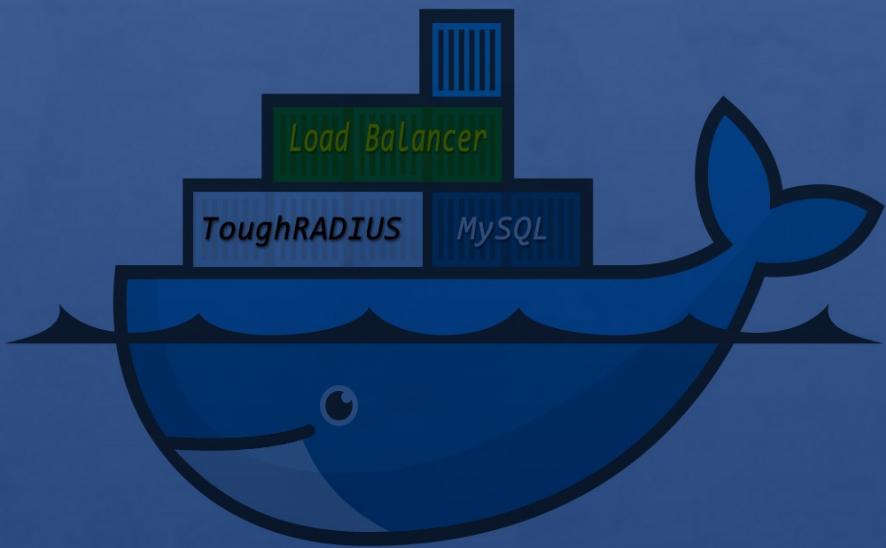


ToughRADIUS

Book



docker

docs.toughradius.net

Table of Contents

Introduction	0
ToughRADIUS 用户手册	1
项目简介	1.1
快速指南 (V1)	1.2
快速指南 (V2)(推荐)	1.3
API接口 (V2)(推荐)	1.4
数据结构定义	1.4.1
返回结果码定义	1.4.2
区域信息管理	1.4.3
区域节点查询	1.4.3.1
NAS接入设备管理	1.4.4
新增NAS接入设备	1.4.4.1
资费套餐管理	1.4.5
资费套餐查询	1.4.5.1
客户管理	1.4.6
客户信息查询	1.4.6.1
客户授权校验	1.4.6.2
客户新开户	1.4.6.3
客户资料修改	1.4.6.4
客户资料删除	1.4.6.5
客户账号删除	1.4.6.6
高级技巧	1.5
系统服务设置	1.5.1
服务端口映射	1.5.2
用户强制下线	1.5.3
使用ToughMySQL(推荐)	1.5.4
客户端691错误诊断	1.5.5
RouterOS 对接	1.6

Linux PPTP 对接	1.7
开发指南 (V1)	1.8
定制你的ToughRADIUS专有版本	1.8.1
mschapv2的认证实现	1.8.2

ToughRADIUS 教程



欢迎阅读ToughRADIUS用户文档。

及时向我们反馈问题是对我们最大的支持。

向我们发送邮件：support@toughstruct.com

加入我们的QQ交流群组：464025428, 247860313

访问我们的网站：<http://www.toughstruct.com>

ToughRADIUS 项目地址

Github 项目地址：<https://github.com/talkincode/ToughRADIUS>

Github 文档地址：<https://github.com/talkincode/ToughRADIUS-GitBook>

感谢

感谢 ToughRADIUS开发团队的家人，没有你们在背后默默的支持，我们不可能顺利的前进。

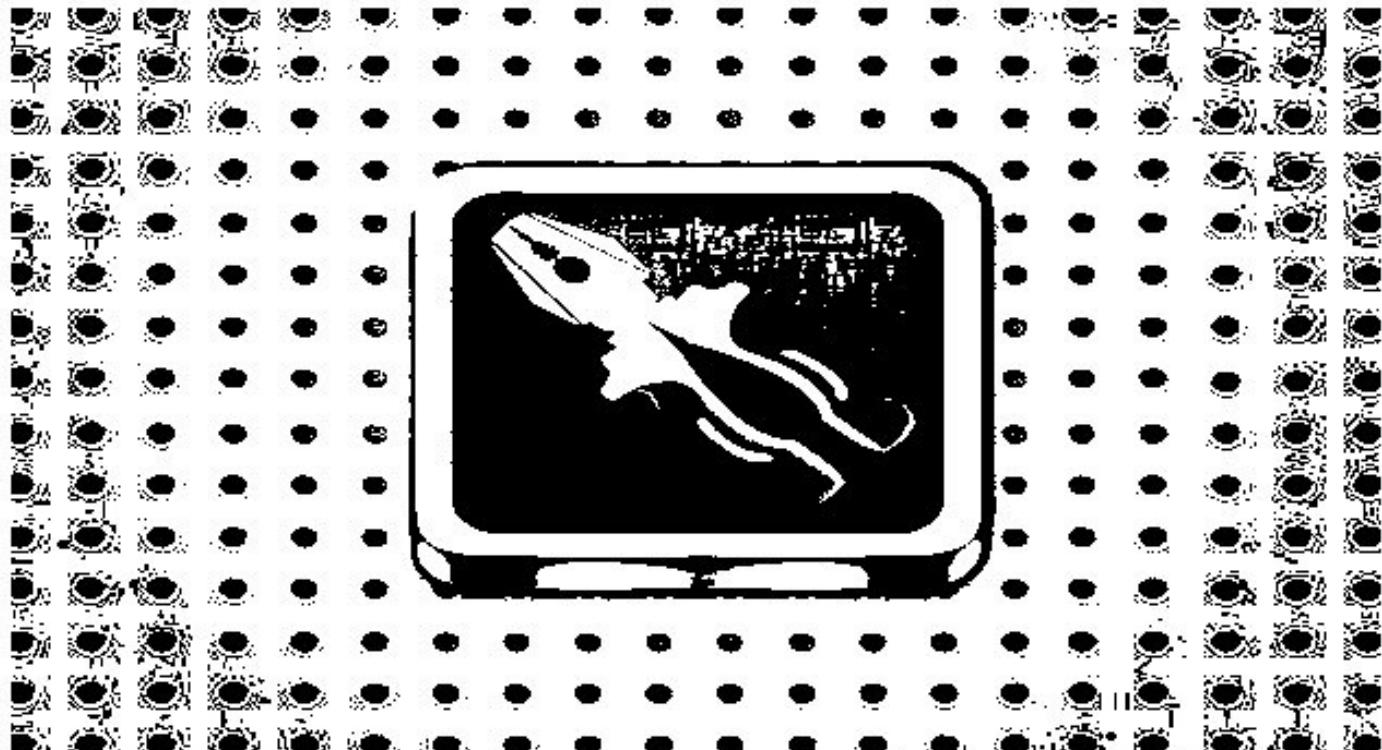
感谢 ToughRADIUS 用户以及各位友人的支持，如果没有你们的支持和反馈，ToughRADIUS不可能这么快速的推进。

感谢 ToughRADIUS 开发团队的默默奉献，我们是一个棒棒哒的团队。

这是最简单实用的 ToughRADIUS 说明书，面向一般使用者，以及高级使用者。

我们倡导简单，使用，我们不再会教你用传统的方式去折腾lamp(linux + apache+mysql+python)或者lnmp (linux + nginx+mysql+python)。

ToughRADIUS的用户应该将注意力集中到如何使用，而不是淹没在各种技术细节中慢慢绝望。



TOUGHRADIUS 项目介绍

ImageLayers.io 803 MB / 53 Layers

ToughRADIUS是一个开源的Radius服务软件，采用于AGPL许可协议发布，从创立之日起，他的宗旨就是服务于中小微ISP，让运营变得更简单。

TOUGHRADIUS支持标准RADIUS协议（RFC 2865, RFC 2866），提供完整的AAA实现。支持灵活的策略管理，支持各种主流接入设备并轻松扩展，具备丰富的计费策略支持。

TOUGHRADIUS支持使用Oracle, MySQL, PostgreSQL, MSSQL等主流数据库存储用户数据，并支持数据缓存，极大的提高了性能。

TOUGHRADIUS支持Windows, Linux, BSD跨平台部署，部署使用简单。

TOUGHRADIUS提供了RADIUS核心服务引擎与Web管理控制台，以及可扩展的API。

TOUGHRADIUS 功能特性

- 标准Radius认证记账支持，提供完整的AAA实现。
- 支持pap, chap, mschap-v2验证。
- 提供基于WEB的管理控制台界面。
- 提供基于WEB的自助服务系统，支持界面定制。
- 基于Python Twisted高性能异步网络框架开发的认证计费引擎。
- Docker支持，支持Windows, Linux, BSD跨平台部署，部署使用简单。
- 支持各种主流接入设备(RouterOS,思科, 华为, 爱立信, 中兴, 阿尔卡特, H3C等)并轻松扩展，支持多设备接入管理。
- 支持使用Oracle, MySQL, PostgreSQL, MSSQL等主流数据库存储数据，并支持高速数据缓存。
- 支持预付费时长，预付费流量，预付费包月，买断包月，买断时长，买断流量资费策略。
- 支持会话时长定制。
- 支持数据库定时备份，支持FTP远程备份。
- 支持用户在线查询，解锁，批量解锁，强制下线。
- 支持用户在线统计，流量统计。
- 支持WEB界面上网日志查询。
- 支持灵活的授权策略扩展。
- 支持操作员权限分级管理。
- 支持第三方支付在线充值续费。
- 支持用户数据，财务数据，记账数据导出管理。

- 支持批量用户导入开户。
- 支持在线实时开通账号使用。
- 支持COA强制下线功能。
- 支持实时记账扣费。
- 支持全局与资费级别的自定义记账间隔下发
- 支持不同类型设备自动限速适配。
- 支持账号到期自动下线。
- 支持到期特定地址池下发。
- 支持到期提前通知，通过邮件和webhook触发实现。
- 详细的操作日志记录，条件查询。

