|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
|  | | |
| 方圆ok1 | | | |
|  | AR开放平台二次开发指南 | | 附件1-16K |
|  | |
| **文档版本** | **02** |
| **发布日期** | **2017-06-14** |
|  | |
| 华为技术有限公司 | |
|  | | |

|  |
| --- |
| 版权所有 © 华为技术有限公司 2016。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  附件3-版权声明页图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暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http:// e.[huawei](http://enterprise.huawei.com).com |

前 言

概述

手册目的是描述AR开放平台架构以及如何在AR开放平台上进行开发，指导客户完成在AR开放平台上的应用开发与运行。

第一段为AR开放平台系统架构介绍；

第二段为AR开放平台组网配置指导；

第三段为AR开放平台交叉编译环境搭建指导与开发举例。

读者对象

本文档（本指南）主要适用于以下工程师：(读者对象，要从“文档指南”中选择，名称要一致。)

* 技术支持工程师
* 维护工程师

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 |
| --- | --- |
| 3.png | 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。 |
| 2.png | 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。 |
| 备份.png | 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。 |
| 0.png | 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、设备性能降低或其它不可预知的结果。  “注意”不涉及人身伤害。 |
| 说明 | 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、设备及环境伤害信息。 |

修改记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 01 (2016-01-15)

第一次正式发布。

目 录

[前 言 ii](#_Toc486543854)

[1 概述 1](#_Toc486543855)

[1.1 边缘计算 1](#_Toc486543856)

[1.2 AR边缘计算开放能力 2](#_Toc486543857)

[1.3 端到端整体方案架构 3](#_Toc486543858)

[2 AR开放平台二次开发指南 3](#_Toc486543859)

[2.1 二次开发须知 3](#_Toc486543860)

[2.2 开放平台概述 4](#_Toc486543861)

[2.2.1 AR开放平台 4](#_Toc486543862)

[2.2.1.1 概述 4](#_Toc486543863)

[2.2.1.2 架构 4](#_Toc486543864)

[2.2.2 网络角度 5](#_Toc486543865)

[2.2.3 二次开发角度 5](#_Toc486543866)

[2.2.3.1 容器内物理接口访问 6](#_Toc486543867)

[2.2.3.2 容器内VRP接口访问 6](#_Toc486543868)

[2.2.4 容器环境介绍 7](#_Toc486543869)

[2.2.5 容器部署 7](#_Toc486543870)

[2.2.5.1 Agile Controller方式 8](#_Toc486543871)

[2.2.5.2 命令行方式 8](#_Toc486543872)

[2.2.5.2.1 进入yangsh命令行界面 8](#_Toc486543873)

[2.2.5.2.2 进入容器目录 8](#_Toc486543874)

[2.2.5.2.3 安装容器OVA镜像 8](#_Toc486543875)

[2.2.5.2.4 激活容器 8](#_Toc486543876)

[2.2.5.2.5 启动容器 9](#_Toc486543877)

[2.2.5.2.6 停止容器 9](#_Toc486543878)

[2.2.5.2.7 去激活容器 (在停止之后操作) 9](#_Toc486543879)

[2.2.5.2.8 卸载OVA容器镜像（在去激活之后操作） 9](#_Toc486543880)

[2.2.5.2.9 删除容器 10](#_Toc486543881)

[2.2.5.2.10 保存配置到FLASH中 10](#_Toc486543882)

[2.3 设备及容器的网络配置指导 10](#_Toc486543883)

[2.3.1 组网需求 10](#_Toc486543884)

[2.3.2 配置思路 11](#_Toc486543885)

[2.3.3 操作步骤 11](#_Toc486543886)

[2.3.3.1 创建容器并启动 11](#_Toc486543887)

[2.3.3.2 配置设备为linux容器系统分配IP地址 11](#_Toc486543888)

[2.3.3.3 配置设备VRP平台物理口ETH0/0/0的IP地址，同时配置私网PC以DHCP方式从设备获取IP地址；完成linux容器与私网用户PC互访 12](#_Toc486543889)

[2.3.3.4 使用SSH登录到容器 13](#_Toc486543890)

[2.3.3.5 配置设备3G/LTE链路，使私网用户通过3G/LTE业务接入internet; 14](#_Toc486543891)

[2.3.4 配置文件 14](#_Toc486543892)

[2.3.5 VRP配置 14](#_Toc486543893)

[2.4 开发指导 15](#_Toc486543894)

[2.4.1 开发步骤 15](#_Toc486543895)

[2.4.2 环境准备 15](#_Toc486543896)

[2.4.2.1 AR路由器 15](#_Toc486543897)

[2.4.2.1.1 连接配置线缆 15](#_Toc486543898)

[2.4.2.1.2 产品手册 18](#_Toc486543899)

[2.4.2.2 交叉编译环境 18](#_Toc486543900)

[2.4.2.2.1 安装Ubuntu 18](#_Toc486543901)

[2.4.2.2.2 安装并启动Docker 19](#_Toc486543902)

[2.4.2.2.3 在Docker中访问Ubuntu用户的资源 21](#_Toc486543903)

[2.4.3 Hello world(example) 21](#_Toc486543904)

[2.4.3.1 二进制程序开发 21](#_Toc486543905)

[2.4.3.2 程序编译 22](#_Toc486543906)

[2.4.3.3 程序安装与运行 23](#_Toc486543907)

[2.4.4 Linux socket (example) 24](#_Toc486543908)

[2.4.4.1 二进制程序开发 24](#_Toc486543909)

[2.4.4.2 程序编译 28](#_Toc486543910)

[2.4.4.3 程序安装 28](#_Toc486543911)

[2.5 附件 31](#_Toc486543912)

[3 应用案例 32](#_Toc486543913)

[迅达电梯 32](#_Toc486543914)

[4 技术支持 33](#_Toc486543915)

# 概述

## 边缘计算

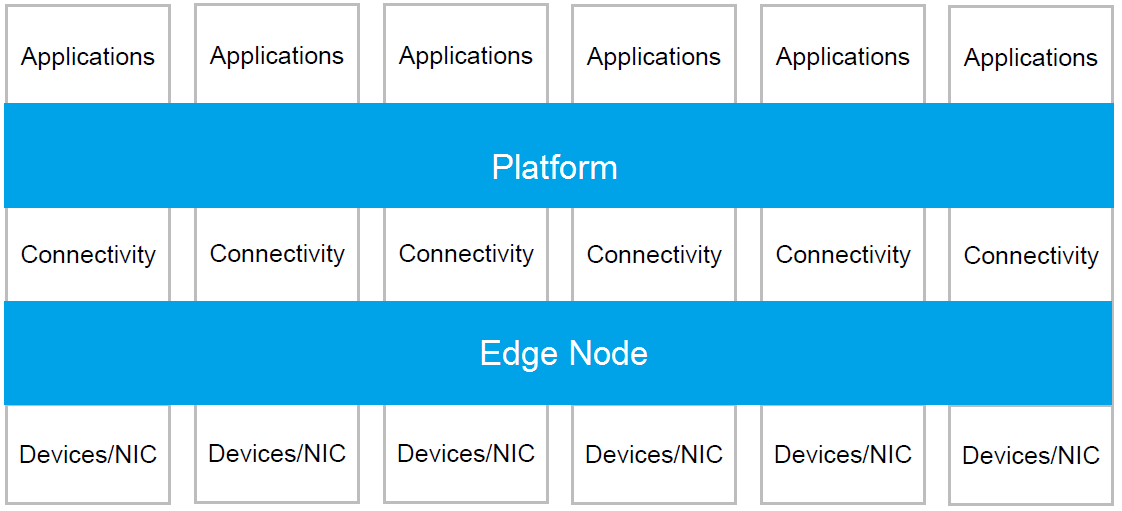
在靠近终端的网络边缘节点上,提供连接、计算、存储、控制与应用功能，满足用户实时、智能、安全和数据聚合等需求。

