Docker 实战实验报告

组员：

张旭升 115034910181； 黄琰 015034910081； 侯小凤 015033910016； 林晨 p201602052；朱瀚超 115030990056

实验步骤：

1. **任选编程语言编写一个 web 应用，要求：**
   a) **暴露 URL 1（/cpu_info），返回当前系统的 CPU 信息。**
   b) **暴露 URL 2（/cpu_rate=$value）,可以通过这个 URL 来调整对系统 CPU 的消耗。**
   c) **将代码上传到 Github**
   d) **在代码路径下包含一个 dockerfile 文件（通过这个来制作 Docker 镜像）**

在实验中,我们选择 Python 作为编程语言,Django(https://www.djangoproject.com/)作为 Web 开发框架。

a) /cpu_info 接口通过 python 的 psutil 模块实现。此模块可以方便的获取系统性能信息。cpu_count()函数获取了 CPU 核心数,cpu_percent()函数获取 CPU 占用率,cpu_stat()函数则列出了一些 cpu 的其他相关信息。返回数据为 json 格式，举例如下：

{"cpu_stats":"scpustats(ctx_switches=8727343474,interrupts=3207016195,soft_interrupts=1449230002,syscalls=0)","cpu_count":"16","cpu_rate": "62.5"}

b) /cpu_state=$value,通过这个接口，可以设置应用对系统 CPU 的消耗。该接口的实现过程中解决了以下问题:

- **空循环实现 CPU 占用率的消耗**

这个消耗是通过空循环对 CPU 占用满来实现的。在一段时间内，

CPU 占用率 = CPU 被占用的时间 / 总时间

在试验中，选取的机器主频为 2.6GHz，则 2.6*1000*1000*1000×2÷5 ,为了使 CPU 占用率保持稳定，在 0.01s 内进行一个运行/暂停周期。在 0.01s 内，如果设定占用率为 percent，那么设定的循环次数为等于 2.6*1000*1000*4*percent。但是由于 server 在运行当中还有调度消耗，所以在实际的设置中还有一些微调。

- **多进程调用实现多核 CPU 的利用**

在多核 CPU 的机器中，由于上述方法只计算了一个核的计算量，所以上述方法在多核机器上显示的 CPU 占用率要小于应当显示的 CPU 占用率。在试验中，选取的是 DELL POWEREDGE R630，显示的 cpu_count 即核心数为 16 个，所以在 CPU 占用消耗程序中，采用了 python 的 multiprocessing 模块，循环开启了 n=“核心数”个进程，进程自动分配到 16 个核上，实现了程序对 CPU 占用率的准确调用。

- **数据库读写+守护进程 实现 CPU 占用率控制和保持**

在问题中，如果将 CPU 占用率调控程序直接写到 Web 服务器的远程过程调用中，则无法保持状态，并且造成响应缓慢。解决办法是将 API 请求的 CPU 占用率指令写到数据库中。Django 默认 splite3 数据库，在数据库中建立了一个表格，里面有建立一行数据，其中一条即为 cpu_state。同时，开启一个 python 的守护进程，每隔几秒钟就读取一次数据库，同时将读取到的数据作为 percent 的取值。这样就实现了实时的 CPU 占用率的控制和保持。

c) 代码上传到张旭升的 github(账号为 alinery9289)，repo 为“CpuHomework”，链接为：https://github.com/alinery9289/CpuHomework。

d) 编写 Dockerfile

在本实验中,采用的系统为 ubuntu 系统,编程语言为 Python,同时用 pip 安装了 django 和 psutil 模块，开放端口 8000，编写 Dockerfile,具体见附录 1。

2. 任选编程语言编写一个 web 应用，要求：

a) 在阿里云容器 hub 创建一个 Github 类型的镜像仓库

b) 通过自动构建生成一个镜像

a) 在阿里云开发者平台界面，点击创建镜像仓库，创建了 cpuhomework 的镜像仓库。仓库地址为 registry.aliyuncs.com/alinery9289/cpuhomework。如下图：