系统架构

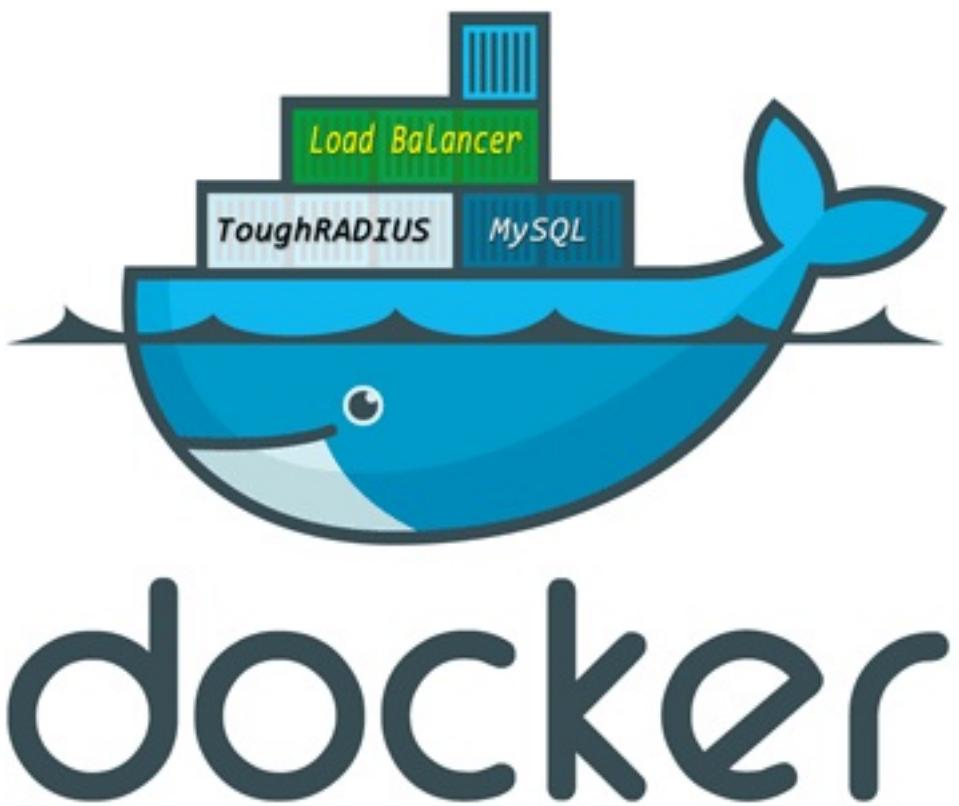


ToughRADIUS快速指南

准备

一台完整的服务器，或者远程VPS，给服务器安装Linux系统，CentOS6以上，ubuntu13以上，或者其他你自己熟悉的Linux发行版。

你要懂一点技术，比如安装操作系统，会在终端敲命令。



ToughRADIUS 是Docker技术的拥抱者，当你决定使用ToughRADIUS，你也需要去学习关于Docker的知识。

安装部署

ToughRADIUS主要采用了Docker镜像部署的模式， ToughRADIUS的镜像基础是CentOS 7。

我们可以把Docker看作一个软件集装箱， 半世纪之前， 集装箱发挥了巨大的力量， 改变了整个运输产业， 也改变了人们的生活。而Docker就类似这样一个集装箱工具， 只不过他封装的是软件。

还记得linux安装lamp的经历吗？现在可以对各种安装配置apache, php等繁琐的工作说再见了。

我们把ToughRADIUS相关的配置， 运行依赖环境等全部打包在一个“Docker集装箱”里， 我们只需要在我们的服务器上简单的安装一个支持运行“Docker集装箱”的环境， 那么我们不用去折腾各种运行环境搭建就能简单的让ToughRADIUS跑起来。

通常我们把封装了软件应用的“Docker集装箱”叫做镜像， 有点类似你可能了解的ISO文件。

Docker 安装

CentOS 6

```
$ sudo yum install http://mirrors.yun-idc.com/epel/6/i386/epel-release-6-8.noarch.rpm  
$ sudo yum install docker-io  
$ sudo service docker start
```

CentOS 7

```
$ sudo yum install docker  
$ sudo service docker start
```

Ubuntu

```
$ sudo apt-get update  
$ sudo apt-get install docker.io  
$ sudo ln -sf /usr/bin/docker.io /usr/local/bin/docker  
$ sudo sed -i '$acomplete -F _docker docker' /etc/bash_completion.d/docker.io
```

Windows

请下载Windows安装文件：<https://github.com/boot2docker/windows-installer/releases/download/v1.8.0/docker-install.exe>

注意：Window下的Docker服务是借助虚拟机来实现的，性能上远不如使用linux主机。

Docker 容器创建

确认Docker服务已经运行，通过以下指令创建ToughRADIUS的容器实例，当看到一串hash打印时，说明容器创建成功了。

```
$ docker run -d --name trserver --net host index.alauda.cn/toughstruct/toughradius
```

这样我们的服务就已经运行了。我们可以通过浏览器来访问我们的应用了。

营业管理：<http://ipaddr:1816> 管理权限 admin/root

自助服务：<http://ipaddr:1817>

系统管理：<http://ipaddr:1819> 管理权限 ctlman/ctlroot

防火墙设置

注意：如果访问不了web，可能是防火墙禁止了相关端口，如果不打算用内置防火墙，可以关闭防火墙。

```
systemctl stop firewalld.service
```

禁止firewall开机启动，防火墙就永久性关闭了。

```
systemctl disable firewalld.service
```

容器的基本管理

启动容器

```
$ docker start trserver
```

停止容器

```
$ docker stop trserver
```

重启容器

```
$ docker restart trserver
```

查看容器日志，*Ctrl+c*退出

```
$ docker logs trserver
```

docker仓库列表

国际docker大本营：[talkincode/toughradius](#)

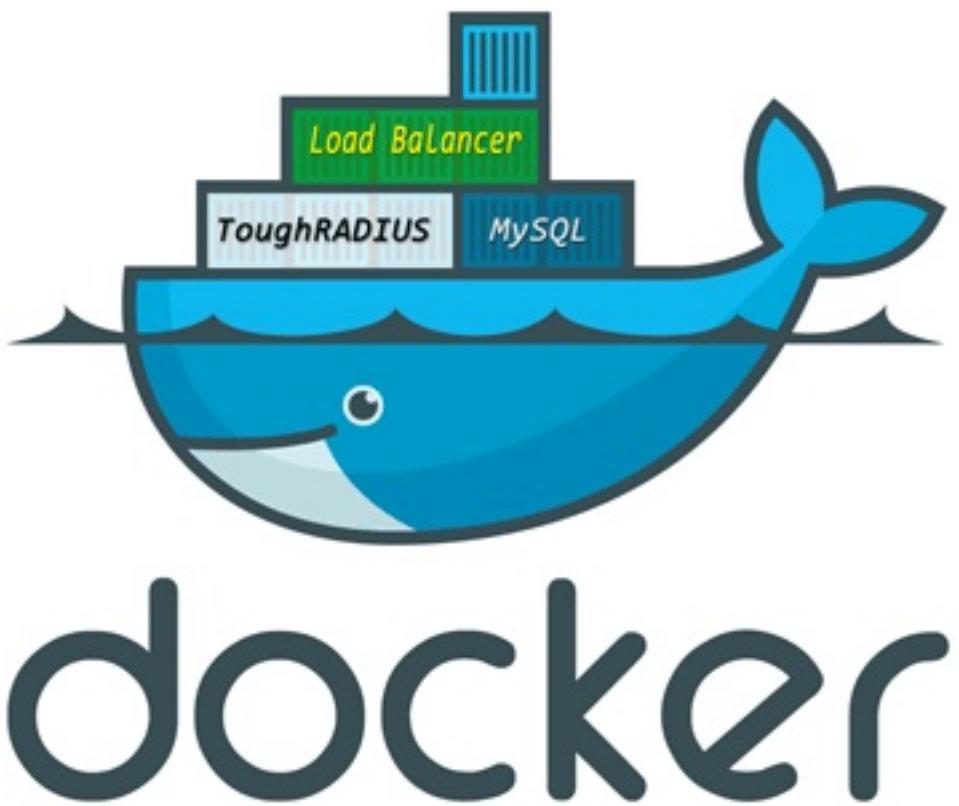
国内灵雀云：[index.alauda.cn/toughstruct/toughradius](#)

ToughRADIUS快速指南

准备

一台完整的服务器，或者远程VPS，给服务器安装Linux系统，CentOS6以上，ubuntu13以上，或者其他你自己熟悉的Linux发行版。

你要懂一点技术，比如安装操作系统，会在终端敲命令。



ToughRADIUS 是Docker技术的拥抱者，如果想更好的使用ToughRADIUS，你也需要去学习关于Docker的知识。

安装部署

ToughRADIUS主要采用了Docker镜像部署的模式， ToughRADIUS的镜像基础是ubuntu 14。

我们可以把Docker看作一个软件集装箱， 半世纪之前， 集装箱发挥了巨大的力量， 改变了整个运输产业， 也改变了人们的生活。而Docker就类似这样一个集装箱工具， 只不过他封装的是软件。

还记得linux安装lamp的经历吗？现在可以对各种安装配置apache， php等繁琐的工作说再见了。

我们把ToughRADIUS相关的配置， 运行依赖环境等全部打包在一个“Docker集装箱”里， 我们只需要在我们的服务器上简单的安装一个支持运行“Docker集装箱”的环境， 那么我们不用去折腾各种运行环境搭建就能简单的让ToughRADIUS跑起来。

通常我们把封装了软件应用的“Docker集装箱”叫做镜像， 有点类似你可能了解的ISO文件。

下载Linux专用脚本工具

```
wget https://raw.githubusercontent.com/talkincode/ToughRADIUS/master/scripts/trshell -O /usr/local/bin/trshell  
chmod +x /usr/local/bin/trshell
```

看看这个工具为我们提供了那些功能

```
trshell help  
  
Usage: /usr/local/bin/trshell [OPTIONS] instance  
  
docker_setup           install docker, docker-compose  
pull                  toughradius docker images pull  
install               install toughradius with already exists mysql  
install_with_lmysql    install toughradius with local docker mysql instance  
install_with_rmysql    install toughradius with remote mysql server  
remove                uninstall toughradius and database  
config                toughradius instance config edit  
status                toughradius instance status  
restart               toughradius instance restart  
stop                  toughradius instance stop  
upgrade              toughradius instance upgrade  
logs                 toughradius instance logs  
dsh                  toughradius instance bash term  
  
All other options are passed to the toughrad program.
```

Docker环境安装

我们首先应该安装配置服务器的Docker运行环境，以下指令会自动根据当前linux版本下载对应的docker版本进行自动安装。

```
trshell docker_setup
```

ToughRADIUS 应用实例创建

注意，trshell创建容器指令需要交互式完成，请根据提示进行输入操作

一键部署 TOUGHRADIUS， 默认使用sqlite数据库

```
$ trshell install t1      # t1表示实例名，可自定义，如果服务器只有一个实例，可以为空
```

一键部署TOUGHRADIUS，连接已有的远程MySQL数据库

```
$ trshell install_with_rmymysql t1      # t1表示实例名，可自定义，如果服务器只有一个实例，可以为空
```

