openEuler操作系统

实验手册



华为技术有限公司

目录

[1构建实验环境 3](#_Toc48901772)

[1.1实验介绍 3](#_Toc48901773)

[1.1.1关于本实验 3](#_Toc48901774)

[1.1.2实验组网介绍 3](#_Toc48901775)

[1.1.3实验设备介绍 4](#_Toc48901776)

[1.2构建实验环境 5](#_Toc48901777)

[1.2.1购买VPC 5](#_Toc48901778)

[1.2.2购买ECS 7](#_Toc48901779)

[1.2.3通过ssh登录系统 9](#_Toc48901780)

[2 openEuler系统环境实验 12](#_Toc48901781)

[2.1实验介绍 12](#_Toc48901782)

[2.1.1关于本实验 12](#_Toc48901783)

[2.1.2实验目的 12](#_Toc48901784)

[2.1.3查看系统信息 12](#_Toc48901785)

[2.1.4测试编程环境 13](#_Toc48901786)

[2.1.5查看ARMv8-64汇编指令 14](#_Toc48901787)

[2.1.6编译和运行汇编程序 17](#_Toc48901788)

[3 iSulad的安装和基本操作 19](#_Toc48901789)

[3.1实验介绍 19](#_Toc48901790)

[3.1.1关于本实验 19](#_Toc48901791)

[3.1.2实验目的 19](#_Toc48901792)

[3.2安装isulad 19](#_Toc48901793)

[3.2.1安装 19](#_Toc48901794)

[3.2.2启动并查看版本 19](#_Toc48901795)

[3.3安装和配置CNI网络接口 20](#_Toc48901796)

[3.3.1前期准备 20](#_Toc48901797)

[3.3.2安装CNI 20](#_Toc48901798)

[3.3.3修改isulad的配置文件 21](#_Toc48901799)

[3.3.4运行容器hello-world 22](#_Toc48901800)

[3.3.5运行容器busybox 23](#_Toc48901801)

[3.4使用isula-build构建容器镜像 25](#_Toc48901802)

[3.4.1安装isula-build 25](#_Toc48901803)

[3.4.2构建容器镜像并导入到isulad 26](#_Toc48901804)

[3.4.3扩展实验：isula-build的其他镜像导出方式 28](#_Toc48901805)

[4智能优化引擎A-Tune实验 29](#_Toc48901806)

[4.1实验介绍 29](#_Toc48901807)

[4.1.1关于本实验 29](#_Toc48901808)

[4.1.2实验目的 29](#_Toc48901809)

[4.2安装A-Tune 29](#_Toc48901810)

[4.2.1安装 29](#_Toc48901811)

[4.2.2启动 30](#_Toc48901812)

[4.3运行atune-adm命令 31](#_Toc48901813)

[4.3.1查看版本 31](#_Toc48901814)

[4.3.2查询负载类型 31](#_Toc48901815)

[4.3.3分析负载类型并自优化 31](#_Toc48901816)

[4.3.4系统信息查询 32](#_Toc48901817)

[4.3.5离线业务自调优 32](#_Toc48901818)

[4.4资源清理 33](#_Toc48901819)

[4.4.1资源清理 33](#_Toc48901820)

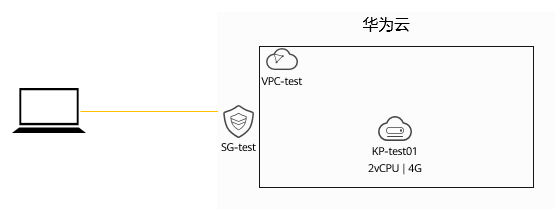
# 构建实验环境

## 实验介绍

### 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。

### 实验组网介绍



openEuler操作系统实验环境– 华为云版

注意：图中的”KP-test01”是ECS的名称，学生可以根据自己的情况设置

### 实验设备介绍

推荐设备配置如下表所示：

实验设备

|  |  |
| --- | --- |
| **云资源** | **规格** |
| ECS 鲲鹏计算 | 2vcpu | 4GB | 40GB |
| EIP带宽 | 5Mbit/s | 按流量计费 |

