毕业设计(论文)

**学生姓名：** **学 号：**

**所在学院：** 信息工程学院

**专 业：** 网络

**设计(论文)题目：** 基于容器的企业服务器初始化部署

**指导教师：**

2020年 5 月 10 日

# 摘 要

最近几年，容器技术在国内外发展的非常迅速，国外已经形成了相对成熟的生态圈，相较国内，国内很多的大中型企业已经慢慢从IaaS向PaaS过渡并取得了许多显著成果，但是也有很多的中小型企业踌躇不定，在平台过渡上徘徊，本篇站在中小型企业的立场上实现了普通云服务器简易化过渡到容器，并且以很简单的方式部署了几个企业级应用，让中小型的企业看到容器的优点，并简易的使用上容器，实现从0到1的过渡。

**[关键词]:Docker;容器;PaaS**

目录

[摘 要 I](#_Toc2337)

[第一章 绪 论 1](#_Toc4551)

[1.1开发背景 1](#_Toc17892)

[1.2研究意义 1](#_Toc23247)

[1.3开发平台与局限 1](#_Toc30206)

[第二章 传统虚拟机与容器对比分析 2](#_Toc3471)

[2.1虚拟机 2](#_Toc26292)

[2.2容器 3](#_Toc10793)

[2.3 虚拟机与容器平台的构建对比 4](#_Toc10188)

[2.3.1 KVM创建CentOS7虚拟机 4](#_Toc30931)

[2.3.2 Docker创建CentOS7容器 5](#_Toc540)

[2.3.3 平台分析对比 6](#_Toc8856)

[第三章 系统概况分析 7](#_Toc14286)

[3.1系统分析 7](#_Toc20765)

[3.1.1系统层面 7](#_Toc908)

[3.1.2应用层面 7](#_Toc5525)

[3.1.3安全层面 9](#_Toc13867)

[3.2设计架构图 9](#_Toc6214)

[第四章 系统详细设计与实现 11](#_Toc3729)

[4.1 系统层面 11](#_Toc23887)

[4.1.1 内核升级 11](#_Toc1511)

[4.1.2 安装初始化Python3 11](#_Toc4244)

[4.1.3 初始化Docker环境 12](#_Toc23136)

[4.2 应用层面 13](#_Toc6641)

[4.2.1 安装初始化Zabbix监控实例 13](#_Toc13586)

[4.2.2 安装初始化Jumpserver跳板机 14](#_Toc1062)

[4.2.3 安装初始化ELK日志实例 15](#_Toc27571)

[4.2.4 Docker-Compose搭建redis集群 15](#_Toc21539)

[4.3 安全层面 17](#_Toc9535)

[4.3.1 cocofirewalld防火墙策略 17](#_Toc7058)

[4.3.2 coconetwork网络连接策略 19](#_Toc8403)

[第五章 系统具体实现 20](#_Toc10451)

[5.1 系统层面 20](#_Toc6200)

[5.2 应用层面 21](#_Toc1996)

[5.3 安全层面 23](#_Toc17398)

[5.3.1 防火墙cocofirewalld 23](#_Toc25141)

[5.3.2 网络coconetwork 28](#_Toc29936)

[总 结 33](#_Toc842)

[参考文献 34](#_Toc6092)

[致 谢 35](#_Toc8712)

# 第一章 绪 论

## 1.1开发背景

云时代的发展如火如荼，容器的优略也慢慢的展现出来，一个企业慢慢步入容器是必然的。在实习期间，本人有幸见证了一个中小型公司从IaaS到PaaS的迁移与过渡，但是在刚开始的时候步履艰难，遇到了很多的坑，本课题在这样的前提下受到启发所开发的，解决了企业在部署安装Docker容器时可能遇到的坑，此外提供了一些自动化部署企业常用应用的的功能。

## 1.2研究意义

本课题探究的意义是简化企业运维人员搭建PaaS平台。题前半部分可以供初识Linux的运维人员快速构建部署一个PaaS平台，并在所构建的PaaS平台上安装部署一些以前传统物理服务器或云服务器上所必备的企业应用程序，由于像JumpServer，Zabbix,Redis集群等服务在原来的传统物理服务器或云服务器上部署起来非常的麻烦与鸡肋，一不小心就可能导致部署失败，排错也非常的麻烦，因此本课题后半部分的研究就是针对传统物理服务器或云服务器上部署服务经常陷入窘境而开发的一套快速部署这些基础服务的脚本。通过这些脚本部署可以相当大程度的对比传统虚拟机与容器在程序部署方面的优势。

## 1.3开发平台与局限

本课题基于VMware所开发，开发环境为Windows10，运行环境为CentOS7，编写语言为shell，运行环境最低配置要求2核4G。由于没有实体机去做真实企业服务器环境，因此下面的一系列实验在VMware虚拟机上实现，模拟企业物理服务器或云服务器上部署安装容器平台及应用。

# 传统虚拟机与容器对比分析

## 2.1虚拟机

虚拟机是一个计算机系统的仿真，简单来说，虚拟机可以实现在一台物理计算机上模拟多台计算机运行任务。操作系统和应用共享一台或多台主机(集群)的硬件资源，每台VM有自己的OS，硬件资源是虚拟化的。管理程序(hypervisor)负责创建和运行VM，它连接了硬件资源和虚拟机，完成server的虚拟化。由于虚拟化技术和云服务的出现，IT部门通过部署VM可以可减少cost提高效率。VM也消耗大量系统资源，每个VM不仅运行一个OS的完整copy并且需要所有硬件的虚拟化copy，这消耗大量RAM和CPU。相比单独计算机，VM是比较经济的，但对于一些应用VM是过度浪费的，需要容器。虚拟机如图2.1所示

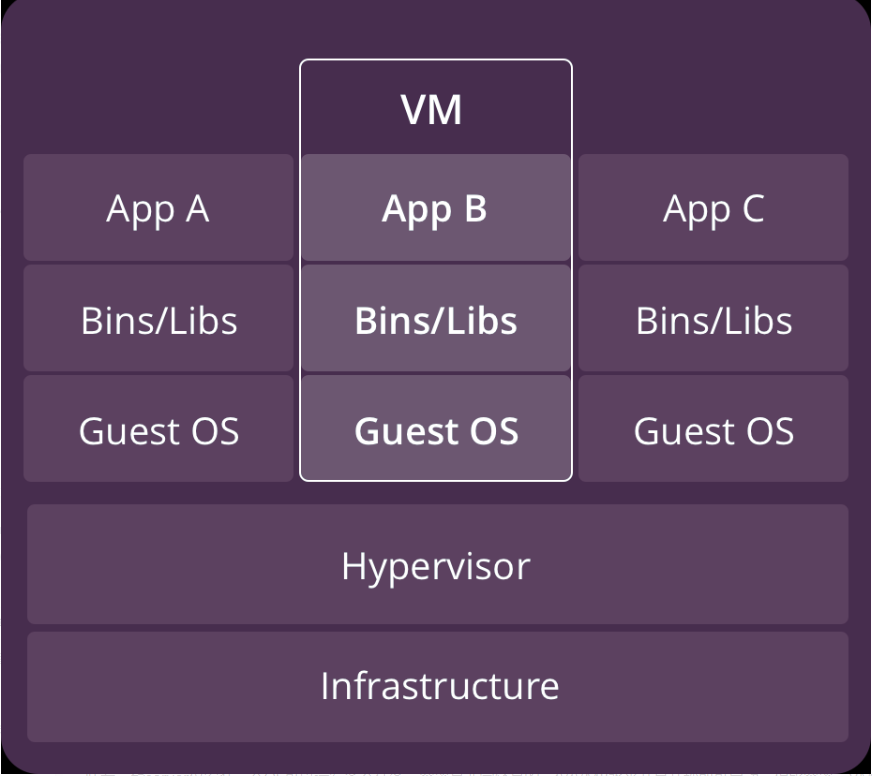


图 2.1虚拟机

## 2.2容器

容器是将操作系统虚拟化，这与VM虚拟化一个完整的计算机有所不同。

容器是在操作系统之上，每个容器共享OS内核，执行文件和库等。共享的组件是只读的，通过共享OS资源能够减少复现OS的代码，意味着一台server仅安装一个OS可以运行多个任务。容器是非常轻量的，仅仅MB水平并且几秒即可启动。相比容器，VM需要几分钟启动，并且大小也大很多。

与VM相比，容器仅需OS、支撑程序和库文件便可运行应用，这意味可以在同一个server上相比VM运行2-3倍多的应用，并且，容器能帮助创建一个可移植的，一致的开发测试部署环境。容器如图2.2所示

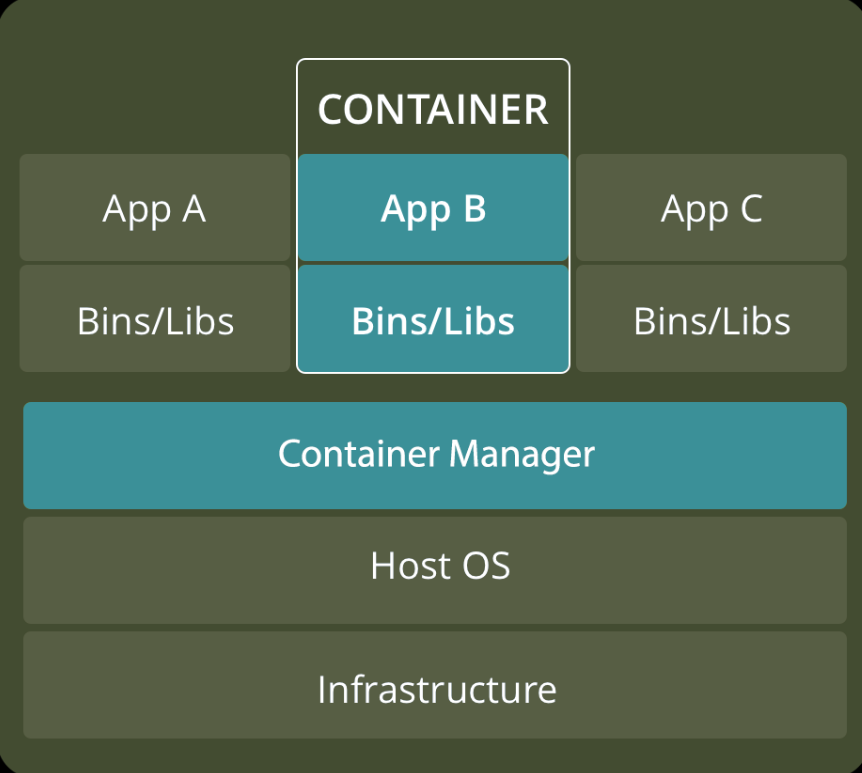


图 2.2容器

## 2.3 虚拟机与容器平台的构建对比

本小节选择KVM与Docker同时去构建一个CentOS7的虚拟机以分析比较虚拟机与容器两者之间的差异。

### 2.3.1 KVM创建CentOS7虚拟机

KVM 是Linux 的一个内核模块，它使 Linux 变成了一个 Hypervisor。简单的可以理解它可以在一台主机上创建多个虚拟机，就像云服务器一样，这里使用KVM创建一台CentOS7虚拟机。构建KVM的先决条件是CPU是否支持虚拟化，此KVM虚机是在VM中构建的，因此我们需要配置开启此处的CPU虚拟化。如图2.3所示：