一键部署TOUGHRADIUS，创建一个本地MySQL实例并连接它。

```
$ trshell install_with_lmysql t1      # t1表示实例名，可自定义，如果服务器只有一个实例，可以为空
```

应用管理

这样我们的服务就已经运行了。我们可以通过浏览器来访问我们的应用了。

营业管理：<http://ipaddr:1816> 管理权限 admin/root

防火墙设置

注意：如果访问不了web，可能是防火墙禁止了相关端口，如果不打算用内置防火墙，可以关闭防火墙。

```
systemctl stop firewalld.service
```

禁止firewall开机启动，防火墙就永久性关闭了。

```
systemctl disable firewalld.service
```


ToughRADIUS接口协议文档

协议说明

- 协议采用基于http的接口调用方式。
- 消息发起方：第三方系统。
- 消息接受方：TOUGHRADIUS系统。
- 接口请求方法：HTTP GET 或者 POST，支持标准http1.1协议。
- 消息报文编码：UTF-8，注意对参数进行 urlencode 编码
- 接口鉴权：MD5签名，`md5(共享密钥+排序的参数值组合字符串)`, 签名校验是双向的，消息接收方对请求消息进行签名验证，同时需对结果进行签名返回，消息发起方对响应结果的签名进行验证。
- 当前接口版本：v1

API应用场景

- **业务功能扩展：** ToughRADIUS V2版本提供了一个精简的管理系统，但是对于一些客户比较复杂的业务需求会显得不够用，利用API可以实现业务功能的扩展，通过已有CRM等管理系统与ToughRADIUS实现对接。
- **自助服务系统定制：** ToughRADIUS V2版本移除了自助服务模块，我们希望把这个模块提供給第三方来实现。
- **移动终端APP开发：** 对于ToughRADIUS V2版本，无论是管理，自服务都是可以扩展到移动终端的，通过API可以支持IOS，Android，微信公众号开发扩展。

示例

```
GET /api/v1/nas/add?sign=C5219018DC9A71232A88606E48554F50&isp_code=default&bas_name=api%20add&i
```

将所有参数的值进行排序相加得到排序的参数值字符串，为了安全，可以增加一个随机参数值，比如nonce=123131414,保证nonce每次都不同，nonce的值参与签名计算。

除了直接GET提交，也可以通过Form表单提交

签名代码参考

- python

请参考：<https://github.com/talkincode/toughlib/blob/master/toughlib/apiutils.py>

```
#!/usr/bin/env python
#coding:utf-8
import logging
from hashlib import md5

def make_sign(api_secret, params=[]):
    """
    >>> make_sign("123456", [1, '2', u'中文'])
    '33C9065427EECA3490C5642C99165145'
    """
    _params = [p for p in params if p is not None]
    _params.sort()
    _params.insert(0, api_secret)
    strs = ''.join(_params)
    mds = md5(strs.encode('utf-8')).hexdigest()
    return mds.upper()

def check_sign(api_secret, msg):
    """
    >>> check_sign("123456", dict(code=1, s='2', msg=u'中文', sign='33C9065427EECA3490C5642C99165145'))
    True
    """
    if "sign" not in msg:
        return False
    sign = msg['sign']
    params = [msg[k].encode('utf-8') for k in msg if k != 'sign']
    local_sign = make_sign(api_secret, params)
    result = (sign == local_sign)
    if not result:
        logging.error("check_sign failure, sign:%s != local_sign:%s" %(sign, local_sign))
    return result
```

数据结构定义

区域

属性	类型 (长度)	可否为空	描述
id	INTEGER	False	区域编号
node_name	VARCHAR(32)	False	区域名
node_desc	VARCHAR(64)	False	区域描述

BAS设备

属性	类型 (长度)	可否为空	描述
id	INTEGER	False	设备id
dns_name	VARCHAR(128)	True	DNS名称
vendor_id	VARCHAR(32)	False	厂商标识
ip_addr	VARCHAR(15)	True	IP地址
bas_name	VARCHAR(64)	False	bas名称
bas_secret	VARCHAR(64)	False	共享密钥
coa_port	INTEGER	False	CoA端口
time_type	SMALLINT	False	时区类型

资费套餐

- 资费类型 product_policy 0 预付费包月 1 预付费时长 2 买断包月 3 买断时长 4 预付费流量 5 买断流量
- 销售状态 product_status 0 正常 1 停用 资费停用后不允许再订购

属性	类型 (长度)	可否为空	描述
id	INTEGER	False	资费id
product_name	VARCHAR(64)	False	资费名称
product_policy	INTEGER	False	资费策略
product_status	SMALLINT	False	资费状态
bind_mac	SMALLINT	False	是否绑定mac
bind_vlan	SMALLINT	False	是否绑定vlan
concur_number	INTEGER	False	并发数
fee_months	INTEGER	True	买断授权月数
fee_times	INTEGER	True	买断时长(秒)
fee_flows	INTEGER	True	买断流量(kb)
fee_price	INTEGER	False	资费价格
input_max_limit	INTEGER	False	上行速率
output_max_limit	INTEGER	False	下行速率
create_time	VARCHAR(19)	False	创建时间
update_time	VARCHAR(19)	False	更新时间

客户信息

属性	类型 (长度)	可否为空	描述
customer_id	INTEGER	False	用户id
node_id	INTEGER	False	区域id
customer_name	VARCHAR(64)	False	用户登录名
password	VARCHAR(128)	False	用户登录密码
realname	VARCHAR(64)	False	用户真实姓名
idcard	VARCHAR(32)	True	用户证件号码
sex	SMALLINT	True	用户性别0/1
age	INTEGER	True	用户年龄
email	VARCHAR(255)	True	用户邮箱
mobile	VARCHAR(16)	True	用户手机
address	VARCHAR(255)	True	用户地址
customer_desc	VARCHAR(255)	True	用户描述
create_time	VARCHAR(19)	False	创建时间
update_time	VARCHAR(19)	False	更新时间

客户账号

- 认证账号表，每个会员可以同时拥有多个账号
- account_number 为每个套餐对应的上网账号，每个账号全局唯一
- 用户状态 0:"预定",1:"正常", 2:"停机" , 3:"销户", 4:"到期"

属性	类型 (长度)	可否为空	描述
account_number	VARCHAR(32)	False	上网账号
customer_id	INTEGER	False	用户id
product_id	INTEGER	False	资费id
password	VARCHAR(128)	False	上网密码
status	INTEGER	False	用户状态
install_address	VARCHAR(128)	False	装机地址
balance	INTEGER	False	用户余额-分
time_length	INTEGER	False	用户时长-秒
flow_length	INTEGER	False	用户流量-kb
expire_date	VARCHAR(10)	False	过期时间- #####-##-##
user_concur_number	INTEGER	False	用户并发数
bind_mac	SMALLINT	False	是否绑定mac
bind_vlan	SMALLINT	False	是否绑定vlan
mac_addr	VARCHAR(17)	True	mac地址
vlan_id1	INTEGER	True	内层vlan
vlan_id2	INTEGER	True	外层vlan
ip_address	VARCHAR(15)	True	静态IP地址
last_pause	VARCHAR(19)	True	最后停机时间
account_desc	VARCHAR(255)	True	用户描述
create_time	VARCHAR(19)	False	创建时间
update_time	VARCHAR(19)	False	更新时间

返回结果码定义

返回码	描述	说明
0	success	处理成功
90001	message sign error	消息签名错误
90002	param parse error	参数解析错误
90003	message verify error	消息校验错误
90004	request timeout	消息校验错误
90005	api limit	频率限制
90006	server process failure	服务器处理失败
99999	unknow error	未知错误

区域信息管理

根据客户资料管理及授权不同，操作员对所管辖的地区进行区域划分，客户开户、移机等需要到指定的区域进行受理。一个操作员可以管理多个区域，一个区域也可以被多个操作员管理。

区域节点查询

接口描述

- 查询系统区域节点

请求URL

- `http://server:port/api/v1/node/query`

请求方式

- GET 或者 POST

示例：

```
GET /api/v1/node/query?sign=1BB40565D6C88276085719A5964BE3E0&node_id=1 HTTP/1.1
```

参数

参数名	必选	类型	说明
node_id	否	string	区域ID

返回结果

返回示例：

```
{
  "code": 0,
  "msg": "success",
  "nodes": [
    {
      "id": 1,
      "node_name": "default",
      "node_desc": "默认区域"
    }
  ],
  "nonce": "1456377457",
  "sign": "91FBD7D83B5F9069E456E3DDCF1CCF9C"
}
```

返回结果描述：

- code：返回结果码
- msg：返回消息
- nodes：区域节点列表，如果参数node_id不为空，则返回指定区域，若区域不存在，返回空。
- nonce：随机时间戳
- sign：消息签名

代码参考

- java

```
import java.io.IOException;
import org.apache.http.client.fluent.*;
public class SendRequest
{
    public static void main(String[] args) {
        sendRequest();
    }
    private static void sendRequest() {
        // node query (GET )
        try {
            // Create request
            Content content = Request.Get("http://127.0.0.1:1816/api/v1/node/query?sign=1BB4056
            // Fetch request and return content
            .execute().returnContent();
            // Print content
            System.out.println(content);
        }
        catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

- php

```
<?php
// Get cURL resource
$ch = curl_init();
// Set url
curl_setopt($ch, CURLOPT_URL, 'http://127.0.0.1:1816/api/v1/node/query?sign=1BB40565D6C88276085');
// Set method
curl_setopt($ch, CURLOPT_CUSTOMREQUEST, 'GET');
// Set options
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
// Send the request & save response to $resp
$resp = curl_exec($ch);
if(!$resp) {
    die('Error: "' . curl_error($ch) . '" - Code: ' . curl_errno($ch));
} else {
    echo "Response HTTP Status Code : " . curl_getinfo($ch, CURLINFO_HTTP_CODE);
    echo "\nResponse HTTP Body : " . $resp;
}
// Close request to clear up some resources
curl_close($ch);
```

- python

```
import requests
def send_request():
    # node query
    # GET http://127.0.0.1:1816/api/v1/node/query
    try:
        response = requests.get(
            url="http://127.0.0.1:1816/api/v1/node/query",
            params={
                "sign": "1BB40565D6C88276085719A5964BE3E0",
                "node_id": "1",
            },
        )
        print('Response HTTP Status Code: {status_code}'.format(
            status_code=response.status_code))
        print('Response HTTP Response Body: {content}'.format(
            content=response.content))
    except requests.exceptions.RequestException:
        print('HTTP Request failed')
```

- go

