# Docker预研报告

本文记述了应用Docker容器技术构建一个简单地分布式应用的过程，以及在构建应用过程中对Docker容器技术的理解。

## 基本概念

### Docker引擎

Docker引擎（Docker Engine1）是一个客户端/服务器结构的应用程序，包含两个组件：

1. 服务守护进程（Docker Daemon）
2. 命令行客户端（Command Line Interface）

服务守护进程是操作系统中的一个守护进程，默认开机启动。命令行客户端则是以docker命令的形式提供。如图 1，命令行客户端通过一套REST API访问服务守护进程，管理镜像、容器、网络、数据卷等资源。



图 1 Docker引擎

### Docker仓库

Docker仓库（Docker Registry2）是一个用来存储以及分发镜像的服务侧应用程序。Docker引擎和仓库之间的关系如图 2所示。命令行客户端提供build命令构建镜像，提供push、pull命令向仓库推送、拉取镜像，提供run命令根据镜像生成容器。命令行客户端只是所有这些服务的请求者，而服务守护进程才是这些服务的执行者。若有需要，可以根据REST API文档构建自己的客户端程序。Portainer3就是一款开源的Docker Web客户端程序。本文使用Portainer辅助展示应用构建过程。

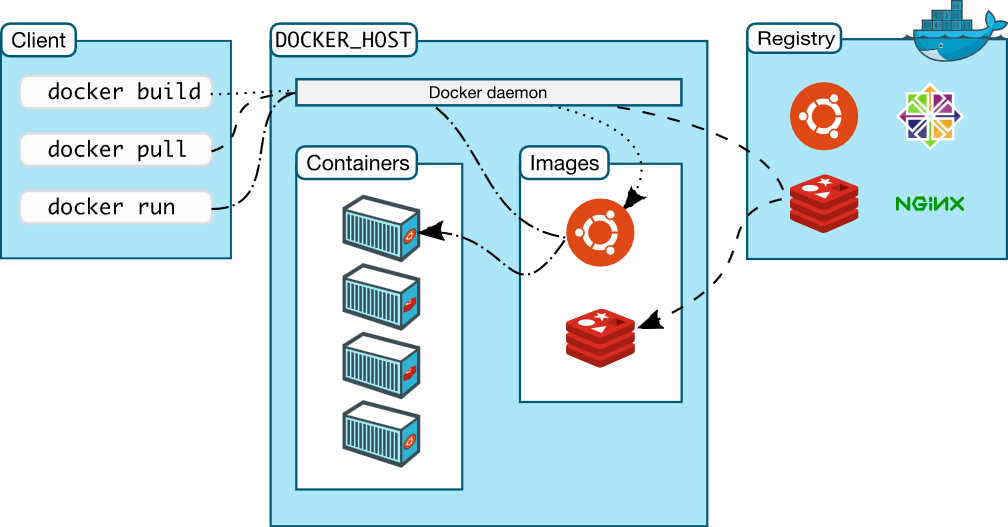


图 2 Docker Architecture

### Docker Swarm

Docker Swarm3是Docker内生支持的集群模式。一组运行了Docker引擎的主机通过内在的通信机制可以构成一个计算集群。集群中的每台主机被称为节点（Swarm Node）。节点有两种类型：管理员（Swarm Manager）和工人（Swarm Worker）。管理员类型的节点可以管理集群中的节点、服务和容器等计算资源，可以将工人类型的节点提升为管理员，也可以将其他管理员节点下降为工人。

管理员类型的节点通过命令docker swarm实现对集群的管理。而工人节点只是单纯地为集群提供计算能力，除了管理自己（比如加入、退出集群等）之外，不能访问集群以及其他节点的任何信息。集群相关的命令有docker swarm、docker service、docker node。

此外，Docker还推出了Docker Compose、Docker Machine两款工具。下面简单介绍应用构建过程中对这些工具使用的理解。

## 工具介绍

在下文构建基于Docker容器技术的分布式应用的过程中，使用了Docker命令行、Docker Compose和Docker Machine三款工具。Docker Compose和Docker Machine需要独立安装，并不与Docker引擎一起安装。Portainer虽然功能强大，只是为方便展示构建过程。

### Docker命令行

Docker命令行4也就是docker命令，主要功能有创建和管理镜像等（图 1）、创建和管理swarm集群。常用的命令有（未列出选项参数）：

1. docker build，创建镜像
2. docker push，将镜像推送到仓库中
3. docker pull，从仓库中拉取镜像
4. docker search，在仓库中搜索镜像
5. docker image，管理本节点镜像
6. docker container，管理本节点容器
7. docker run，根据镜像创建容器
8. docker swarm init，节点以管理员的角色开启swarm模式
9. docker swarm join，节点以管理员或者工人的角色加入swarm集群
10. docker swarm leave，节点脱离swarm集群
11. docker service，管理swarm集群中运行的多容器多主机服务
12. docker node，管理swarm集群中的节点

### Docker Compose

Docker Compose5是Docker推出的用于定义、运行和管理多容器服务的工具，以命令行工具docker-compose的方式提供使用。多容器服务是指一个服务由若干容器子功能组成，这些容器子功能互相协作完成服务的整体功能。

创建多容器服务的第一步是定义YAML服务编排文件（Compose File9）。服务编排文件定义了各容器的镜像、容器使用的计算资源（cpu、内存等）、容器使用的存储以及网络资源。在定义好服务编排文件之后，在命令行模式下进入到文件保存目录，执行docker-compose up命令启动服务。docker-compose还提供了其他的子命令可以方便地管理服务。

简单理解，Docker Compose以docker-compse命令和YAML服务编排文件的方式简化了多容器服务的定义和管理，提供了比docker命令原语更高一级的容器服务操作接口。

### Docker Machine

Docker Machine6是Docker推出的方便于创建、管理安装了Docker引擎的虚拟机以及操作虚拟机的工具，以命令行工具docker-machine的方式提供使用。常用的命令有（未列出选项参数）：

1. docker-machine create --driver，以指定的虚拟机技术创建虚拟主机
2. docker-machine ssh，使用SSH协议登陆虚拟机或者在虚拟机环境执行命令

### Portainer

Portainer7是一个开源的Docker环境管理Web客户端程序。Portainer可以管理安装了Docker引擎的主机、添加Docker仓库（包括Docker Hub和私有仓库）、上传下载镜像到Docker仓库、管理Swarm集群以及管理用户。本文主要使用Portaniner展示构建过程中Docker主机以及Swarm集群的变化。

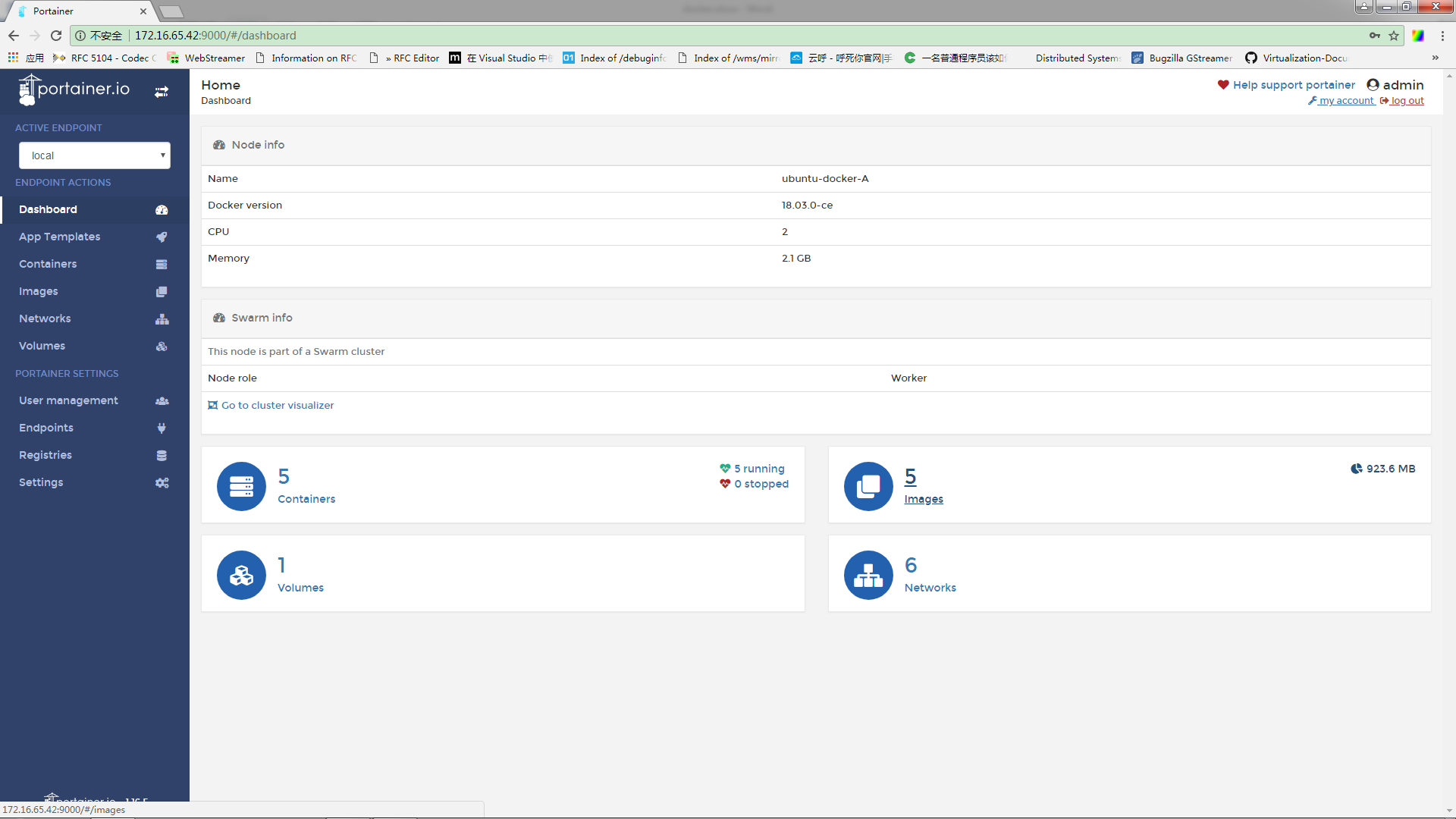


图 3 Portainer Web UI

## 镜像与服务定义

基于Docker容器技术的分布式应用构建涉及两种定义文件：镜像构建定义文件Dockfile、YAML服务编排文件（.yml）。Docker对这两种定义文件的格式提供了详细的参考文档，这里作简单介绍。