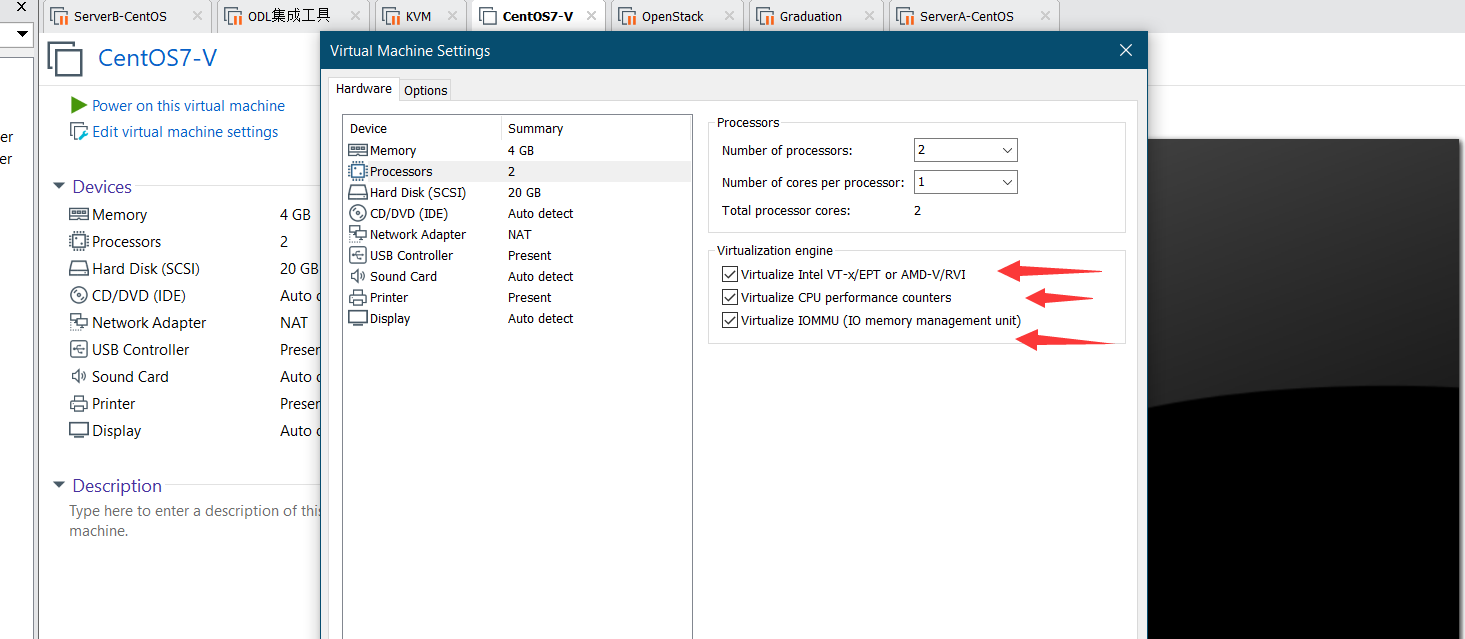


图 2.3 开启CPU虚拟化

这里的虚拟机使用带界面的便于观察，在图形界面的虚拟机中部署KVM，安装好KVM后，使用上传好的镜像创建一个CentOS7的虚拟机如图2.4所示：

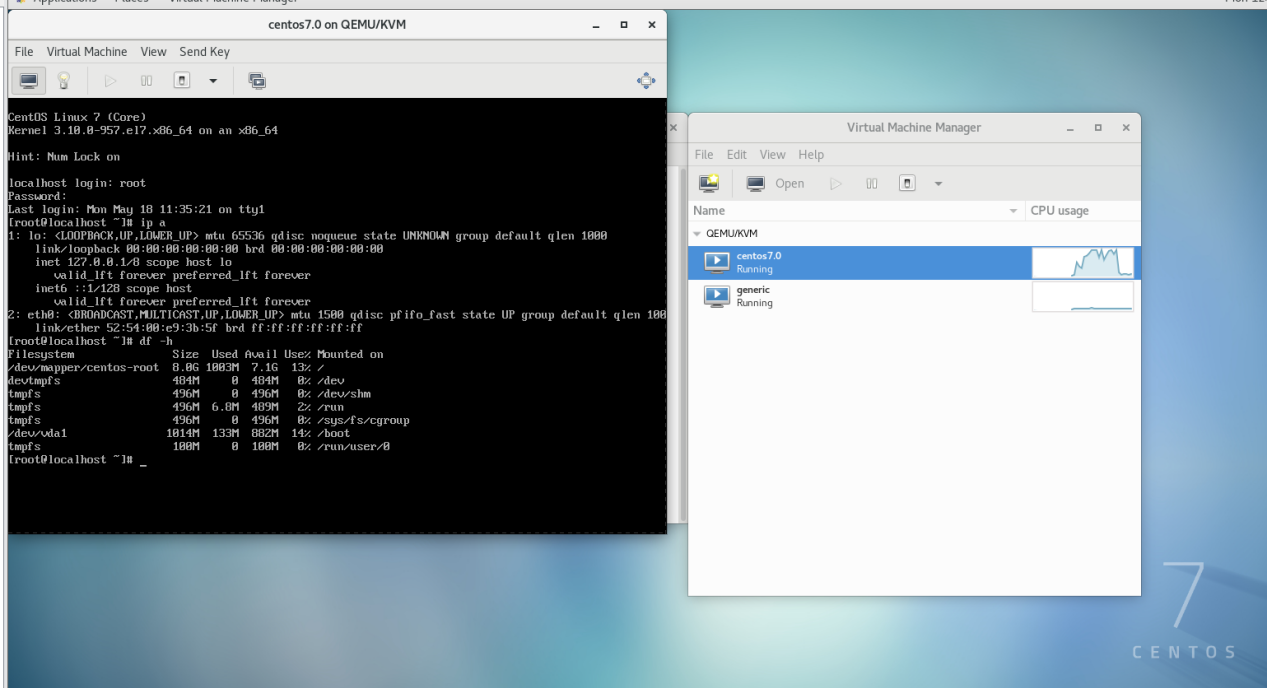


图 2.4 KVM虚拟机运行情况

### 2.3.2 Docker创建CentOS7容器

Docker是PaaS的典型代表，它使用了沙盒机制，可以方便的打包或移植所构建的应用程序，也可以随时运行别人已经构建好的容器。这部分从Dockerhub上拉取了一个CentOS7的镜像容器，进入容器直接运行，如图2.5所示：

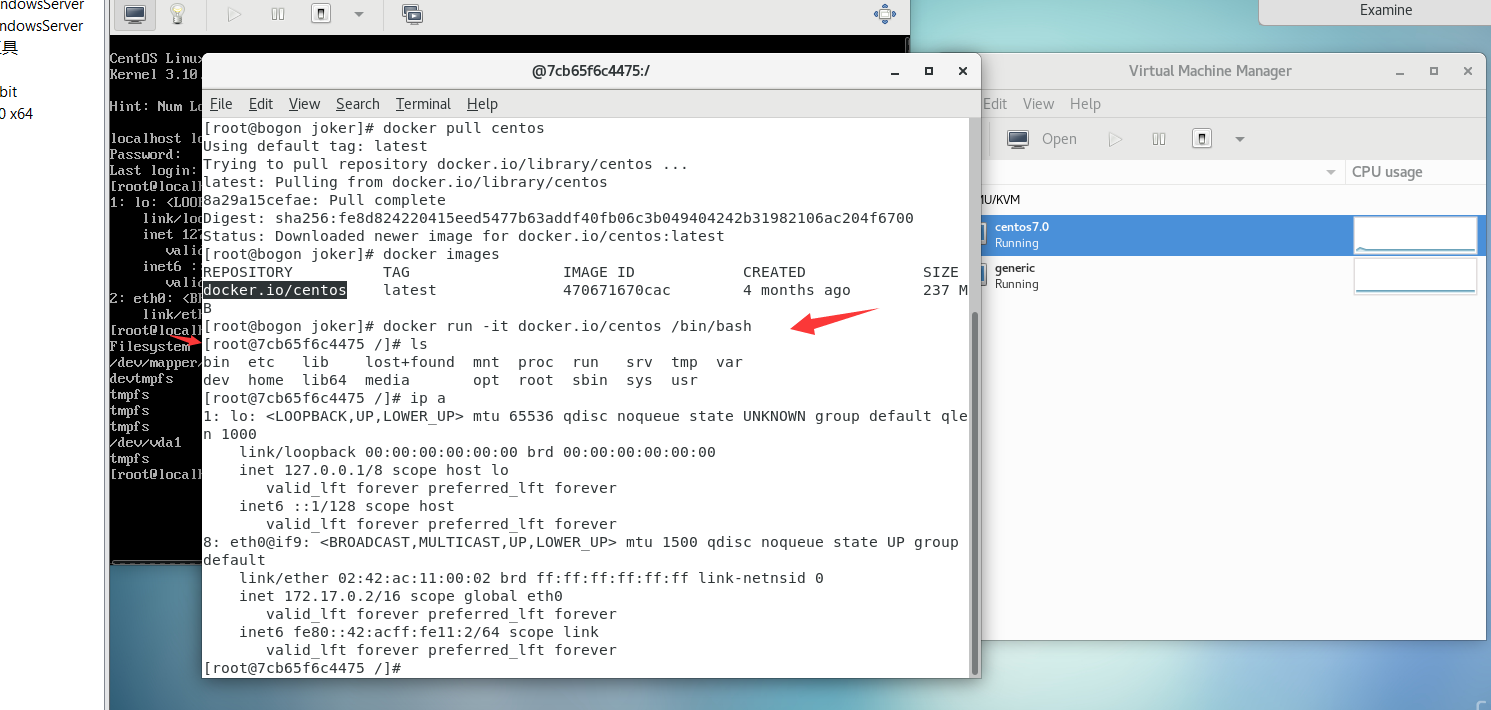


图 2.5 Docker运行情况

### 2.3.3 平台分析对比

构建上面两个虚机很明显的对比前者和安装一个普通的虚拟机一样，非常慢，而后者只需运行镜像即可，构建是秒级的，很明显的原因是构建一个KVM虚机是从底层硬件全部虚拟构建出来来了，而构建一个容器就像开启了一个进程一样简单。具体分析如下：

KVM(IaaS)

KVM对比于容器也有一个比较大的优势就是可以使用不同的操作系统或内核。所以，举例说，你可以使用微软Azure，同时运行Windows Server2012的实例和SUSE Linux企业级服务器的实例。至于Docker，所有容器都必须使用同样的操作系统和内核。

Docker(PaaS)

Docker容器可以在秒级启动，比起KVM创建虚拟机要快的多，它对CPU以及内存的使用率是非常高的，可以在一台主机上运行很多个Docker容器。

构建一个容器非常的简单，只需编写对应的Dockerfile即可构建一个服务层面的容器，程序开发者通过Dockerfile指导安装配置环境和依赖，这样大大提高了程序交付的时间。

# 第三章 系统概况分析

## 3.1系统分析

由于企业的服务器是面向生产环境的，因此对服务器要求比较高，从服务器的选型，操作系统的选择，操作系统内核的选择，以及平台安全性，以及软件的部署速度各方各面都有所要求。

考虑到这些问题，本设计从下面几种层面分别出发做了相应的脚本。

本系统运行后目录可选项如下：

1. 更新服务器内核及BASH
2. 安装初始化Python3环境
3. 安装初始化Docker环境
4. Docker安装初始化ELK日志监控
5. Docker安装初始化Zabbix性能监控
6. Docker安装初始化Jumpserver跳板机
7. Docker-Compose搭建Redis三主三从集群
8. cocofirewalld防火墙策略
9. coconetwork网络连接策略
10. 查看服务器基本信息

q.退出

上述目录选项大致可分为三个层面，分别为系统层面，应用层面，安全层面。

下面将分析这三个层面的功能。

### 3.1.1系统层面

可以自由的选择是否升级内核与Bash，以及安装Python3和Docker。

### 3.1.2应用层面

用Docker来打包部署开发人员所写的程序以及部署几种常用的应用服务，下面的三种应用服务是很多企业都会用到的，对应关系如下

Zabbix-服务器/应用性能

Jumpserver-服务器安全

ELK-服务器/应用日志分析

最后第四个Redis所对应的是企业应用程序，下面的Redis将搭建一个三主三从的集群模拟企业应用程序互连关系。

在这之前先简单的介绍一些这几个应用层面的服务：

#### 3.1.2.1 Zabbix

为了服务器的各项性能监控，我们使用zabbix,它可以很方便的将服务器或者服务器上的某些应用程序状态等各种指标一一展现在web界面上，方便管理员对服务器或者这些应用快速的定位和解决各项问题。

