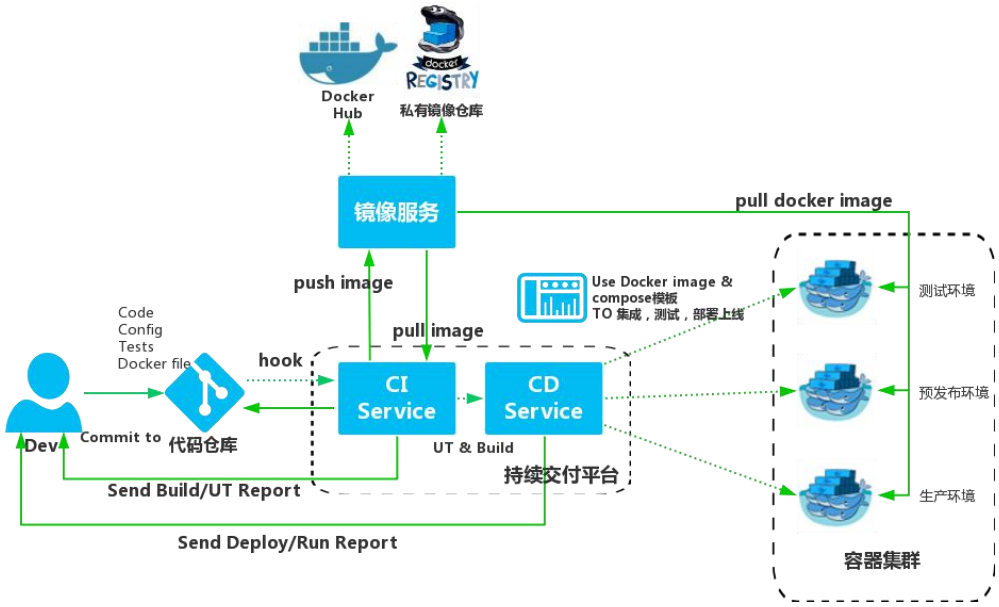
金山云容器解决方案

北京金山云网络技术有限公司

概述

本文指导用户使用金山云产品，构建用于生产环境的 docker 系统，加速系统开发、测试、部署，提高软件交付质量。一个完整的 docker 工作流（如下图），通常包含构建（build）、分发（ship）、运行（run）等几个环节，用户可以根据需要，选择其中的一个或者多个环节。



- Build
  - 使用 jenkins 作为 CI、CD 的引擎
- Ship
  - 提供 docker registry 的 KS3 驱动，结合 harbor 等开源系统，用户可以在金山云构建低成本，高可靠的企业级镜像存储中心
- Run
  - 用户可以选择云主机或者云物理主机（国内独家）搭建容器集群
  - 用户可以选择 mesos、kubernetes、swarm 等作为容器集群管理系统
  - 金山云强大的网络组件（VPC/LB/NAT）,提供安全、可靠、高性能的容器网络
  - 用户可选择 RDS、KCS、KS3 等服务，存储持久化数据，免去自己搭建维护数据库、缓存的工作

适用场景

- CI/CD（持续集成/持续交付）
- Devops
- 自动化运维
- 混合云管理
- 微服务系统构建

创建容器集群

推荐您在 VPC 机房（目前有北京 6 区和上海 2 区）购买云服务器或者云物理主机，搭建容器集群。金山云 VPC 提供高效、安全、可靠的网络服务，并且便于您构建混合云。关于 VPC 的介绍，请参考 VPC 产品文档。

创建 VPC 及子网

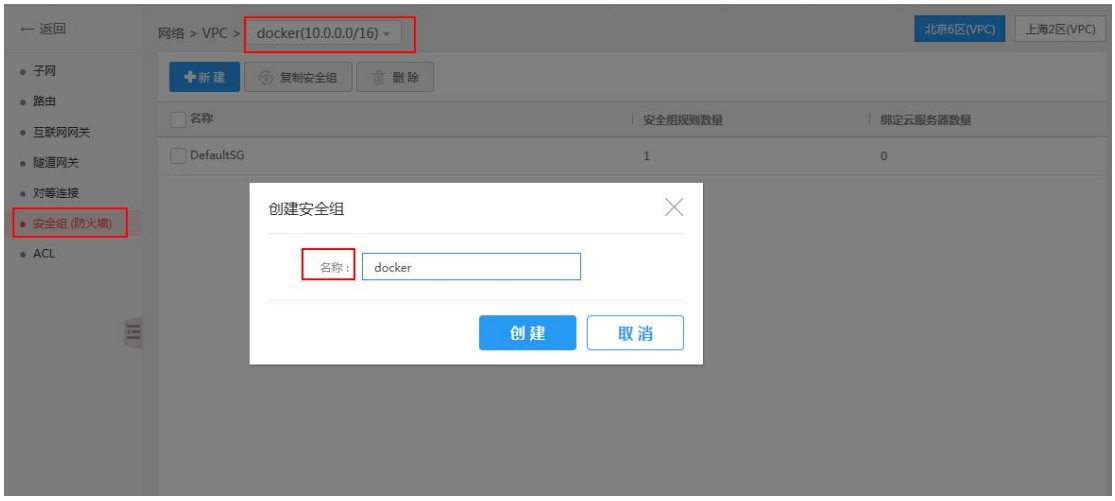
在购买云服务器之前，您应该至少拥有一个 VPC 和一个子网。请参考 VPC 快速入门中的“创建 VPC”和“创建子网”章节。

- 创建一个名称为“docker”的 VPC，以及一个名为“docker-vnet-1”子网，您可以自定义网段，或者使用默认值。这里同时创建了一个类型为“终端子网”的子网 2，在使用金山云 RDS/KCS 等服务时，需要用到“终端子网”。

创建安全组

您可以使用 VPC 的默认安全组 DefaultSG，或者参考快速入门中的“创建安全组”创建一个新的安全组。

- 创建一个名称为“docker”的安全组



创建公网 NAT

如果您的容器集群有访问公网的需求（如访问 docker hub 拉取 docker 镜像，或者您部署在容器集群中的服务需要访问其他公网服务），推荐您使用 VPC 的 NAT 功能。NAT 功能使您容器集群中的机器，共享外网访问能力。请参考快速入门中的“创建 NAT”章节，创建一个 BGP 类型的 NAT。