### Dockerfile

Dockerfile8是Docker提供的用于定义镜像构建过程的文本文件。每次创建新的Docker镜像，用户首先需要编写符合规格的Dockerfile文件。该文件是由若干Dockerfile命令组成的，并且必须以FROM命令开始。FROM命令指定了目标镜像构建的基础镜像（Base Image）。基础镜像为后续Dockerfile命令提供了上下文环境。任何有效的镜像都可以作为基础镜像。Dockerfile定义好之后，用户使用docker build创建镜像。

### YAML Compose File

见Docker Compose中的介绍（见上方）。

## 分布式应用构建

下文通过一个简单的分布式应用的构建展示如何使用Docker开发分布式应用。该分布式应用由两个服务组成：Web服务、数据库服务。Web服务基于FLASK框架（一个Python编写的Web微框架）实现，数据库服务采用Redis。Redis已经推出官方Docker镜像，需要创建Web服务镜像（见下方）。

该应用已部署在小组长拷环境中，可通过Portainer（http://172.16.65.42:9000）查看。

### 环境搭建

分布式应用运行在两台计算机组成的Docker Swarm集群中。配置如下，

1. 主机：ubuntu-docker-A（172.16.65.42）、ubuntu-docker-C（172.16.65.44）
2. 操作系统：Ubuntu 16.04.2 x86\_64 （xenial）
3. Docker：18.03.0~ce-0~ubuntu\_amd64