实验软件介绍：

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 / scp为例进行讲解。

## 构建实验环境

### 购买VPC

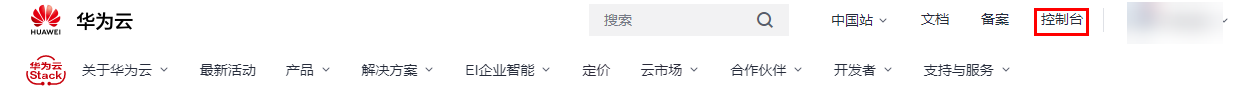
在浏览器地址栏输入[www.huaweicloud.com](http://www.huaweicloud.com)，登录华为云官网。



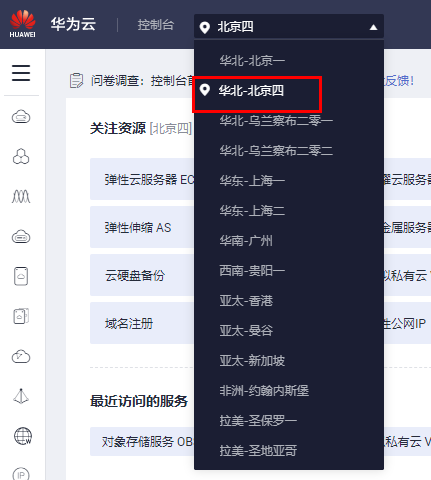
点击右上角的登录，打开登录窗口。

按要求输入账号密码，登录华为云。

点击右上角的“控制台”，打开控制台界面。



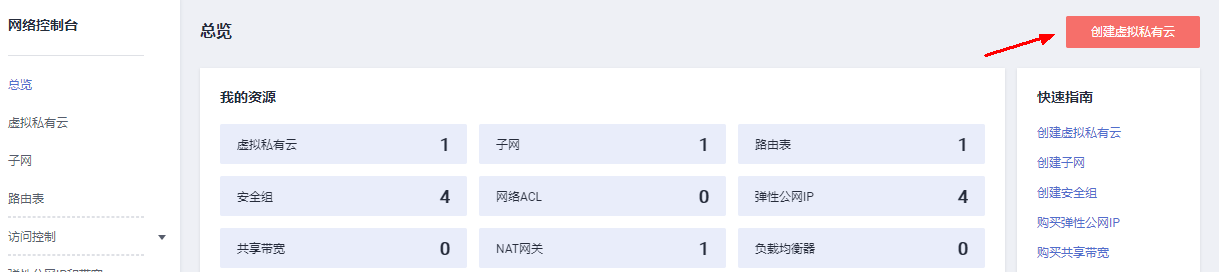
切换区域为“华北-北京四”（或“亚太-香港”）



点击控制台界面“关注资源”下面的“虚拟私有云VPC”，进入VPC控制台。



点击右上角的“创建虚拟私有云”。



按照如下表配置VPC参数。

|  |  |
| --- | --- |
| **参数** | **配置** |
| 区域 | 华北-北京四（或亚太-香港） |
| 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 |

注意：区域也可以配置成“亚太-香港”等其他地区。

配置完成后，点击“立即创建”，系统会自动回到VPC控制台。

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。



点击右上角的“创建安全组”。



在弹出的对话框中按如下图配置安全组参数，然后点击“确定”。



### 购买ECS

点左侧导航栏的“弹性云服务器ECS”，进入ECS控制台。



在ECS控制台界面点击右上角的“购买弹性云服务器”。



按照如下表配置弹性云服务器的基础配置参数。

|  |  |
| --- | --- |
| **参数** | **配置** |
| 计费模式 | 按需计费 |
| 区域 | 华北-北京四（或亚太-香港） |
| 可用区 | 随机分配 |
| 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：“区域”也可以是“亚太-香港”等其他地区，要和VPC的区域保持一致。“系统盘”也可配置成“高IO”。

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 |

注意：为加快实验进度，带宽大小也可以配置成10Mbit/s。

配置完成后，点击“下一步：高级配置”，按如下表配置ECS高级配置参数。

|  |  |
| --- | --- |
| **参数** | **配置** |
| 云服务器名称 | kp-test01（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* |
| 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* |

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。

在提交任务成功后，点击“返回云服务器列表”，返回ECS控制台。

### 通过ssh登录系统

在ECS控制台查看ECS弹性公网IP地址。



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@121.36.45.64

（注意：此处的IP地址121.36.45.64即是刚刚购买的弹性公网IP地址。）

第一次登录时会有安全性验证的提示：

The authenticity of host '121.36.45.64 (121.36.45.64)' 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 '121.36.45.64' (ECDSA) to the list of known hosts.