借助成熟的通信技术，在位于网络边缘的节点上分布式部署计算、存储、安全等能力，把中心节点的计算、存储、通信压力分散到计算能力稍弱的边缘节点，同时实现了服务的低时延、高可靠、低成本，也有效的保护了用户的边缘隐私，支持网络从成本中心向商业价值中心的转移。

边缘计算核心价值：

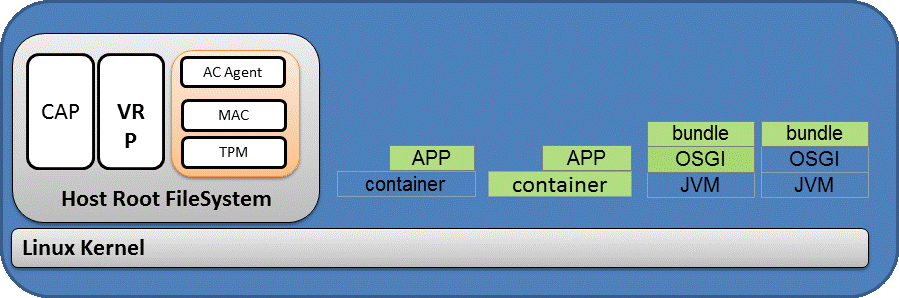
* 支持实时性业务：可以做到ms级的数据实时分析、事件实时响应
* 支持边缘智能分析处理：业务边缘部署、灵活调整，网络自动运维
* 数据聚合：消除数据碎片化、屏蔽无效噪声、数据按需上传
* 私有的安全域：包括数据安全、节点安全、网络安全

边缘计算体系架构：

******

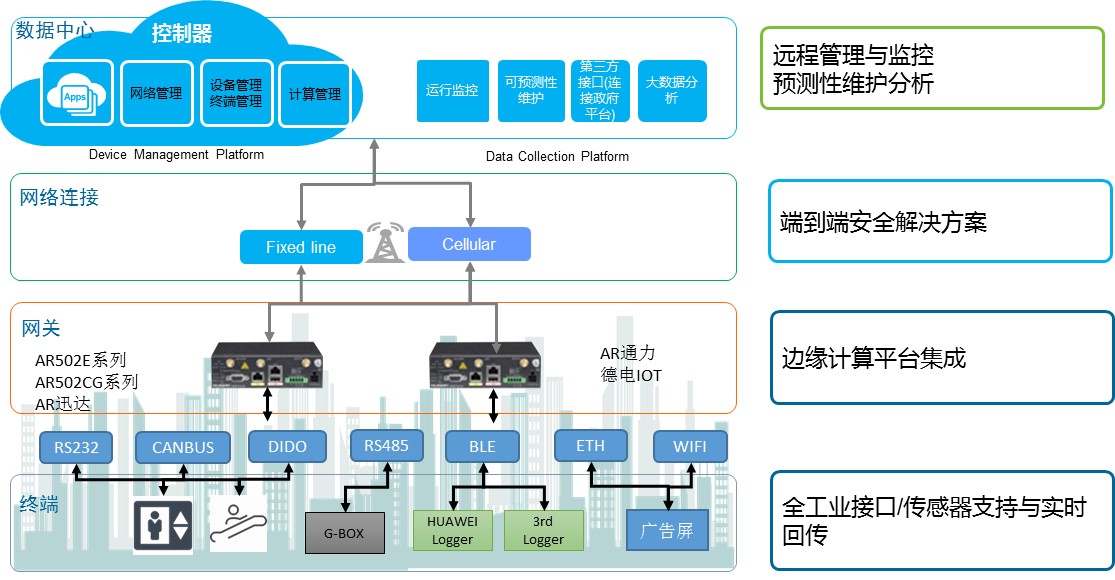
AR IOT网关设备处于上图中Edge Node位置，作为网络节点提供边缘计算能力。

## AR边缘计算开放能力



|  |  |
| --- | --- |
| **1 提供容器级对应工具链** | **客户直接开发部署自己的应用** |
| 2 提供APP级容器 | 客户直接将自己的应用作为容器，直接在设备上部署，docker方式 |
| 3 提供JVM虚拟机 | 客户直接开放JAVA软件在设备上运行 |
| 4 提供JVM虚拟机+ OSGI框架 | 客户只要开发bundle，实现业务APP在OSGI框架内 |

## 端到端整体方案架构



AR网关提供边缘计算平台，具备二次开放能力，开发者可以根据场景选择对应的款型，部署开发自己的应用，采集终端数据，进行处理，然后与远端IOT平台进行交互。

控制器由华为提供，可以实现对网关的管理。

# AR开放平台二次开发指南

## 二次开发须知

本文档的读者应该具备下面技能：

* 熟悉AR设备。
* 熟悉linux系统的管理和应用开发。
* 熟悉嵌入式设备的交叉开发流程。
* 熟悉C/C++语言二进制应用开发。



本文档涉及的代码仅供学习参考。

## 开放平台概述

### AR开放平台

#### 概述

AR设备支持集成开源、或者客户开发的应用，以扩展设备功能。即AR设备作为一个开放平台，支持客户开发、部署自己的应用。

#### 架构



AR软件架构抽象为双系统：VRP网关系统、linux系统。双系统间通过以太网络进行通信。VRP网关系统作为应用也运行在同一个linux操作系统上。

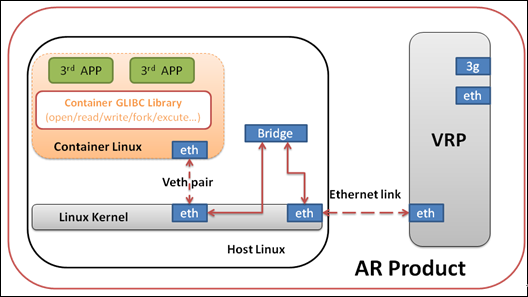
双系统的分工：

VRP网关系统管理所有的网络接口（WIFI/3G/ETH等）。利用VRP协议栈的路由、安全、AAA用户管理、VPN、多核转发等能力，提供功能丰富、安全、高性能的网络通道

Linux系统管理（RS232/RS485/CANBUS/DIDO）接口和存储设备，作为宿主机，提供容器运行环境，将这些物理接口映射到容器中，开放给容器中的第三方客户应用；客户应用运行在容器中，就好比运行在一个LINUX系统中。