### 私有仓库搭建

通过Docker发布的仓库镜像registry，搭建私有镜像仓库。

1. 通过PuTTY11 SSH登陆主机ubuntu-docker-A
2. 运行registry镜像启动仓库容器

docker run -d -p 5000:5000 --restart=always --name private-hub \

-e REGISTRY\_STORAGE\_DELETE\_ENABLED=true \

-v /var/docker/registry:/var/lib/registry registry

1. 访问<http://172.16.65.42:5000/v2/_catalog>，验证仓库容器创建成功（如图 4）。

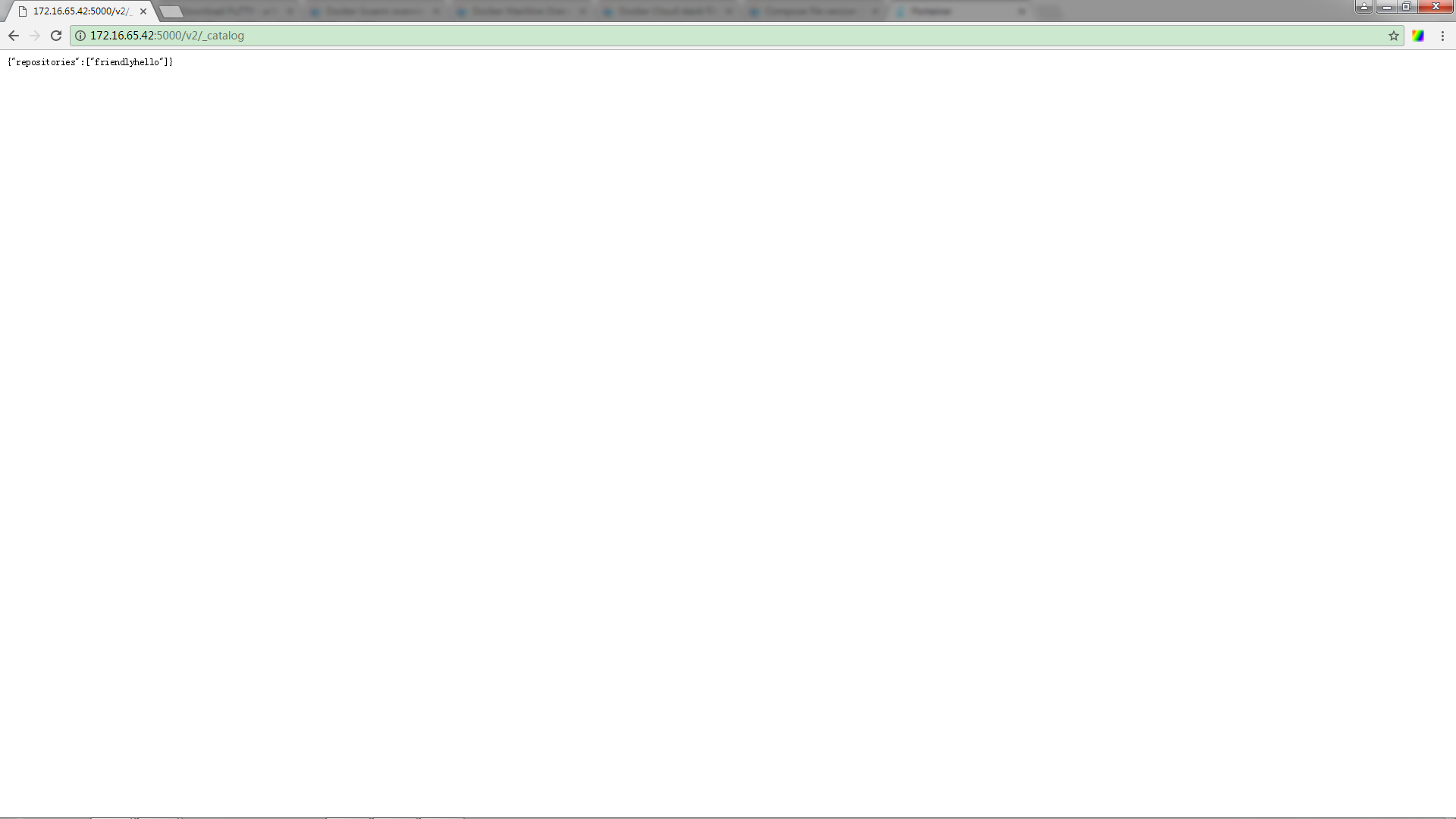


图 4 私有镜像仓库

1. 运行hyper/docker-registry-web镜像启动仓库内容浏览容器

docker run -d -p 8080:8080 --restart=always --name private-hub-webui \

--link private-hub -e REGISTRY\_NAME=172.16.65.42:5000 \

-e REGISTRY\_URL=http://172.16.65.42:5000/v2 hyper/docker-registry-web

1. 访问<http://172.16.65.42:8080/>，验证仓库内容浏览容器创建成功（如图 5）。

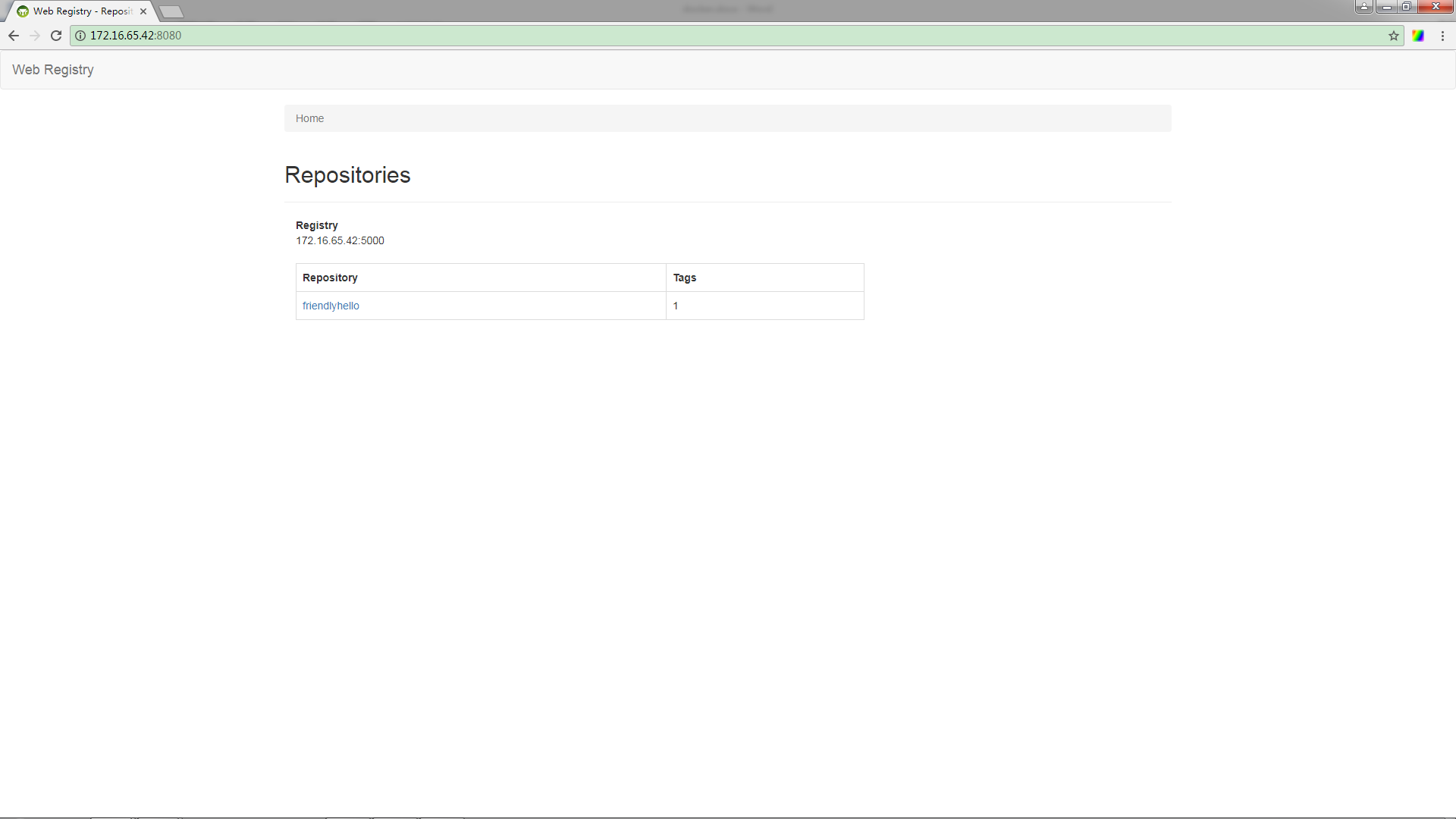


图 5 私有仓库浏览Web UI

### Portainer管理前端

1. 通过PuTTY SSH登陆主机ubuntu-docker-A
2. 运行portainer/portainer镜像启动Portainer容器

docker run -d -p 9000:9000 --restart always --name docker-webmanager \

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

-v /var/docker/portainer:/data portainer/portainer

1. 访问<http://172.16.65.42:9000>，进入Portainer登陆主页（用户名admin密码admin123）

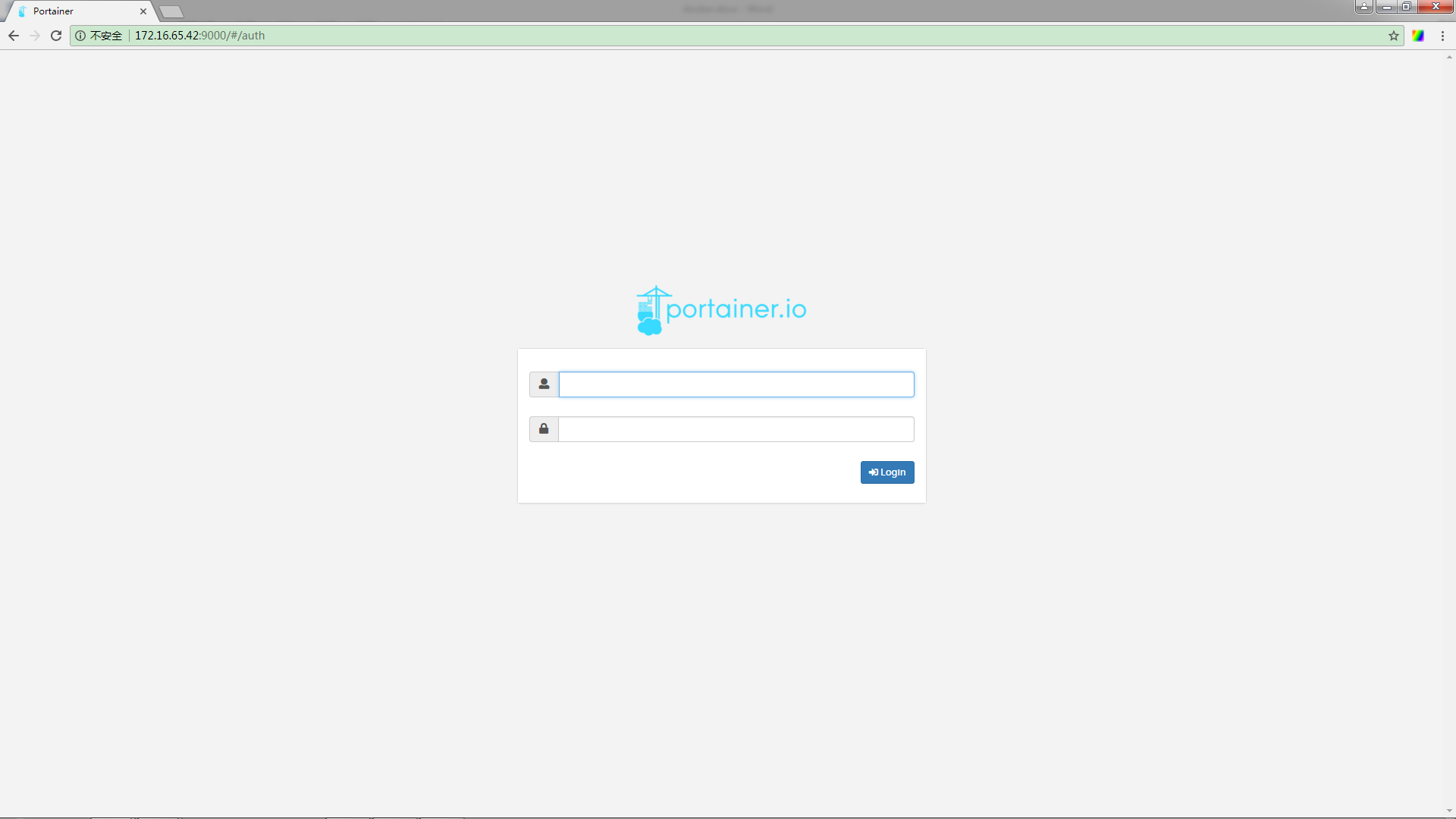


图 6 Portainer登陆主页

1. 开启Docker Daemon端口2375 TCP监听12
2. 向Portainer中添加Docker引擎主机（Portainer称为Endpoint），如图 7。