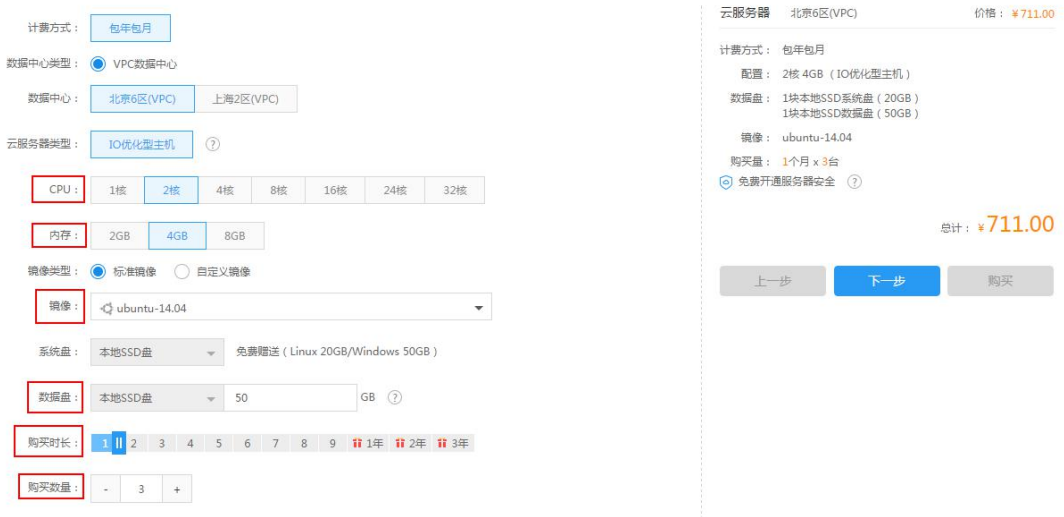
- 新建的 VPC 中，会默认创建一个类型为“金山云内网”的 NAT。需要先把这个 NAT 删掉，才能创建公网类型的 NAT。

- 创建一个链路类型为“BGP”的 NAT

创建云服务器

参考云服务器购买指导。

- 根据需要选择云服务器配置，如 CPU、内存、数据盘、购买时长，购买数量，镜像推荐使用 ubuntu-14.04，为了搭建容器集群，购买数量一般最少为 3 个。

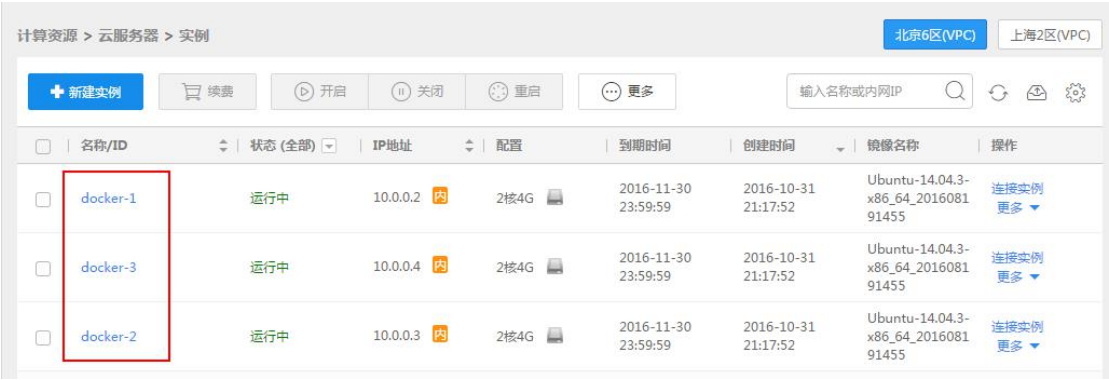


- 选择弹性 IP 页面，选择“稍后购买”

- 设置 VPC 页面，选择 VPC、子网、安全组

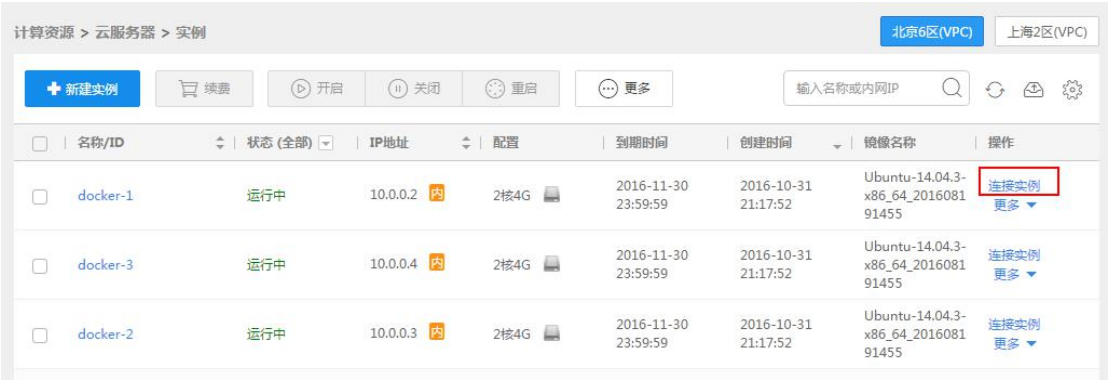
- 设置基本信息页面，设置服务器名称，密码等信息后，点击“购买”