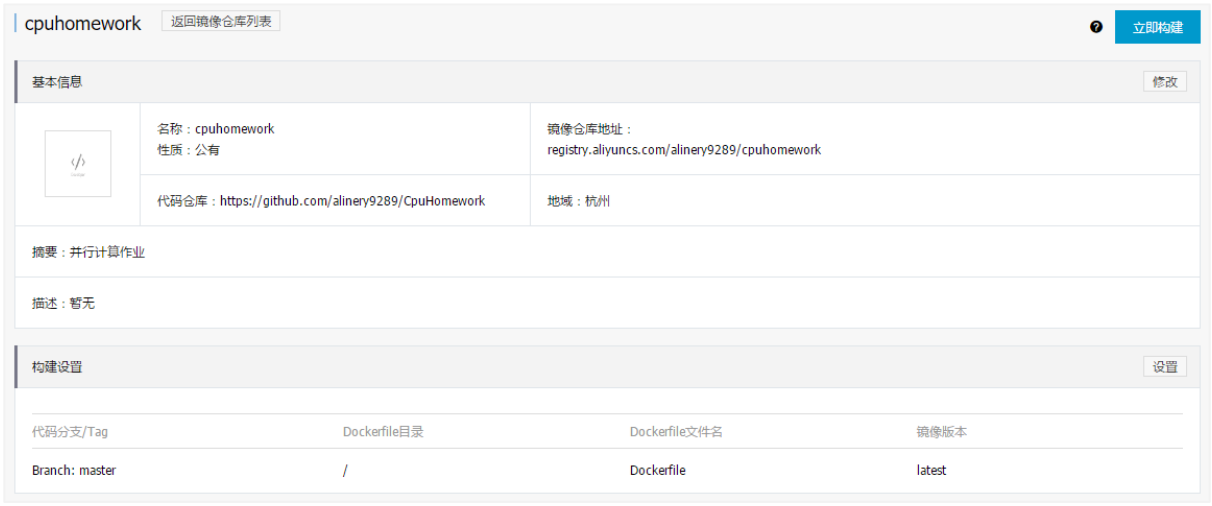

图 1 阿里云镜像仓库

b) 在点击立即构建时，在执行“RUN pip install psutil”(pip 安装 psutil 模块)命令时，发现出错。查看出错原因，发现 setup.py 安装过程中报错，猜测是阿里云和 pip 的官网网络连接问题，于是改用下载安装包的方式安装，仍然有问题，于是采用源码安装。猜测是 ubuntu 镜像的 python 环境中的 setuptools 版本不对的原因，于是先安装了 setuptools，并升级了 python-dev，最终成功安装。

另外，在本机上生成镜像中，并没有产生此类错误，可否检测一下问题的原因。

生成镜像的界面截图如下：

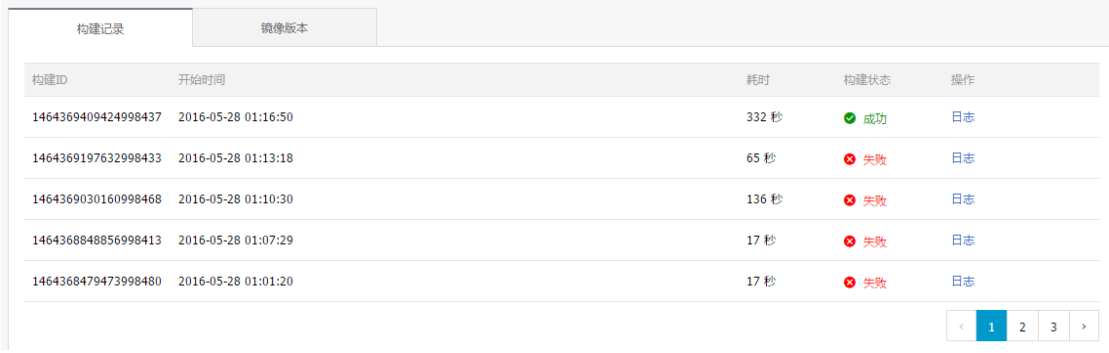

图 2 阿里云 docker 镜像生成

3. **在个人电脑或笔记本上通过 Docker 命令行运行这个镜像**

   a) **使用上一步生产的镜像，通过 docker run 启动两个容器 A 和 B，并保证 A 对服务器的 CPU 占用率是 B 的 2 倍。（通过调节应用的 URL 参数和 docker run 的 cpu 参数来观察效果）**

a) 使用 docker pull registry.aliyuncs.com/alinery9289/cpuhomework 命令拉取了在阿里云上生成的镜像。并使用