```
package main
import (
    "fmt"
    "io/ioutil"
    "net/http"
)
func sendNodeQuery() {
    // node query (GET http://127.0.0.1:1816/api/v1/node/query?sign=1BB40565D6C88276085719A5964
    // Create client
    client := &http.Client{}
    // Create request
    req, err := http.NewRequest("GET", "http://127.0.0.1:1816/api/v1/node/query?sign=1BB40565D6
    parseFormErr := req.ParseForm()
    if parseFormErr != nil {
        fmt.Println(parseFormErr)
    }
    // Fetch Request
    resp, err := client.Do(req)
    if err != nil {
        fmt.Println("Failure : ", err)
    }
    // Read Response Body
    respBody, _ := ioutil.ReadAll(resp.Body)
    // Display Results
    fmt.Println("response Status : ", resp.Status)
    fmt.Println("response Headers : ", resp.Header)
    fmt.Println("response Body : ", string(respBody))
}
```

NAS 接入设备管理

NAS设备信息必须在系统中登记才能和RADIUS进行通信。NAS设备对应着实际的物理接入网关设备，是具备PPPOEServer功能的路由器或交换机等设备，NAS设备以IP地址唯一标识，每个NAS与RADIUS配置对应的共享密钥。系统支持标准的RADIUS协议接入，通常大多数NAS设备可以以标准类型配置，但是对于很多厂家的NAS设备，在标准协议下并不能工作的很完美，比如MAC地址，VLAN往往以各自的私有协议封装，RADIUS为了准确的处理这些信息，必须对这些NAS设备进行类型标识，并在消息处理时进行解析。

NAS 接入设备新增

接口描述

- 增加NAS接入设备。

请求URL

- `http://server:port/api/v1/nas/add`

请求方式

- GET 或者 POST

示例：

```
GET /api/v1/nas/add?ip_addr=192.168.31.153&bas_name=test%20bas&dns_name=&bas_secret=123456&vend
```

参数

支持的设备类型

```
{  
    '0': '标准',  
    '3041': '阿尔卡特',  
    '2352': '爱立信',  
    '2011': '华为',  
    '25506': 'H3C',  
    '3902': '中兴',  
    '10055': '爱快',  
    '14988': 'RouterOS'  
}
```

参数名	必选	类型	说明
ip_addr	否	string	接入设备地址
dns_name	否	string	DNS域名
bas_name	是	string	接入设备名称
bas_secret	是	string	共享秘钥
vendor_id	是	int	接入设备类型
coa_port	是	int	授权端口 默认3799

返回结果

返回示例：

```
{
  "code": 0,
  "msg": "success",
  "nonce": "1456970950",
  "sign": "1E5AD43D7794267B1C08939B7A0871BC"
}
```

返回结果描述：

- code：返回结果码
- msg：返回消息
- nonce：随机时间戳
- sign：消息签名

资费套餐管理

资费是对计费规则的定义，决定了对每个上网账号如何收费，如何扣费的规则。

系统资费类型有以下几种：

- 预付费包月：按月定义资费单价，用户可以按需要订购月数。
- 买断包月：预付费包月的一种打包形式，将多个月打包为一个资费，例如：将12个月打包为一个年资费套餐，
- 预付费时长：按小时定义资费单价，用户只需要充值，在实际使用时周期性扣费。
- 买断时长：预付费时长的一种打包形式，一次性将时长打包为一个套餐。
- 预付费流量：按流量（MB）定义资费单价，用户只需要充值，在实际使用中按使用的流量进行扣费。
- 买断流量：预付费流量的一种打包形式，一次性将流量打包为一个套餐。

资费套餐查询

接口描述

- 查询资费套餐

请求URL

- `http://server:port/api/v1/product/query`

请求方式

- GET 或者 POST

示例：

```
GET /api/v1/product/query?sign=1BB40565D6C88276085719A5964BE3E0&product_id=1 HTTP/1.1
```

参数

参数名	必选	类型	说明
product_id	否	string	资费ID

返回结果

返回示例：

```
{  
    "code": 0,  
    "msg": "success",  
    "products": [  
        {  
            "id": 1,  
            "product_name": "测试2M包月20元",  
            "product_policy": 0,  
            "product_status": 0,  
            "bind_mac": 0,  
            "bind_vlan": 0,  
            "concur_number": 0,  
            "fee_months": 0,  
            "fee_times": 0,  
            "fee_flows": 0,  
            "fee_price": 2000,  
            "input_max_limit": 1048576,  
            "output_max_limit": 2097152,  
            "create_time": "2016-02-20 17:59:09",  
            "update_time": "2016-02-20 17:59:09"  
        }  
    ],  
    "nonce": "1456369956",  
    "sign": "AD1E70D8CEB98D4F6DDE193E7860F587"  
}
```

返回结果描述：

- code: 返回结果码
- msg: 返回消息
- products: 产品套餐列表，如果参数product_id不为空，则返回指定套餐，若套餐不存在，返回空。
- nonce: 随机时间戳
- sign: 消息签名

客户管理

客户指拥有认证账号使用权的终端客户，每个客户都可以拥有多个认证账号，每个客户的认证账号都是独立的，客户账号是由营业操作员在营业系统中受理开户。

客户信息查询

接口描述

- 查询系统客户信息，以及客户的认证账号列表。

请求URL

- `http://server:port/api/v1/customer/query`

请求方式

- GET 或者 POST

示例：

```
GET /api/v1/customer/query?sign=F1670B9A51DD7AE9882B36D32C48C89F&account_number=test01&customer_name=
```

参数

account_number 与 customer_name 必须有一个不为空

参数名	必选	类型	说明
account_number	是	string	客户认证账号
customer_name	是	string	客户账号名

返回结果

如果指定了 account_number 参数，返回的 accounts 结果集中将只会包含 account_number 的信息

返回示例：

```
{
  "code": 0,
  "msg": "success",
  "customer": {
    "customer_id": 1,
    "node_id": 1,
    "customer_name": "test01",
    "realname": "tester",
    "idcard": "4123123",
    "sex": 1,
    "age": 0,
    "email": "",
    "mobile": "13655555555",
    "address": "小区E-01栋5楼",
    "customer_desc": null,
    "create_time": "2016-03-03 09:18:20",
    "update_time": "2016-03-03 09:18:20"
  },
  "accounts": [
    {
      "account_number": "test01",
      "customer_id": 1,
      "product_id": 1,
      "group_id": null,
      "status": 1,
      "install_address": "小区E-01栋5楼",
      "balance": 0,
      "time_length": 0,
      "flow_length": 0,
      "expire_date": "2017-03-03",
      "user_concur_number": 0,
      "bind_mac": 0,
      "bind_vlan": 0,
      "mac_addr": "",
      "vlan_id1": 0,
      "vlan_id2": 0,
      "ip_address": null,
      "last_pause": null,
      "account_desc": null,
      "create_time": "2016-03-03 09:18:20",
      "update_time": "2016-03-03 09:18:20"
    }
  ],
  "nonce": "1456967907",
  "sign": "5F9FF91F9D761F0C4AD440B9CA7EEFC5"
}
```

返回结果描述：

- code: 返回结果码
- msg: 返回消息

- customer: 客户基本信息
- accounts: 客户所有认证账号。
- nonce: 随机时间戳
- sign: 消息签名

客户授权校验

接口描述

- 客户登陆校验，支持自助服务名登陆，支持上网账号登陆

请求URL

- `http://server:port/api/v1/customer/auth`

请求方式

- GET 或者 POST

示例

```
GET /api/v1/customer/auth?sign=45D0920A65AD46529FF9BCCC3CD45497&password=888888&customer_name=t
```

参数

customer_name 与 account_number 必须有一个不为空

参数名	必选	类型	说明
customer_name	是	string	客户账号名
account_number	是	string	客户认证账号
password	否	string	客户账号密码或认证账号密码

返回结果

返回示例：

```
{
  "code": 0,
  "msg": "success",
  "nonce": "1456970950",
  "sign": "1E5AD43D7794267B1C08939B7A0871BC"
}
```

返回结果描述：

- code：返回结果码
- msg：返回消息
- nonce：随机时间戳
- sign：消息签名

客户信息新增

接口描述

- 新开客户账号，同时开通默认认证账号。

请求URL

- `http://server:port/api/v1/customer/add`

请求方式

- GET 或者 POST

示例：

```
GET /api/v1/customer/add?sign=880E93FA8FA22273F24DE5C0F2BE779D&account_number=test03&customer_n
```

参数

参数名	必选	类型	说明
realname	否	string	用户名
node_id	否	int	区域id
idcard	是	string	证件号码
mobile	是	string	用户手机号码
email	是	string	用户Email
address	是	string	用户地址
customer_name	是	string	客户自助服务账号
account_number	否	string	用户认证账号
product_id	否	int	资费id
password	否	string	用户密码
begin_date	否	string	开通日期
expire_date	否	string	过期日期
balance	是	int	用户余额
time_length	是	int	用户时长
flow_length	是	int	用户流量

返回结果

返回示例：

```
{
  "code": 0,
  "msg": "success",
  "nonce": "1456970950",
  "sign": "1E5AD43D7794267B1C08939B7A0871BC"
}
```

返回结果描述：

- code：返回结果码
- msg：返回消息
- nonce：随机时间戳
- sign：消息签名

客户资料修改

接口描述

- 客户基本信息修改，包括姓名，密码，手机，邮箱等基本资料。

请求URL

- `http://server:port/api/v1/customer/update`

请求方式

- GET 或者 POST

客户信息删除

接口描述

- 删除客户资料，以及所有认证账号,相关数据

请求URL

- `http://server:port/api/v1/customer/delete`

请求方式

- GET 或者 POST

示例：

```
GET /api/v1/customer/delete?sign=EBB9DDBF180B35D7C7C8145DD13620D4&customer_name=test03 HTTP/1.1
```

参数

参数名	必选	类型	说明
customer_name	否	string	客户账号名

返回结果

返回示例：

```
{
  "code": 0,
  "msg": "success",
  "nonce": "1456973309",
  "sign": "6FDA5A8BFA9413E79F0F391783ECAD52"
}
```

返回结果描述：

- code：返回结果码
- msg：返回消息

- nonce: 随机时间戳
- sign: 消息签名

客户认证账号删除

接口描述

- 客户账号删除，删除客户账号资料及相关数据，但不删除客户信息

请求URL

- `http://server:port/api/v1/account/delete`

请求方式

- GET 或者 POST

示例：

```
GET /api/v1/account/delete?sign=EBB9DDBF180B35D7C7C8145DD13620D4&account_number=test03 HTTP/1.1
```

参数

参数名	必选	类型	说明
account_number	否	string	客户认证账号

返回结果

返回示例：