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

root@121.36.45.64'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 ~]#

# openEuler系统环境实验

## 实验介绍

### 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。

### 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。

### 查看系统信息

查看总体架构

[root@localhost ~]# uname –a

查看操作系统信息

[root@localhost ~]# cat /etc/os-release

查看CPU信息

[root@localhost ~]# lscpu

查看内存信息

[root@localhost ~]# free

查看磁盘信息

[root@localhost ~]# fdisk -l

查看系统资源实时信息

[root@localhost ~]# top

查看各种信息后按“q”退出。

### 测试编程环境

下面以C语言开发环境为例，查看基于鲲鹏架构的openEuler操作系统的编程环境。

查看gcc版本

[root@localhost ~]# 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）。

按照以下文件准备源代码

//

// test\_char.c

//

// The default type of char is 'unsigned char' in Kunpeng Platform

// gcc test\_char.c

#include <stdio.h>

#define JUDGE\_NEGATIVE\_OR\_NOT(x) \

((x) < 0 ? printf("negative") : \

((x) == 0) ? printf("zero") : \

printf("positive"))

int main()

{

unsigned char ch1 = -1;

char ch2 = -1;

signed char ch3 = -1;

// original code of 1: 0000 0001b

// ones' complement of 1: 1111 1110b

// two's complement of -1: 1111 1111b

printf("unsigned char ch1 = 0x%x, %d, ", ch1, ch1), JUDGE\_NEGATIVE\_OR\_NOT(ch1), printf("\n");

printf(" char ch2 = 0x%x, %d, ", ch2, ch2), JUDGE\_NEGATIVE\_OR\_NOT(ch2), printf("\n");

printf(" signed char ch3 = 0x%x, %d, ", ch3, ch3), JUDGE\_NEGATIVE\_OR\_NOT(ch3), printf("\n");

return 0;

}

可以用vi编辑该源代码：

[root@localhost ~]# vi test\_char.c

编译源文件

[root@localhost ~]# gcc test\_char.c

查看执行结果

[root@localhost ~]# ./a.out

unsigned char ch1 = 0xff, 255, positive

char ch2 = 0xff, 255, positive

signed char ch3 = 0xffffffff, -1, negative

带编译选项重新编译该文件

[root@localhost ~]# gcc -fsigned-char test\_char.c

再次查看结果，看两次结果有什么不同？

[root@localhost ~]# ./a.out

unsigned char ch1 = 0xff, 255, positive

char ch2 = 0xffffffff, -1, negative

signed char ch3 = 0xffffffff, -1, negative

找出原因

在shell中运行下面的命令：

[root@localhost ~]# lscpu | grep Archit | awk '{print $2}'

aarch64

如果有条件，再到X86机器的Linux上运行此命令，看两次结果有何不同？（可以看出，是不同CPU的架构、指令集导致差异。）

思考题：如果你知道计算机中的整数可以用补码表示就可以很好解释以上结果，相对于原码、反码，补码表示法有何优点？

### 查看ARMv8-64汇编指令

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架构下的汇编指令对比。

按照以下文件准备源代码

// abc.c

//

// "gcc -S abc.c" to generates assembly code

// "gcc abc.c" to generates binary code

//

#include <stdio.h>

int main()

{

int a = 1;

int b = 2;

int c = a + b;

printf("c = %d\n", c);

return 0;

}

可以用vi创建和编辑：

[root@localhost ~]# vi abc.c

编译源文件

[root@localhost ~]# gcc abc.c

查看执行结果是否正确

[root@localhost ~]# ./a.out

c = 3

带编译选项重新编译该文件

[root@localhost ~]# gcc -S abc.c

[root@localhost ~]# 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？

# abc\_openEuler@aarch64(ARMv8)

.arch armv8-a

.file "abc.c"

.text

.section .rodata

.align 3

.LC0:

.string "c = %d\n"

.text

.align 2

.global main

.type main, %function

main:

.LFB0:

.cfi\_startproc

stp x29, x30, [sp, -32]!