### 网络角度

从网络通信的角度看：Linux系统是台PC，VRP是一个路由器网关，即逻辑上是两个设备。Linux系统和外部进行网络通信时，需要通过VRP系统进行转发。在Linux系统（宿主机）运行的容器是虚拟出来的另一台PC，容器需要与外部通信时通过veth pair先将报文转发到宿主LINUX，然后再通过bridge将报文转发到VRP，通过VRP转发出去。



### 二次开发角度

设备在容器内集成mqtt bridge（mosquito），在容器内通过json over mqtt消息接口经过华为集成的OPEN APP去访问VRP，获取系统信息，比如LTE接口状态、系统版本、CPU占用率、内存占用率、存储空间占用灯信息。

在容器内客户可以部署IOT Agent采集串口、CAN口所连接终端的信息，并且去连接远端平台，将终端信息发送到远端IOT平台。



#### 容器内物理接口访问

设备提供的串口、Canbus、BLE接口会映射到容器内部，体现为标准的linux tty设备节点，开发者可以直接使用标准LINUX读写接口去访问设备节点。

#### 容器内VRP接口访问



通过Json Over Mqtt消息接口访问VRP；MQTT通过发布与订阅进行消息交互，属于异步方式，开发者在开发时需要了解，具体实现可以参考下面方式实现。

Mqtt消息分为三类：

系统静态消息，OpenAPP会主动获取并设置retained标志，等待3rd APP主动获取；



动态与配置消息，需要3rd APP主动发起获取，OpenAPP获取后会发布给3rd APP；



事件消息，OpenAPP会主动发布，并设置retained标志，3rd APP需要订阅



Json消息格式详见附录部分。

注：上图中静态消息会在系统启动后，openapp主动获取并进行发布；事件消息，只有在设备中事件被触发后，实时进行通知；

### 容器环境介绍

设备内部容器系统选用Jessie Debian linux, 源于标准debia。当前AR5XX款型选用架构为armel(不支持硬浮点)。

在容器内部包含systemd系统服务管理软件、coreutil、apt-get deb包管理工具、rsyslog日志服务、Cron服务、dhcp client、ssh service等一些基础工具，详细信息可以在容器内通过dpkg –l命令查看提供了哪些软件包。

对于开发者，可选集成python、java vm运行环境，开发者在容器启动完成后通过apt-get命令从debian社区进行下载，自行完成应用安装.

容器内部默认包含有veth0网络口，用于经过VRP路由连接到外部网络，veth0网络IP地址默认由VRP经过分配，当前固定分配地址为192.168.100.254，客户可以连接设备物理eth口通过SSH

登录到容器内，默认用户名与密码都为root。

注：HAUWEI默认发布带有基础工具的容器镜像，如果有特殊需求，可在商业销售时进行注明。

### 容器部署

容器部署方式有两种，一种是通过华为Agile Controller远程管理设备，在设备上管理容器，另一种就是在设备通过命令行方式管理容器。

容器镜像包采用OVA包方式，里面包含容器rootfs镜像以及容器默认配置信息。

注：容器OVA镜像可从华为support网站上进行获取。（当前尚未发布release版本）。对于定制款型，会在设备中预置容器，无需再进行安装。

#### Agile Controller方式

Agile Controller是华为开发用来远程管理设备的云管理工具，支持设备管理、设备监控、网络配置下发、容器管理、APP管理。详情可以参考Agile Controller资料。

#### YANGSH命令行方式

首先请从华为support网站获取对应设备款型容器镜像（当前尚未正式发布），然后通过FTP或者U盘方式拷贝到设备内。

假设需要部署容器镜像包为lxc01.ova,容器名称为lxc01.进入Host Linux的shell，开始命令行方式管理容器。

##### 进入yangsh命令行界面

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/mnt/squash/local/lib;/mnt/squash/local/bin/yangsh

##### 进入容器目录

cd huawei-virtualization

##### 安装容器OVA镜像

/huawei-virtualization$ install-ova

[install-ova]$ set name lxc01

[install-ova]$ set ova-file lxc01.ova

[install-ova]$ commit

##### 激活容器

/huawei-virtualization$ cd managed-services-config/

/huawei-virtualization/managed-services-config$ cd services/

/huawei-virtualization/managed-services-config/services$ add-listentry name lxc01

/huawei-virtualization/managed-services-config/services$ ls

services[name=lxc01]

/huawei-virtualization/managed-services-config/services$ cd services[name=lxc01]/

/huawei-virtualization/managed-services-config/services[name=lxc01]$ set autostart false

/huawei-virtualization/managed-services-config/services[name=lxc01]$ set hyperv linux-lxc

/huawei-virtualization/managed-services-config/services[name=lxc01]$ cd configuration-file/

/huawei-virtualization/managed-services-config/services[name=lxc01]/configuration-file$ set name lxc01\_config

/huawei-virtualization/managed-services-config/services[name=lxc01]/configuration-file$ set type lxc-conf

/huawei-virtualization/managed-services-config/services[name=lxc01]/configuration-file$ cd ..

/huawei-virtualization/managed-services-config/services[name=lxc01]$ set autostart true

/huawei-virtualization/managed-services-config/services[name=lxc01]$ cd ..

/huawei-virtualization/managed-services-config/services[name=lxc01]$ save

/huawei-virtualization/managed-services-config/services$ save

/huawei-virtualization/managed-services-config/services$ commit

##### 启动容器

/huawei-virtualization/managed-services-config/services$ cd ..

/huawei-virtualization/managed-services-config$ cd ..

/huawei-virtualization$ service-oper

[service-oper]$ set name lxc01

[service-oper]$ set hyperv linux-lxc

[service-oper]$ set command start

[service-oper]$ commit

##### 停止容器

/huawei-virtualization$

/huawei-virtualization$ service-oper

[service-oper]$ set name lxc01

[service-oper]$ set hyperv linux-lxc

[service-oper]$ set command stop

[service-oper]$ commit

##### 去激活容器 (在停止之后操作)

/huawei-virtualization$ cd managed-services-config/

/huawei-virtualization/managed-services-config$ cd services/

/huawei-virtualization/managed-services-config/services$ ls

services[name=lxc01]

/huawei-virtualization/managed-services-config/services$ delete services[name=lxc01]/

/huawei-virtualization/managed-services-config/services$commit

##### 卸载OVA容器镜像（在去激活之后操作）

/huawei-virtualization$ uninstall-ova

[uninstall-ova]$ set name lxc01

[uninstall-ova]$ set ova-path lxc01.ova

[uninstall-ova]$ set rm-image true

[uninstall-ova]$ commit

##### 删除容器

/huawei-virtualization$ destruct-conf

[destruct-conf]$ set name lxc01

[destruct-conf]$ set hyperv linux-lxc

[destruct-conf]$ cd configuration-file/

[destruct-conf]configuration-file$ set name lxc01\_config

[destruct-conf]configuration-file$ set type lxc-conf

[destruct-conf]configuration-file$ cd ..