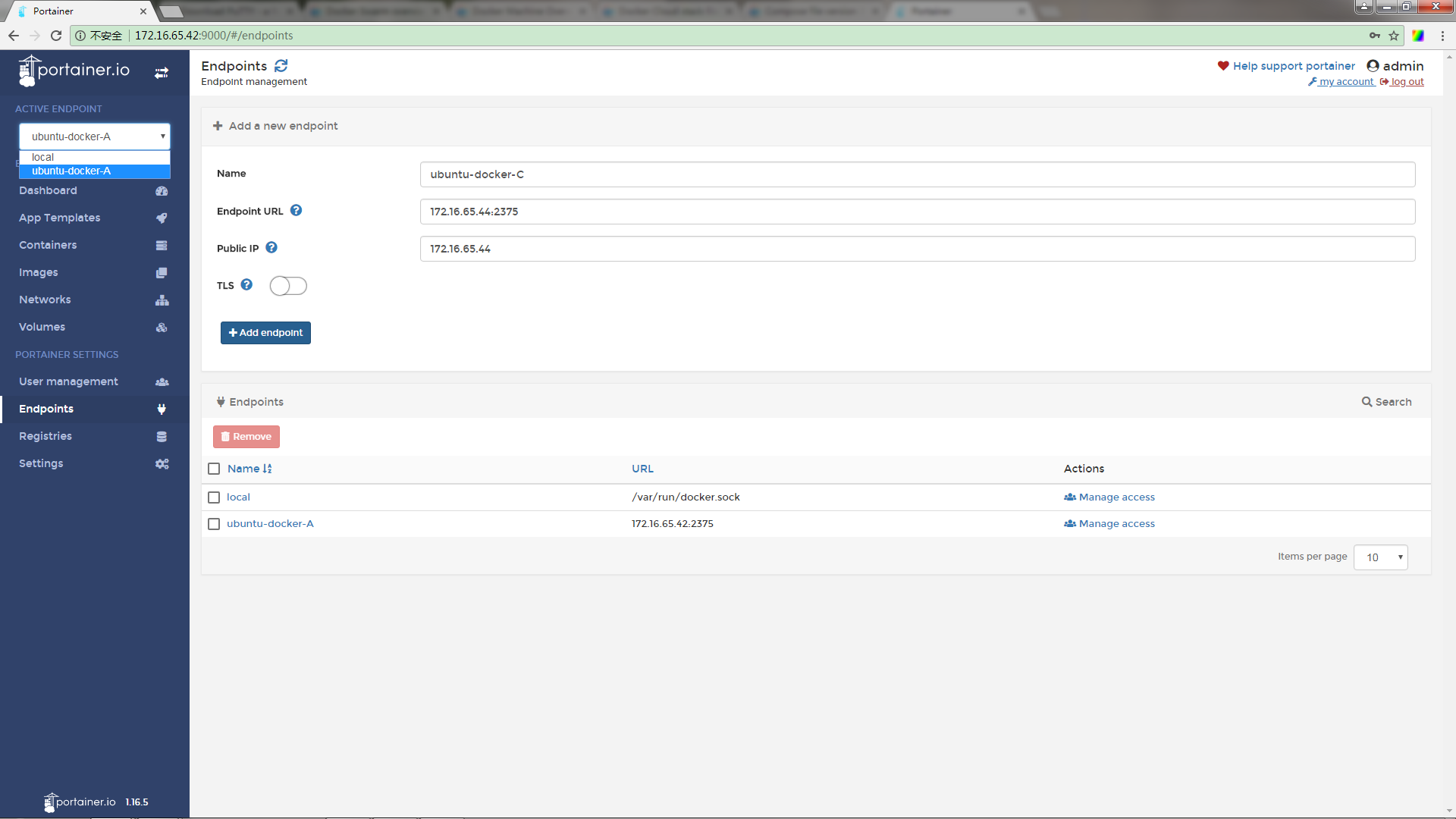


图 7 Portainer Docker主机管理

1. 向Portainer添加私有仓库（如图 8）

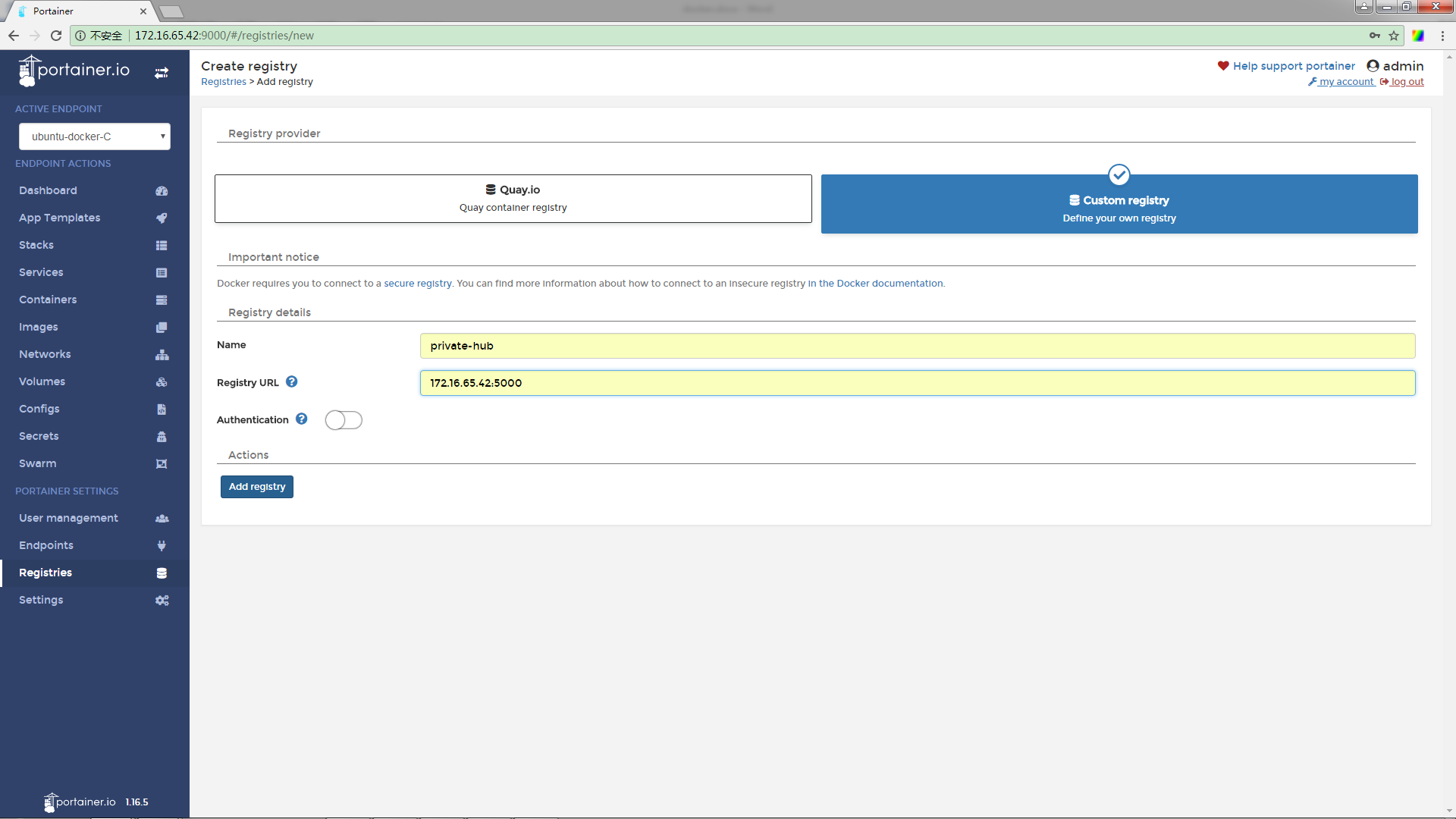


图 8 Portainer 私有仓库管理

1. 浏览Portainer

通过Portainer页面左侧的快速访问标签，可以查看指定主机的容器、镜像、网络、数据卷等资源信息，以及所有已添加主机、私有仓库信息。除此之外，还可以通过Portainer创建镜像、向仓库推送拉取镜像、创建容器、创建网络、创建数据卷。这里不一一演示，可访问长拷机试验（<http://172.16.65.42:9000>）。

### 创建Web服务镜像

1. Dockerfile定义

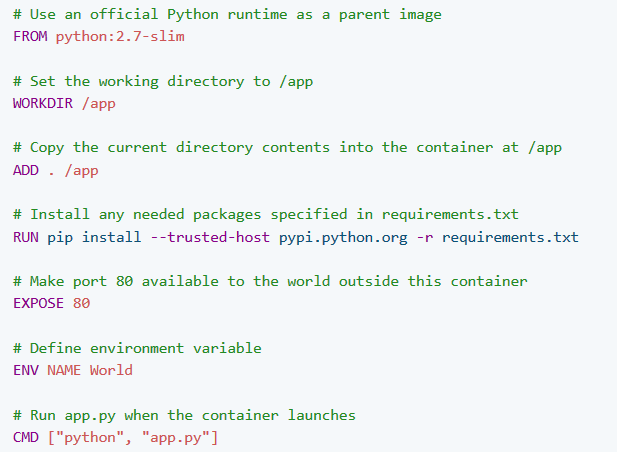


图 9 Web服务镜像Dockerfile

命令说明：

* + 1. FROM，指定Web服务镜像的基础镜像为python，版本为2.7-slim。
    2. WORKDIR，为其后的ADD、RUN、CMD命令（还包括ENTRYPOINT、COPY命令，这里未使用）设置工作目录/app。
    3. ADD，将Docker主机中目录“.”（也即当前目录）的内容添加到镜像文件系统目录/app中。

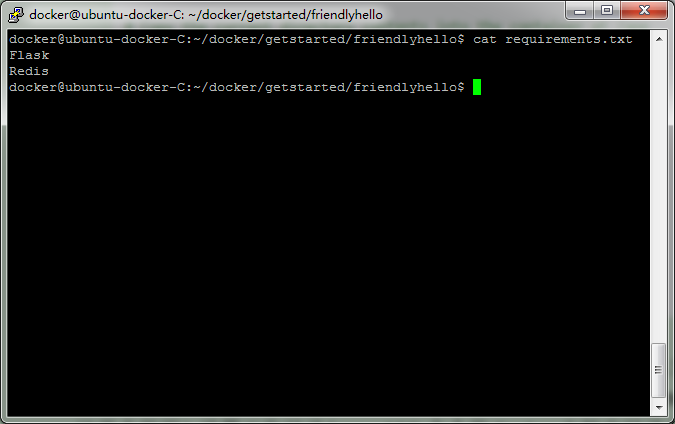


图 10 requirements.txt

* + 1. RUN，在当前镜像的顶层执行指定命令。在这里是安装requirements.txt（如图 10）指定的python模块：FLASK、Redis。FLASK是轻量级Web框架，Redis是python语言redis客户端。
    2. EXPOSE，指定容器的监听端口，默认TCP协议。因此Web服务容器监听端口80。
    3. ENV，设置环境变量NAME值为“World”，对后续所有指令有效。
    4. CMD，执行shell命令。这里是运行python程序app.py（如图 11）。程序app.py实现了一个简单的Web服务。该服务工作在80端口，返回给客户端一段文本字符串。在该字符串中，包含了环境变量、服务主机名以及页面访问次数。环境变量展示了Dockerfile指令ENV是有效的。页面访问次数是保存在Redis数据库中的，因此要使用Redis服务。