```
{
  "code": 0,
  "msg": "success",
  "nonce": "1456974137",
  "sign": "E65A4352B6DAD0576EBAD28AFCFCAA35"
}
```

返回结果描述：

- code：返回结果码
- msg：返回消息

- nonce: 随机时间戳
- sign: 消息签名

高级技巧



通过 `systemd` 设置系统服务（V1版本）

目前很多Linux的发行版已经开始采用`systemd`系统，通过`systemd`来设置ToughRADIUS的服务是一件很容易的事情。

创建文件 `/usr/lib/systemd/system/toughradius.service`

内容如下：

```
[Unit]
Description=ToughRADIUS Service
After=docker.service
Requires=docker.service

[Service]
Type=forking
RemainAfterExit=yes
ExecStart=/usr/bin/docker start trserver
ExecStop=/usr/bin/docker stop trserver
ExecReload=/usr/bin/docker restart trserver

[Install]
WantedBy=multi-user.target
```

其中`trserver`就是你的容器实例的名称。`toughradius`是服务名，通过以下指令设置开机启动开关。

```
systemctl enable toughradius
```

由于我们的Docker容器都依赖于`docker`服务本身，因此也要保证`docker`服务开启。

```
systemctl enable docker
```

ToughRADIUS V2版本设置系统服务

在ToughRADIUS V2版本中，只需要设置`docker`服务本身即可

```
systemctl enable docker
```

ToughRADIUS 是通过`docker`内置服务机制来实现自启动的。

服务端口映射(V1版本)

默认情况下，ToughRADIUS的Docker容器采用的是host模式的网络配置，注意缺省创建容器指令：

```
$ docker run -d --name trserver --net host index.alauda.cn/toughstruct/toughradius
```

其中--net host表示使用服务主机网络配置，这种模式无需任何NAT转换，这将获得最大的网络性能。如果需要灵活的改变端口，我们可以使用bridge模式，这种模式下容器将会创建虚拟的网络配置，并与主机网络进行桥接，我们需要将容器的端口映射到主机端口上：

```
$ docker run -d --name trserver -p 1812:1812/udp -p 1813:1813/udp -p 80:1816 -p 1815:1815 -p 1
```

如上所示，我们将主机端口与容器端口进行一对一映射，上面将主机的80端口映射到容器的管理控制台默认1816端口上。容器本身服务的端口保持不变，但主机映射端口可以灵活变化。

我们接着访问<http://主机地址>即可访问ToughRADIUS管理控制台。

服务端口映射(V2版本)

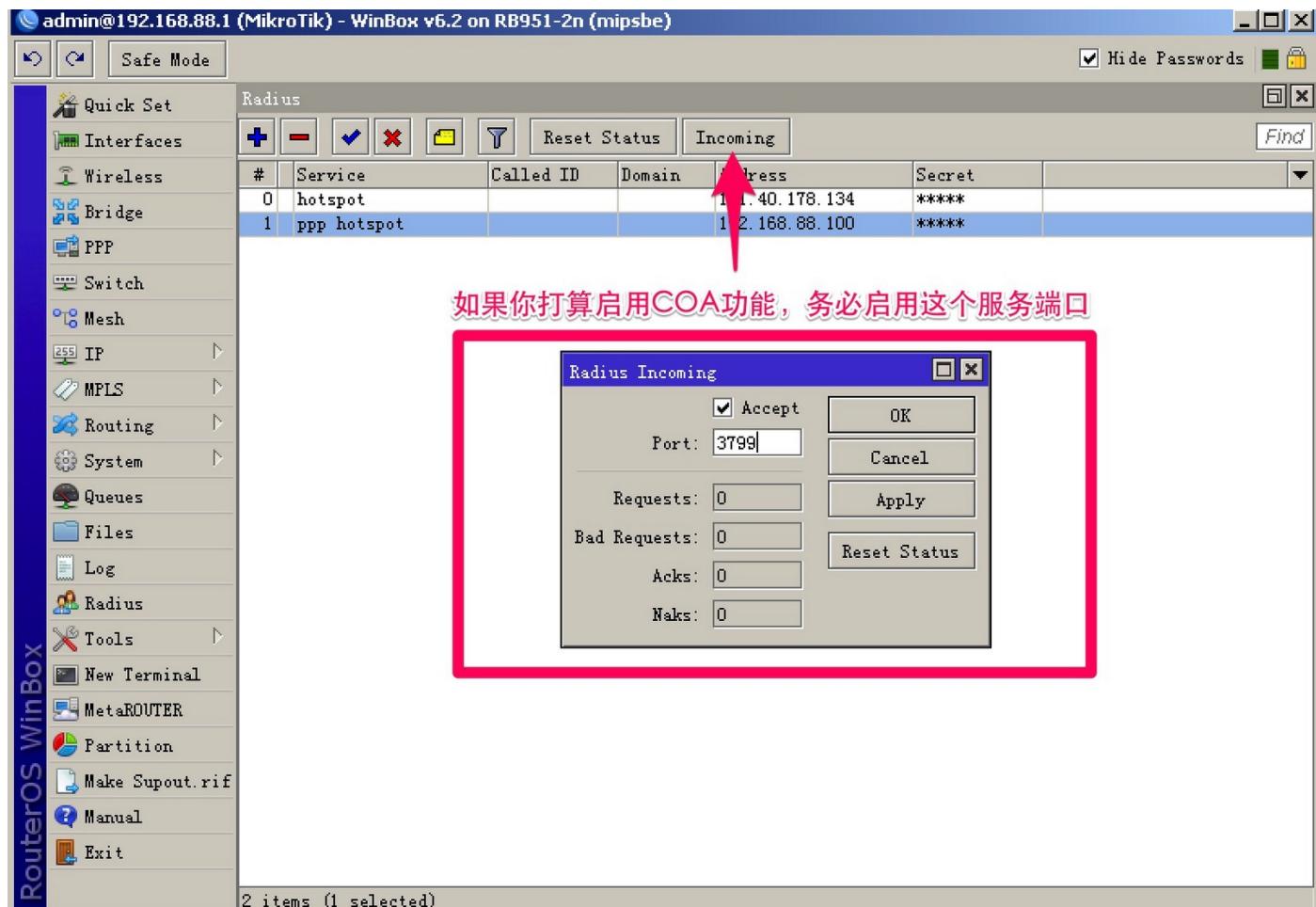
对于ToughRADIUS V2版本，在使用安装脚本安装时，可以直接指定服务端口。

用户强制下线

在ToughRADIUS中有一项很重要的功能，对在线用户强制下线。但在使用这一功能之前，管理员务必清楚的理解这一功能。在ToughRADIUS版本的更新迭代过程中，这一功能在不断完善。

首先必须了解，要正常使用强制下线功能，对接的NAS（路由器设备）必须支持并且必须在设备中开启相关选项，不同的设备配置方法可能不一样，你可以去咨询了的设备经销商和相关技术支持。

我们以RouterOS为例，开启这一功能有如下配置：



你应该注意到了，端口3799与在ToughRADIUS对应的BAS设备配置是一致的，

配置设备的功能后，你还需要开放ToughRADIUS设备的管理端口，默认是1815，ToughRADIUS通过这一端口进行一些实时性更强的消息通讯，使用websocket实现，因此你必须保证这个端口和ToughRADIUS web管理的端口一样的访问权限。

这一端口是可以配置的，如果你修改了这一端口，或者使用了防火墙映射。另外也可以配置外部使用的地址。

系统参数管理

	Radius设置
Radius认证模式	强制密码认证
是否允许查询用户密码	是
Radius服务地址	127.0.0.1
Radius服务管理端口	1815
Radius记账间隔(秒)	120
Radius最大会话时长(秒)	86400
拒绝延迟时间(秒)(0-9)	0
更新	

主动强制下线

在ToughRADIUS的在线用户查询界面中，可以对指定的在线用户直接解锁。在正常情况下，ToughRADIUS向NAS（路由器）设备发起强制下线请求，用户被断开，同时系统会生成上网记录。

被动强制下线

用户余额不足，过期等条件会自动触发用户强制下线请求，这一过程是系统自动完成的，无需人工介入。

强制下线与解锁的不同

用户解锁和强制下线最大的差别就是ToughRADIUS不会向NAS（路由器）设备发起请求，仅仅是清除在线纪录，并生成上网日志，它仍然可以解决用户并发限制后挂死的情况。如果一个NAS设备不支持强制下线功能，那么只能通过解锁来实现清理挂死用户。

使用ToughMySQL为ToughRADIUS系统提供数据存储

ToughMySQL是一个基于Docker技术的MySQL应用，一开始它就是为了ToughRADIUS提供一个简单可靠易用的数据库服务。

ToughRADIUS默认采用了SQLite存储数据，通常这足够运营上千的用户量了，不过当系统对数据的可管理性，系统的性能有更高的要求时，我们建议采用MySQL数据库来替换。

功能特性：

- 实现MySQL Docker容器部署。
- 提供针对不同服务器配置环境的优化配置。
- 提供一键脚本快速安装。
- 提供备份脚本，支持7天以上备份自动删除。
- 提供主从，互为主备的快速配置。

快速指南

备份当前数据库

如果是首次安装，可略过，如果是迁移数据库，则务必进行备份。

安装脚本

tmshell是一个自动化安装和管理脚本，通过这个脚本，提供了很多有用的管理功能

```
$ wget https://github.com/talkincode/toughmysql/raw/master/tmshell -O /usr/local/bin/tmshell  
$ chmod +x /usr/local/bin/tmshell  
$ tmshell install
```

直接输入 tmshell 可以看到支持的指令操作

```
usage: tmshell [OPTIONS] instance

    docker_setup           install docker, docker-compose
    pull                  mysql docker images pull
    install               install default mysql instance
    remove                uninstall mysql instance
    config                mysql instance config edit
    status                mysql instance status
    restart               mysql instance restart
    stop                 mysql instance stop
    logs                 mysql instance logs
    showmaster            mysql instance show master status
    showslave              mysql instance show slave status
    upmaster              mysql instance update master sync config
    backup                mysql instance backup database
    dsh                  mysql instance bash term
```

All other options are passed to the tmshell program.

完整的安装过程

安装过程是一个交互式的过程，根据实际情况修改具体参数

```
[root@i-jahnM3dt ~]# tmshell install
# 默认创建的mysql数据库用户
mysql user [raduser]:
# 默认创建的mysql数据库用户密码
mysql user password [radpwd]:
# 默认创建的mysql数据库名
mysql database [radiusd]:
# 默认mysqlroot密码
mysql root password [none]:
# 默认的mysql专用复制用户密码
mysql replication password [replication]:
# mysql服务端口
mysql port [3306]:
# 如果打算以热备模式部署，需要输入server id
mysql server id [1,2](default none): 1
# mysql服务使用的最大内存
mysql max memory [512M,1G,4G](default none):