[destruct-conf]$ set rm-image false

[destruct-conf]$ commit

##### 保存配置到FLASH中

/huawei-virtualization$ cd ..   
  
/$ copy-config running startup

#### VRP命令行方式

##### 进入容器配置视图

[Huawei]lxc

[Huawei-lxc]

##### 创建容器

[Huawei-lxc]container-create name lxc0 rootfs lxc-esdk-root-squash-new

##### 启动容器

[Huawei-lxc]container-start lxc0

##### 停止容器

[Huawei-lxc]container-stop lxc0

[Huawei-lxc]container-del lxc0

##### 查看容器状态

[Huawei-lxc]display container-status

Name:lxc0

Status:active

Config:/mnt/flash/lxc\_dir/lxc0/lxc0\_config

Rootfs:/mnt/flash/lxc-esdk-root-squash-new

##### 查看容器信息

[Huawei-lxc]display container-info lxc0

Name: lxc0

State: RUNNING

PID: 29211

IP: 200.1.1.253

CPU use: 16.86 seconds

Memory use: 36.54 MiB

KMem use: 0 bytes

##### 查看容器配置

[Huawei-lxc]display container-cfg lxc0

lxc.mount.entry = proc proc proc nodev,noexec,nosuid 0 0

lxc.mount.entry = sysfs sys sysfs defaults 0 0

lxc.network.type = veth

lxc.network.flags = up

lxc.network.link = br0

lxc.network.name = veth1

lxc.network.mtu = 1500

lxc.pts = 9

lxc.tty = 9

lxc.cgroup.devices.deny = a

lxc.cgroup.devices.allow = c 1:3 rwm

lxc.cgroup.devices.allow = c 1:5 rwm

lxc.cgroup.devices.allow = c 5:1 rwm

lxc.cgroup.devices.allow = c 5:0 rwm

lxc.cgroup.devices.allow = c 1:9 rwm

lxc.cgroup.devices.allow = c 1:8 rwm

lxc.cgroup.devices.allow = c 136:\* rwm

lxc.cgroup.devices.allow = c 5:2 rwm

lxc.cgroup.devices.allow = c 10:200 rwm

lxc.hook.autodev = sh -c "modprobe tun; cd ${LXC\_ROOTFS\_MOUNT}/dev; mkdir net; mknod net/tun c 10 200; chmod 0666 net/tun"

lxc.autodev = 1

lxc.kmsg = 0

lxc.mount.entry = /mnt/sock mnt/sock none rw,bind 0.0

lxc.rootfs = overlayfs:/mnt/flash/lxc\_dir/lxc0/lower:/mnt/flash/lxc\_dir/lxc0/upper

lxc.mount = /mnt/flash/lxc\_dir/lxc0/lower/etc/fstab

lxc.utsname = lxc0

lxc.network.hwaddr = 24:9e:ab:ba:60:b4

##### 连接容器

container-attach lxc0 /sbin/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

co:0 (0.0 B)

veth1 Link encap:Ethernet HWaddr 24:9e:ab:ba:60:b4

inet addr:200.1.1.253 Bcast:200.1.1.255 Mask:255.255.255.0

inet6 addr: fe80::269e:abff:feba:60b4/64 Scope:Link

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

RX packets:10 errors:0 dropped:0 overrun0 carrier:0

collisions:0 txqueuelen:1000

RX bytes:1286 (1.2 KiB) TX bytes:1152 (1.1 KiB)

##### 重定向到容器

[Huawei-lxc]container-console lxc0

[Huawei-lxc]

Connected to tty 1

Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself

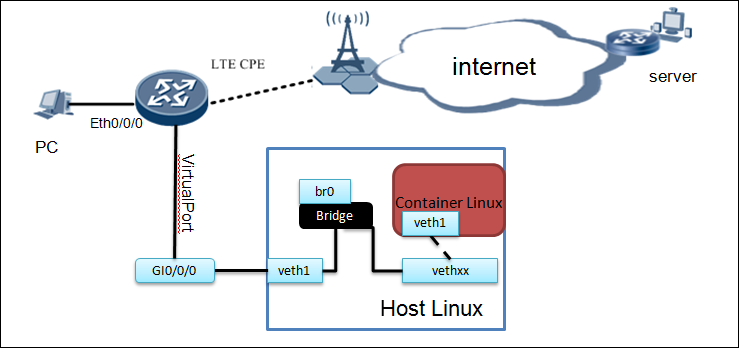
Debian GNU/Linux 8 huawei tty1

huawei login:

## 设备及容器的网络配置指导

### 组网需求

AR路由器提供VRP平台及LXC系统服务；完成linux容器系统安装后，配置设备为linux容器系统分配IP地址；PC作为私网网络用户，能够通过设备与linux容器系统通信，同时linux容器系统能够通过设备上行3G/LTE业务接入internet；



### 配置思路

1. 配置设备通过DHCP方式为linux容器系统分配IP地址
2. Linux容器与私网用户PC互访
3. 配置上行3G/LTE链路，设备cellular接口获取IP地址；
4. 私网PC用户和linux容器能够通过上行3G/LTE业务接入internet

### 操作步骤

#### 创建容器并启动

创建步骤详情可以参考1.2.5容器部署章节。

#### 配置设备为linux容器系统分配IP地址

1. 查看虚拟接口

[AR505G-U-D]display current-configuration

可以找到某个接口下有description VirtualPort字样，该接口即为虚接口。



1. 配置设备虚接口IP地址，配置虚接口下的client端从接口地址池中获取IP地址

[AR505G-U-D]int GigabitEthernet 0/0/0

[AR505G-U-D-GigabitEthernet0/0/0]ip address 200.1.1.1 255.255.255.0

[AR505G-U-D-GigabitEthernet0/0/0]dhcp select interface

[AR505G-U-D-GigabitEthernet0/0/0]dhcp server static-bind ip-address 200.1.1.253 mac-address 8a46-4349-b200

[AR505G-U-D-GigabitEthernet0/0/0]dhcp server dns-list 200.1.1.1

注：MAC地址8a46-4349-b200为容器中网口地址，通过上述配置确保容器中分配的IP地址为200.1.1.253，方便通过设备eth口进行远程访问；

1. 登陆容器，查看容器系统网络信息（容器系统默认DHCP方式获取地址）

[AR505G-U-D-diagnose]shell

[AR505G-U-D-diagnose]

INFO: Enter exit or press Ctr+D to quit.