#### 3.1.2.2 Jumpserver

企业中可能会用到大量的服务器，为了这些服务器的集中管理与安全，企业会选择搭建Jumpserver开源跳板机来管理这些机器。Jumpserver 是一款由python编写开源的跳板机（堡垒机）系统，实现了跳板机应有的功能。基于ssh协议来管理，客户端无需安装 agent。Jumpserver能够实现跳板机常见需求，如授权、用户管理、服务器基本信息记录等，同时又可批量执行脚本等功能；它里面的高级功能有录像回访，实时监控等功能。简单的来说，Jumpserver的作用相当于一个高级的防火墙。

#### 3.1.2.3 ELK

在很多场景下，企业可能有大量对日志分析的需求，如果是小规模的日志排查问题，那么可以通过简单搜索定位排查问题的所在，但是如果有大量的日志需要一个一个排查问题，那么传统的排查将非常的费时费力，ELK的出现就是为了解决这种企业窘境。

ELK分为三部分：

Logstash：负责数据的收集，处理和储存

Elasticsearch：负责数据的检索和分析

Kibana：负责可视化。

#### 3.1.2.4 Redis

做开发的人没有不知道redis的，redis是一个key-value键值型的数据库，它的速度非常的快。是一个内存级别的数据库，可以用作数据库，缓存以及消息中间件等应用场景。

### 3.1.3安全层面

部署上述应用后服务后，服务器应该放行对应的端口使这些服务在web界面或者远程进行连接。为了方便这里编写了两个和容器关系不大的脚本，简单化以字符交互界面控制网络与防火墙，并且在防火墙中添加了一项防止暴力破解服务器端口，并根据情况发送邮件到管理员邮箱的功能。设计这两个脚本仅仅是为了方便防火墙与网络的操作。

## 3.2设计架构图

本系统开发是采用CentOS7系统下的bash shell,作用是为了快速初始化一个CentOS7服务器使之过渡向容器化，并在其上面部署Docker以及一些常用监控运维工具，并且开发了两个小工具，作用分别是管理防火墙和管理网络。具体设计架构图3.1所示：

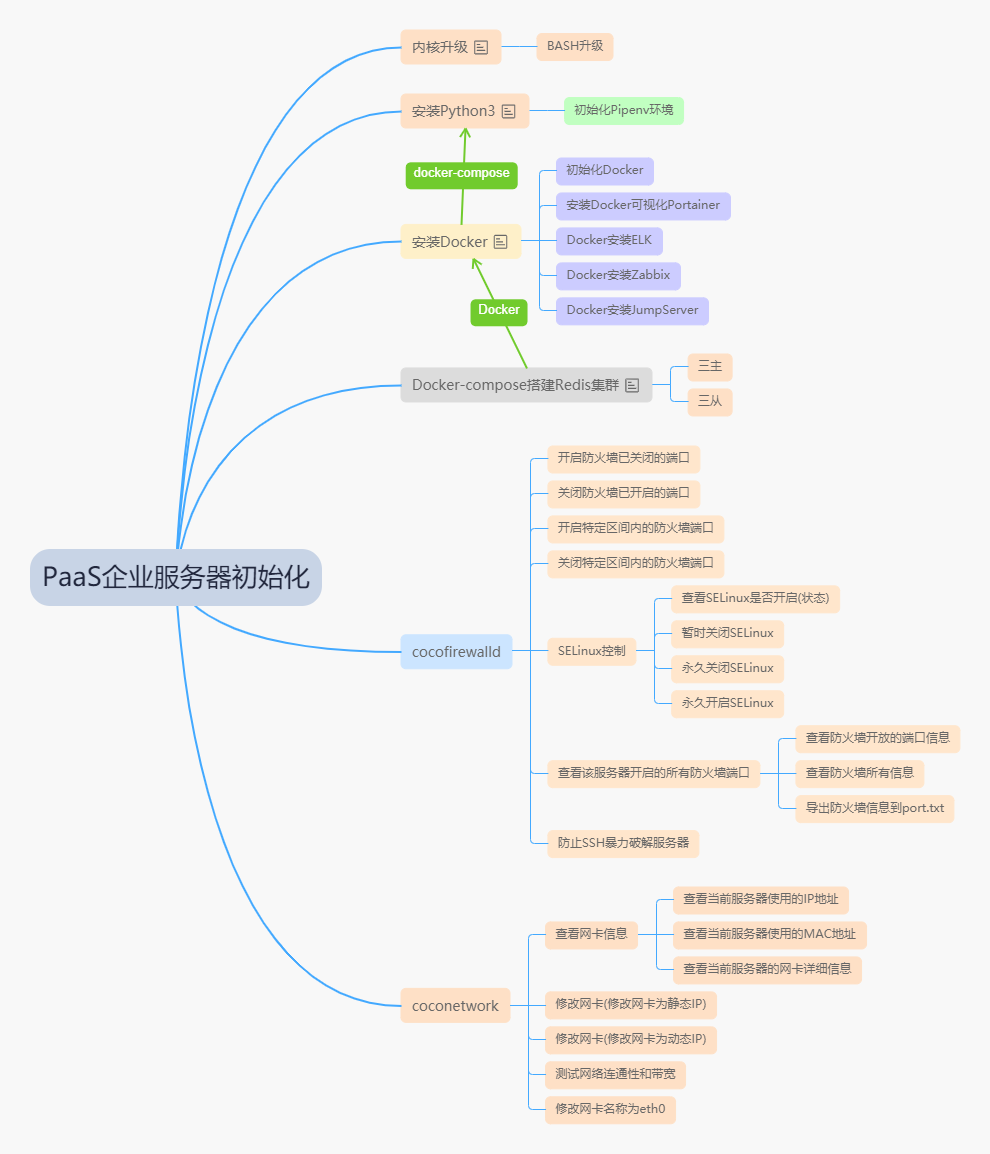


图 3.1架构图

# 第四章 系统详细设计与实现

本程序运行的先决条件是网络通畅，由于需要下载一系列软件，本程序需要在网络通畅的条件下进行，之所以选择网络安装是因为下载过多的包组不容易之间上传，另外还可能安装不全组件，因此本程序设计之初就是在网络通畅的情况下运行的，由于本系统是使用shell脚本所编写，因此网络和防火墙一些不需要网络的脚本程序在有无网络的环境下均不受影响。一运行本程序，程序会下载一些必备组件包，修改中文，更换阿里云源等，等一切安装就绪后进入如字符交互界面，由于不习惯每次同一种颜色的界面，于是设置了颜色随机，每次进入终端颜色可能会不同。

## 4.1 系统层面

### 4.1.1 内核升级

这部分的实际实现为：使用yum从阿里云官网下载安装最新版本稳定内核，同时安装bash5.0，CentOS7系统默认内核版本3.10，bash版本为4.2，安装完成重启服务器内核版本更新至5.6.8，bash版本为5.0，做此步骤是因为Docker中某些高级功能不支持内核版本太低，因此这里需要更新服务器内核。更新内核是从elrepo下载稳定内核的源，然后再下次重启的时候引导下载的新内核，引导成功开机正常后可以删除以前的内核。

### 4.1.2 安装初始化Python3

这部分的实际实现为：使用yum安装Python3和很多必要的依赖，安装Python3主要是为了安装后续所需的Docker-Compose，安装完成下次登入系统需要使用source /opt/py3/bin/activate启动一下pipenv环境，这样安装Python的第三方库会安装到一个虚拟的环境中，不会直接安装到系统中，这样的好处是实现不同版本的隔离。安装docker-compose使命令pip3 install docker-compose，只有在安装完成docker-compose后才能使用yml文件直接启动redis集群。在这步中其实还做了换源等操作，方便后续开发的使用。

### 4.1.3 初始化Docker环境

这部分的实际实现为：首先使用yum安装Docker和各种依赖组件，然后通过shell脚本在安装完毕后更换了Docker的源，此外安装完成Docker后使用Docker安装了一个Docker的可视化管理平台Portainer，Portainer是使用docker pull命令从docker hub上拉取的镜像。安装完成后访问ip+9000端口会出现如下图4.1页面。这个地方启动命令为：