docker run -p 8000:8000 -d registry.aliyuncs.com/alinery9289/cpuhomework:latest

docker run -p 8001:8000 -d registry.aliyuncs.com/alinery9289/cpuhomework:latest

分别将应用的端口映射到了服务器的 8000 端口和 8001 端口。

4. **创建 Docker Compose file 将这两个容器编排起来。**

   a) **要求保证 A 在 B 之前启动。**

   b) **在本地可以通过执行 docker-compose up 运行。**

   c) **提交 compose.yml 到 Github 相同的 repo。**

a) 在生成 docker-compose file 时，要使 A 在 B 之前启动，需要让 B 的-link 为 A

b) 生成后可以通过执行 docker-compose up 运行

c) compose.yml 见附录 1，已提交到 https://github.com/alinery9289/CpuHomework。

实验结果：

1. 只启动一个 Web Server 时，设置/cpu_state=20 为例：

   设置完成后，页面显示：

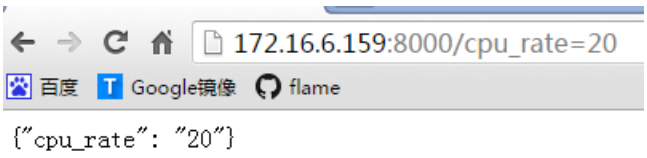

图 3 设置 cpu_rate

应用占用 20%左右，而由于系统自身仍占用部分，所以查询 cpu_info，显示为 24%左右，如下图：

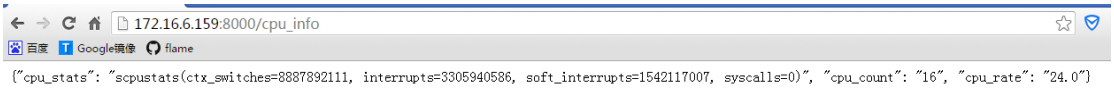

图 4 cpu_info API 查询系统 CPU 占用率

用 ubuntu 下的 top 命令查询，发现共启动了 16 个消耗 CPU 的 Python 程序（还有一个 Python 为基于 Python 语言的 Web 服务器），对应 16 个核，平均每个占用率在 20%左右。如下图所示：