- 购买并成功支付后，可在云服务器控制台看到您购买的主机

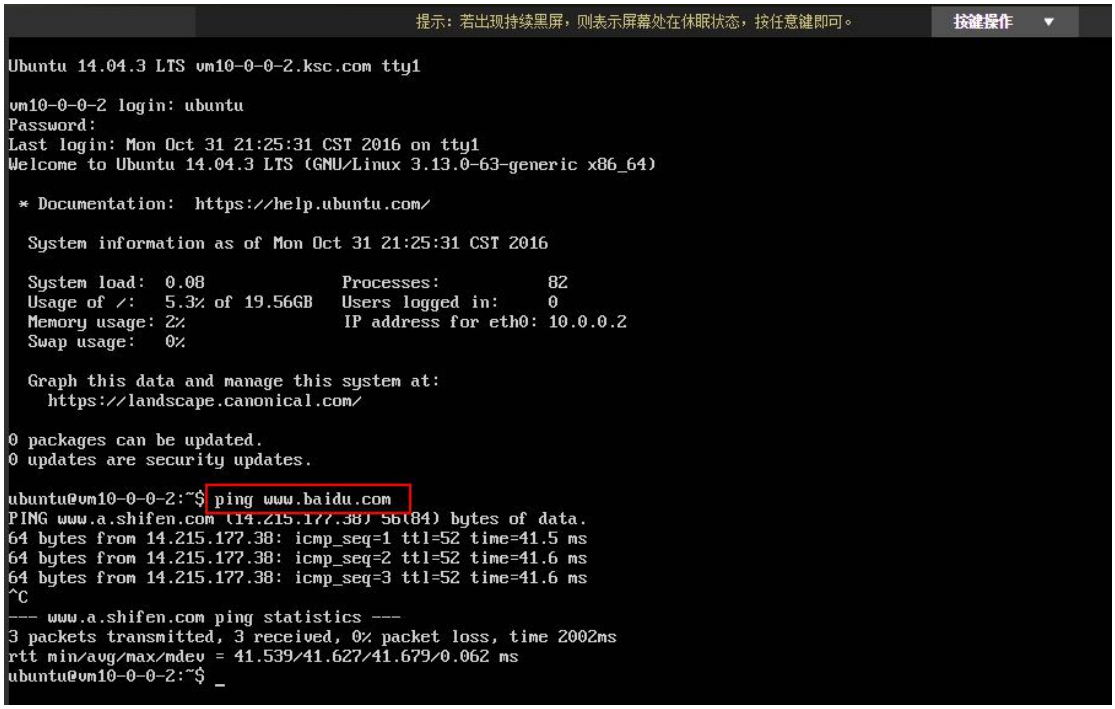


登录云服务器

- 在云服务器列表页，点击“连接实例”，可通过 VNC 控制台登录主机



- 输入用户名和密码登录后，检测外网是否联通



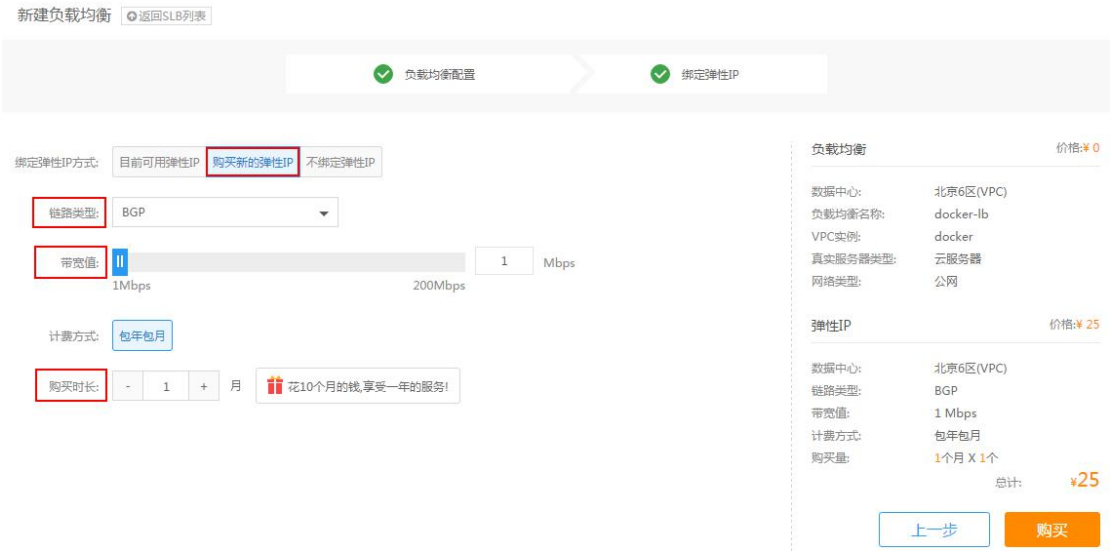
如果无法 ping 通外网，请检查是否正确配置了公网 NAT。

通过 ssh 访问云服务器

如果您习惯 ssh 方式登录云服务器，我们推荐使用负载均衡（LB）将云服务器的 22 端口暴露到公网。另外，您还需要在安全组中放行 22 端口。参考负载均衡快速入门的“创建负载均衡”以及 VPC 快速入门的“创建安全组规则”。

- 创建一个名称为“docker-lb”的负载均衡，网络类型选择“公网”

- 在弹性 IP 页面，选择“购买新的弹性 IP”

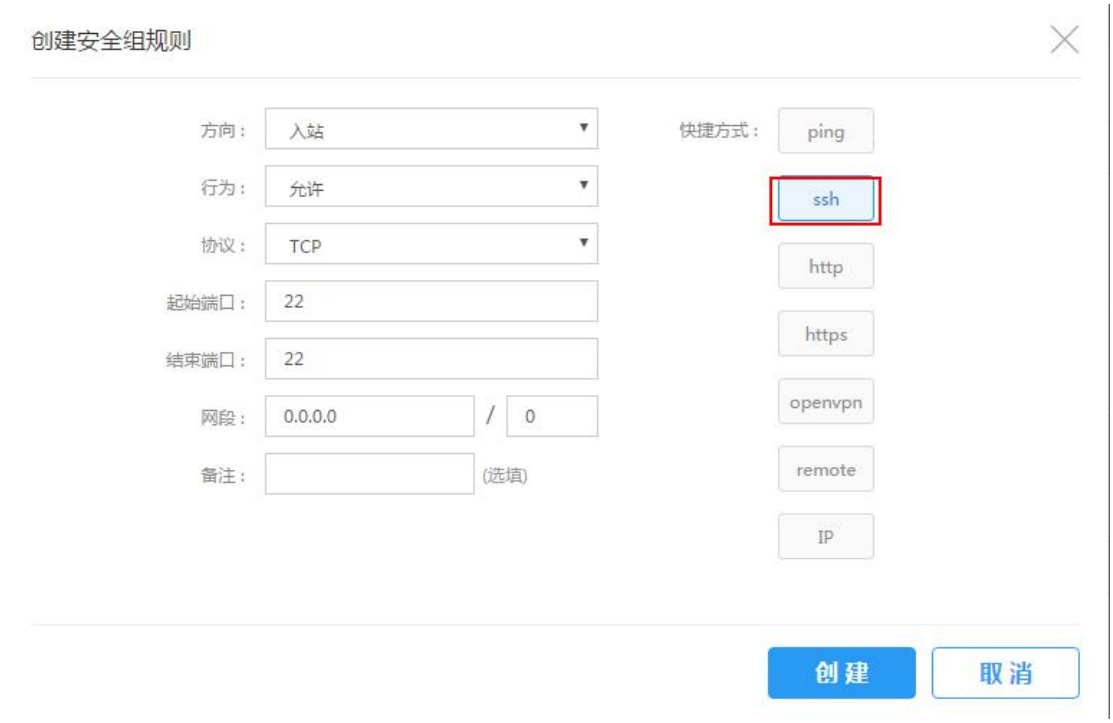


- 支付成功后，在负载均衡列表页可以看到刚创建好的 LB 和弹性 IP，点击“进入负载均衡”可以对 LB 进行配置。

- 创建一个名为 ssh_1 的监听器，监听 10001 端口，真实服务器（RS）设置为 docker-1（10.0.0.2），真实服务器的端口为 22。创建成功以后，我们即把 10.0.0.2 的 22 端口映射到了负载均衡器的 10001 端口。