ToughMySQL instance config:

instance name: mysql
mysql_user: raduser
mysql_password: radpwd
mysql_database: radiusd
mysql_root_password:
mysql_repl_password: replication
mysql_port: 3306
```

```
serverid: 1
mysql_max_mem:

database:
  container_name: db_mysql
  image: "index.alauda.cn/toughstruct/mysql"
  privileged: true
  ports:
    - "3306:3306"
  ulimits:
    nproc: 65535
    nofile:
      soft: 20000
      hard: 40000
    environment:
      - SERVERID=1
      - MYSQL_MAX_MEM=
      - MYSQL_USER=raduser
      - MYSQL_PASSWORD=radpwd
      - MYSQL_DATABASE=radiusd
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_REPL_PASSWORD=replication
  restart: always
  volumes:
    /home/toughrun/mysql/dbmysql:/var/lib/mysql
    /home/toughrun/mysql/backup:/var/backup
```

Creating db_mysql

Name	Command	State	Ports
<hr/>			
db_mysql	/usr/local/bin/run	Up	0.0.0.0:3306->3306/tcp

/home/toughrun/mysql/dbmysql 目录是映射到主机上的MySQL数据文件目录

/home/toughrun/mysql/backup 目录是映射到主机上的备份目录

双机热备模式部署

请参考 [《ToughMySQL, 为ToughRADIUS提供数据库双机热备支持》](#)

客户端认证报691的问题诊断

通常用户终端拨号会出现一个常见的691错误，一般客户端都会说是密码错误，这实在是太草率太过时的一个提示。

要诊断691背后的错误，还真的去求助于RADIUS系统，以RADIUS协议来说，只要返回一个认证拒绝消息，客户端就会收到一个691错误，我们需要找到RADIUS系统认证拒绝的原因。

通过系统日志来诊断691

要通过系统日志来诊断，请务必开启系统的DEBUG开关，以获取尽量多的信息。

通过终端进行日志定位，这要求对linux下的环境和指令都很熟悉。

如果你的ToughRADIUS部署的容器名是 trserver，通过以下指令进入容器终端：

```
docker exec -it trserver bash
```

ToughRADIUS的radius日志默认保存在 /var/log/radiusd.log，你可以通过grep, awk, vim等工具来定位与指定用户相关的输出信息。

另外，ToughRADIUS的系统控制管理台，也提供了一个日志查看功能，不过只能查看最近的一部分内容，如果内容滚动过快，很难找到相关信息，如果系统用户很少，日志滚动不是很快，则通过这个功能也是可行的。

The screenshot shows the ToughRADIUS system control interface. On the left is a sidebar with '功能导航' (Function Navigation) containing links for Control Panel, Configuration Management, Backup Management, and Log Management. Under Log Management, there are three options: Radius System Log, System Log, and Self-service System Log. The main area is titled '/var/log/radiusd.log' and displays a log file with the following content:

```

-09-16 22:31:40+0800 [AdminServerProtocol,3,127.0.0.1] Client connecting: tcp4:127.0.0.1:42229
2015-09-16 22:31:40+0800 [AdminServerProtocol,3,127.0.0.1] WebSocket connection open.
2015-09-16 22:32:06+0800 [-] Received SIGTERM, shutting down.
2015-09-16 22:32:06+0800 [autobahn.twisted.websocket.WebSocketServerFactory] (TCP Port 1815 Closed)
2015-09-16 22:32:06+0800 [autobahn.twisted.websocket.WebSocketServerFactory] Stopping factory
2015-09-16 22:32:06+0800 [AdminServerProtocol,3,127.0.0.1] WebSocket connection closed: connection was closed uncleanly (peer dropped the TCP connection without previous WebSocket closing handshake)
2015-09-16 22:32:06+0800 [AdminServerProtocol,2,127.0.0.1] WebSocket connection closed: connection was closed uncleanly (peer dropped the TCP connection without previous WebSocket closing handshake)
2015-09-16 22:32:06+0800 [CoAClient (UDP)] (UDP Port 38497 Closed)
2015-09-16 22:32:06+0800 [CoAClient (UDP)] Stopping protocol
2015-09-16 22:32:06+0800 [RADIUSAccounting (UDP)] (UDP Port 1813 Closed)
2015-09-16 22:32:06+0800 [RADIUSAccounting (UDP)] Stopping protocol
2015-09-16 22:32:06+0800 [RADIUSAccess (UDP)] (UDP Port 1812 Closed)
2015-09-16 22:32:06+0800 [RADIUSAccess (UDP)] Stopping protocol
2015-09-16 22:32:06+0800 [-] Main loop terminated.
[36;2m[INFO] - use config:/etc/radiusd.conf0m
running radiusd server...
2015-09-16 22:32:08+0800 [-] Log opened.
2015-09-16 22:32:08+0800 [-] serve listen 0.0.0.0
2015-09-16 22:32:08+0800 [-] RADIUSAccess starting on 1812
2015-09-16 22:32:08+0800 [-] Starting protocol
2015-09-16 22:32:08+0800 [-] RADIUSAccounting starting on 1813
2015-09-16 22:32:08+0800 [-] Starting protocol
2015-09-16 22:32:08+0800 [-] WebSocketServerFactory starting on 1815
2015-09-16 22:32:08+0800 [-] Starting factory
2015-09-16 22:32:09+0800 [AdminServerProtocol,0,127.0.0.1] Client connecting: tcp4:127.0.0.1:42236

```

通过用户日志来诊断691

在ToughRADIUS管理界面的维护管理栏目下，提供了一个用户消息跟踪的功能，该功能提供了对用户RADIUS认证和记账消息的实时跟踪，以及查询最近消息的功能，可以很方便的查找用户认证失败的原因。

常见的691原因

- 接入设备没有在ToughRADIUS中注册，请在BAS信息管理里添加，注意信息准确，如果不准确也会导致认证被拒绝。
- 用户授权时间已经过期，请提醒用户续费。
- 用户密码错误，请与用户核对修正。
- 用户余额不足，请提醒用户充值。
- 用户并发控制，即已经有用户认证成功在线，但用户并发数被限制，请确认该在线是不是异常断开导致挂死，如果是，使用解锁或强制下线。
- 用户MAC或VLAN已经绑定，如果用户需要更换认证设备，请解绑。

RouterOS对接指南

Linux PPTP 对接

以 ubuntu14 为例，谈谈PPTP对接ToughRADIUS

安装pptpd服务

```
sudo apt-get update -y  
sudo apt-get install -y pptpd iptables libfreeradius-client2 libfreeradius-client-dev
```

如果/etc/radiusclient目录不存在，建立一个radius配置目录链接

```
ln -s /usr/local/etc/radiusclient /etc/radiusclient
```

配置pptpd与radius

修改配置文件 /etc/pptpd.conf

```
option /etc/ppp/pptpd-options  
#debug  
#timeout 10  
logwtmp  
#bcrelay eth1  
#delegate  
#connections 100  
localip 10.79.97.1  
remoteip 10.79.97.10-200
```

修改配置文件 /etc/ppp/pptpd-options，注意这里采用了maschapv2认证方式，并采用mppe128位加密模式。

```
name pptpd
refuse-pap
refuse-chap
refuse-mschap
require-mschap-v2
require-mppe-128

# Network and Routing
ms-dns 8.8.8.8
ms-dns 8.8.4.4
proxyarp
nodefaultroute

#Logging
#debug
#dump
#logfile /var/log/pptpd.log

# Miscellaneous
lock
nobsdcomp
novj
novjccomp
#nologfd

plugin /usr/lib/pppd/2.4.5/radius.so
plugin /usr/lib/pppd/2.4.5/radattr.so
radius-config-file /etc/radiusclient/radiusclient.conf
```

配置/etc/radiusclient/radiusclient.conf , 注意配置authserver, acctserver为你实际的radius服务器地址和端口。

```
auth_order radius
login_tries 4
login_timeout 60
nologin /etc/nologin
authserver radius.toughctruc.net:1812
acctserver radius.toughctruc.net:1813
servers /etc/radiusclient/servers
dictionary /etc/radiusclient/dictionary
seqfile /var/run/radius.seq
mapfile /etc/radiusclient/port-id-map
default_realm
radius_timeout 10
radius_retries 3
login_local /bin/login
```

如果 /etc/radiusclient/port-id-map 不存在, 建立一个空文件

```
echo "" > /etc/radiusclient/port-id-map
```

配置radius服务器和共享密钥 /etc/radiusclient/servers

```
radius.toughstruct.net      testing123
```

为了支持mschapv2认证，需要加入 dictionary.microsoft字典，修改字典文件 /etc/radiusclient/dictionary，在末尾务必加上：

```
INCLUDE /etc/radiusclient/dictionary.microsoft
```

如果目录中没有这个字典，可以下

载：<https://raw.githubusercontent.com/talkincode/ToughVPN/master/radius/dictionary/dictionary.microsoft>

修改防火墙配置并修改内核转发支持

注意IP地址与/etc/pptpd.conf中配置的一致

```
iptables -t nat -A POSTROUTING -s 10.79.97.0/24 -o eth0 -j MASQUERADE  
iptables -A FORWARD -s 10.79.97.0/24 -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -j TCPMSS --
```

设置内核转发支持

```
sysctl -w net.ipv4.ip_forward=1
```

启动pptpd服务

```
service pptpd start
```

配置ToughRADIUS

在ToughRADIUS系统中，需要把PPTP作为接入设备加入，在BAS信息管理里增加一个标准配置即可。

增加资费，新增用户信息，接下来就可以进行拨号测试了。

在拨号过程中，可以通过用户消息跟踪或系统日志查看进行调试，使用mschapv2认证时，用户消息必定会有两个特定属性：MS-CHAP-Challenge 和 MS-CHAP2-Response，如果用户消息中没有此属性，则可能的原因是：

- pptp服务没有配置require-mschap-v2 和 require-mppe-128
- 系统内核不支持mppe
- 没有加入dictionary.microsoft
- 如果没有上面的问题，试着修改 require-mppe-128 为 require-mppe

注意事项

要支持mschapv2，需要系统内核支持MPPE，输入指令：