.cfi\_def\_cfa\_offset 32

.cfi\_offset 29, -32

.cfi\_offset 30, -24

add x29, sp, 0

.cfi\_def\_cfa\_register 29

mov w0, 1

str w0, [x29, 28]

mov w0, 2

str w0, [x29, 24]

ldr w1, [x29, 28]

ldr w0, [x29, 24]

add w0, w1, w0

str w0, [x29, 20]

adrp x0, .LC0

add x0, x0, :lo12:.LC0

ldr w1, [x29, 20]

bl printf

mov w0, 0

ldp x29, x30, [sp], 32

.cfi\_restore 30

.cfi\_restore 29

.cfi\_def\_cfa 31, 0

ret

.cfi\_endproc

.LFE0:

.size main, .-main

.ident "GCC: (GNU) 7.3.0"

.section .note.GNU-stack,"",@progbits

如果有条件，试着在X86\_64的机器上做同样的操作，看两次结果有什么不同？以下是在基于X86\_64架构CentOS生成的汇编代码，请做比较：

# abc\_CentOS@X86\_64

.file "abc.c"

.section .rodata

.LC0:

.string "c = %d\n"

.text

.globl main

.type main, @function

main:

.LFB0:

.cfi\_startproc

pushq %rbp

.cfi\_def\_cfa\_offset 16

.cfi\_offset 6, -16

movq %rsp, %rbp

.cfi\_def\_cfa\_register 6

subq $16, %rsp

movl $1, -4(%rbp)

movl $2, -8(%rbp)

movl -8(%rbp), %eax

movl -4(%rbp), %edx

addl %edx, %eax

movl %eax, -12(%rbp)

movl -12(%rbp), %eax

movl %eax, %esi

movl $.LC0, %edi

movl $0, %eax

call printf

movl $0, %eax

leave

.cfi\_def\_cfa 7, 8

ret

.cfi\_endproc

.LFE0:

.size main, .-main

.ident "GCC: (GNU) 4.8.5 20150623 (Red Hat 4.8.5-39)"

.section .note.GNU-stack,"",@progbits

用objdump命令观察汇编代码

[root@localhost ~]# gcc -g abc.c

[root@localhost ~]# objdump -S ./a.out

请查看与C语句对应的汇编代码。

思考题：在基于X86\_64机器的Linux操作系统上完成同样的步骤，观察两者机器码的不同。你能说出ARMv8\_64指令集相对于X86\_64的一些特点吗？

### 编译和运行汇编程序

按以下文件内容编辑源代码

# hello.s

.text

.global start

start:

mov x0,#0

ldr x1,=msg

mov x2,len

mov x8,64

svc #0

mov x0,123

mov x8,93

svc #0

.data

msg:

.ascii "Hello openEuler!\n\0"

len=.-msg

编译源文件

[root@localhost ~]# as -o hello.o hello.s

产生可执行文件

[root@localhost ~]# ld -o hello hello.o

ld: warning: cannot find entry symbol \_start; defaulting to 00000000004000b0

此处的告警信息请忽略。

查看运行结果

[root@localhost ~]#./hello

Hello openEuler!

