虚拟机的 *folder2* 文件夹及其中的文件、子文件夹拷贝到本地 PC 当前目录。

注意以上 scp 命中 IP 地址之后有一个冒号（以红色字体标出）。

8.3 终端软件的使用

在 Windows 10 中，我们一般选择 cmd 命令窗或 Windows Terminal 作为终端软件。在终端窗口中按下鼠标右键即可将系统剪贴板中的当前文本复制到当前终端命令行。以鼠标左键选中终端窗口中的文本再按鼠标右键，即可将选中的文本复制到系统剪贴板。

9 缩略语表

缩略语	英文全称	中文全称
ECS	Elastic Cloud Server	弹性云服务器
EIP	Elastic IP	弹性IP地址
PC	Personal Computer	个人电脑
SG	Security Groups	安全组
VPC	Virtual Private Cloud	虚拟私有云