```
modprobe ppp-compress-18 && echo ok
```

如果输出ok，则系统内核支持。

在测试过程中，可以开启debug收集日志进行诊断，以及配合Radius服务器的日志进行诊断。

更多帮助，请参考 <http://poptop.sourceforge.net/dox/>

另外，你也可以关注我们的这个开源项目：<https://github.com/talkincode/ToughVPN>，这个项目计划实现更简单的一键安装以及常识Docker模式的部署。

开发指南



Git是一个分布式的版本控制系统，最初由Linus Torvalds编写，用作Linux内核代码的管理，现在它已经成为最流行的版本管理工具。Github是一个开源代码库以及版本控制系统，同时GitHub还是一个开源协作社区，ToughRADIUS的代码库正式托管在此。如果你不喜欢Github和Docker，我们是很难在一起愉快的玩耍了。

定制你的ToughRADIUS专有版本

想拥有自己的ToughRADIUS专有版本吗，想保持与官方版本的同步同时加入自己的定制特性而不冲突吗，这当然是可以的。

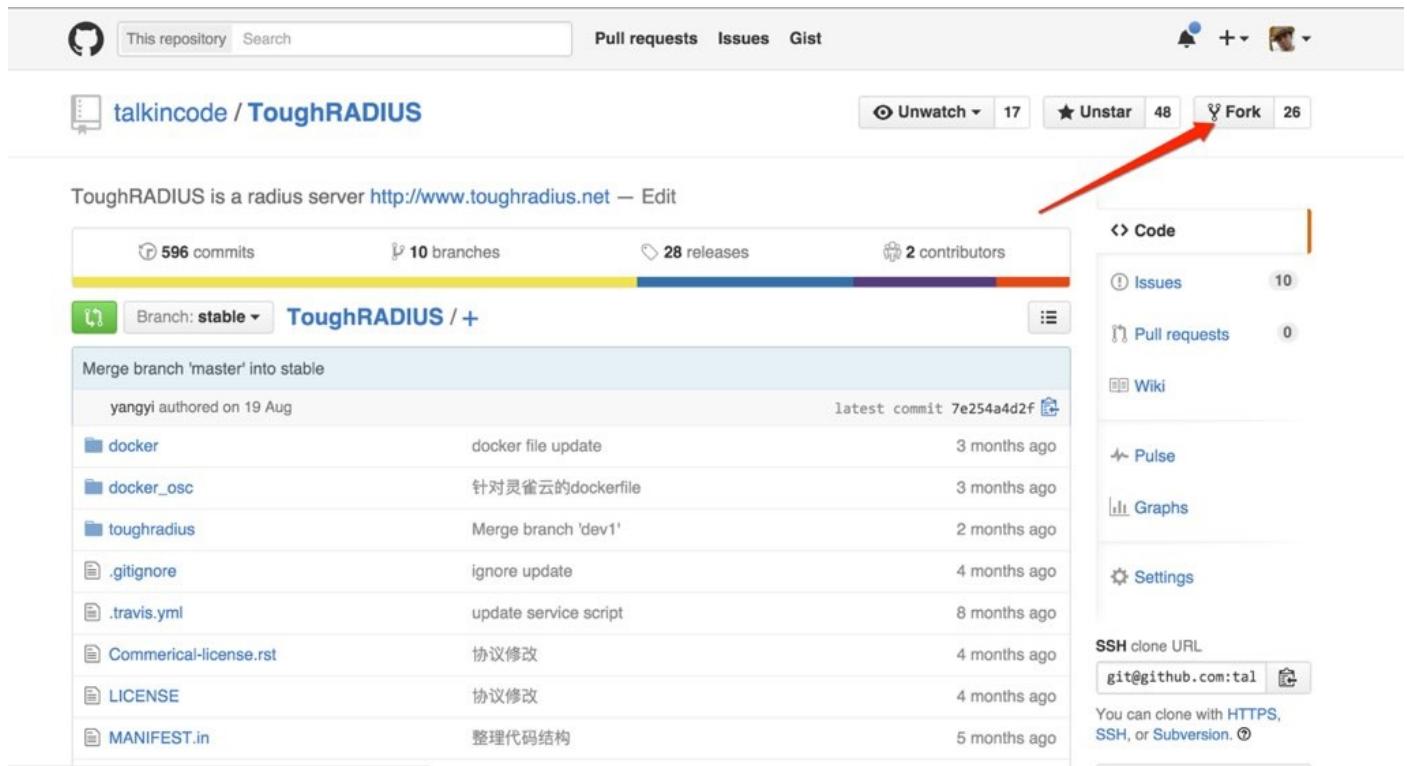
准备

你是一个程序猿或接近猿类。

拥有一个自己的Github帐号，<https://github.com/>

拥有一个Docker容器云平台的帐号，推荐国内灵雀云，<http://www.alauda.cn/>

fork一个ToughRADIUS仓库



The screenshot shows the GitHub repository page for `talkincode/ToughRADIUS`. The top navigation bar includes links for `Pull requests`, `Issues`, and `Gist`. In the top right corner, there are buttons for `Unwatch`, `Unstar`, and `Fork`. A red arrow points to the `Fork` button. The main content area displays the repository statistics: 596 commits, 10 branches, 28 releases, and 2 contributors. Below this, a list of recent commits is shown, including updates to Docker files, merges from the `master` branch, and updates to configuration files like `.gitignore` and `.travis.yml`. On the right side, there's a sidebar with options for `Code`, `Issues` (10), `Pull requests` (0), `Wiki`, `Pulse`, `Graphs`, and `Settings`. At the bottom, there's an `SSH clone URL` section with the address `git@github.com:tal/ToughRADIUS`.

在windows平台，你可以使用 [GitHub Desktop](#) 来进行本地仓库管理。

遵循git版本管理的规范进行开发是你要自己把控的，保证自己的定制特性，同时保持对官方版本的同步合并。

选择开发工具

通常pycharm这样的ide是不错的选择。

```

FROM centos:centos7
MAINTAINER jamiesun <jamiesun.net@gmail.com>

ADD docker_osc/radiusd.conf /etc/radiusd.conf
ADD docker_osc/supervisord.conf /etc/supervisord.conf
ADD docker_osc/toughrad /usr/bin/toughrad

RUN chmod +x /usr/bin/toughrad

RUN mkdir -p /var/toughradius/data

ADD docker_osc/privkey.pem /var/toughradius/privkey.pem
ADD docker_osc/cacert.pem /var/toughradius/cacert.pem

RUN yum update -y
RUN yum install -y libffi-devel openssl-devel git gcc crontabs python-devel python-setuptools
RUN yum install -y mysql-devel MySQL-python
RUN yum clean all

RUN easy_install pip
RUN pip install supervisor
RUN pip install Mako==0.9.0
RUN pip install Beaker==1.6.4
RUN pip install MarkupSafe==0.18
RUN pip install PyYAML==3.10
RUN pip install SQLAlchemy==0.9.8
RUN pip install Twisted==14.0.2
RUN pip install autobahn==0.9.3-3
RUN pip install bottle==0.12.7
RUN pip install six==1.8.0
RUN pip install tablib==0.10.0
RUN pip install zope.interface==4.1.1
RUN pip install pycrypto==2.6.1
RUN pip install pyOpenSSL==0.14
RUN pip install service_identity

RUN git clone -b stable https://github.com/talkincode/ToughRADIUS.git /opt/toughradius
RUN ln -s /opt/toughradius/toughctl /usr/bin/toughctl && chmod +x /usr/bin/toughctl
RUN /opt/toughradius/toughctl --initdb

```

但ide并非必须的，一个编辑器足以胜任，尤其是像sublime text 这样的精品。

```

FROM centos:centos7
MAINTAINER jamiesun <jamiesun.net@gmail.com>

ADD docker_osc/radiusd.conf /etc/radiusd.conf
ADD docker_osc/supervisord.conf /etc/supervisord.conf
ADD docker_osc/toughrad /usr/bin/toughrad

RUN chmod +x /usr/bin/toughrad

RUN mkdir -p /var/toughradius/data

ADD docker_osc/privkey.pem /var/toughradius/privkey.pem
ADD docker_osc/cacert.pem /var/toughradius/cacert.pem

RUN yum update -y
RUN yum install -y libffi-devel openssl-devel git gcc crontabs python-devel python-setuptools
RUN yum install -y mysql-devel MySQL-python
RUN yum clean all

RUN easy_install pip
RUN pip install supervisor
RUN pip install Mako==0.9.0
RUN pip install Beaker==1.6.4
RUN pip install MarkupSafe==0.18
RUN pip install PyYAML==3.10
RUN pip install SQLAlchemy==0.9.8
RUN pip install Twisted==14.0.2
RUN pip install autobahn==0.9.3-3
RUN pip install bottle==0.12.7
RUN pip install six==1.8.0
RUN pip install tablib==0.10.0
RUN pip install zope.interface==4.1.1
RUN pip install pycrypto==2.6.1

```

定制Docker构建脚本

修改 script/toughrun 构建脚本，修改以下内容：

```
git clone -b master https://github.com/talkincode/ToughRADIUS.git /opt/toughradius
```

修改为：

```
git clone -b master <你的仓库地址> /opt/toughradius
```

开始定制你的专有版本

现在可以开始加入你自己的代码了。

广告：欢迎加入ToughRADIUS的QQ开发群组讨论：487229323

可以修改一个自己的酷炫版本号：

文件在 ./toughradius/__init__.py