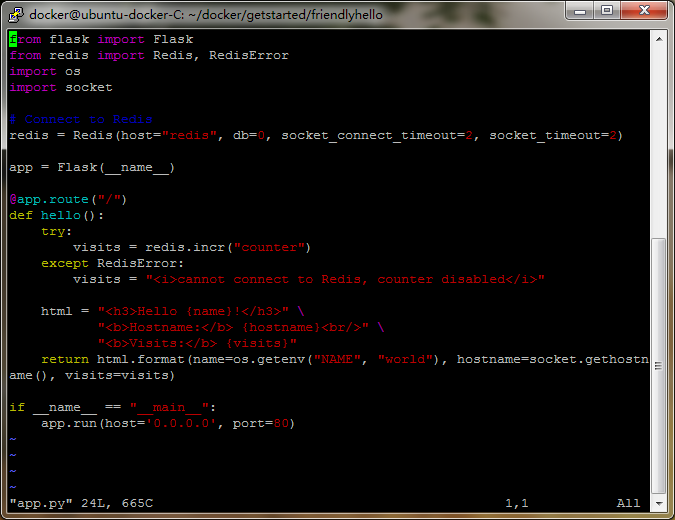


图 11 app.py

1. 创建镜像

docker build –t friendlyhello

1. 配置Docker Daemon（如图 12）以向私有仓库上传镜像

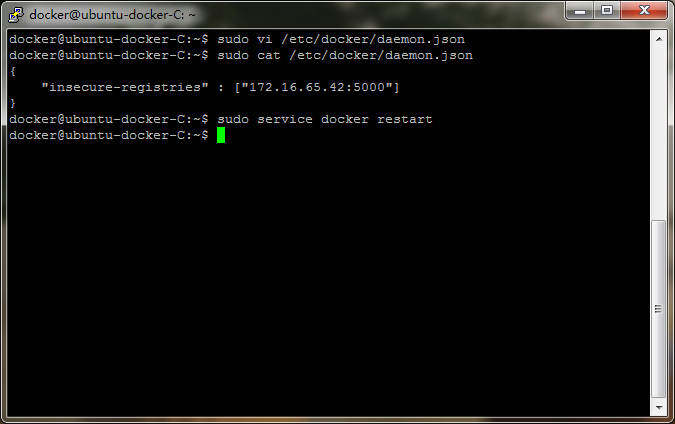


图 12 Docker Deamon配置向仓库上传镜像

1. 上传镜像（如图 13）

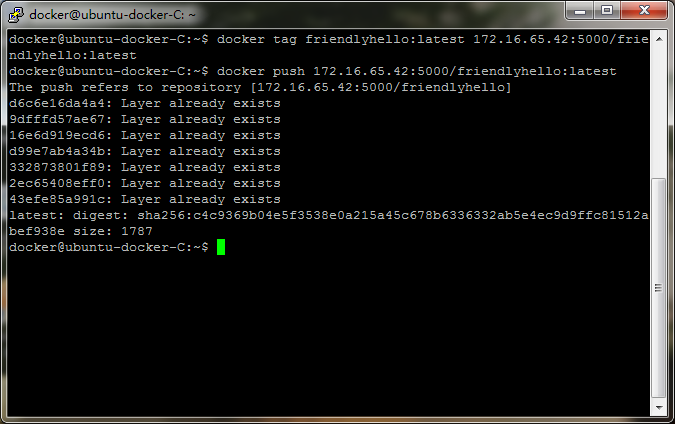


图 13 使用docker命令上传镜像到仓库

访问<http://172.16.65.42:8080/> 查看镜像仓库内容，图 14显示镜像已经上传成功。

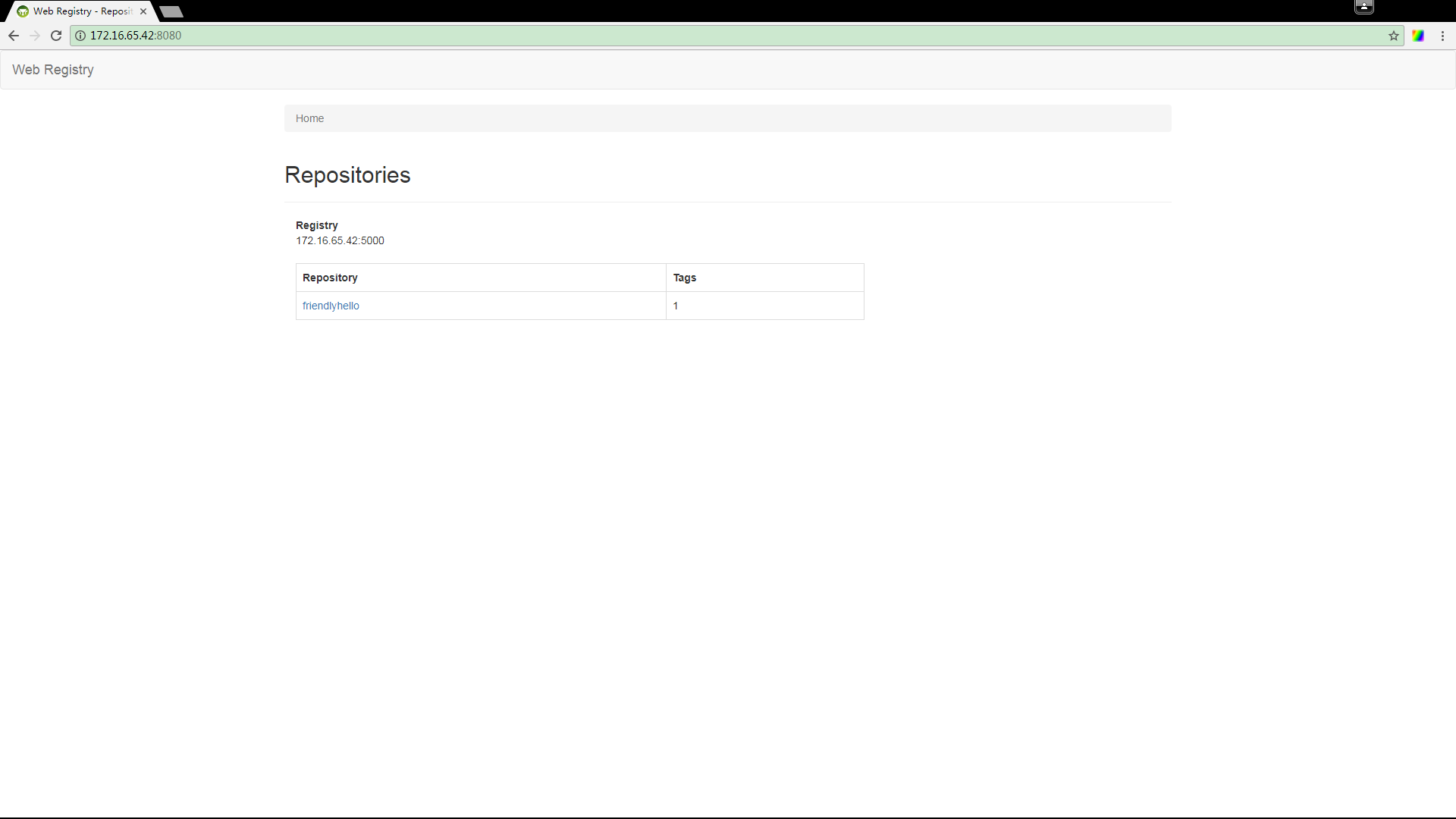


图 14 私有仓库镜像列表

1. 运行镜像启动Web服务容器

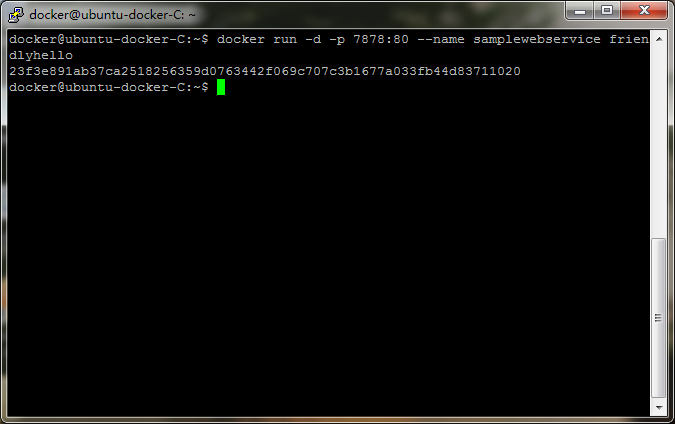


图 15 运行Web服务容器

在浏览器地址栏输入<http://172.16.65.44:7878/>访问本节定义的Web服务。返回页面中“*cannot connect to Redis counter disabled*”，指示Web服务不能访问Redis服务，页面访问次数值无效。下面小节中同时启动Web服务和Redis服务时，页面访问次数字段会显示正确的结果。