- 依次创建另外两个监听器，分别把 docker-2（10.0.0.3）和 docker-3（10.0.0.4）的 22 端口，映射到负载均衡器（docker-lb）的 10002 和 10003 端口。完成后如图所示。

- 最后，我们在安全组中添加针对 22 端口的入站规则。



现在，可以通过 LB（IP 地址为 120.x.x.x）的 10001、10002、10003 端口，以 ssh 协议访问三台云服务器。

安装 docker-engine

参考 docker 官方的安装教程，在 3 台云服务器上安装 docker-engine。建议选择 1.12 以上版本，以便获取 swarm mode 等特性。

- 安装完成后，执行 docker version 命令，确认安装的 docker 版本。

```
root@vm10-0-0-2:~# docker version
Client:
 Version:      1.12.3
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   6b644ec
 Built:        Wed Oct 26 21:44:32 2016
 OS/Arch:      linux/amd64

Server:
 Version:      1.12.3
 API version:  1.24
 Go version:   go1.6.3
 Git commit:   6b644ec
 Built:        Wed Oct 26 21:44:32 2016
 OS/Arch:      linux/amd64
```

创建容器

我们以 nginx 镜像为例，在云服务器 docker-1（IP 地址为 10.0.0.2）创建一个 web 容器。

- 运行 docker run 命令，创建一个名为 web 的容器，将容器的 80 端口映射到主机的 80 端口。

```
root@vm10-0-0-2:~# docker run -d -p 80:80 --name web nginx
43731c27248c7e27d9271f561d508b5211e8d9dee02189127223cc95b029da60
```

- 运行 docker ps 命令，确定容器已经在运行

```
root@vm10-0-0-2:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
  NAMES
43731c27248c        nginx               "nginx -g 'daemon off"   6 minutes ago       Up 6 minutes        0.0.0.0:80->80/tcp, 443/tcp
  web
```

- 通过主机的 80 端口，访问该容器提供的服务

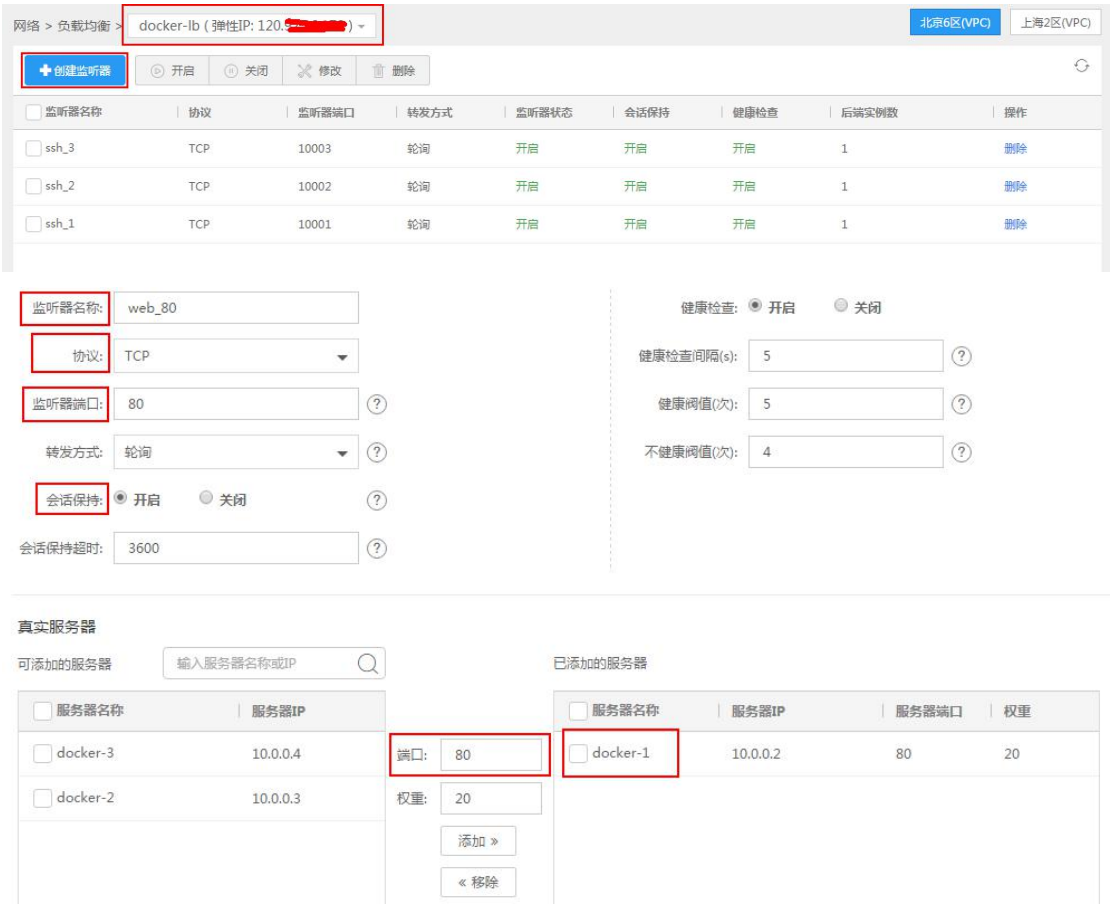
```
root@vm10-0-0-2:~# curl localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

将容器端口暴露到公网

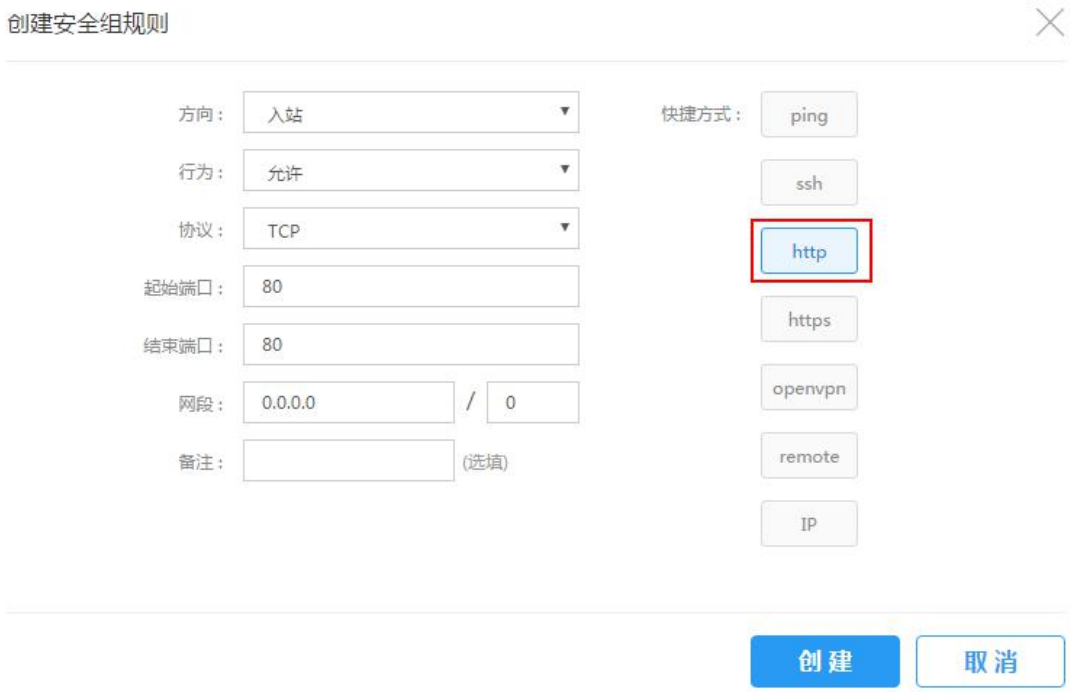
推荐使用负载均衡，将容器服务的端口暴露到公网。在“通过 ssh 访问云服务器”一节，我

们已经介绍了基本步骤。您可以购买一个新的负载均衡，或者使用之前创建好的负载均衡（docker-lb）。

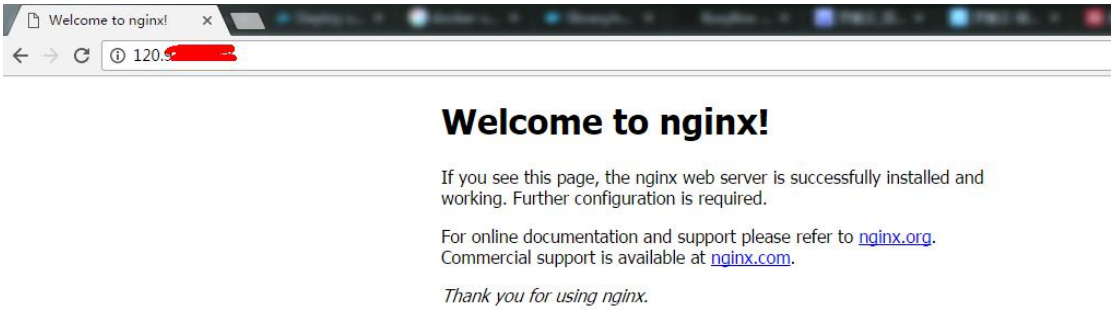
- 在负载均衡 docker-lb 的配置页面，创建一个新的监听器 web_80，监听 80 端口，真实服务器设置为 docker-1 的 80 端口



- 在 VPC 安全组中，增加针对 80 端口的入站规则



- 现在，我们可以通过负载均衡的 IP 地址，在浏览器中访问容器的 web 服务



安装集群管理系统

通过上面的步骤，我们已经安装了 docker engine，并且可以创建容器。但是，单个容器的作用是有限的。在一个实际的系统中，往往需要管理大量容器，并且组织容器之间的关系以便构建分布式服务，这时，要借助容器集群管理系统。
目前，主流的容器集群管理系统有 Kubernetes、Mesos、Swarm 等。您可以选择并安装自己熟悉的系统。如果您之前没有了解过这些系统，推荐您使用 Swarm。Swarm 是 docker 官方的集群管理工具，并且在 docker 1.12 版本以后，Swarm 已经内置在 docker engine 里，安装和配置都比较简单。

创建 Swarm

参考官方的教程：https://docs.docker.com/engine/swarm/swarm-tutorial/

- 我们的 3 台云服务器，设定其中一台为管理节点（manager），另外两台为工作节点（worker）

节点名称	节点 IP 地址	角色
docker-1	10.0.0.2	manager
docker-2	10.0.0.3	worker
docker-3	10.0.0.4	worker

- 在管理节点上，执行 swarm init 命令，创建一个新的 swarm