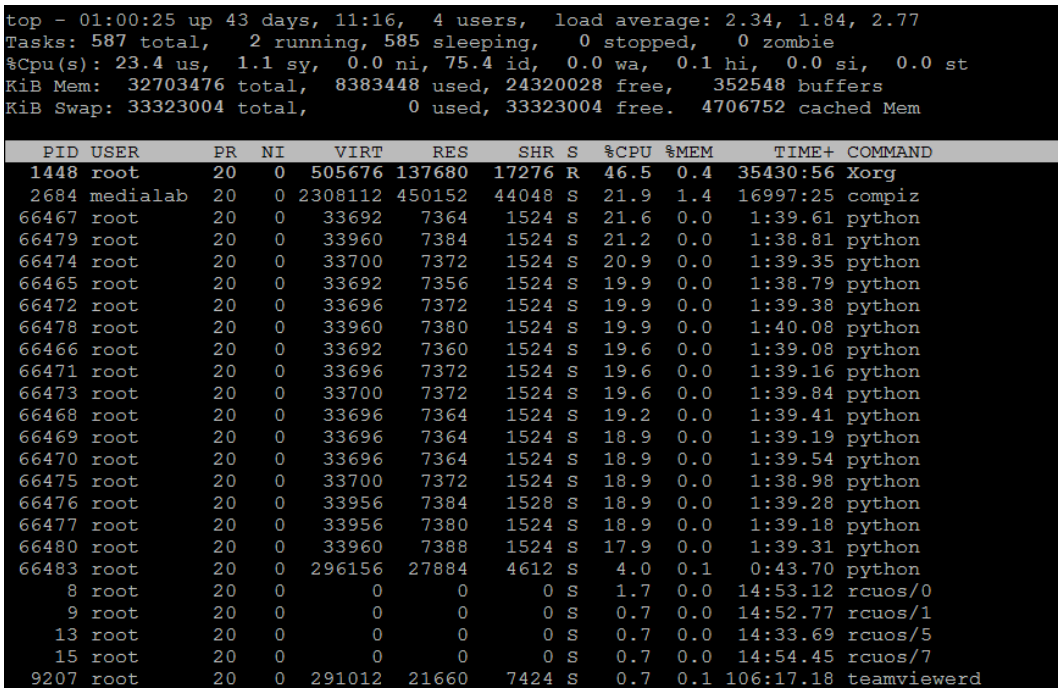

图 5 top 命令查询 CPU 占用率

设置/cpu_rate=70, top 查询后如图：

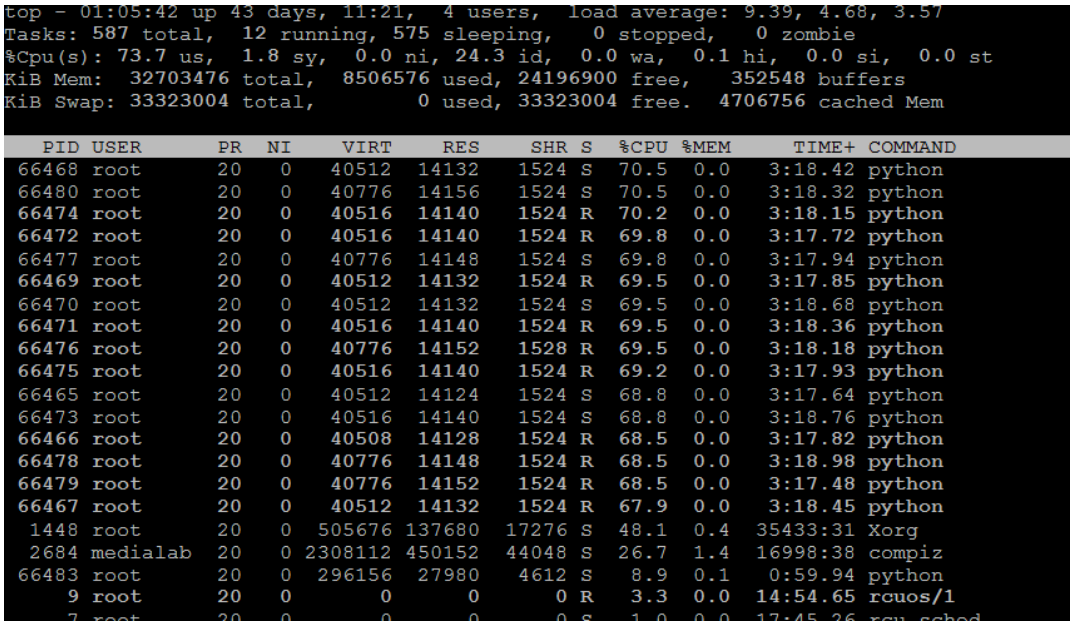

图 6 top 命令查询 CPU 占用率

2. 只启动一个 Web Server 时，通过 docker-compose up 启动，启动效果如下图，分别访问了两个 web 应用的 API。

```
[00:31][root@medialab] CpuHomework (master) # docker-compose up
Creating cpuhomework_A_1
Creating cpuhomework_B_1
Attaching to cpuhomework_A_1, cpuhomework_B_1
A_1   | [28/May/2016 17:10:39] "GET /cpu_info HTTP/1.1" 200 152
B_1   | [28/May/2016 17:10:57] "GET /cpu_info HTTP/1.1" 200 152
B_1   | [28/May/2016 17:11:27] "GET /cpu_rate=15 HTTP/1.1" 200 18
A_1   | [28/May/2016 17:11:55] "GET /cpu_rate=30 HTTP/1.1" 200 18
```