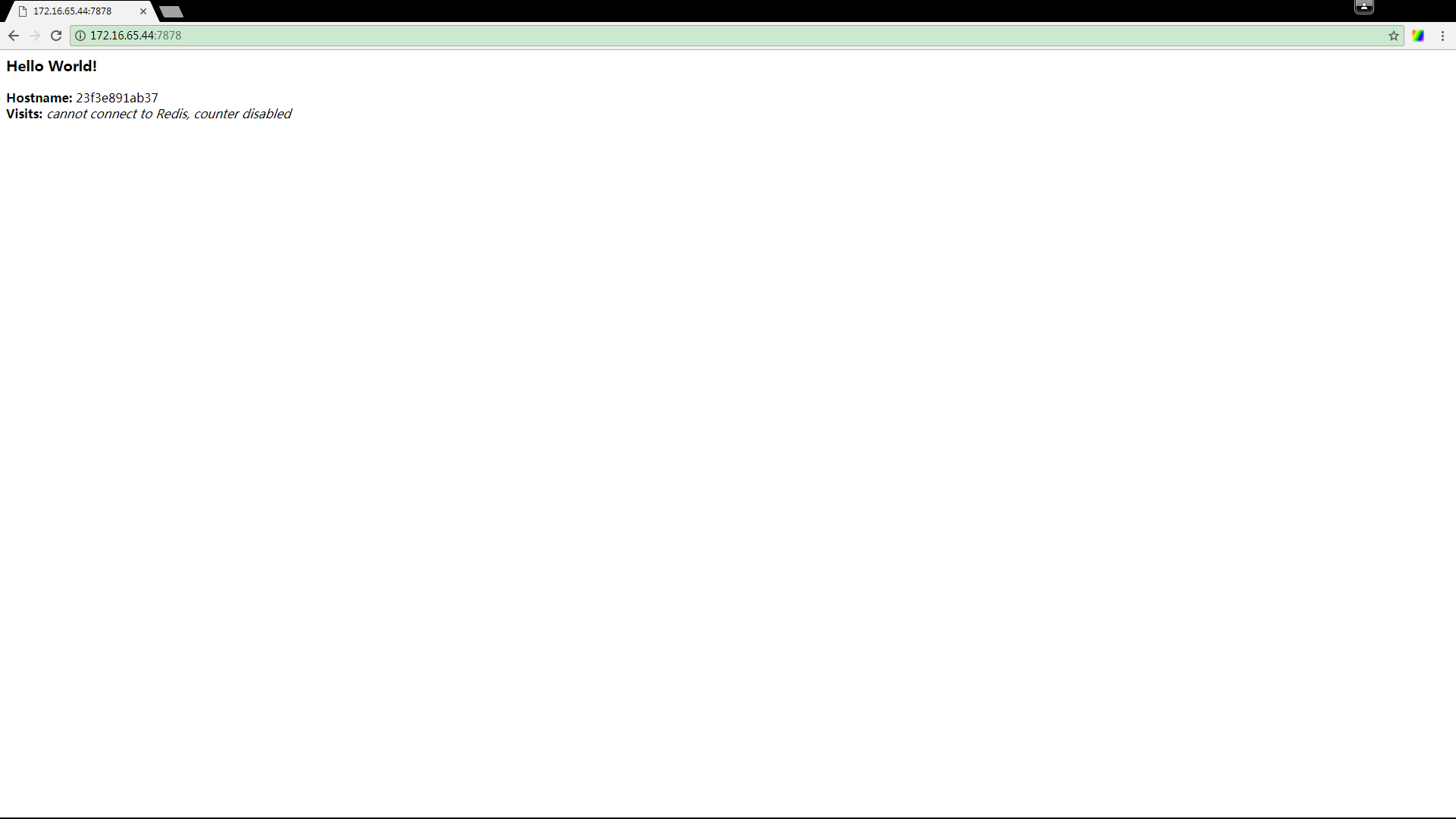


图 16 浏览Web服务

### 服务编排

1. YAML服务编排文件定义

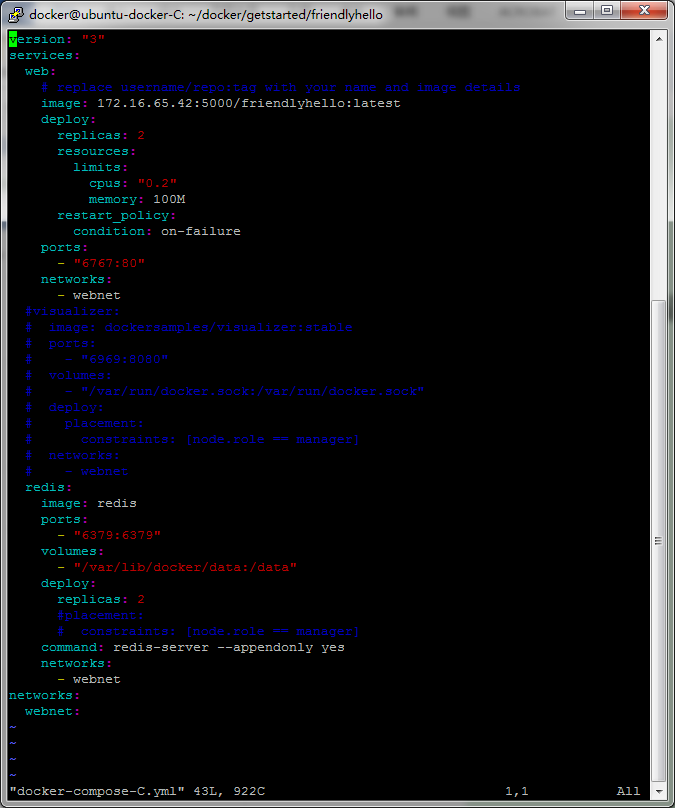


图 17 YAML服务编排文件

图 17的服务编排文件定义了两个服务：web、redis。

1. 字段version指示编排文件遵守版本3。
2. web服务：image字段指定容器镜像为上节创建并已上传到私有仓库的镜像；deploy字段指定了容器复制个数（2个实例）、计算资源约束（cpu使用0.2，内存100M）和重启策略（运行失败时）；ports字段指定容器端口80映射到主机端口6767；networks指定容器使用默认网络配置。
3. redis服务：command字段指示redis服务开启数据持久化；volumes字段指示容器目录/data映射到主机目录/var/lib/docker/data上（持久化数据保存在该主机目录中，即使容器重启持久化数据也不会丢失）；其他与web服务类似。
4. 启动服务

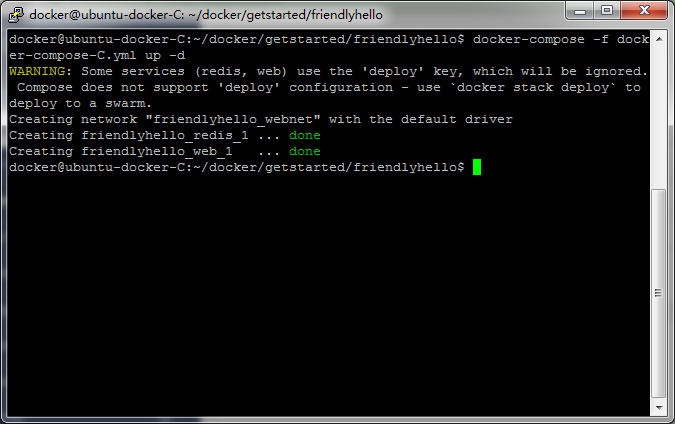


图 18 docker-compose启动服务

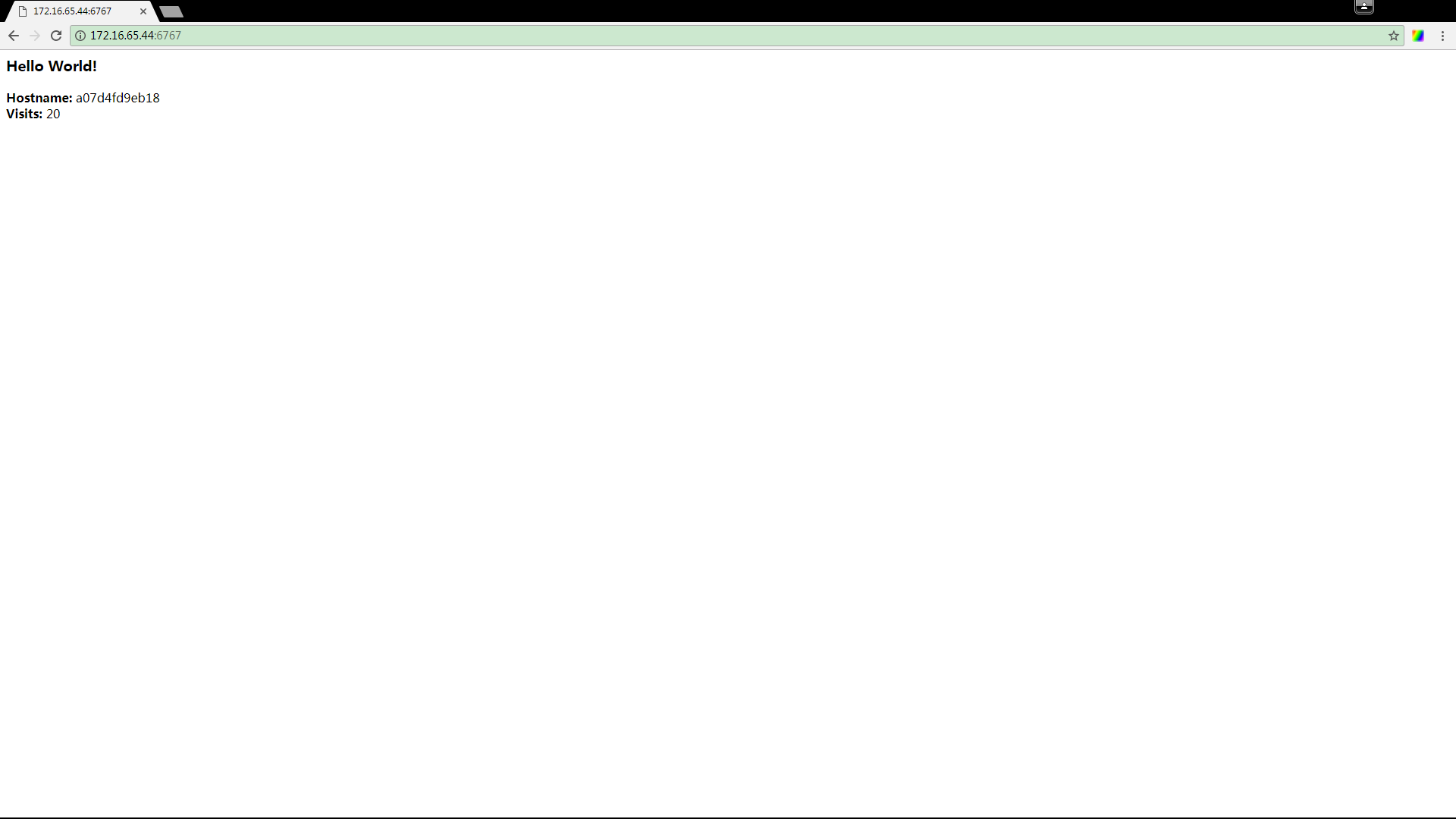


图 19 访问docker-compose所启动服务

图 18中警告deploy字段无效。这是因为deploy字段需要在Swarm模式下调用docker stack deploy命令部署服务时才会生效。访问<http://172.16.65.44:6767/>，显示web服务、redis服务都已经启动。