```
root@vm10-0-0-2:~# docker swarm init --advertise-addr 10.0.0.2
Swarm initialized: current node (dqxpssl5jjq510817dxn3webi) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-194rumn9pqng50jn2sygvza7mw7io7vh4jahx0izz1v8ulue8i-4dep8r9yyf0w45y69xb6jru8q \
    10.0.0.2:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

- 在两个工作节点上，执行 swarm join 命令，将工作节点加入集群

```
root@vm10-0-0-3:~# docker swarm join \
>     --token SWMTKN-1-194rumn9pqng50jn2sygvza7mw7io7vh4jahx0izz1v8ulue8i-4dep8r9yyf0w45y69xb6jru8q \
>     10.0.0.2:2377
This node joined a swarm as a worker.
```

```
root@vm10-0-0-4:~# docker swarm join \
>     --token SWMTKN-1-194rumn9pqng50jn2sygvza7mw7io7vh4jahx0izz1v8ulue8i-4dep8r9yyf0w45y69xb6jru8q \
>     10.0.0.2:2377
This node joined a swarm as a worker.
```

- 在管理节点上，执行 node ls 命令，确认集群创建成功并且包含 3 个节点

```
root@vm10-0-0-2:~# docker node ls
ID                            HOSTNAME              STATUS  AVAILABILITY  MANAGER STATUS
7jfsome23nixu0ohhb068k6r8     vm10-0-0-3.ksc.com    Ready   Active
7qw3g03wm16kgy3bnzlsivxvf     vm10-0-0-4.ksc.com    Ready   Active
dqxpssl5jjq510817dxn3webi *   vm10-0-0-2.ksc.com    Ready   Active        Leader
```

创建服务

参考官方教程：https://docs.docker.com/engine/swarm/services/

- 在管理节点上，执行 service create 命令，创建一个名为 my_web 的服务，副本数设置为 3，公开端口（publish port）为 8080

```
root@vm10-0-0-2:~# docker service create --name my_web --replicas 3 --publish 8080:80 nginx
21f8cesimbalzd77fwn9qyad1
```

- 在管理节点上，执行 service ps 命令，可以看到 3 个容器被调度到了集群中运行

```
root@vm10-0-0-2:~# docker service ps my_web
ID                  NAME      IMAGE  NODE                DESIRED STATE  CURRENT STATE
8nxgdcg46a1539nba1xh3bs9u  my_web.1  nginx  vm10-0-0-4.ksc.com  Running        Running 3 minutes ago
3deo9do99yx0czlhpyxmw8waa  my_web.2  nginx  vm10-0-0-3.ksc.com  Running        Running 2 minutes ago
0peaxhy3v2uutg7dg2jotyszg  my_web.3  nginx  vm10-0-0-3.ksc.com  Running        Running 2 minutes ago
```