图 7 compose up 启动两个 web 应用

从上图可以看出，A 先启动，B 后启动。且设置了 A 的 CPU 占用率为 B 的 CPU 占用率的两倍。得到的效果如下图：

```
top - 01:18:36 up 43 days, 11:34,  4 users,  load average: 8.53, 8.38, 6.49
Tasks: 608 total,  11 running, 597 sleeping,   0 stopped,   0 zombie
%Cpu(s): 61.6 us,  2.9 sy,  0.0 ni, 35.3 id,  0.0 wa,  0.2 hi,  0.0 si,  0.0 st
KiB Mem:  32703476 total,  8534128 used, 24169348 free,   352548 buffers
KiB Swap: 33323004 total,        0 used, 33323004 free.  4707088 cached Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 1448 root      20   0  505676 137680  17276 S  50.2  0.4  35439:51 Xorg
66727 root      20   0   34836   8492   1524 S  37.4  0.0   2:28.47 python
66719 root      20   0   34828   8476   1524 S  37.1  0.0   2:27.47 python
66725 root      20   0   34832   8488   1524 R  37.1  0.0   2:27.72 python
66729 root      20   0   34836   8492   1524 S  36.7  0.0   2:27.83 python
66720 root      20   0   34828   8480   1524 S  36.4  0.0   2:28.08 python
66723 root      20   0   34832   8484   1524 S  36.4  0.0   2:27.65 python
66724 root      20   0   34832   8484   1524 R  36.4  0.0   2:26.43 python
66733 root      20   0   35096   8504   1524 S  36.4  0.0   2:27.72 python
66734 root      20   0   35096   8508   1524 S  36.4  0.0   2:26.70 python
66722 root      20   0   34832   8480   1524 R  36.1  0.0   2:26.61 python
66726 root      20   0   34832   8488   1524 S  36.1  0.0   2:26.70 python
66730 root      20   0   35092   8500   1528 R  36.1  0.0   2:27.62 python
66732 root      20   0   35096   8496   1524 S  36.1  0.0   2:27.66 python
66721 root      20   0   34828   8480   1524 R  35.8  0.0   2:26.53 python
66728 root      20   0   34836   8492   1524 S  35.8  0.0   2:26.21 python
66731 root      20   0   35096   8496   1524 S  35.8  0.0   2:26.75 python
 2684 medialab  20   0 2308112 450152  44048 S  23.3  1.4  17001:50 compiz
66780 root      20   0   33132   6672   1524 S  21.6  0.0   1:33.48 python
66782 root      20   0   33388   6684   1528 R  21.6  0.0   1:31.09 python
66775 root      20   0   33128   6664   1524 S  21.3  0.0   1:31.17 python
66776 root      20   0   33128   6668   1524 S  21.3  0.0   1:29.94 python
66779 root      20   0   33128   6672   1524 S  21.3  0.0   1:30.08 python
66784 root      20   0   33388   6684   1524 R  21.3  0.0   1:30.64 python
66786 root      20   0   33392   6688   1524 S  21.3  0.0   1:30.30 python
66787 root      20   0   33392   6688   1524 R  21.3  0.0   1:30.15 python
66789 root      20   0   33392   6692   1524 S  21.3  0.0   1:30.44 python
66777 root      20   0   33128   6668   1524 S  21.0  0.0   1:29.90 python
66778 root      20   0   33128   6672   1524 S  21.0  0.0   1:30.37 python
66781 root      20   0   33132   6676   1524 R  21.0  0.0   1:29.68 python
66783 root      20   0   33388   6684   1528 R  21.0  0.0   1:30.21 python
66788 root      20   0   33392   6692   1524 S  21.0  0.0   1:30.07 python
66790 root      20   0   33396   6696   1524 S  21.0  0.0   1:29.83 python
66785 root      20   0   33392   6684   1524 S  20.7  0.0   1:30.21 python
66738 root      20   0  296156  28896   4612 S   7.9  0.1   0:31.76 python
66793 root      20   0  222432  29296   4612 S   7.5  0.1   0:31.16 python
   19 root      20   0       0      0      0 S   2.6  0.0   6:43.97 rcuos/11
   11 root      20   0       0      0      0 S   1.3  0.0  14:49.32 rcuos/3
   13 root      20   0       0      0      0 S   1.3  0.0  14:40.26 rcuos/5
    7 root      20   0       0      0      0 S   1.0  0.0  17:50.76 rcu_sched
   12 root      20   0       0      0      0 S   1.0  0.0  14:47.34 rcuos/4
```

图 8 top 命令查看系统资源消耗图

从图中可以看出，系统的资源消耗率平均值为 35%的为 A 服务器控制的，平均值为 20%的为 B 服务器控制的，两者相对于设定值均有 5%的增加。这是由于服务器只有 16 核，而两个服务器占用了 32 个进程，CPU 核心对于进程的切换时也消耗了较多 CPU 资源。最后

的两个 python 程序占用 7.5%左右，为两个 Web 服务器。

实验结论和总结：

本小组完成了作业的前四项内容。成功的实现了通过 API 查询 CPU 消耗和控制应用对 CPU 的消耗。了解了 Docker 和 Github 的基本使用，同时也对在阿里云上结合两者，更高效的使用有了一定的理解。
本次作业对于零基础的我们是一项非常大的挑战，最终凭借着小组的团结合作，和不懈的努力完成了此次作业，受益匪浅。在以后的软件设计和系统搭建中，docker 可以成为一个非常有用的工具，让系统更加的健壮，可移植和可拓展。
同时也谢谢阿里云的老师，这是一个非常棒的课程。

附录 1：

1. Dockerfile：

```
#cpu manege
#based ubuntu,python,django,psutil
FROM ubuntu

MAINTAINER zhangxusheng sdzhangxusheng@163.com

RUN apt-get update && apt-get install -y python-pip && apt-get install -y vim-gtk
RUN pip install django
RUN pip install setuptools
RUN apt-get install -y python-dev
COPY psutil-4.2.0.tar.gz /opt/psutil-4.2.0.tar.gz
RUN pip install /opt/psutil-4.2.0.tar.gz
COPY /CPUManage /opt/CPUManage

EXPOSE 8000
WORKDIR /opt/CPUManage
CMD python manage.py runserver 0.0.0.0:8000 & python test.py
```

2. docker-compose.yml

```
B:
    image: "registry.aliyuncs.com/alinery9289/cpuhomework:latest"
    ports:
        - "8001:8000"
    links:
        - A

A:
    image: "registry.aliyuncs.com/alinery9289/cpuhomework:latest"
    ports:
        - "8000:8000"
```