### Swarm服务栈

1. 启动swarm模式

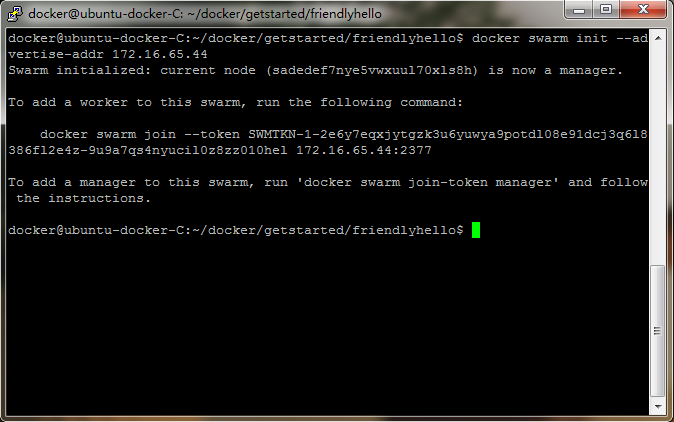


图 20 Swarm Manager（ubuntu-docker-C）

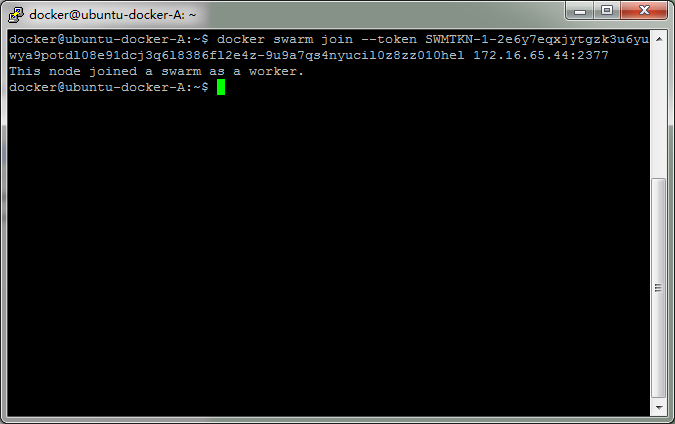


图 21 Swarm Worker（ubuntu-docker-A）

1. 访问Portainer查看Swarm模式状态

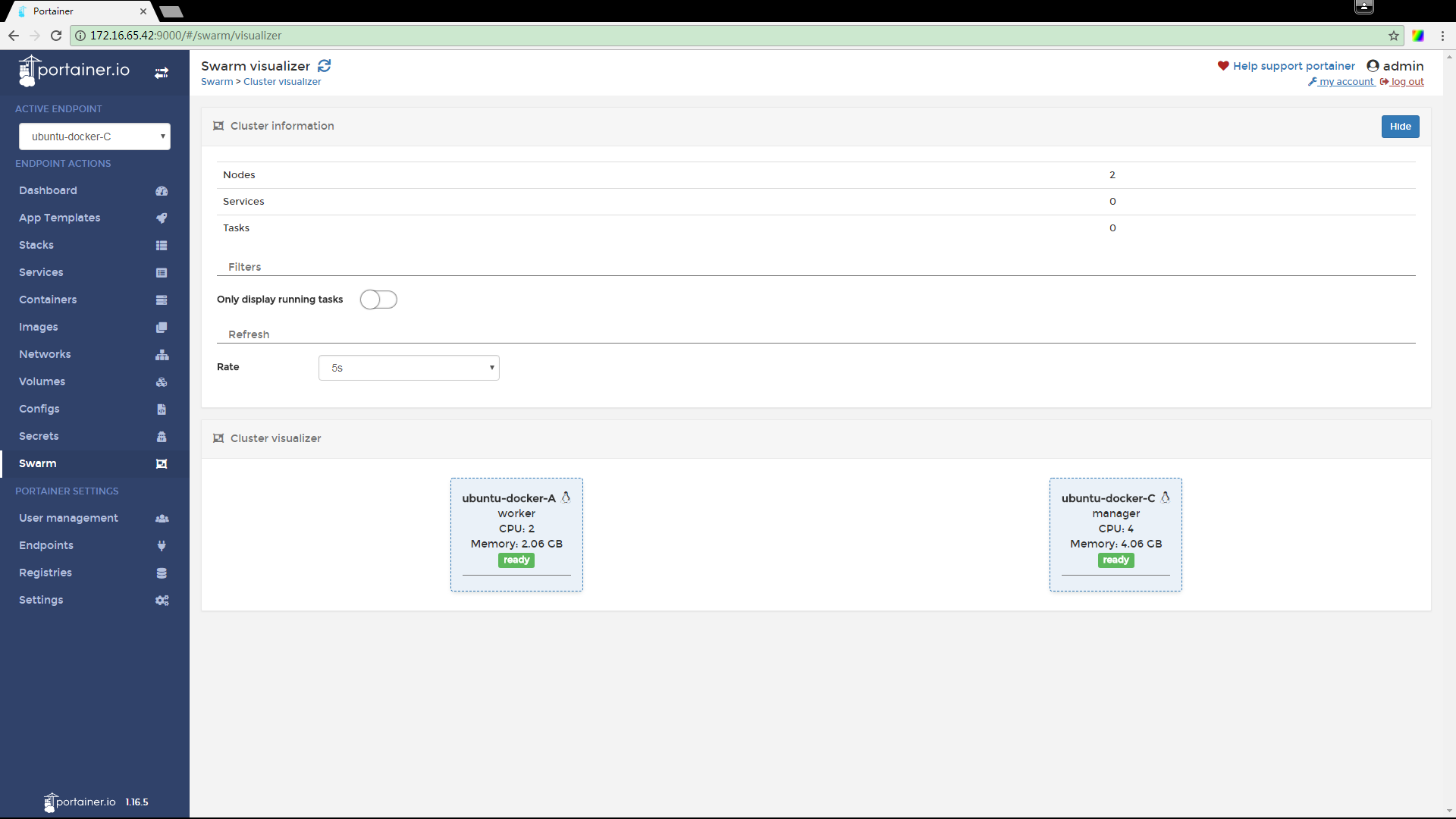


图 22 Portainer展示Swarm模式

1. 部署服务栈

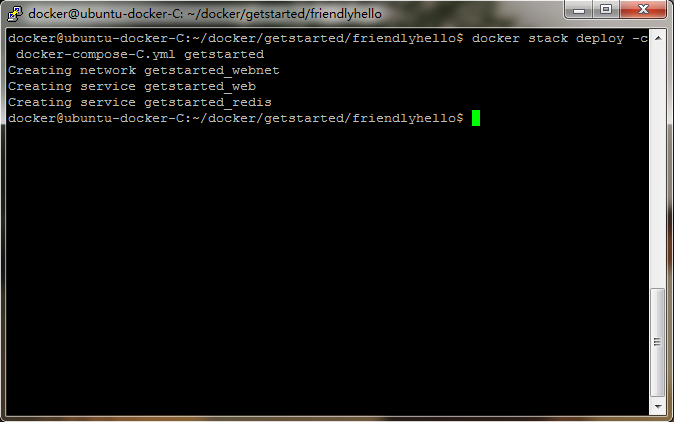


图 23 部署web、redis服务栈

1. 访问Portainer查看服务栈部署状态

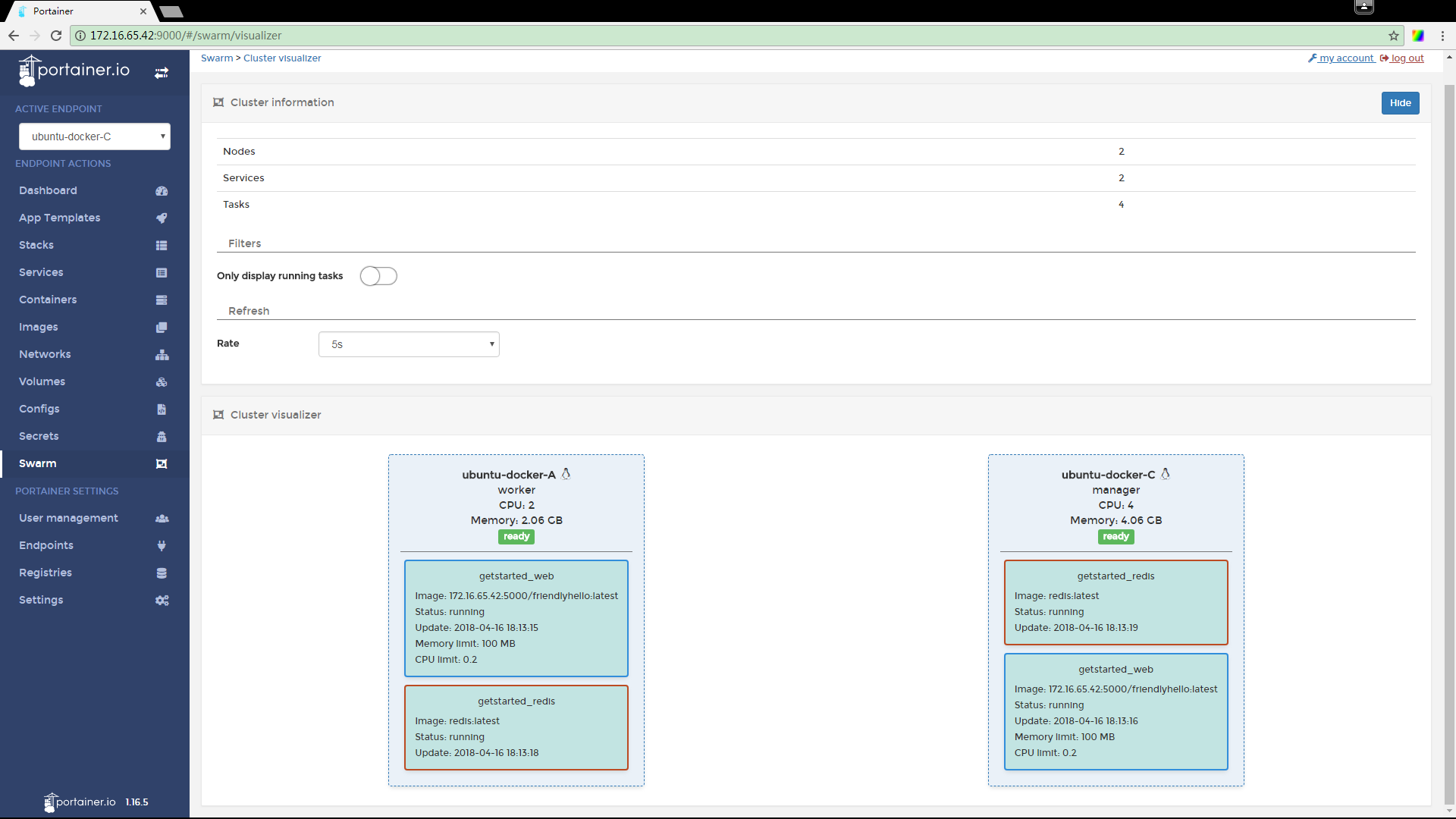


图 24 web、redis服务栈部署状态

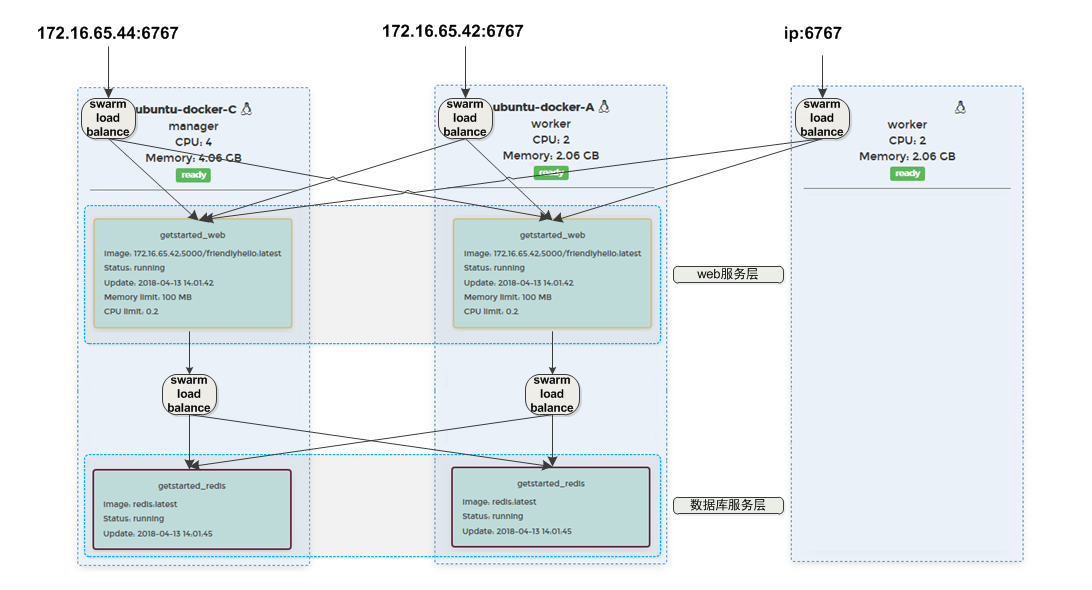


图 25 Swarm进入路由网格

图 24显示web服务、redis服务各启动了两个容器，按照默认均衡机制分别部署在swarm集群的两个节点上。不光是容器部署，对于服务的访问Swarm也提供了负载均衡机制。这种机制Docker术语称为进入路由网格（ingress routing mesh），如图 25。路由网格机制保证了可以从swarm集群的任意主机访问web服务。集群中的web服务、redis服务逻辑分层形成所谓的服务栈（service stack）。

1. 服务栈验证

访问<http://172.16.65.44:6767/、http://172.16.65.42:6767/>多次得到图 26所示结果。在这个结果里，两个redis数据库保持了数据一致性。