- 在集群的任意节点上，访问节点的 8080 端口，可以访问该服务。

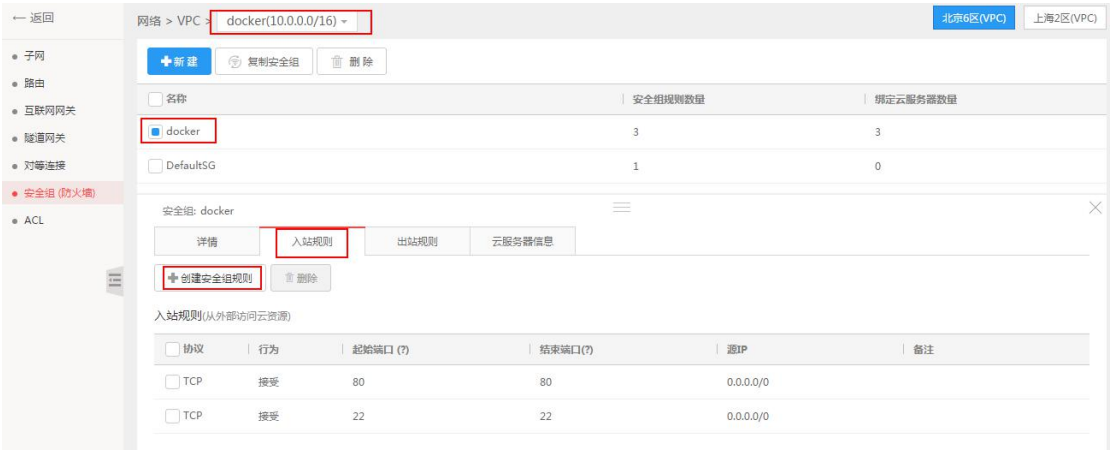
```
root@vm10-0-0-2:~# curl 10.0.0.4:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

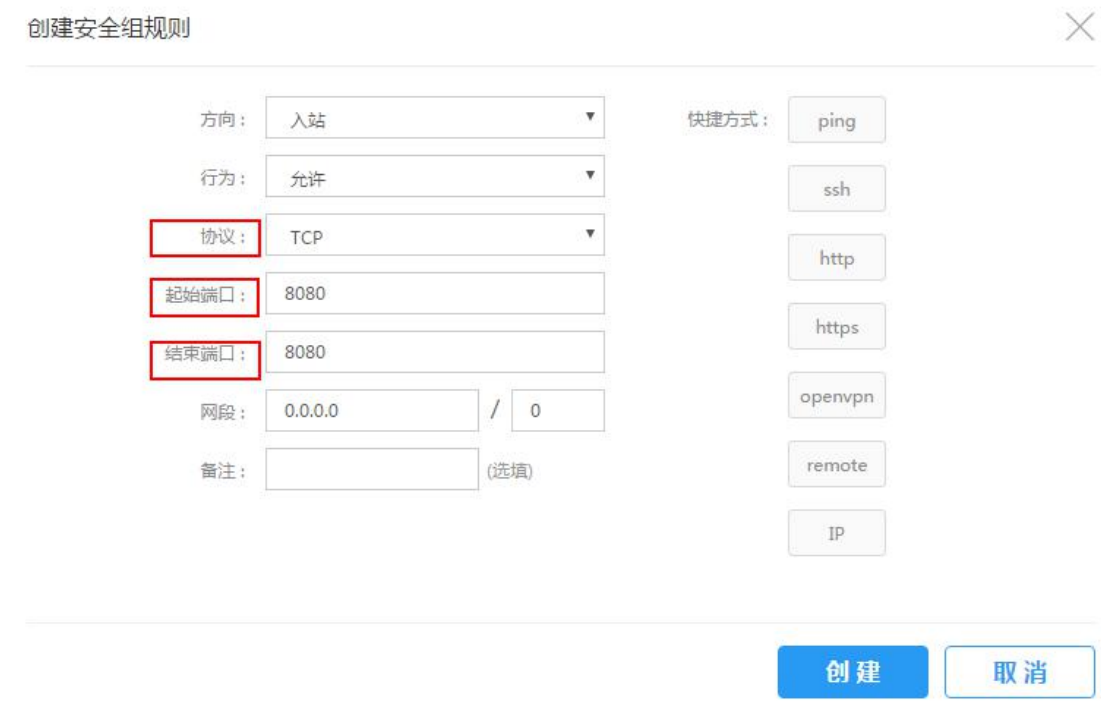
将服务暴露到公网

使用负载均衡把 my_web 发布到公网。注意，前面创建的负载均衡 docker-lb，其 80 端口已经被占用。如果继续使用这个负载均衡，只能选择其他端口（下面的示例，选择 8080 端口）。或者，购买一个新的负载均衡发布到 80 端口。

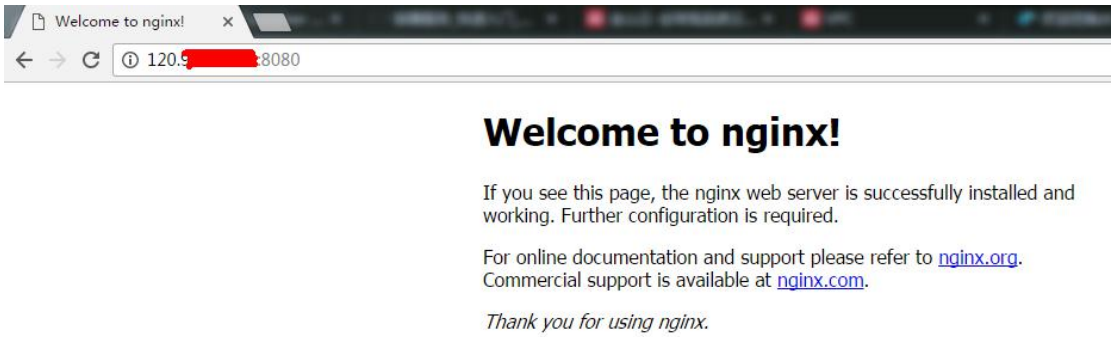
- 在负载均衡 docker-lb 的配置页面，创建一个新的监听器 web_8080，监听 8080 端口，真实服务器设置为 docker-1、docker-2、docker-3 的 8080 端口

- 在 VPC 安全组中，增加针对 8080 端口的入站规则





- 现在，可以通过负载均衡（IP 地址为 120.x.x.x）的 8080 端口访问 my_web 服务。



使用 KS3 存储 docker 镜像

Docker Registry 是构建 docker 镜像仓库的核心组件，docker 官方提供了多种存储驱动，使得用户可以把镜像存储到本地磁盘或者亚马逊 S3 等云储存服务上。

KS3（Kingsoft Standard Storage Service）是金山云提供的无限制、多备份、分布式的低成本存储空间服务，解决存储扩容、数据可靠安全以及分布式访问等相关复杂问题。

如果您想构建私有镜像仓库并且使用金山云对象存储（KS3）作为后端存储，请使用我们定制的 registry 镜像：ksyun/registry。详细的配置，请参考配置文档。

搭建企业级的镜像仓库

我们推荐您使用 Harbor，结合金山云 KS3 存储，构建企业级镜像仓库服务。

Harbor 是 VMware 中国研发中心开发的一款镜像仓库管理工具，提供了企业级用户需要的用户管理、权限管理、镜像复制、LDAP/AD 支持等功能。

本节介绍了如何使用 KS3 + Harbor 搭建企业级镜像仓库。

安装 Docker

如果您还没有安装 docker，执行以下命令安装 Docker