```
#!/usr/bin/env python

__version__ = 'ohmyradius_v1.2.0.1'
__license__ = 'AGPL'
__author__ = 'jamiesun.net@gmail.com'
```

完成之后就提交你的代码吧。

在灵雀云建立自己的构建镜像库

使用你的帐号进入灵雀云控制台，选择镜像仓库，创建镜像构建仓库。

The screenshot shows the LinqCloud web interface. At the top, there's a navigation bar with links for Home, Products, Prices, Activities, Documentation, Demonstrations, Blogs, and Mirror Center. On the right side of the header, there's a user profile icon for 'toughtech' and a sign-out link. Below the header, there are several service icons: Services, Mirror Library, Build, Storage Backup, and Dedicated Host. To the right of these are more icons for Organization Settings, Free Acceleration, Service Status, Work Orders, and Bills. The main content area has a heading '您还没有创建任何镜像仓库' (You haven't created any mirror library yet). Below this is a large blue button with a '+' icon and the text '创建镜像仓库'. A dropdown menu from this button shows two options: 'Mirror Library' and 'Mirror Build Library', with 'Mirror Build Library' being the one highlighted with a red box. At the bottom of the page, there's a footer with copyright information: 'Copyright © 2015 - LinqCloud Technology Beijing ICP Registration No. 15011102' and an email address: 'info@mathildetech.com'.

你可以在灵雀云关联绑定你的Github仓库，也可以选择快速创建。

The screenshot shows the LinqCloud web interface. The layout is similar to the previous one, with a navigation bar at the top and service icons below. The main content area has a heading '选择一个代码仓库' (Select a code repository) and a '快速创建' (Quick Create) button. Below this, there's a note: '温馨提示：对于快速创建方式创建的镜像构建仓库，您需要手动触发该镜像仓库的构建。' (Tip: For quick creation, you need to manually trigger the build of the mirror build library.) followed by two bullet points: '当使用SSH下载地址创建的镜像构建仓库时，我们会为您生成一个部署公钥，代码仓库的访问权限可以是私有的或公有的。' and '当使用HTTPS下载地址创建的镜像构建仓库时，我们不会为此创建部署公钥，请确保代码仓库的访问权限是公开的。' There are input fields for '仓库名称*' (Repository name*) containing 'toughtech / ohmyradius' and '代码仓库下载链接*' (Repository download link*) containing 'https://github.com/jamiesun/ToughRADIUS.git'. At the bottom are '创建' (Create) and '重置' (Reset) buttons. The footer is identical to the previous screenshot, with copyright information and an email address.

修改构建配置，在构建节点的选择上注意，如果选择国际节点，在构建速度上会有优势，因为 ToughRADIUS 依赖的很多开发库在国外，但是如果你的部署服务器国际带宽不够大，那么在初次部署时会比较慢。选择国内节点则反过来，如何选择请根据自己的情况判断，可以随时重新调整。

The screenshot shows the configuration page for the 'ohmyradius' image on the ToughRADIUS platform. At the top, there's a navigation bar with links for Home, Products, Prices, Activities, Documentation, Demonstrations, Blogs, and Image Center. On the right side of the header are user profile icons and account management links for 'tougttech'.

The main content area has a breadcrumb trail: '主页 / 镜像仓库'. Below it, the title 'tougttech > ohmyradius' is displayed with a back arrow icon.

修改构建配置

A note in a callout box states: '如果您需要开启持续构建, 选择“开启持续构建”并保存构建配置, 我们会为您生成Web钩子URL。由于您使用的是快速构建方式创建此镜像构建仓库, 所以我们无法将Web钩子URL自动添加到您的代码仓库设置中, 请您手动添加。'

Code repository: [https://github.com/jamiesun/ToughRADIUS.git \(Simple\)](https://github.com/jamiesun/ToughRADIUS.git)

Build service node: International (dropdown menu) Complete-time email notification

Image version: latest Add time version Continuous build

Code branch: master Continuous build

Build sub-directory: /docker Dockerfile location directory: /docker

Note: Branches and image repositories must be different, and names must be unique. When continuous builds are enabled, we will create push hooks for you, which will trigger automatic builds when code is updated. Additionally, we will create deploy keys to facilitate code download. Please ensure you have the necessary permissions for the code repository.

Buttons: 保存 (Save) 取消 (Cancel)

Copyright information at the bottom: 版权 © 2015 - 云雀科技 京ICP备15011102号-2 联系我们: info@mathildetech.com

点击开始构建按钮，等待构建完成。

The screenshot shows the Alauda Cloud Platform interface. At the top, there's a navigation bar with links for Home, Products, Prices, Activities, Documentation, Demonstrations, Blogs, and Image Center. On the right side of the header, there's a user profile icon for 'toughtech' and a search bar. Below the header, there are several icons for different services: Services, Image Library, Build, Storage Backup, and Dedicated Host. To the right of these are more icons for Organization Settings, Free Acceleration, Service Status, Workbench, and Bills.

In the main content area, the URL 'toughtech > ohmyradius' is shown, along with a green button labeled 'Image Library'. Below this, there's a table with details about the 'ohmyradius' image:

ohmyradius				
属性	公共的			
创建时间	2015-10-05 02:25:35	更新时间	2015-10-05 03:04:24	
构建服务节点	国际	代码仓库	https://github.com/jamiesun/ToughRADIUS.git	
分享链接	https://hub.alauda.cn/repos/toughtech/ohmyradius			

Below the table, there are three tabs: 'Information' (selected), 'Version', and 'Build'. Under 'Build', there's a table showing the build configuration:

Image Version	Code Branch	Build Context Directory	Dockerfile Location
latest	master	/docker	/docker

At the bottom of the build section, there's a summary of the last build:

成功 2d92b34d	toughstruct 0dec0965	toughtech/ohmyradius latest, v20151004.183229	2015-10-05 02:32:29 耗时 27分钟 53秒
----------------	-------------------------	--	------------------------------------

At the very bottom of the page, there's a footer with copyright information: 'Copyright © 2015 - 云雀科技 京ICP备15011102号-2 联系我们: info@mathildetech.com'.

可以实时查看构建过程：

The screenshot shows the Alouda Cloud platform's build interface. At the top, there are navigation links for Home, Products, Prices, Activities, Documentation, Demonstrations, Blogs, and Mirror Center. On the right, there are user profile and account management links. Below the navigation, there are several service icons: Services, Mirror Library, Build, Storage Snapshots, and Dedicated Host. To the right of these are more management links: Organization Settings, Free Accelerator, Service Status, Tools, and Bills. The main content area shows a build progress bar for a build ID: 2d92b34d. The status is "进行中" (In Progress). The build details include the source repository: https://github.com/jamiesun/ToughRADIUS.git, commit: latest, v20151004.183229, branch: master, and author: 0dec0965. A "Delete" button is visible at the top right of the build card. Below the card, a "Build Progress" section displays the terminal logs of the build process, which includes dependency resolution and package installation steps.

主页 / 镜像仓库

toughtech > 2d92b34d 进行中 删除

构建ID: 2d92b34d-9e45-46d1-b94c-49c9fba27fb7

toughstruct	toughtech/ohmyradius	https://github.com/jamiesun/ToughRADIUS.git
国际	latest, v20151004.183229	master
2015-10-05 02:32:29		由 0dec0965

构建进度

```
2015-10-05 02:34:07 =====
2015-10-05 02:34:07 Updating:
2015-10-05 02:34:07 coreutils           x86_64      8.22-12.el7_1.2      updates      3.2 M
2015-10-05 02:34:16 => Processing Dependency: perl-LWP::UserAgent 4.5:16.3-285.el7      tor-patch-4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Socket) >= 1.3 for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Scalar::Util) >= 1.10 for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl-macros for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl-LibXS for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(threads::shared) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(threads) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(constant) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Time::Local) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Time::HiRes) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Storable) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Socket) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Scalar::Util) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Pod::Simple::XHTML) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Pod::Simple::Search) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Filter::Util::Call) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: perl(Carp) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Processing Dependency: libperl.so()(64bit) for package: 4:perl-5.16.3-285.el7.x86_64
2015-10-05 02:34:16 --> Package perl-Error.noarch 1:0.17020-2.el7 will be installed
2015-10-05 02:34:16 --> Package perl-Exporter.noarch 0:5.68-3.el7 will be installed
2015-10-05 02:34:16 --> Package perl-File-Path.noarch 0:2.09-2.el7 will be installed
2015-10-05 02:34:16 --> Package perl-File-Temp.noarch 0:0.23.01-3.el7 will be installed
2015-10-05 02:34:16 --> Package perl-Getopt-Long.noarch 0:2.40-2.el7 will be installed
2015-10-05 02:34:16 --> Processing Dependency: perl(Pod::Usage) >= 1.14 for package: perl-Getopt-Long-2.40-2.el7.noarch
2015-10-05 02:34:16 --> Processing Dependency: perl(Text::ParseWords) for package: perl-Getopt-Long-2.40-2.el7.noarch
2015-10-05 02:34:16 --> Package perl-Git.noarch 0:1.8.3.1-4.el7 will be installed
2015-10-05 02:34:16 --> Package perl-PathTools.x86_64 0:3.40-5.el7 will be installed
2015-10-05 02:34:16 --> Package perl-TermReadKey.x86_64 0:2.30-20.el7 will be installed
2015-10-05 02:34:16 --> Package python-backports-ssl_match_hostname.noarch 0:3.4.0.2-4.el7.noarch will be installed
2015-10-05 02:34:16 --> Processing Dependency: python-backports for package: python-backports-ssl_match_hostname-3.4.0.2-4.el7.noarch
2015-10-05 02:34:16 --> Package rsync.x86_64 0:3.0.9-15.el7 will be installed
```

版权 © 2015 - 云雀科技 京ICP备15011102号-2
联系我们: info@mathilde.tech

构建完成后，就可以在你的服务器上使用Docker模式部署了，只是Docker仓库地址是你自己的了。
具体部署请参考快速指南

是不是酷毙了，你甚至可以把你得意的定制版本向更多朋友分享。

继续保持对你专有ToughRADIUS版本的更新维护吧。

mschapv2在Radius中的认证实现

在Radius的认证请求AccessRequest包中如果包含 MS-CHAP2-Response 和 MS-CHAP-Challenge 属性则意味着需要实现ms-chap-v2认证。

客户端 MS-CHAP2-Response 和 MS-CHAP-Challenge 生成的规则

MS-CHAP-Challenge

MS-CHAP-Challenge (即AuthChallenge) 是客户端生成的随机16字节。

MS-CHAP2-Response

随机生成16字节属性 PeerChallenge, 连同AuthChallenge, UserName, Password作为输入参数, 调用方法 GenerateNTResponse 得到 NtResponse.

```
GenerateNTResponse(AuthChallenge, PeerChallenge, UserName, Password)
```

GenerateNTResponse 方法的实现参考 <http://tools.ietf.org/html/rfc2759.html#section-8.1>

封装50字节的 MS-CHAP2-Response 属性:

[0 : 2]	Flags \x00\x00
[2 : 18]	PeerChallenge
[18 : 26]	Reserved \x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
[26 : 50]	NtResponse

服务端认证规则

校验 MS-CHAP2-Response 长度, 长度不等于50应该丢弃, 并发送拒绝认证。

从 MS-CHAP2-Response 提取 PeerChallenge, NtResponse

```
NtResponse = MS-CHAP2-Response[26 : 50]
PeerChallenge = MS-CHAP2-Response[2 : 18]
```

调用 GenerateNTResponse 方法得到 MyNtResponse

```
GenerateNTResponse(AuthChallenge, PeerChallenge, UserName, Password)
```

比较 MyNtResponse 与 NtResponse，不相等则验证失败。

Radius 认证响应

调用 GenerateAuthenticatorResponse 方法得到 AuthenticatorResponse

```
GenerateAuthenticatorResponse(  
    Password,  
    NtResponse,  
    PeerChallenge,  
    AuthChallenge  
    UserName  
)
```

GenerateAuthenticatorResponse 方法的实现参考 <http://tools.ietf.org/html/rfc2759.html#section-8.7>

设置Radius响应消息属性 MS-CHAP2-Success = AuthenticatorResponse