docker run -d -p 9000:9000 \

--restart=always \

-v /var/run/docker.sock:/var/run/docker.sock \

--name prtainer \

docker.io/portainer/portainer

--restart=always选项设置启动docker后自动启动portainer。

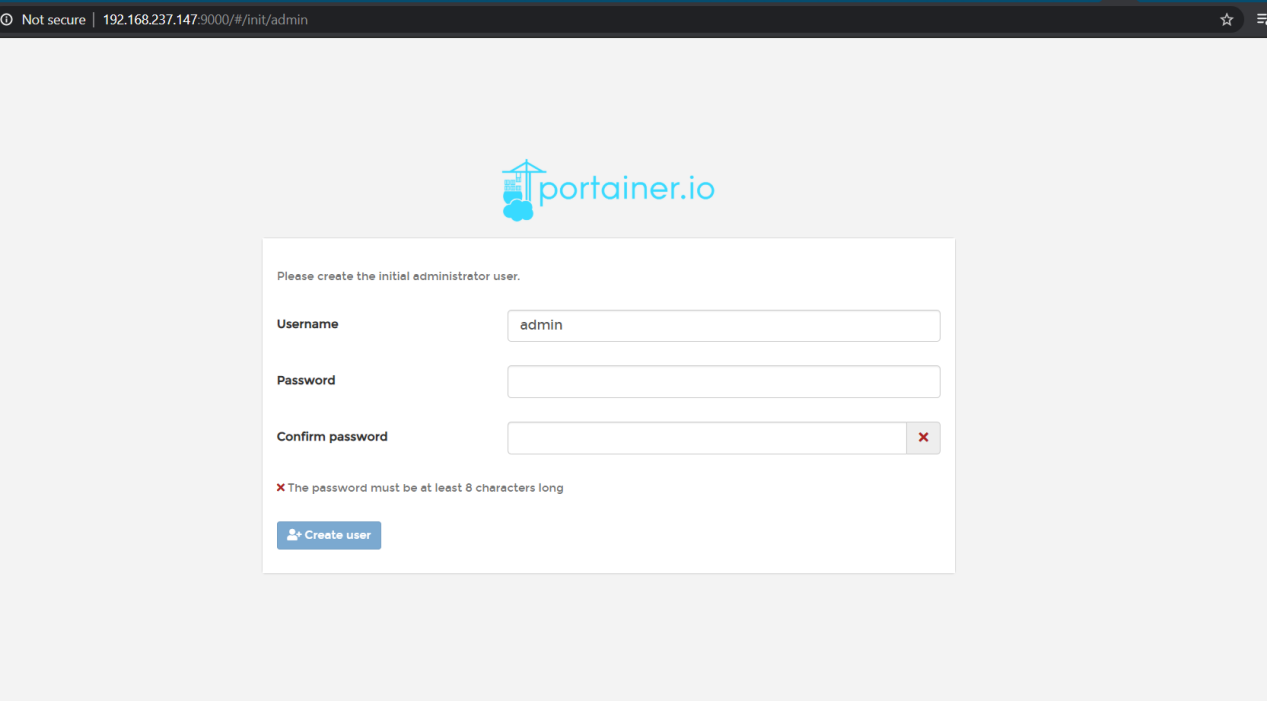


图 4.1 portainer登录界面

设置用户名密码之后，后续很多操作都会在这个页面上进行，如图4.2所示：

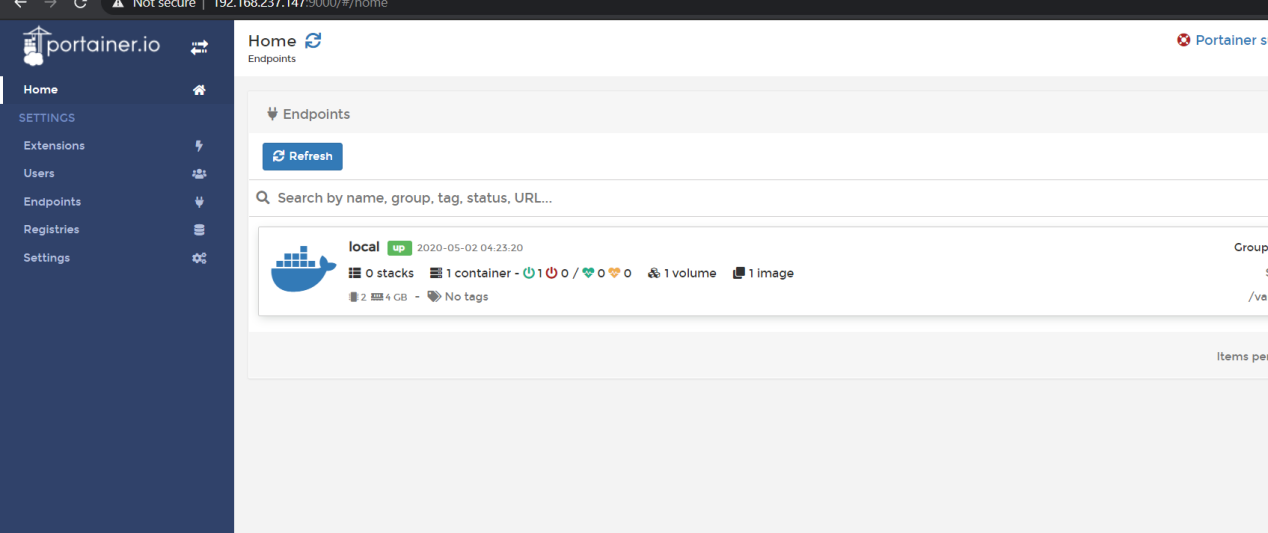


图 4.2 portainer操作界面

## 4.2 应用层面

### 4.2.1 安装初始化Zabbix监控实例

这部分实际的实现是：使用docker pull命令从docker hub上拉取mysql5.7，zabbix/zabbix-server-mysql，zabbix/zabbix-web-nginx-mysql三个镜像，这三个镜像的启动依赖关系如下：

1.启动空的 MySQL 服务器实例

2.启动 Zabbix server 实例，并将其关联到已创建的 MySQL server 实例Zabbixserver 实例将 10051/TCP 端口（Zabbix trapper）暴露给主机。

3.启动 Zabbix Web 界面，并将其关联到已创建的 MySQL server 和 Zabbix server 实例。安装成功运行界面如4.3所示：

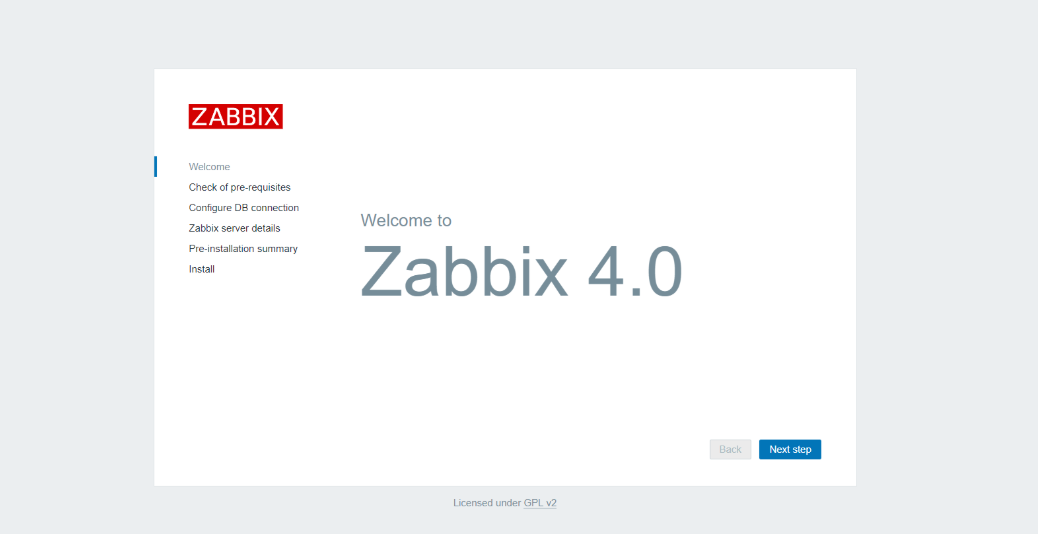


图 4.3 zabbix安装界面

### 4.2.2 安装初始化Jumpserver跳板机

这部分实际的实现是：使用docker pull从docker hub拉取出Jumpserver的集成镜像，之所以拉取集成镜像的原因是Jumpserver组件众多，不便管理。 Jumpserver跳板机的作用是管理很多的服务器，将这些资产全部接入到一台可以统一管理的机器上，这样很大程度保护了服务器的安全。单机部署Jumpserver的话会遇到很多问题因为Jumpserver中的luna,coco,guacamole是分开安装的，一不小心配置就会使某个组件不好用。而且Jumpserver官方说的是最好在新的系统上部署安装。安装成功如图4.4所示:



图 4.4 Jumpserver界面

### 4.2.3 安装初始化ELK日志实例

这部分实际的实现是：使用docker pull从docker hub上拉取elk集成镜像，集成镜像由Logstash，Elasticsearch，Kibana三部分组成。如果在普通服务器不使用容器部署的话配置这三部分组件连通是比较麻烦的，并且会浪费很多的时间，运行起来如图4.5所示

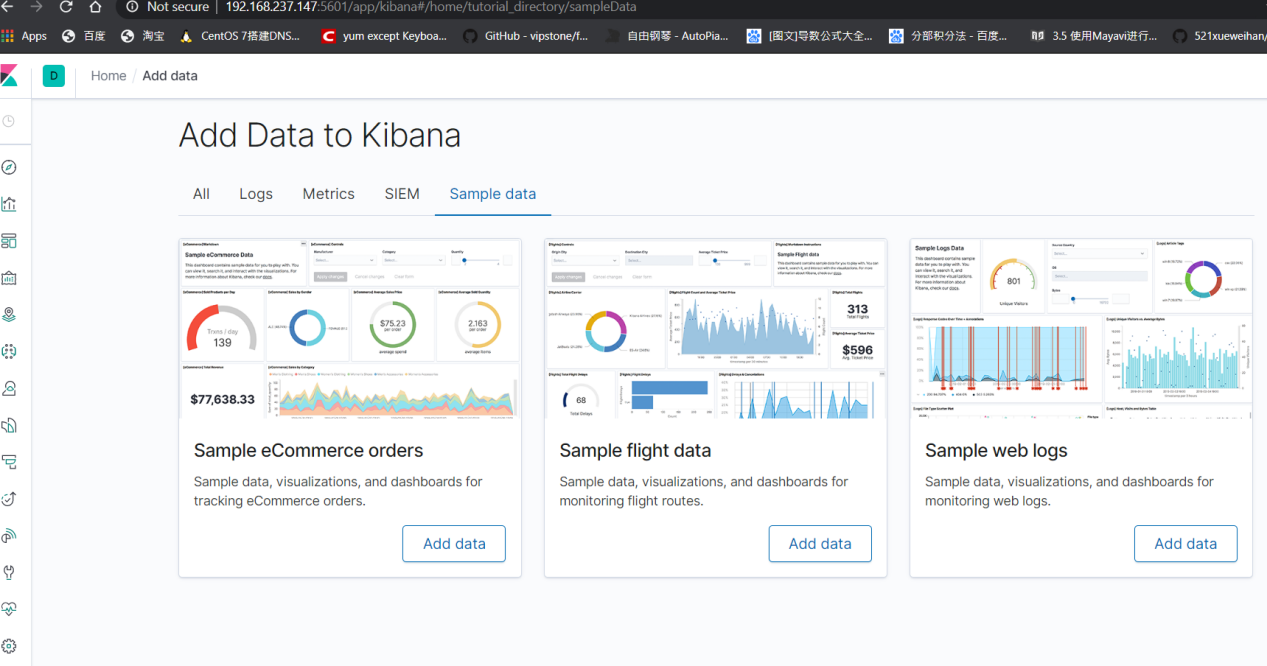


表 4.5 ELK的Kibana界面

### 4.2.4 Docker-Compose搭建redis集群

这部分比起上面三部分一键式安装麻烦一些，作用是模拟企业部署容器应用之间的互联，具体实现是：使用docker pull拉取docker hub上面的redis镜像，然后编写一个docker-compose.yml文件，然后把所有配置，挂载卷，网桥等全部写入docker-compose.yml文件中，一次性启动即可，主节点和从节点放入不同的网段中，如下下图可视化界面中所示:

通过Portainer可以看到我们的集群是存活的。如图4.6和4.7所示：

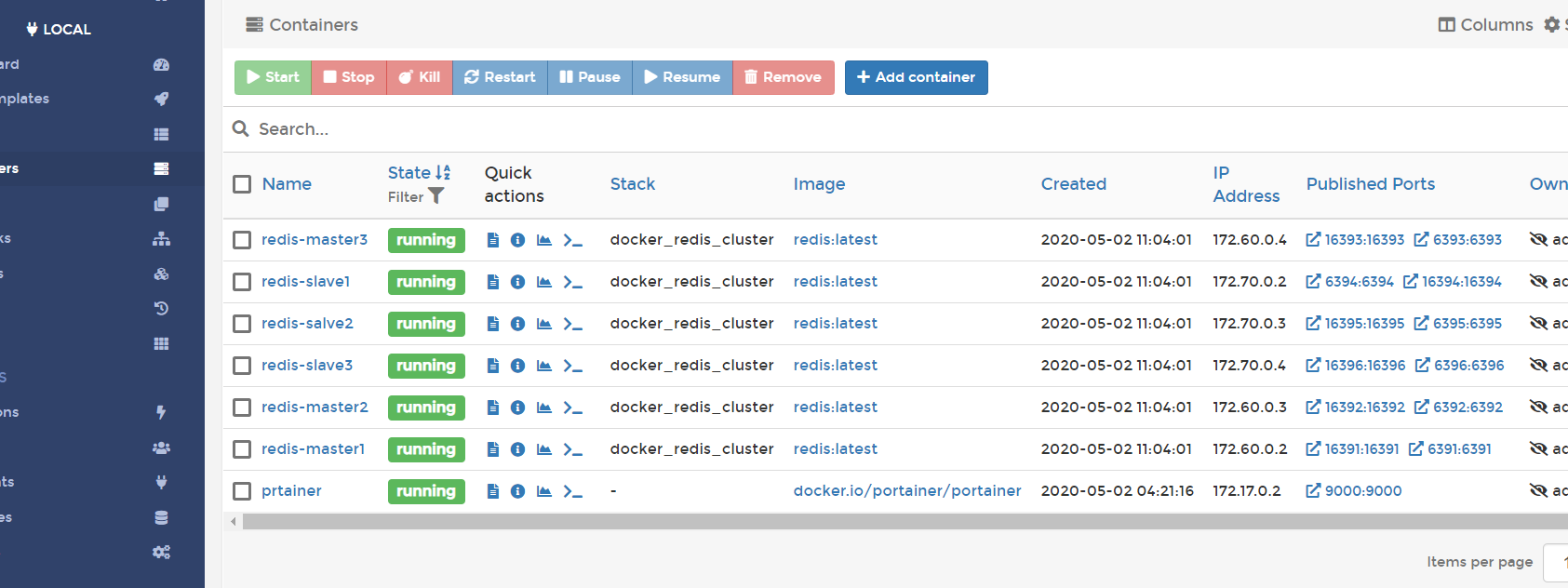


图 4.6 集群状态

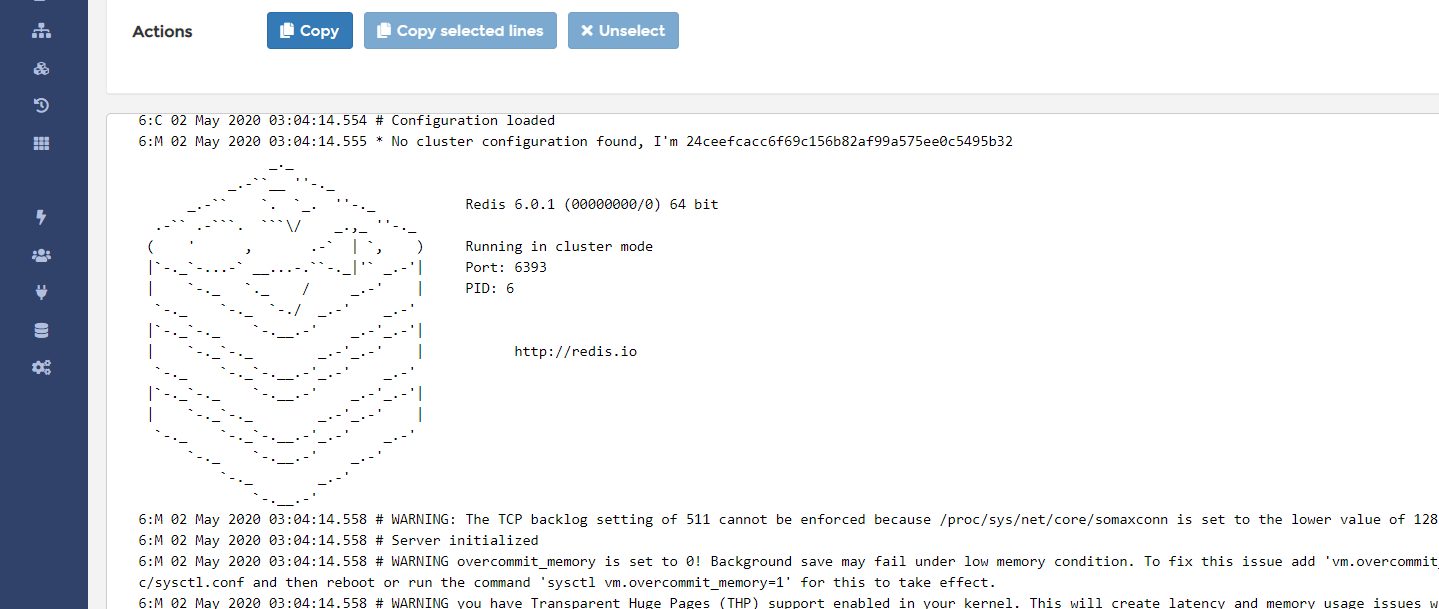


图 4.7 单个节点信息

这里贴出1号主节点的配置信息，代码解释并入注释如下所示：

redis-master1:

image: redis:latest # 基础镜像

container\_name: redis-master1 # 容器服务名

working\_dir: /config # 工作目录

environment: # 环境变量

- PORT=6391 # 跟 config/nodes-6391.conf 里的配置一样的端口

ports: # 映射端口，对外提供服务

- "6391:6391" # redis 的服务端口

- "16391:16391" # redis 集群监控端口

stdin\_open: true # 标准输入打开

networks: # docker 网络设置

redis-master:

ipv4\_address: 172.60.0.2

tty: true

privileged: true # 拥有容器内命令执行的权限

volumes: ["./config:/config"] # 映射数据卷，配置目录

entrypoint: # 设置服务默认的启动程序

- /bin/bash

- redis.sh

## 4.3 安全层面

### 4.3.1 cocofirewalld防火墙策略

这里的实现全部是由shell所写，为了方便开发或者初级运维接管服务器管理防火墙，因此将这些命令通过脚本整合成一个小程序。放在这里的目的和意义是帮助Docker部署应用端口的开启，可用的功能有

1. 开启防火墙已关闭的端口
2. 关闭防火墙已开启的端口
3. 关闭特定区间的防火墙端口
4. 开启特定区间的防火墙端口
5. SELinux控制
6. 防止SSH暴力破解服务器密码

其中上面第5项SELinux控制部分的功能有

1.查看SELinux是否开启(状态)

2.暂时关闭SELinux

3.永久关闭SELinux

4.永久开启SELinux

运行界面如图4.8，4.9所示：

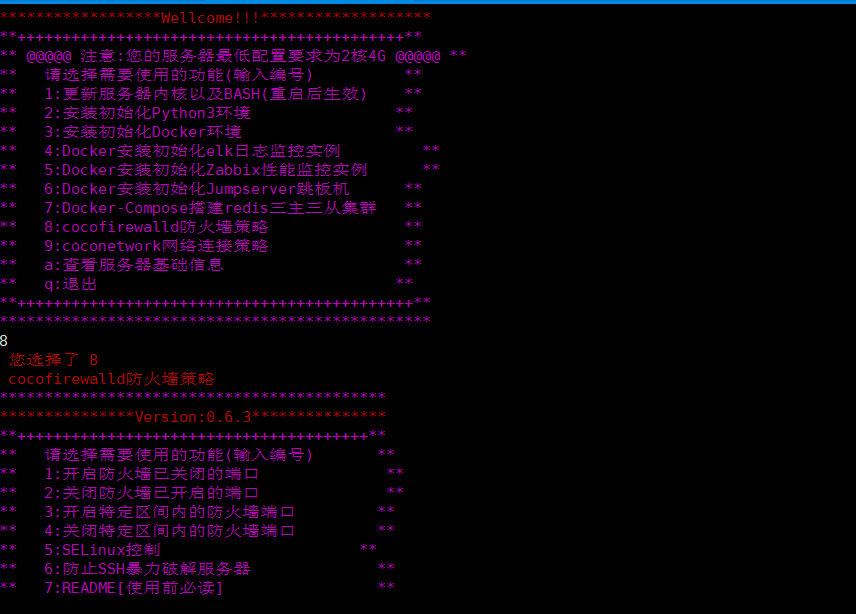


图 4.8防火墙策略

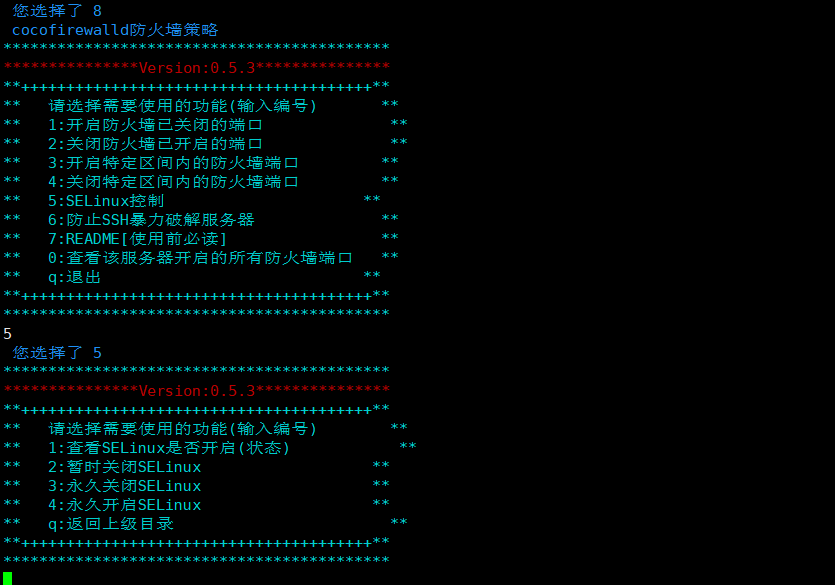


图 4.9 SELinux控制

### 4.3.2 coconetwork网络连接策略

修改或者查看服务器的网络，由于网络也是运维人员必须要注意的，因此这里使用shell配置网卡的一些必要信息。功能实现使用shell脚本，修改信息写入文件主要用的是sed命令以及正则表达式。放在这里的目的和意义是可以随时控制网络是否是静态IP还是动态IP，如果有这种需求的话，可以在运行完成本脚本的最后运行这个模块以很方便的修改网络，可用功能如下所示：