```
curl -fsSL https://get.docker.io | bash
```

安装 docker-compose

默认的官方文档 安装命令如下 ：

```
curl  -L  https://github.com/docker/compose/releases/download/1.6.2/docker-compose-`uname  -s`-`uname -m` > /usr/local/bin/docker-compose
```

下载安装包

Harbor 发行版的安装包可以在 Github 下载，有在线（online）安装和离线（offline）安装两种安装方式选择。
以在线安装示例，通过 wget 命令下载，使用 tar 命令将其解压：

```
wget https://github.com/vmware/harbor/releases/download/0.4.5/harbor-online-installer-0.4.5.tgz
tar -zxvf harbor-online-installer-0.4.5.tgz
```

配置 Harbor

Harbor 的常用配置参数主要存放在 harbor.cfg，直接编辑 harbor.cfg 即可：

```
vim harbor.cfg
# 其配置信息如下
hostname = reg.yourdomain.com                    # Harbor 服务器域名
ui_url_protocol = https                          # UI 组件访问协议
email_server = smtp.mydomain.com                 # email 服务器地址
```

```
email_server_port = 25                              # email 端口
email_username = sample_admin@mydomain.com          # email 账号
email_password = abc                                # email 密码
email_from = admin <sample_admin@mydomain.com>      # email 发件人
email_ssl = false                                   # 是否启用 SSL
harbor_admin_password = Harbor12345                 # Harbor 初始化管理员(admin)密码
auth_mode = db_auth                                 # 权限管理模型(db_auth/ldap_auth)
ldap_url = ldaps://ldap.mydomain.com                # ldap 地址
ldap_basedn = uid=%s,ou=people,dc=mydomain,dc=com # ldap 权限模型
db_password = root123                               # 数据库 管理员密码
self_registration = on                              # 是否打开自动注册
use_compressed_js = on                              # 是否启用压缩 js
max_job_workers = 3                                 # 最大任务数
token_expiration = 30                               # token 超时
verify_remote_cert = on                             # 是否验证远程证书
customize_crt = on                                  # 是否启用自定义证书
# 以下为自定义证书信息
crt_country = CN
crt_state = State
crt_location = CN
crt_organization = organization
crt_organizationalunit = organizational unit
crt_commonname = example.com
crt_email = example@example.com
```

配置后端存储到 KS3

在默认情况下，Harbor 的镜像存储在本地磁盘/data/registry 目录下。但是在生产环境中，出于高可用、高吞吐、安全可靠等因素，我们可能需要将镜像数据存储在 KS3、Ceph 等。

金山云提供了基于 Docker registry v2 封装了 KS3 驱动的官方镜像 ksyun/registry。该镜像提供了 Docker registry v2 和 KS3 的完美对接，通过该镜像，用户可以很方便将镜像仓库数据存储到 KS3 上。

使用 KS3 作为 Harbor 镜像仓库的后端存储，你需要两步配置：

首先，修改 harbor/docker-compose 文件，修改 registry 模块，以支持 KS3 驱动的官方镜像。

```
regsitry:
    image: ksyun/registry:latest    # 修改为支持 KS3 驱动的镜像
    container_name: registry
    restart: always
    volumes:
        # 此处注释掉 harbor 默认挂在本地目录 vlomues 的存储方式
```

```
      # - /data/registry:/storage
    - ./common/config/registry/:/etc/registry/
  environment:
    - GODEBUG=netdns=cgo
  command:
    ["serve", "/etc/registry/config.yml"]
  depends_on:
    - log
  logging:
    driver: "syslog"
  options:
    syslog-address: "tcp://127.0.0.1:1514"
      tag: "registry"
...
```

其次，配置 template/registry/config.yml 文件，修改 storage 模块，为 KS3 的配置数据。

```
storage:
  cache: inmemory
  ks3:
    accesskey: "your ks3 accesskey"          # KS3 的访问私钥
    secretkey: "your ks3 secretkey"          # KS3 的访问秘钥
    internal: false                         # 默认值 false，是否使用内网加速，true 代表使用内网加速，（高速，
不计流量）
    region:  ks3-cn-beijing                  # 区域设置
    bucket : "your bucket name"              # KS3 存储桶的名字
    encrypt: false                          # 默认值 false，是否对镜像数据加密
    secure: false                           # 默认值 false，是否使用 https（ssl）
    chunksize: 5242880                      # 块大小
    storage_page: /                         # 存储路径
```

更多 KS3 存储配置，参见 Docker Registry KS3 存储驱动配置：https://github.com/softlns/distribution/blob/master/docs/storage-drivers/ks3.md。

配置 HTTPS

创建 CA 证书

```
openssl req \
  -newkey rsa:4096 -nodes -sha256 -keyout ca.key \
  -x509 -days 365 -out ca.crt
```

证书签名

```
openssl req \
  -newkey rsa:4096 -nodes -sha256 -keyout yourdomain.com.key \
  -out yourdomain.com.csr
```

初始化 CA 信息

```
mkdir demoCA
cd demoCA
touch index.txt
echo '01' > serial
cd ..
```

配置 Nginx

```
# 复制证书
cp registry.mritd.me.crt config/nginx/cert
cp ca/registry.mritd.me.key config/nginx/cert
# 备份配置
mv config/nginx/nginx.conf config/nginx/nginx.conf.bak
# 使用模板文件
mv config/nginx/nginx.https.conf config/nginx/nginx.conf

# 修改 nginx.conf 配置
server {
  listen 443 ssl;
  server_name harbordomain.com;
  ...
server {
  listen 80;
  server_name harbordomain.com;
  rewrite ^/(.*) https://$server_name:443/$1 permanent;
```

安装并启动

配置完成后，执行 Harbor 提供的安装脚本（install.sh），来生成私有配置，在线安装版本会到 Docker Hub 拉取镜像，并启动 Harbor 服务。