思考题：分析该汇编代码各个语句的作用，有条件的话与X86平台的汇编指令比较异同。

# iSulad的安装和基本操作

## 实验介绍

### 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。

### 实验目的

学会安装iSula容器引擎isulad；

学会如何为isulad配置CNI网络接口；

学会如何建立、运行、停止以及删除一个容器；

学会构建自己的容器镜像并运行之。

## 安装isulad

### 安装

用yum命令安装

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

### 启动并查看版本

用systemctl start命令启动

[root@openeuler ~]# systemctl start isulad

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

## 安装和配置CNI网络接口

### 前期准备

安装jq (后面的操作可能会用到jq)

[root@openeuler ~]# yum install –y jq

（虽然建议自己输入命令行，但如果拷贝粘贴此命令的话，注意参数-y的“-”被正确粘贴。最好将命令拷贝至记事本再进行粘贴。）

### 安装CNI

首先建立netconf文件以描述网络

[root@openeuler ~]# mkdir CNI

[root@openeuler ~]# cd CNI

[root@openeuler CNI]# mkdir -p /etc/cni/net.d

[root@openeuler CNI] # vi /etc/cni/net.d/10-mynet.json #在此处编辑该文件

[root@openeuler CNI] # cat /etc/cni/net.d/10-mynet.json

{

"cniVersion": "0.3.0",

"name": "default",

"type": "bridge",

"bridge": "cni0",

"ipam": {

"type": "host-local",

"subnet": "10.1.0.0/16",

"gateway": "10.1.0.1"

},

"dns": {

"nameservers": [ "10.1.0.1" ]

}

}

构建CNI的plugins

[root@openeuler CNI]# git clone https://github.com/containernetworking/plugins.git

[root@openeuler CNI]# cd plugins/

[root@openeuler plugins]# ./build\_linux.sh

[root@openeuler plugins]# mkdir -p /opt/cni/bin

[root@openeuler plugins]# cp ./bin/\* /opt/cni/bin/

至此，CNI安装完毕。

### 修改isulad的配置文件

修改配置文件daemon.json

[root@openeuler plugins]# cd

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

[root@openeuler iSula]# cp /etc/isulad/daemon.json /etc/isulad/daemon.json.bak

[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": [

"docker.io"

],

"insecure-registries": [],

"pod-sandbox-image": "gcrio/pause-arm64:3.1",

"image-opt-timeout": "5m",

"image-server-sock-addr": "unix:///var/run/isulad/isula\_image.sock",

"native.umask": "secure",

"network-plugin": "cni",

"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：

设"registry-mirrors"的值为"docker.io"

设"network-plugin"为"cni"

注意：

"cni-bin-dir"的默认值即为"/opt/cni/bin"

"cni-conf-dir"的默认值即为"/etc/cni/net.d"

我们上一步安装CNI时即是按照这样的目录结构安排的。

另外"pod-sandbox-image"的值我们设成了"gcrio/pause-arm64:3.1"，这是为以后可能使用CRI接口准备的，此处暂不详细解释。

设置环境变量

[root@openeuler iSula]# export CNI\_PATH=/opt/cni/bin

[root@openeuler iSula]# export PATH=$PATH:$CNI\_PATH

[root@openeuler iSula]# export NETCONFPATH=/etc/cni/net.d

重启isulad服务

[root@openeuler iSula]# systemctl restart isulad

### 运行容器hello-world

用isula run命令直接运行

[root@openeuler iSula]# isula run hello-world

Unable to find image 'hello-world' locally

Image "hello-world" pulling

Image "a29f45ccde2ac0bde957b1277b1501f471960c8ca49f1588c6c885941640ae60" pulled

Hello from Docker!

This message shows that your installation appears to be working correctly.

由于这是第一次运行，所以会拉取hello-world的镜像，然后会运行它的一个实例，该实例会打印这样一个字符串：

Hello from Docker!

这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。

### 运行容器busybox

创建容器并启动之

[root@openeuler iSula]# isula create -it busybox

Unable to find image 'busybox' locally

Image "busybox" pulling

Image "5a9d3c3dd268611214b65066c72d5fe73cf948f8b6195227de70119858bab258" pulled

bf76b1e5c1c624faaa8a1f81887607ca002ebe3939047508738988134dcd00ec

[root@openeuler iSula]# isula start bf76b1e5c1c6

直接运行

[root@openeuler iSula]# isula run -itd busybox

6c1d81467d3367a90dd6e388a16c80411d4ba76316d86b6f56463699306e1394

可以看出是以一个新的实例运行的。

交互式运行

[root@openeuler iSula]# isula run -it busybox

/ # ls

bin dev etc home proc root sys tmp usr var

/ # 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

/ # ifconfig

lo Link encap:Local Loopback

inet addr:127.0.0.1 Mask:255.0.0.0

inet6 addr: ::1/128 Scope:Host

UP LOOPBACK RUNNING MTU:65536 Metric:1

RX packets:0 errors:0 dropped:0 overruns:0 frame:0

TX packets:0 errors:0 dropped:0 overruns:0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

/ # busybox –help #这是个错误命令，但是可以看到busybox的版本

BusyBox v1.32.0 (2020-07-27 19:11:43 UTC) multi-call binary.