1. 查看网卡信息
2. 修改网卡(修改网卡为静态IP)
3. 修改网卡(修改网卡为动态IP)
4. 测试网络连通性和带宽
5. 修改网卡名称为eth0

运行界面如图4.10所示：

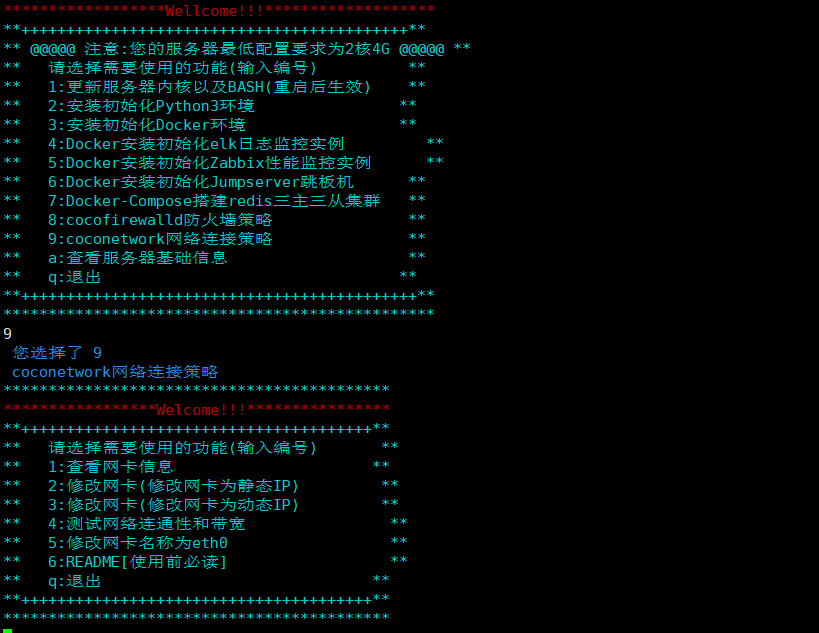


图 4.10 网络策略

# 第五章 系统具体实现

## 5.1 系统层面

系统初始化完成后，加载引入其他几个脚本进入菜单选项，但是初始化系统完成之后选择更新内核或者bash要重启服务器后才会生效。运行图5.1所示：

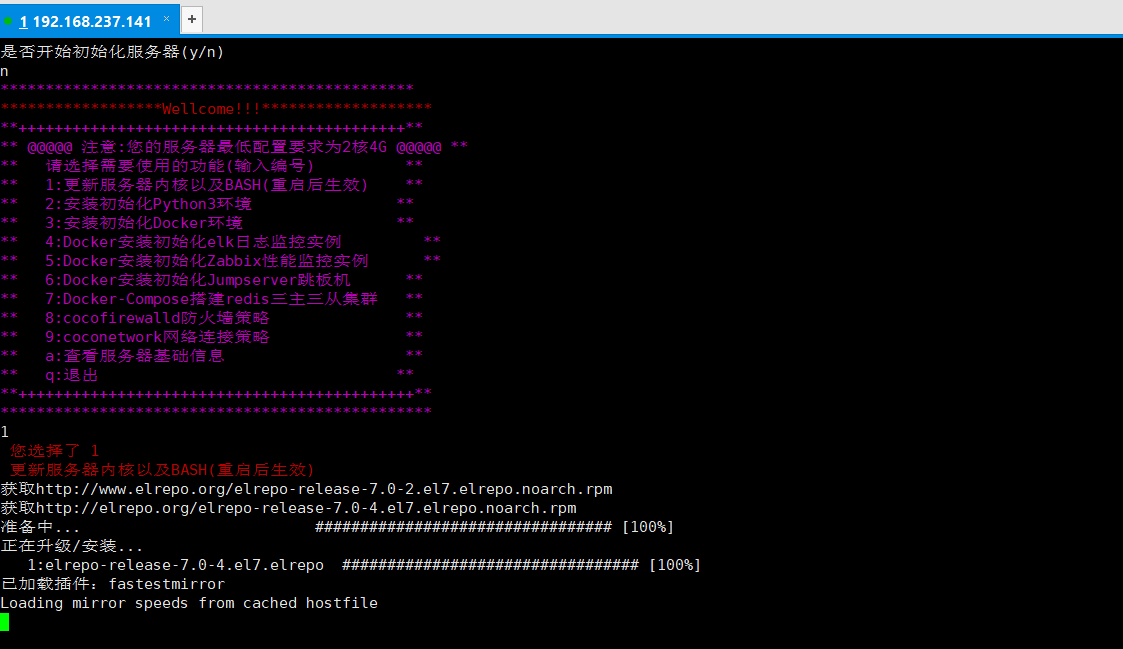


图 5.1 更新内核

重启验证：重启服务器发现内核和bash已经升级成功，查看结果图5.2：

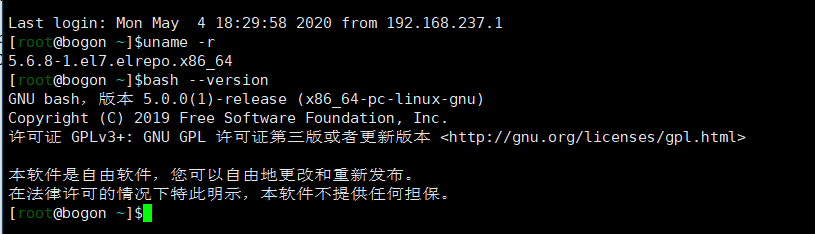


图 5.2 内核升级成功

验证Python3的pipenv虚拟环境，效果如图5.3所示：

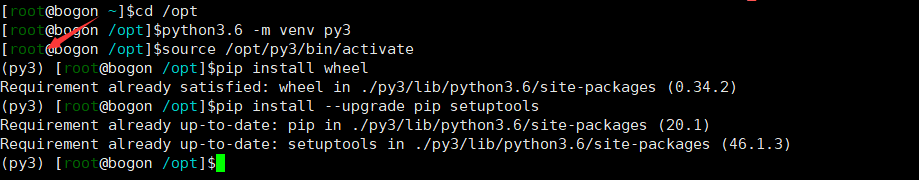


图 5.3 验证Python3环境

## 5.2 应用层面

这一部分主要是使用docker pull从docker hub上拉取docker的镜像然后运行，如图5.4所示：

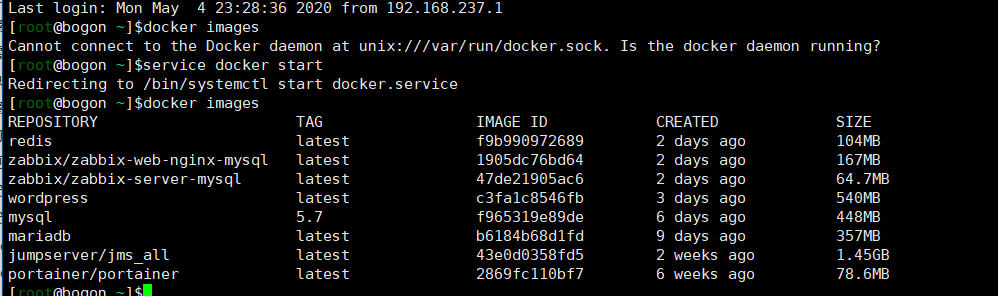


图 5.4 镜像

这部分的安装运行很简单，最终只要再网页上ip+端口访问即可，这里的其余效果图在上一章节中已经列出。

这里重点测试一下上面章节使用docker-compose.yml搭建的redis集群，首先使用docker exec -it redis-master1 /bin/bash 进入1号主节点容器，进行集群初始化。

初始化集群命令如下：

redis-cli --cluster create 192.168.237.148:6391 \

192.168.237.148:6392 \

192.168.237.148:6393 \

192.168.237.148:6394 \

192.168.237.148:6395 \

192.168.237.148:6396 \

--cluster-replicas 1

Redis操作部分命令

# 验证测试集群

# redis-cli -h 192.168.237.148 -p 6391

# set test 1

# redis-cli -c -h 192.168.237.133 -p 6391 set test 1

# redis-cli -c -h 192.168.237.133 -p 6395 get test

集群初始化激活效果如图5.5，5.6所示：

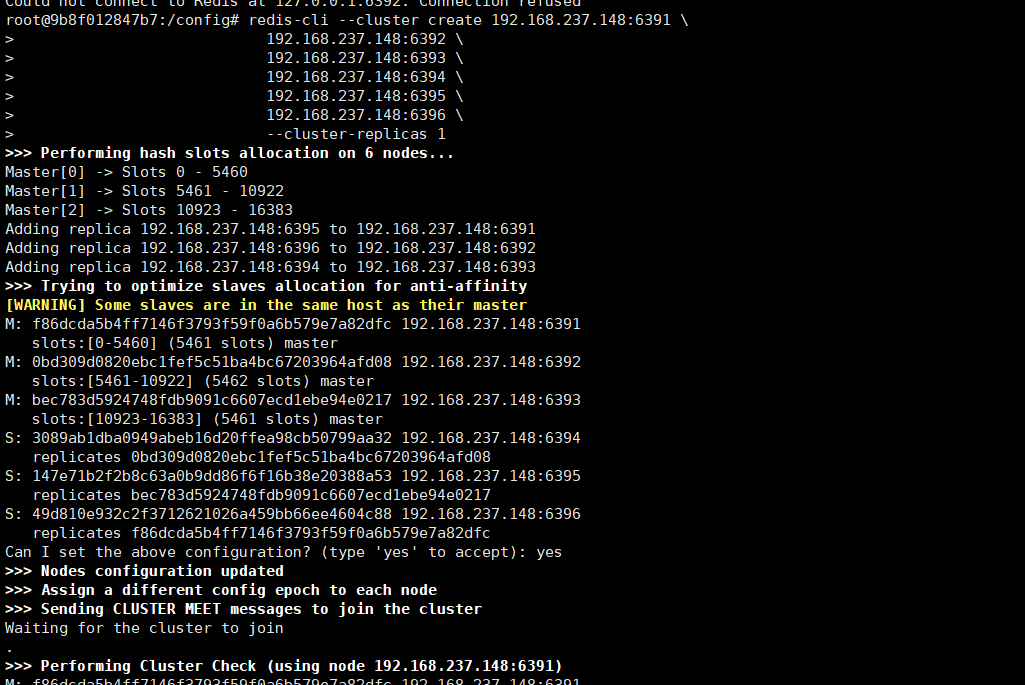


图 5.5 集群激活

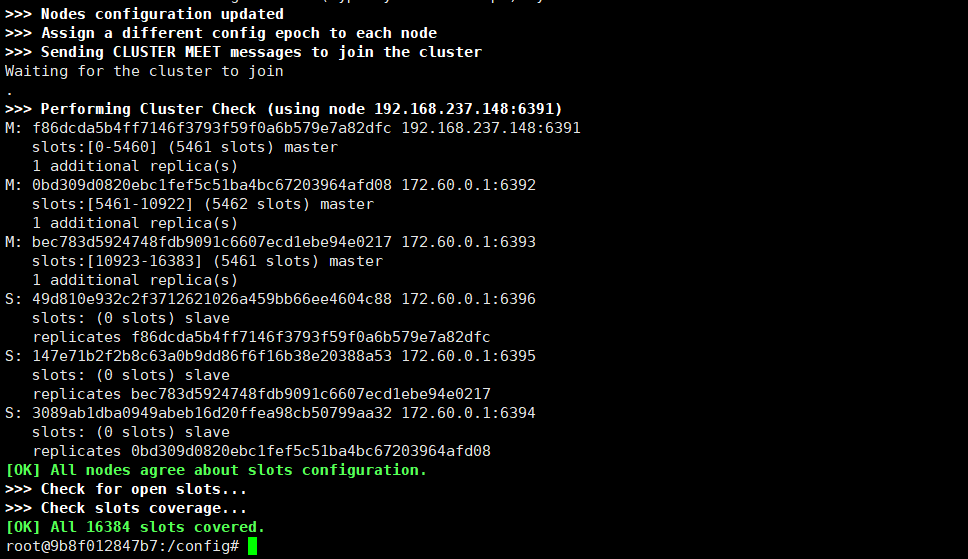


图 5.6 集群激活成功

查看集群信息如下图5.7：

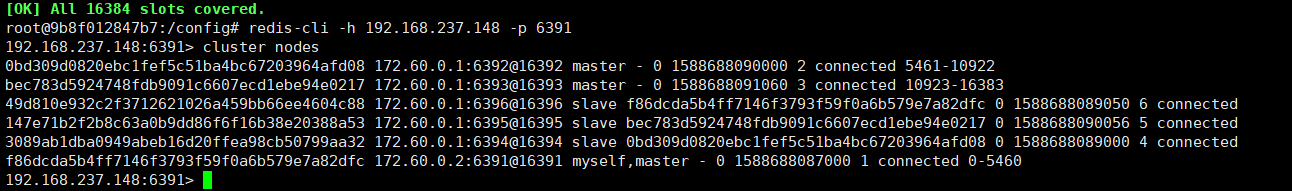


图 5.7 集群信息

普通模式连接：在6391上操作redis，set进去一个key-value键值对，由于 test 根据哈希槽计算，是分布在 6392 服务上。所以这里会提示转到 6392。效果图如5.8所示。