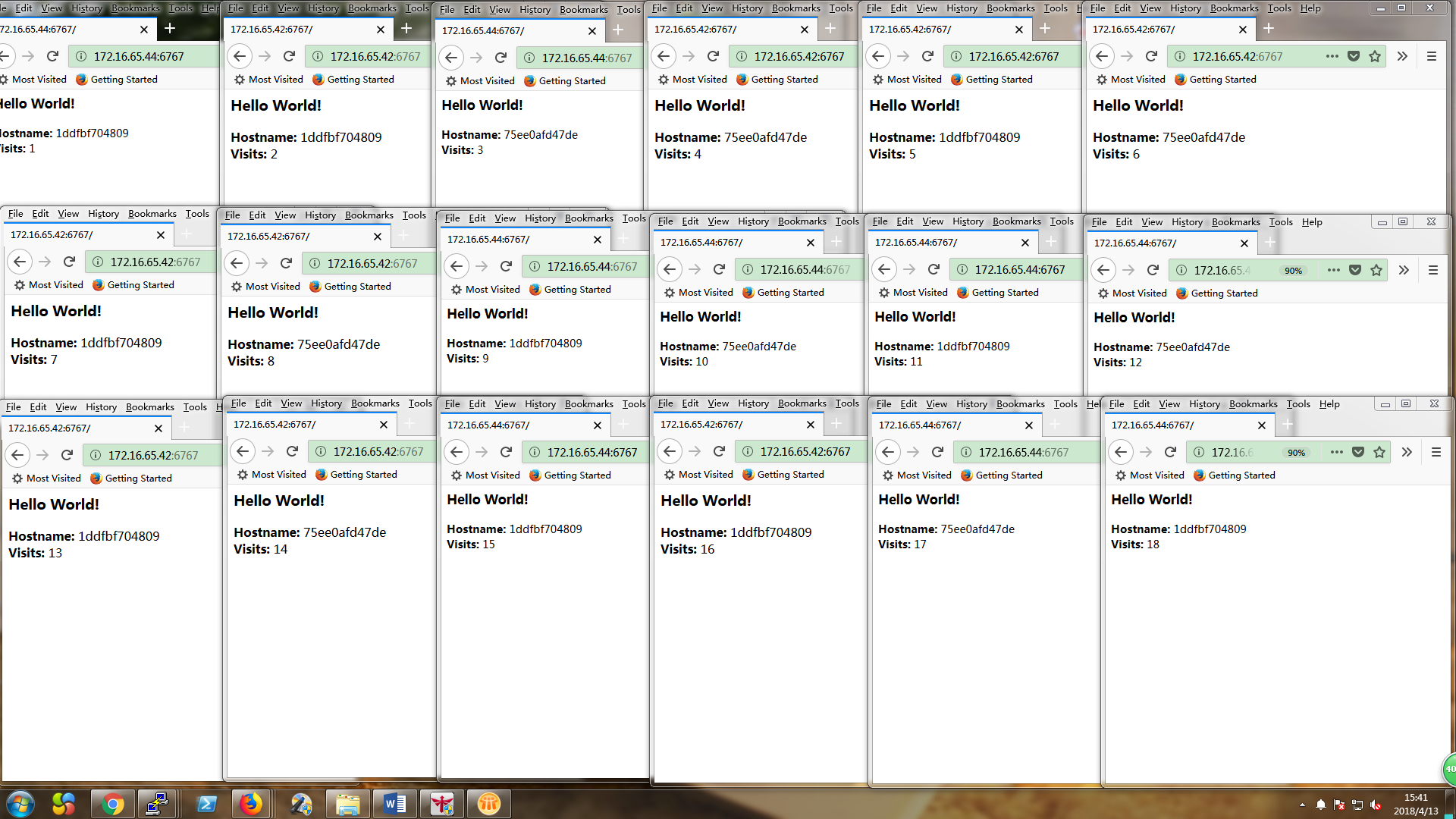


图 26 web、redis服务栈负载均衡

## Docker WebStreamer

### Docker GStreamer镜像构建

本节记述GStreamer基础镜像构建的详细步骤。

### WebStreamer-Compose.yml

后续工作。

## 参考文献

1. <https://docs.docker.com/engine/docker-overview/>
2. <https://docs.docker.com/registry/>
3. <https://docs.docker.com/swarm/overview/>
4. <https://docs.docker.com/engine/reference/commandline/cli/>
5. <https://docs.docker.com/compose/overview/>
6. <https://docs.docker.com/machine/overview/>
7. <https://github.com/portainer/portainer>
8. <https://docs.docker.com/engine/reference/builder/>
9. <https://docs.docker.com/compose/compose-file/>
10. <http://flask.pocoo.org/>
11. <https://www.putty.org/>
12. <https://docs.docker.com/install/linux/linux-postinstall/#configure-where-the-docker-daemon-listens-for-connections>
13. <https://docs.docker.com/engine/swarm/ingress/>