……

/ # exit

暂停/恢复一个容器

[root@openeuler iSula]# isula pause bf76b1e5c1c6

bf76b1e5c1c6

[root@openeuler iSula]# isula unpause bf76b1e5c1c6

bf76b1e5c1c6

先停止，再删除一个容器

[root@openeuler iSula]# isula stop bf76b1e5c1c6

bf76b1e5c1c6

[root@openeuler iSula]# isula rm bf76b1e5c1c6

bf76b1e5c1c624faaa8a1f81887607ca002ebe3939047508738988134dcd00ec

查看正在运行着的容器

[root@openeuler iSula]# isula ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

6c1d81467d33 busybox "sh" 37 minutes ago Up 37 minutes 6c1d81467d3367a90dd6e388a16c80411d4ba76316d86b6f56463699306e1394

直接删除

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

6c1d81467d3367a90dd6e388a16c80411d4ba76316d86b6f56463699306e1394

查看镜像关联到容器没有

[root@openeuler iSula]# isula ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

c17c95817f73 busybox "sh" 54 minutes ago Exited (130) 52 minutes ago c17c95817f7375b4b1bcafdff5c51442b96185ef598335a53a3f3f9ec8f4f2f6

先将关联到镜像的容器销毁

[root@openeuler iSula]# isula rm -f c17c95817f7375

c17c95817f7375b4b1bcafdff5c51442b96185ef598335a53a3f3f9ec8f4f2f6

然后删除镜像

[root@openeuler iSula]# isula rmi busybox

Image "busybox" removed

通过另一个名字建立容器

[root@openeuler iSula]# isula create -t -n test busybox

Unable to find image 'busybox' locally

Image "busybox" pulling

Image "5a9d3c3dd268611214b65066c72d5fe73cf948f8b6195227de70119858bab258" pulled

132a5fd8cadd1afffb8cab67a3d79f6543f355484a15fc50a7a97fd2b2cdd18b

然后启动该容器

[root@openeuler iSula]# isula start test

[root@openeuler iSula]# isula ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

132a5fd8cadd busybox "sh" 3 minutes ago Up 7 seconds test

杀死该容器

[root@openeuler iSula]# isula kill test

test

[root@openeuler iSula]# isula ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

[root@openeuler iSula]# isula ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

132a5fd8cadd busybox "sh" 5 minutes ago Exited (137) 32 seconds ago test

最后删除该容器

[root@openeuler iSula]# isula rm test

132a5fd8cadd1afffb8cab67a3d79f6543f355484a15fc50a7a97fd2b2cdd18b

[root@openeuler iSula]# isula ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

## 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docker.io下载的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。

### 安装isula-build

用yum命令安装

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

用systemctl start命令启动

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

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 20 02:16:07 2020

OS/Arch: linux/arm64

Server:

Version: 0.9.2

Go Version: go1.13.3

Git Commit: a96cf18

Built: Thu Aug 20 02:16:07 2020

OS/Arch: linux/arm64

修改配置

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

cp /etc/isula-build/registries.toml /etc/isula-build/registries.toml.bak

vi /etc/isula-build/registries.toml

……

[registries.search]

registries = ["docker.io"]

……

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: 3.12 GB

MemFree: 2.07 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:

docker.io

Insecure Registries:

在Search Registries 配置项里看到”docker.io”赫然纸上。

### 构建容器镜像并导入到isulad

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]# catDockerfile

FROM busybox

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busybox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 76 Aug 21 12:31 Dockerfile