/mnt/squash/mpu # lxc-console -n lxc01

Connected to tty 1

Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter

Ctrl+a itself

Debian GNU/Linux 8 huawei tty1

huawei login: root

Password:

Last login: Fri Jan 6 05:34:55 UTC 2017 on tty1

Linux huawei 4.1.36 #1 SMP PREEMPT Wed May 24 16:02:58 CST 2017 armv7l

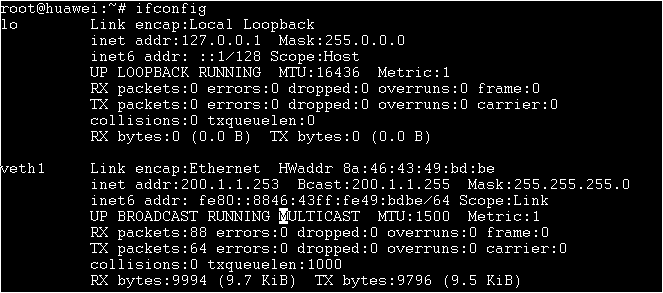
The programs included with the Debian GNU/Linux system are free software;

the exact distribution terms for each program are described in the

individual files in /usr/share/doc/\*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent

permitted by applicable law.

root@huawei:~# ifconfig 

执行Ctrl+a q命令退出linux容器系统

#### 配置设备VRP平台物理口ETH0/0/0的IP地址，同时配置私网PC以DHCP方式从设备获取IP地址；完成linux容器与私网用户PC互访

[AR505G-U-D]int Ethernet 0/0/0

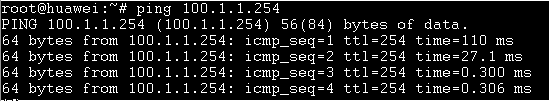
[AR505G-U-D-Ethernet0/0/0]undo portswitch

[AR505G-U-D-Ethernet0/0/0]ip address 100.1.1.1 255.255.255.0

[AR505G-U-D-Ethernet0/0/0]dhcp select interface

查看私网PC的IP地址为100.1.1.254（PC配置略）

Linux容器能够与私网网络PC互通



#### 使用SSH登录到容器

在OVA包中的容器配置文件中已经指定容器的硬件地址：



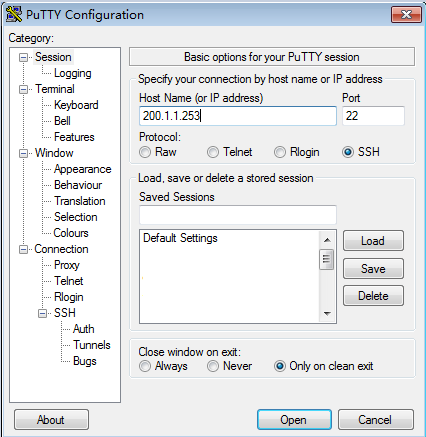
结合虚拟接口的DHCP绑定IP地址的分配策略，则容器可以获得固定的IP地址：



私网网络PC能够和Linux容器的网段互通后就可以通过SSH登录到容器。（互通方式参考1.3.3.3章节）

使用SSH工具putty登录容器

Putty配置示例：



容器缺省账号**root**和密码**root**

#### 配置设备3G/LTE链路，使私网用户通过3G/LTE业务接入internet;

[AR505G-U-D]dialer-rule

[AR505G-U-D-dialer-rule]dialer-rule 1 ip permit

[AR505G-U-D]int Cellular 0/0/0

[AR505G-U-D-Cellular0/0/0]dialer enable-circular

[AR505G-U-D-Cellular0/0/0] dialer-group 1

[AR505G-U-D-Cellular0/0/0]dialer number \*99#

[AR505G-U-D-Cellular0/0/0]apn-profile private.vpdn

[AR505G-U-D-Cellular0/0/0]dialer timer autodial 10

[AR505G-U-D-Cellular0/0/0]ip address negotiate

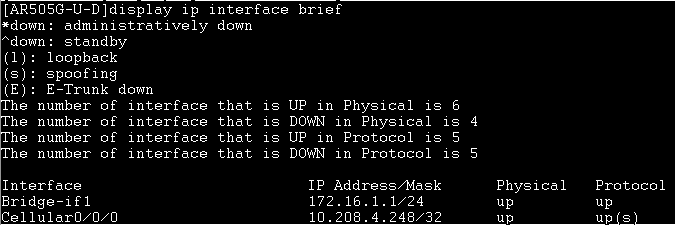
[AR505G-U-D]acl 3000

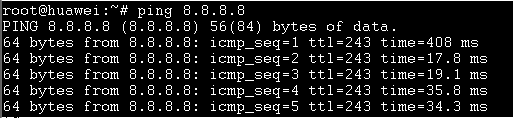
[AR505G-U-D-acl-adv-3000]rule permit ip

[AR505G-U-D-Cellular0/0/0]nat outbound 3000

[AR505G-U-D]ip route-static 0.0.0.0 0 Cellular 0/0/0

接口获取3G/LTE地址，linux容器能够通过上行3G/LTE业务接入internet





### 配置文件

### VRP配置

#

acl number 3000

rule 5 permit ip

#

interface Ethernet0/0/0

undo portswitch

ip address 100.1.1.1 255.255.255.0

dhcp select interface

#

interface GigabitEthernet0/0/0

description VirtualPort

ip address 200.1.1.1 255.255.255.0

dhcp select interface

#

interface Cellular0/0/0

dialer enable-circular

dialer-group 1

apn-profile private.vpdn

dialer timer autodial 10

dialer number \*99#

nat outbound 3000

ip address negotiate

#

dialer-rule

dialer-rule 1 ip permit

#

ip route-static 0.0.0.0 0.0.0.0 Cellular0/0/0

#

## 开发指导

### 开发步骤

按照如下操作步骤进行二次开发程序的开发与使用：

环境准备，包括AR设备和交叉编译环境；

开发，编译，调试C/C++环境；

程序启动与运行

注意：对于定制款型，容器环境在设备出厂时已经预装集成，会将可以开放的物理接口包扩但不限于RS232/RS485/DIDO/CANBUS接口在容器中进行映射，供容器中应用进行直接开发；同时也会将容器与外部通信网络默认配置好，客户只要聚焦于容器中应用的开发。

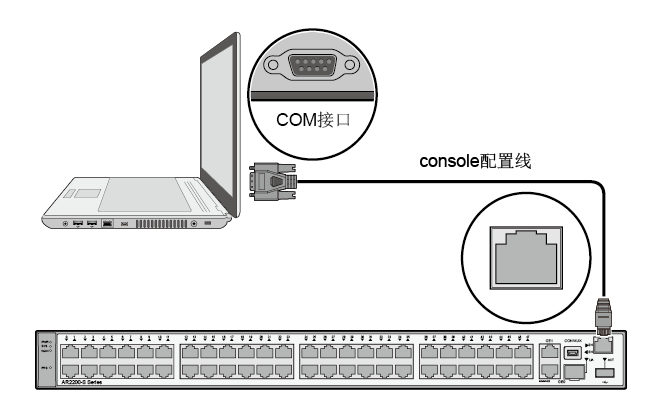
### 环境准备

#### AR路由器

##### **连接配置线缆**

连接console配置线缆（即串线）

* 将console配置线缆的一端（RJ45）连接到路由器的CON/AUX接口（RJ45）上。
* 将console配置线缆的另一端（DB9）连接到管理PC的串行接口（COM）上。