分别用集群其余节点获取6391上set进去的值，发现可以正常获取，效果图如图5.9所示。

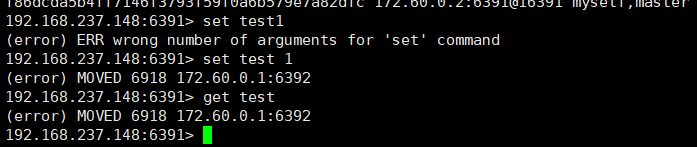


图 5.8 测试redis

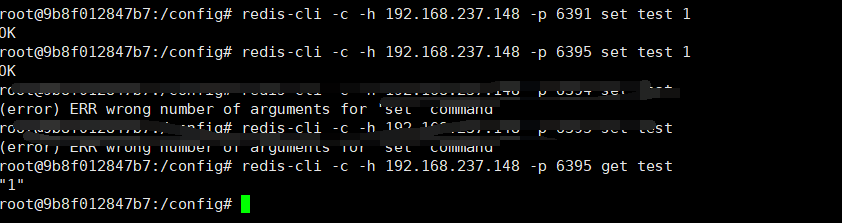


图 5.9 验证redis

## 5.3 安全层面

### 5.3.1 防火墙cocofirewalld

这部分功能是在防火墙开启的情况下使用的，如果没有开启首先会询问是否开启防火墙。选择y，进入cocofirewalld字符界面，效果如图5.11所示：

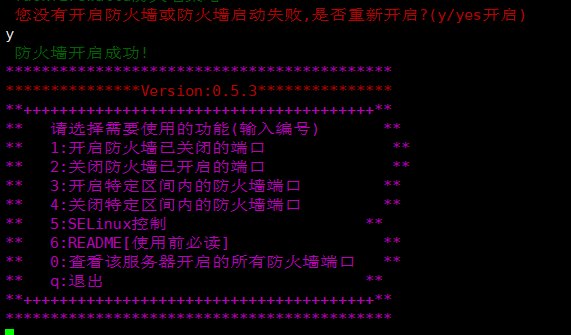


图 5.11 cocofirewalld主界面

选择1，任意开启一个tcp的8888端口，效果如图5.12所示。

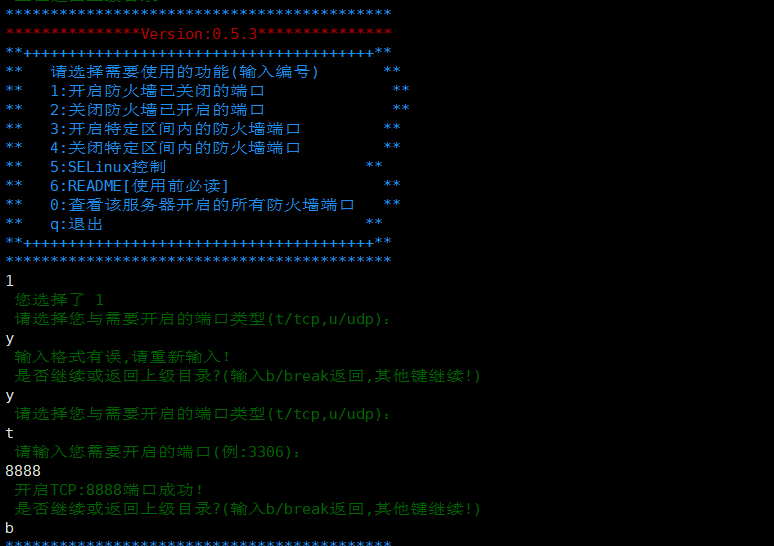


图 5.12

选择0，查看防火墙开启的端口，发现8888端口已经打开，效果图如图5.13：

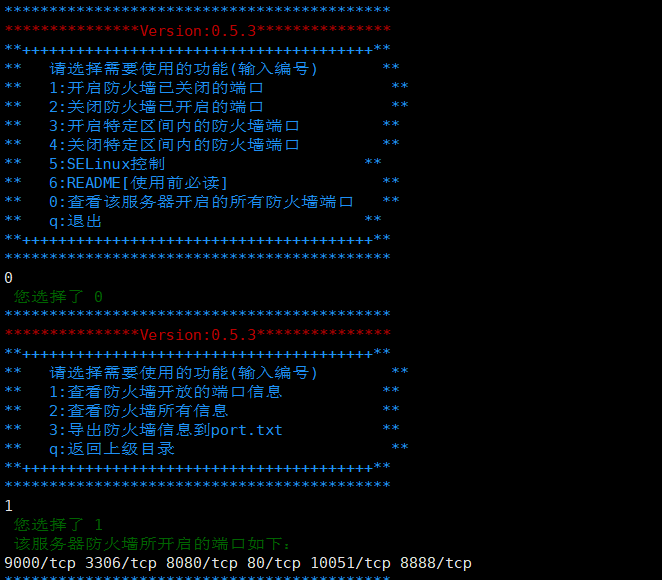


图 5.13

选择3，任意开启udp/10000-10100区间内的端口，图5.14，5.15：

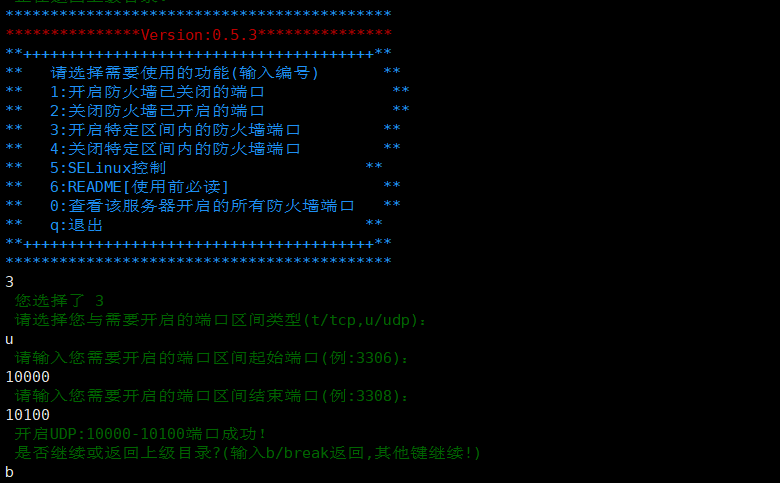


图 5.14



图 5.15

在这个界面内选择2查看端口开放的所有信息，如果选择导出防火墙信息到port.txt则会在程序运行目录下生成这个文件，并将防火墙信息写入。效果图如5.16，5.17所示：



图 5.16

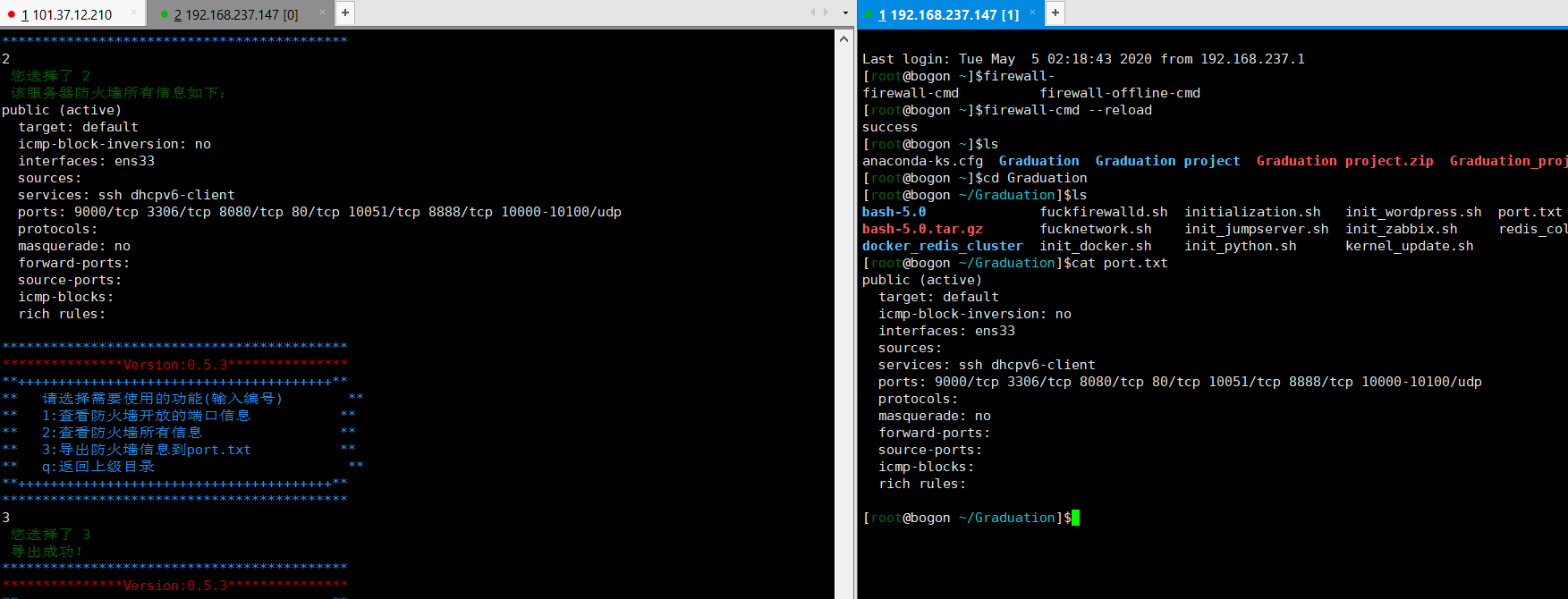


图 5.17

选择4，任意关闭上面打开的10000-10100/udp端口。按提示选择，发现端口已经关闭。效果图如5.18，5.19所示：

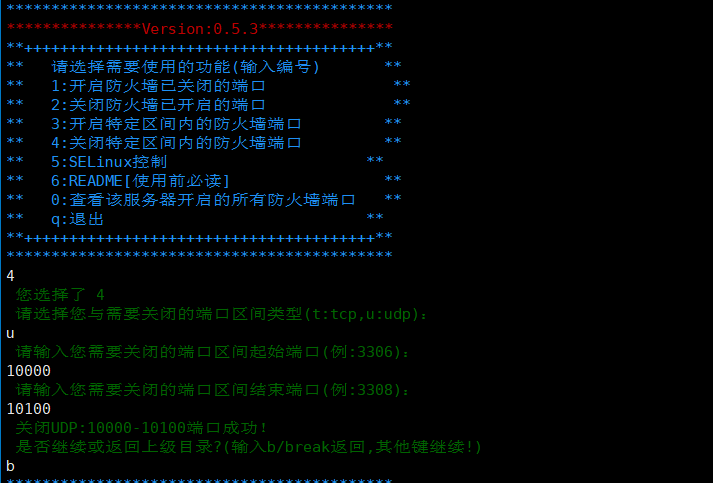


图 5.18



图 5.19

### 5.3.2 网络连接coconetwork

这部分为配置网络部分，由于动态IP是由DHCP分配的会有租期，可能IP隔段时间就会改变，所以我们会使用静态IP限制IP改变。运行本模块效果图如下图5.20所示：

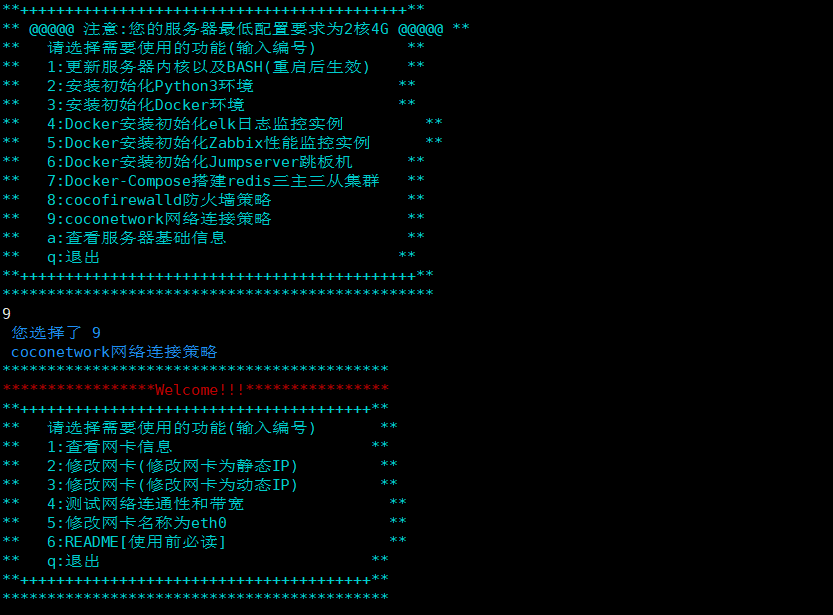


图 5.20 coconetwork界面

先查看一下IP和MAC地址。筛选IP使用正则表达式ip addr | grep 'state UP' -A2 | grep inet | egrep -v '(127.0.0.1|inet6|docker)' | awk '{print $2}' | tr -d "addr:" | head -n 1 | cut -d / -f1进行筛选出来正在使用的IP，egrep是扩展正则表达式。awk是一种类似于shell的编程语言，支持二维数组，这里用awk截取出来了正则表达式挑选好的第二列。MAC地址的筛选如下：ip addr | grep 'state UP' -A2 | grep link | egrep -v '(127.0.0.1|inet6|docker)' | awk '{print $2}，特别的地方是docker网桥分配的MAC地址也可以挑选出来，第一个则是本机的MAC地址。效果图为图5.21