-rwx------ 1 root root 38 Aug 21 12:32 hello.sh

以上即是我们刚刚创建的2个文件。

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 busybox

STEP 2: COPY hello.sh /usr/bin/

STEP 3: CMD ["sh", "-c", "/usr/bin/hello.sh"]

…

Build success with image id: b195dbe534918dbb67694ab9ee9e15a02a7ae16018c93e212debd71072b4086b

查询构建出来的容器镜像

[root@openeuler test]# isula-build ctr-img images

-------------------------- ----------- ----------------- ------------------------ ------------

REPOSITORY TAG IMAGE ID CREATED SIZE

-------------------------- ----------- ----------------- ------------------------ ------------

hello-isula-build v0.1 b195dbe53491 2020-08-11 08:01:27 1.45 MB

docker.io/library/busybox latest 018c9d7b792b 2020-07-28 00:19:37 1.45 MB

------------------------------ ----------- ----------------- ------------------------ ------------

[root@openeuler test]# isula images

REPOSITORY TAG IMAGE ID CREATED SIZE

hello-isula-build v0.1 b195dbe53491 2020-08-11 16:01:27 1.380 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

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!

### 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' .

在时间充足的情况下，可以按照命令行帮助信息中给出的详细示例进行扩展实验。

# 智能优化引擎A-Tune实验

## 实验介绍

### 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。

### 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。

## 安装A-Tune

### 安装

安装系统软件的依赖包

回到root用户目录：

[root@openeuler test]# cd

用yum命令进行安装：

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

安装Python的依赖包

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

[root@openeuler ~]# pip3 install hyperopt lhsmdu

下载A-Tune的源代码

[root@openeuler ~]#git clone –b master 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（注意：虽然建议手动输入命令，但如果直接从这里复制粘贴的话注意不要遗漏字母b前面的“-”符号。）

编译及安装

cd A-Tune

make && make install

编译的时候会花一定时间，请耐心等待。

### 启动

修改A-Tune的配置文件

[root@openeuler A-Tune]# ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.1.88 netmask 255.255.255.0 broadcast 192.168.1.255

……

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

……

[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

[root@openeuler A-Tune]# cp /etc/atuned/atuned.cnf /etc/atuned/atuned.cnf.bak

[root@openeuler A-Tune]# vi /etc/atuned/atuned.cnf

[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。

装载并启动atuned服务

[root@openeuler A-Tune]# systemctl daemon-reload

[root@openeuler A-Tune]# systemctl start atuned

[root@openeuler A-Tune]# systemctl start atune-engine

[root@openeuler A-Tune]# ps -ef | grep atuned #此一步是为查看atuned有没有启动

root 15696 1 0 12:44 ? 00:00:00 /usr/bin/atuned

root 15703 15696 5 12:44 ? 00:00:00 python3 /usr/libexec/atuned/analysis/app.py /etc/atuned/atuned.cnf

root 15733 1 12 12:45 ? 00:00:01 /usr/bin/python3 /usr/libexec/atuned/analysis/app\_engine.py /etc/atuned/atuned.cnf

root 15758 7251 0 12:45 pts/0 00:00:00 grep --color=auto atuned

检查A-Tune服务运行状态

[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

按“q”或“Q”键从查看信息的状态中退出。

## 运行atune-adm命令

### 查看版本

查看atune-adm版本信息

[root@openeuler A-Tune]# atune-adm --version

atune-adm version 0.2(9d6cacf)

### 查询负载类型

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。

### 分析负载类型并自优化

负载类型识别

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

采集系统的实时统计数据并进行负载类型识别（注意：如果直接从这里复制粘贴此命令注意不要遗漏选项characterization前的“--”符号。）

负载类型识别及自动优化

[root@openeuler A-Tune]# atune-adm analysis

采集系统的实时统计数据进行负载类型识别，并进行自动优化。

### 系统信息查询

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。

### 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。

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@openeuler compress]# ls

compress\_client.yaml compress.py compress\_server.yaml enwik8.zip prepare.sh README

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 compress.py.bak

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。

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。

## 资源清理

### 资源清理

回到ECS控制台，勾选kp-test01虚拟机，然后点击“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。