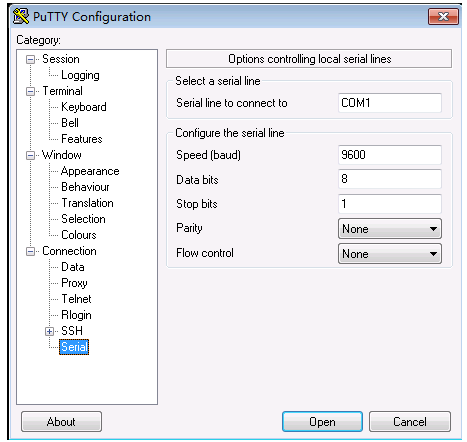


使用串口工具IPOP或PUTTY连接设备：

串口参数：

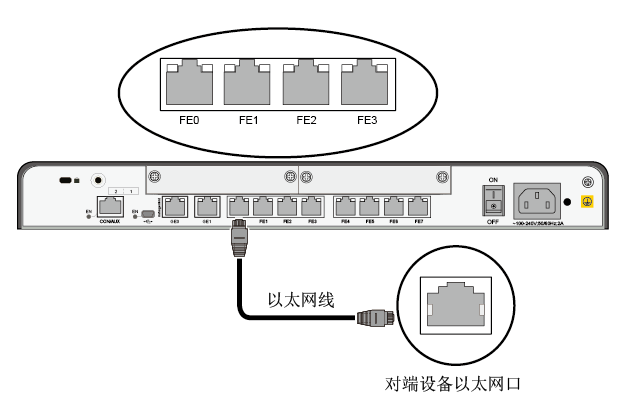
* 每秒位数：9600
* 数据位：8
* 奇偶校验：无
* 停止位：1
* 数据流控制：无

Putty配置示例：



新的AR设备缺省账号**admin**和密码**Admin@huawei**

连接以太网线：



管理口将PC的Ethernet接口与设备的WAN以太接口相连，并配置同网段的IP地址，通过telnet或stelnet对设备进行控制操作。

* Telnet配置视频教程

<http://support.huawei.com/enterprise/docinforeader.action?contentId=DOC1000088197&idAbsPath=7919710|21432787|7923148|9858988|21399251>

##### **产品手册**

更多AR详细信息参见AR产品手册：

* [V2R7C00 产品描述链接](http://support.huawei.com/enterprise/docinforeader.action?contentId=DOC1000085853&idPath=7919710|21432787|7923148|9858988|19907398)
* [V2R7C00 产品文档链接](http://support.huawei.com/ehedex/hdx.do?docid=DOC1000085851&lang=zh&idPath=7919710|21432787|7923148|9858988|19907398)

#### 交叉编译环境

##### **安装Ubuntu**

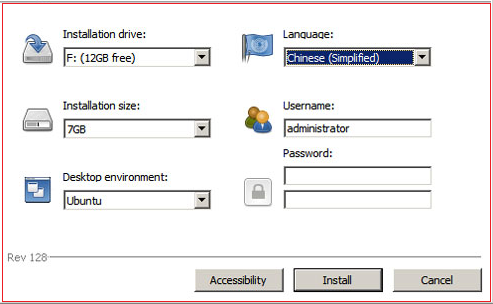
软件下载路径：<http://releases.ubuntu.com/14.04/>

如下载的iso文件为ubuntu-14.04.4-desktop-amd64.iso

可以直接用安装盘安装，也可以用下面方法安装。

在Windows下可以不用重新分区，直接像安装一个应用程序那样安装Ubuntu，安装方法是，创建一个文件夹并命名为ubuntu，将wubi.exe文件解压到这个目录中，并将ubuntu-14.04.4-desktop-amd64.iso文件放入同一个文件夹中。断网，双击运行wubi.exe。在Windows下直接安装而无需分区。

接着出现下面的安装界面，选择一个磁盘，然后将语言选择为“Chinese（Simplified）简体中文”，Installation size为Ubuntu环境的总共磁盘大小，然后是登录用户名和密码，设置好了以后就点安装继续。



然后根据提示选择，重启后进入Ubuntu安装界面。

后面的安装操作很简单，不需要手动干预就可以直接安装好整个操作系统，大部分的硬件驱动都可以自动安装好。提示安装完毕后，重启系统，就可以使用Ubuntu了。

Ubuntu 14.04安装后常见问题解决

系统安装完重启后进系统时出现”为/检查磁盘时发生严重错误“的提示。

解决方法：

重启，进入紫色的grub启动器画面。按键盘上e键，进入编辑模式。找到以“linux”开头的那行，将其中的“ro rootflags=sync”，改为“rw rootflags=sync”。修改完毕后再按F10启动。

启动后打开终端，修改/etc/grub.d/10\_lupin文件的第150行；将ro修改为rw；





修改后



然后再运行sudo update-grub命令更新grub；



GNOME下终端打开时提示“创建此终端的子进程时出错“，无法使用terminal。

解决方法：

切换到其他用户，如按下<Ctrl+Alt+F1>进入login shell，然后打开/etc/fstab文件，追加以下几行：

none /proc proc defaults 0 0

none /dev/pts devpts gid=5,mode=620 0 0

none /dev/shm tmpfs defaults 0 0

##### 安装并启动Docker

安装完Ubuntu后，首先登录系统，并进行网络配置，连接网络；

更新系统软件仓库列表

sudo apt-get update

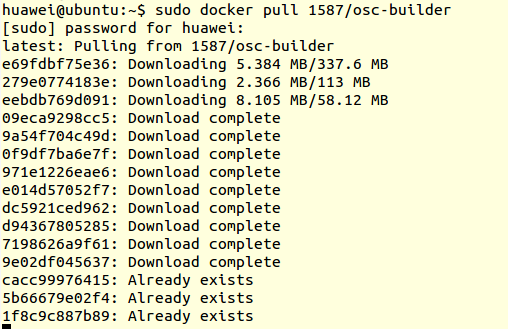
安装docker.io以及qemu等必要的软件

sudo apt-get install docker.io

安装编译环境镜像

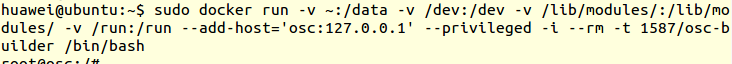
下载osc-builder镜像，该过程需要花费较长的时间，请耐性等待，如果失败请重试该条命令

sudo docker pull 1587/osc-builder

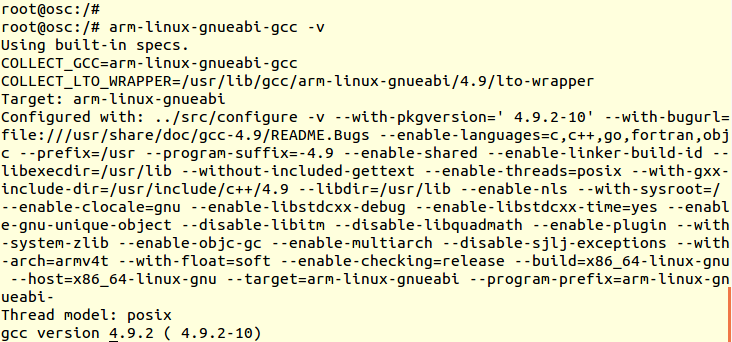


启动docker镜像

sudo docker run –v ~:/data –v /dev:/dev –v /lib/modules/:/lib/modules/ -v /run:/run -–add-host=’osc:127.0.0.1’ -–privileged –i –-rm –t 1587/osc-builder /bin/bash