图 5.21

选择2，修改IP为192.168.200.200做测试，使用nmcli查看发现IP已经设置为静态IP。这里也可以使用ip addr或者ifconfig查看ip。修改静态ip的本质是修改网卡的配置文件。效果图5.22，5.23：

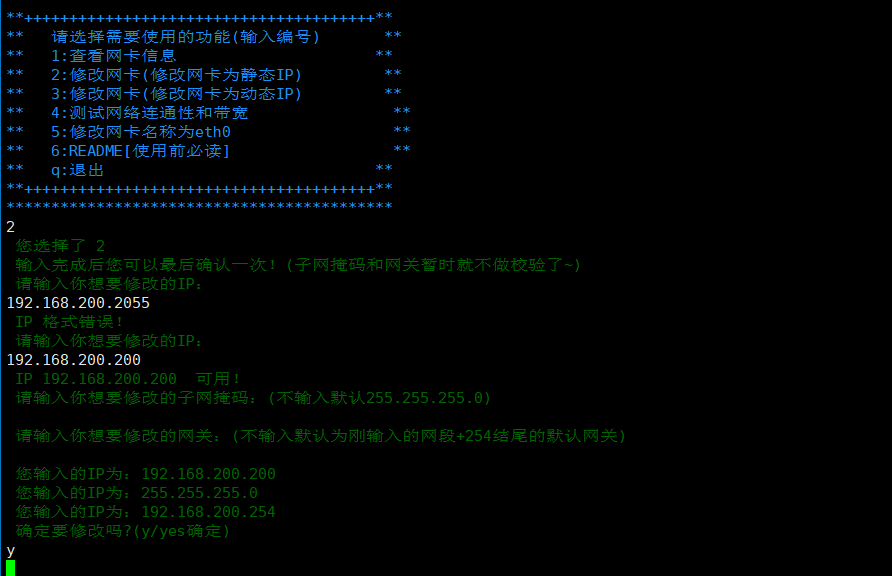


图 5.22

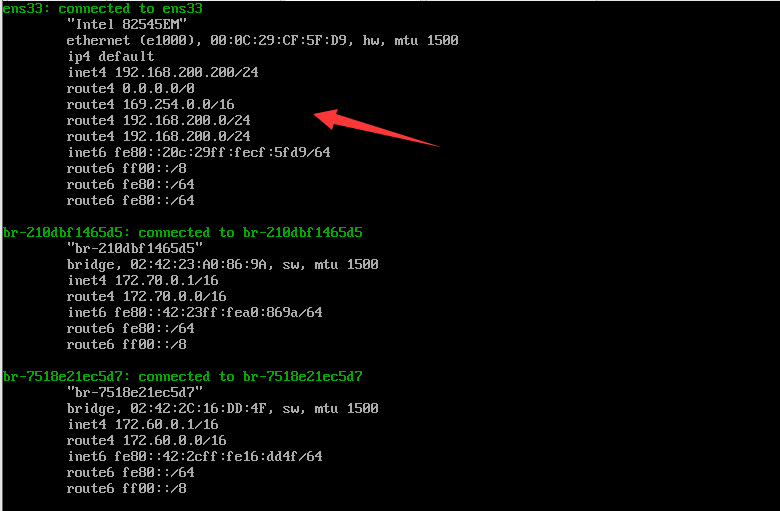


图 5.23

重新修改为动态IP。如图5.24：

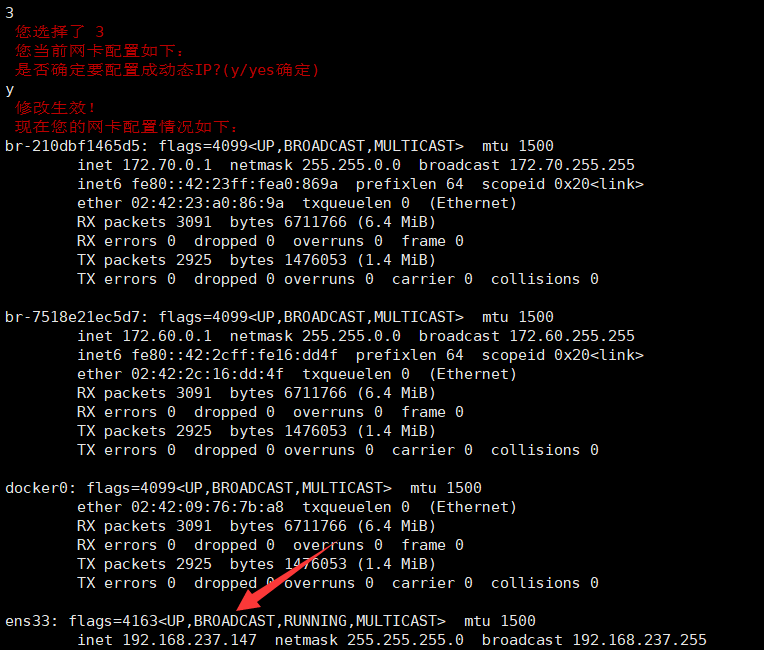


图 5.24

测试网络连通和检测网速，由于我的服务器速度比较快，就把这个模块拉到我的服务器上测试了。测试网络速度使用网上开源的一个Python测速程序。因此需要网络条件比较好的情况下测速。效果如图5.25所示：



图 5.25

CentOS7上把网卡的名称改成了ens开头的，因为CentOS7之前的服务器版本都使用的eth为网卡名称，不修改的话某些程序会出现莫名奇妙的问题。修改这里需要修改内核的一个参数，在/etc/default/grub中需要将GRUB\_CMDLINE\_LINUX后面加个参数net.ifnames=0。这里主要使用的是sed命令，sed -i.bak -r '/GRUB\_CMDLINE\_LINUX/s@(.\*)"$@\1 net.ifnames=0"@' /etc/default/grub替换/etc/default/grub中的参数。如图5.26所示：

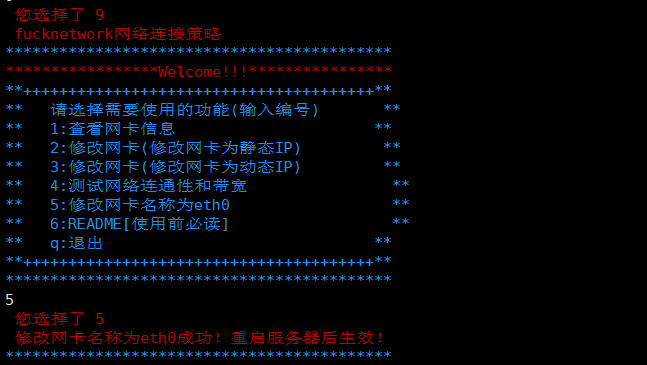


图 5.26

显示服务器网卡名是eth0。效果如图5.27

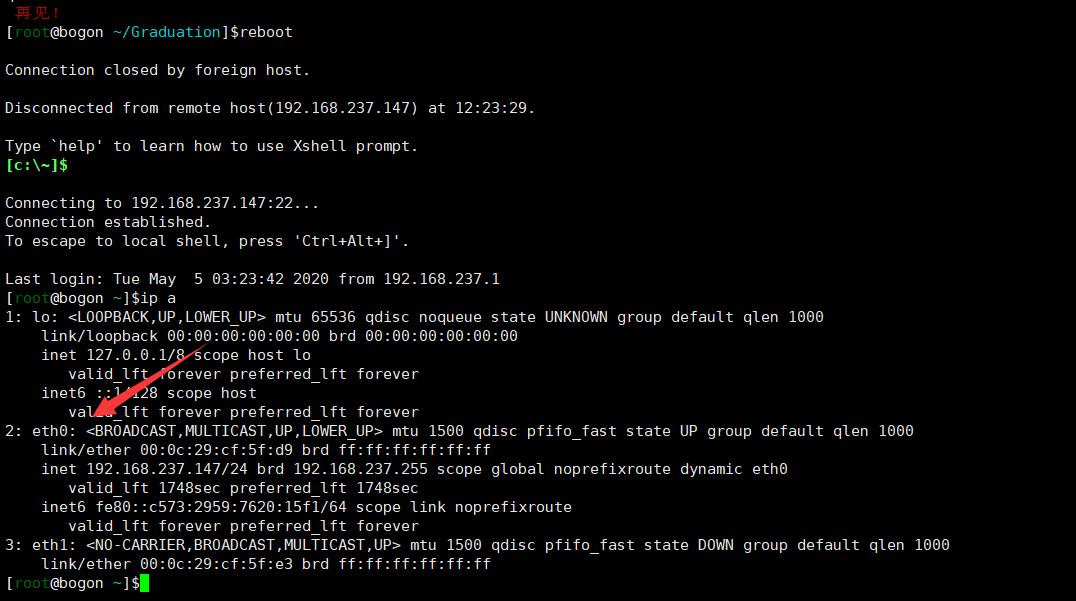


图 5.27

# 总 结

用了很长的时间，我终于完成了我的毕业设计及论文，从一开始拿到毕业设计课题的毫无头绪到最后的完成，其中我花费了许多精力和时间，在此期间我通过查找资料和不断地探索，最终完成了我的毕业设计。通过这一段时间对课题的研究，我对专业知识有了更深的认识。

由于我的专业就是计算机网络，虽然在实习期间一直做的软件开发，这次选择这个课题也是为了挑战一下自己，云时代已经很长时间了，公有云，私有云选型以及各种容器技术都是一个企业应该考虑的。容器技术的发展很快，最主流的指的的就是Docker，它需要比较的其实是现有的虚拟机技术，为什么叫容器而不叫虚拟机，就是因为它的轻量级。当然，并不是所有应用都适合容器技术。看业务，看生产，不能为了技术使用技术。在本次的课题中我有很多的不足之处，前期本来想使用OpenStack和Kubernetes去做IaaS与PaaS平台的对比，但是由于OpenStack在我的虚机上经常莫名奇妙的出现错误而放弃了，不过我相信我会在以后的学习中更加深入的研究“云”的。

# 参考文献

1. 刘志国《练就Linux系统高手教程》延边出版社 2006.1
2. 高性能Linux服务器构建实战
3. 白帽子讲web安全
4. 大型网站技术架构
5. Docker技术入门与实战
6. OpenStack开源云王者归来 云计算、虚拟化、Nova、Swift、Quantum与Hadoop
7. Zabbix企业级分布式监控系统
8. Linux云计算从入门到精通
9. 毛德操 胡希明 《Linux内核源代码情景分析》
10. 林慧琛《Red hat Linux服务器配置与应用》人民邮电出版社 2006.1
11. 岳 浩《Linux操作系统教程》机械工业出版社 2005.4
12. 社大鹏《Red hat Linux因特网服务器》中国水利水电出版社 2004.2
13. 冉林仓.Red Hat Linux 9编程开发与网络管理.北京：电子工业出版社

# 致 谢

通过本次毕业设计，我学会了很多的专业知识，我从我的导师身上学到了许多东西，在此感谢他细心，耐心的辅导。同样也感谢一些同学在设计中给我的帮助。

此外，我还要感谢我在大学上课时给我授课的每一位老师。你们不但教会了我计算机的基础知识，还给我鼓励和支持。就是因为你们的细心教导，才使我有了这个系统的制造基础知识，和对计算机设计的热爱。真的非常感谢你们。

最终我还要感谢学校这几年来的栽培和给我们提供便利的设施，让我们学到各类的知识，让我们成长，才能为社会做出贡献。