检查arm-linux-gnueabi-gcc编译工具是否已经存在



后面就可以使用arm-linux-gnueabi-XXX工具链进行编译开发，将可执行程序拷贝到设备容器中执行即可。

##### 在Docker中访问Ubuntu用户的资源

启动Docker时可以通过-v选项，将Ubuntu宿主用户的目录挂载到Docker对应的目录中；如 –v ~:/data选项就可以把Ubuntu当前用户的HOME目录挂载到Docker的/data目录；

如，在Ubuntu的huawei用户的宿主目录创建了docker目录，并编写了“helloworld”的C源程序文件；



可以在启动docker时加上–v ~:/data选项，在进入docker后就可以到/data目录访问该资源。



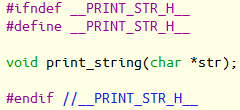
### Hello world(example)

#### 二进制程序开发

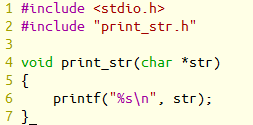
分别编写.h和 .c文件以及Makefile文件



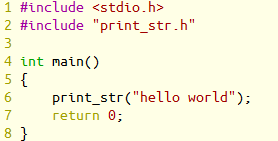
以下是print\_str.h文件



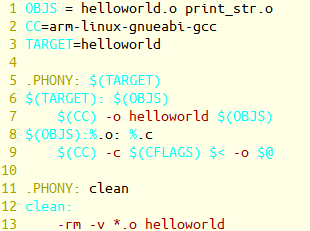
以下是print\_str.c文件



以下是helloworld.c文件



下面是Makefile文件



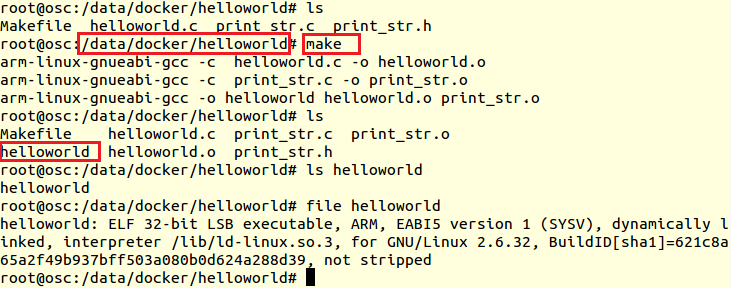
#### 程序编译

使用arm-linux-gnueabi-gcc / g++工具链编译源文件，上面的Makefile中已经制定编译器为arm-linux-gnueabi-gcc编译器。

helloworld.c Makefile print\_str.c print\_str.h这些文件都存在于Ubuntu 1404的huawei用户的~/docker/helloworld/目录；



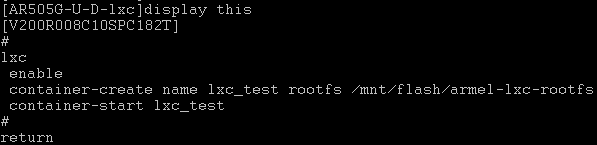
启动Docker后切换到/data目录，并进入到docker/helloworld/目录，执行make命令，即可编译出helloworld可执行文件；



#### 程序安装与运行

1.下载helloworld到容器中，确定容器已经启动，并登录容器，确保容器与ftp服务器网络互通。

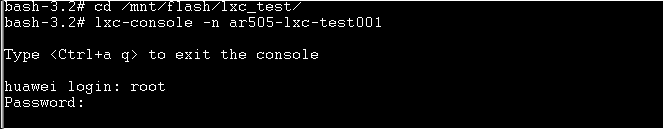
[AR505G-U-D-lxc]display this



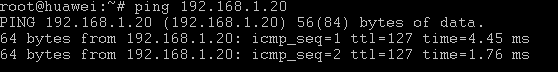
[AR505G-U-D-lxc]container-console lxc\_test

huawei login: root

password:root



root@huawei:~# ping 192.168.1.20

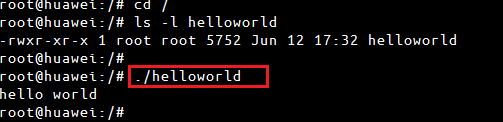


使用wget直接下载helloworld

root@huawei:~# wget -q -P / <ftp://user:pass@192.168.1.20:21/helloworld>

其中/ 是存放下载文件的位置，这里将下载的文件存放到根目录下，user和pass分别是ftp服务器的用户名和密码，192.168.1.20是ftp服务器的ip地址。

查看下载的文件并运行



### Linux socket (example)

#### 二进制程序开发

本程序是一个简单的socket编程的实例，有服务端和客户端程序组成；

在server目录下有Makefile server.c两个文件；client目录有Makefile client.c两个文件；

在server端按照标准的socket编程步骤：create socket、bind、listen、accept、read、write；

Client端按照：create socket、connect、write、read；

参考代码如下：

Server.c参考代码

/\*

\* server

\* \*/

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <errno.h>

int sockfd;

char\* IP = "127.0.0.1";

short PORT = 10111;

typedef struct sockaddr SA;

char dir\_name[20] = "files";

char rcvmsg[256] = { 0 }; //recieve client message

/\* 1 server init network \*/

void init()

{

printf("Server starting...\n");

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd == -1) {

perror("socket");

exit(-1);

}

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(PORT);

addr.sin\_addr.s\_addr = inet\_addr(IP);

/\* solve "address already in use" issue \*/

int reuseaddr = 1;

setsockopt(sockfd, SOL\_SOCKET, SO\_REUSEADDR, &reuseaddr,

sizeof(reuseaddr));

if (bind(sockfd, (SA \*) & addr, sizeof(addr)) == -1) {

perror("bind");

exit(-1);

}

if (listen(sockfd, 100) == -1) {

perror("listen");

exit(-1);

}

printf("Server init OK!\n");

}

/\* recieve client message \*/

void rcv\_proc(int sfd)

{

int size = 0;

char buf[512] = { };

while (size = read(sfd, buf, sizeof(buf))) {

printf("Client %d say>: %s", sfd, buf);

if (!write(sfd, buf, strlen(buf) + 1)) {

perror("write");

}

memset(buf, 0, 100);

}

printf("%d has disconnect!\n", sfd);

}

/\* connect with clients \*/

void service()

{

printf("Server has started!\n");

while (1) {

struct sockaddr\_in fromaddr;

socklen\_t len = sizeof(fromaddr);

printf("\nWaitint for clients to connect...\n");

int fd = accept(sockfd, (SA \*) & fromaddr, &len);

if (fd == -1) {

printf("Client connect failed!\n");

continue;

} else {

printf("Has connect to client: %d\n", fd);

}

rcv\_proc(fd);

}

}

void sig\_close()

{

close(sockfd);

printf("Server is going down...\n");

exit(0);

}

int main()

{

signal(SIGINT, sig\_close);

printf("Press <Ctrl+C> to stop server.\n");

init();

service();

return 0;

}

Client.c参考代码

/\*

\* client

\* \*/

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <signal.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int sockfd;

char\* IP="127.0.0.1";

short PORT=10111;

typedef struct sockaddr SA;

/\* 1 start client, connect to server \*/

void init()

{

printf("client starting...\n");

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

struct sockaddr\_in addr;

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(PORT);

addr.sin\_addr.s\_addr = inet\_addr(IP);

if(connect(sockfd, (SA\*)&addr, sizeof(addr)) == -1){

perror("Can not connect to server.");

printf("Client start failed!\n");

exit(-1);

}

printf("Client start succeeded!\n");

printf("Type <bye> to exit.\n");

}

int get\_user\_input(char \*str, int size)

{

printf("Client>: ");

fgets(str, size, stdin);

return strlen(str);

}

/\* 2 communication \*/

void start()

{

int size = 0;

char msg[100] = {};

char str[100] = {};

do {

memset(msg, 0, sizeof(msg));

size = get\_user\_input(msg, sizeof(msg));

printf("Send message to Server: %s", msg);

write(sockfd, msg, size + 1);

read(sockfd, str, sizeof(msg));

printf("Server return message: %s", str);

if (!strcmp(msg, "bye\n"))

{

exit(0);

}

} while(1);

close(sockfd);

}

int main()

{

init();

start();

return 0;

}

Server目录下的Makefile参考代码

CC=arm-linux-gnueabi-gcc

OBJS=server

CFLAGS=-o

SRC=server.c

OBJS: $(SRC)

$(CC) $(SRC) $(CFLAGS) $(OBJS)

clean:

-rm -rv $(OBJS)

Client目录的Makefile参考代码

CC= arm-linux-gnueabi-gcc

OBJS=client

CFLAGS=-o

SRC=client.c

OBJS: $(SRC)

$(CC) $(SRC) $(CFLAGS) $(OBJS)

clean:

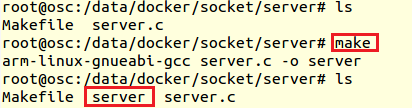
-rm -vf $(OBJS)

#### 程序编译

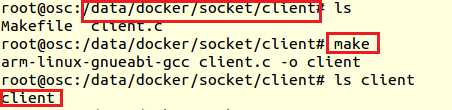
使用arm-linux-gnueabi-gcc / g++工具链编译源文件

以上示例代码的server.c和对应的Makefile位于~/docker/socket/server/目录下，client.c和对应的Makefile位于~/docker/socket/client/目录下；

启动Docker后切换到/data目录，并进入到docker/socket/目录；进入server目录执行make命令，即可编译出server可执行文件；



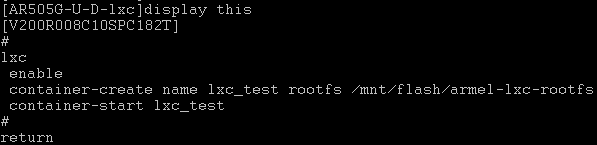
进入client目录执行make命令，即可编译出client可执行文件；



#### 程序安装

1.下载程序到容器中，确定容器已经启动，并登录容器，确保容器与ftp服务器网络互通。

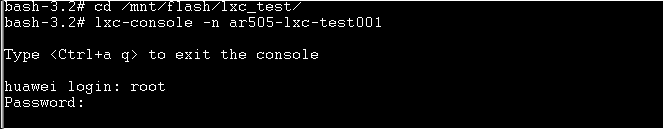
[AR505G-U-D-lxc]display this



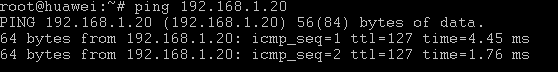
[AR505G-U-D-lxc]container-console lxc\_test

huawei login: root

password:root



root@huawei:~# ping 192.168.1.20



使用wget直接下载server和client

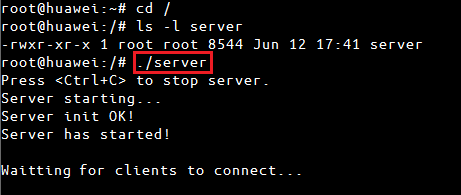
root@huawei:~# wget -q -P / <ftp://user:pass@192.168.1.20:21/server>

root@huawei:~# wget -q -P / <ftp://user:pass@192.168.1.20:21/client>

其中/ 是存放下载文件的位置，这里将下载的文件存放到根目录下，user和pass分别是ftp服务器的用户名和密码，192.168.1.20是ftp服务器的ip地址。

查看下载的文件并运行

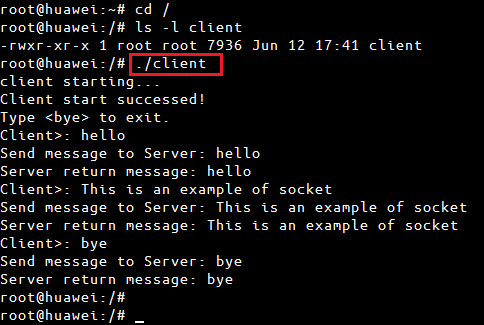
2.进入存放server的路径，通过./执行APP程序进行调试。



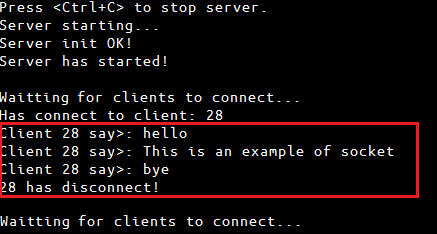
3.在另一个终端上使用telnet方式登录设备，并使用container-attach命令进入容器，然后依照上面的步骤进入存放client的路径，通过./执行APP程序进行调试。

[AR505G-U-D-lxc] container-attach lxc\_test bash

root@huawei:/#



4.再回到刚才启动server的终端，可以看到server端与client交互的信息；



## 附件



# 应用案例

## 迅达电梯



迅达电梯采用基于边缘计算平台架构的AR IOT网关，结合GE Predix平台、Schindler Ahead平台实现电梯的预测性维护功能，降低电梯维护成本，并提供相关增值业务，给乘客提供最安全与智能化的电梯服务。

|  |  |
| --- | --- |
| 边缘计算网关 | 1负责物理连接电梯控制器，采集数据；  2 支持开放平台，将RS232/RS485/CANBUS/DI/DO接口开放给迅达APP使用，迅达APP实现对电梯数据的采集：  a支持部署GE Predix软件连接到GE的Edge Manger平台  b 支持部署迅达应用连接迅达的 Schindler Ahead平台  3 支持被华为Agile Controller远程管理； |
| Agile Controller | 负责华为网关的管理，设备状态监控； |
| GE Edge Manager | 负责迅达应用的管理； |
| Schindler Ahead | 迅达平台，负责电梯数据分析与状态监控； |

# 技术支持

AR物联网关资料地址:

<http://support.huawei.com/enterprise/zh/universal-service-router/ar500-pid-21247181>

AR开放平台二次开发指南维护人：

[paul.sunbin@huawei.com](mailto:paul.sunbin@huawei.com) 孙斌

[stan.hujian@huawei.com](mailto:stan.hujian@huawei.com) 胡剑