```
sudo ./install.sh
```

测试

访问 Harbor UI

在浏览器中打开 http://reg.yourdomain.com/ ，以 admin 账号（初始账号/密码：admin/Harbor12345）登录，可以查看 Harbor 的控制面板。

测试 docker client 登录、推拉镜像

在控制面板（Harbor UI）中创建一个项目 myproject，docker client 登录认证，并测试推送私有镜像：

```
docker login reg.yourdomain.com

docker push reg.yourdomain.com/myproject/myrepo:mytag
```

管理 Harbor 生命周期

你可以通过 docker-compose 工具来管理 Harbor 的生命周期，常用命令如下：

```
# 停止 Harbor
$ sudo docker-compose stop
Stopping harbor_proxy_1 ... done
Stopping harbor_ui_1 ... done
Stopping harbor_registry_1 ... done
Stopping harbor_mysql_1 ... done
Stopping harbor_log_1 ... done
Stopping harbor_jobservice_1 ... done

# 启动 harbor
$ sudo docker-compose start
Starting harbor_log_1
Starting harbor_mysql_1
Starting harbor_registry_1
Starting harbor_ui_1
Starting harbor_proxy_1
Starting harbor_jobservice_1

# 更新 Harbor 配置
$ sudo docker-compose down
$ vim harbor.cfg
$ sudo install.sh
```

基于 Docker 的持续集成、持续交付

本节介绍了，如何通过 gitlab、jenkins 等基础服务来搭建持续集成、持续部署流水线。

搭建基础服务

CI、CD 涉及的基础服务有：

- 代码仓库（以 Gitlab 示例）
- 镜像仓库（以搭建私有镜像仓库示例）
- 代码构建（以 Jenkins 示例）
- 容器集群（以 swarm 示例）

代码仓库（构建 gitlab 服务）

GitLab 是一个利用 Ruby on Rails 开发的开源应用程序，实现一个自托管的 Git 项目仓库，可通过 Web 界面进行访问公开的或者私人项目；类似于 Github。

选一台主机以 docker 化的方式来搭建 Gitlab 服务。（已有私有 Gitlab 仓库或使用 Github，可跳过此步骤）

我们选择 Docker Hub 上的 sameersbn/gitlab 镜像来快速构建 Gitlab 服务。根据 sameersbn/docker-gitlab 给出的文档，我们需要启动三个容器组件：

```
# 启动 postgresql 容器
docker run --name gitlab-postgresql -d \
    --env 'DB_NAME=gitlabhq_production' \
    --env 'DB_USER=gitlab' \
    --env 'DB_PASS=password' \
    --env 'DB_EXTENSION=pg_trgm' \
    --volume /srv/docker/gitlab/postgresql:/var/lib/postgresql \
    sameersbn/postgresql:9.5-3
# 启动 redis 容器
docker run --name gitlab-redis -d \
    --volume /srv/docker/gitlab/redis:/var/lib/redis \
    sameersbn/redis:latest
# 启动 gitlab 容器
docker run --name gitlab -d \
    --link gitlab-postgresql:postgresql --link gitlab-redis:redisio \
    --publish 10022:22 --publish 10080:80 \
    --env 'GITLAB_PORT=10080' \
     --env 'GITLAB_SSH_PORT=10022' \
```

```
--env 'GITLAB_SECRETS_DB_KEY_BASE=long-and-random-alpha-numeric-string' \
--env 'GITLAB_SECRETS_SECRET_KEY_BASE=long-and-random-alpha-numeric-string' \
--env 'GITLAB_SECRETS_OTP_KEY_BASE=long-and-random-alpha-numeric-string' \
--volume /srv/docker/gitlab/gitlab:/home/git/data \
sameersbn/gitlab:latest
```

测试：
执行 docker ps 命令查看容器运行状态。
浏览器访问 http://host:port，查看 gitlab 网页能够正常打开，并设置 root 账号初始密码。

镜像仓库（构建 Docker 私有仓库服务）

在 CI、CD 中，需要一个 Docker 镜像仓库来存放每次构建的镜像。从镜像的安全、可靠、访问速度等因素考虑，搭建一个私有的镜像仓库，对企业级开发和实践是很有必要的，推荐使用 KS3 搭建高可用的企业级私有镜像仓库。

如果仅为了测试，可以使用 registry 镜像搭建一个简单的仓库：

```
docker run -d --restart=always --name registry \
-v /mnt/docker/registry:/var/lib/registry -p 15000:5000 registry:2
```

测试：
执行 docker ps 命令查看容器运行状态。
测试镜像的推送/拉取（docker push/docker pull）

注：docker 私有镜像仓库 拉取/推送 失败，提示不能使用 http 连接。解决方法：
1、设置 https 访问，参见企业级镜像仓库的搭建。
2、在 docker 启动参数里添加--insecure-registry ip:port，然后重启 docker。

代码构建（构建 Jenkins 服务）

Jenkins 是基于 Java 开发的一种开源持续集成工具，具有开源，支持多平台和插件扩展，安装简单，界面化管理等特点。Jenkins 使用 job 来描述每一步工作，节点是用来执行项目的环境。Master 节点是 Jenkins job 的默认执行环境，也是 Jenkins 应用本身的安装环境。

由于 Jenkins 本身版本比较老，插件多，但许多插件缺少维护更新，加上本身环境比较陈旧，且各工程之间插件依赖关系难以管理；因此建议使用 Docker 来搭建 Jenkins 服务。

在 Jenkins 容器中调用 docker 的方式有三种：

1、通常在单节点下，可将节点 docker 挂载到 docker 容器中，需添加如下参数：

```
-v /var/run/docker.sock:/var/run/docker.sock -v $(which docker):/usr/bin/docker
```

2、通常在集群中，可以使用 jenkins 的 docker 集群，连接到 Docker server REST API（http://master-ip:2375），需安装一下插件：

```
* Docker plugin
* Docker Commons Plugin
* docker-build-step
```

3、通常在 jenkins 的集群中，插件的方式并不是很方便，仍然需要在 shell 中执行 docker 命令，这时候可以使用 DinD（Docker- in-Docker）的方式来构建 jenkins 服务，参见 Github 项目 https://github.com/jpetazzo/dind。

这里我们使用单节点的方式，使用 Docker Hub 上金山云的 ksyun/jenkins 镜像来构建 jenkins 服务：

```
docker run  docker run  -d  -p 8080:8080 \
  --name jenkins  --restart=always
  -v /mnt/jenkins_home:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock
  --env 'DOCKER_SOCKET=/var/run/docker.sock' \
  --env 'DOCKER_GROUP=dockerhost' \
  --env 'DOCKER_HOST=unix:///var/run/docker.sock' \
  --env 'JENKINS_USER=jenkins'  ksyun/jenkins
```

测试：
执行 docker ps 命令查看容器运行状态。
在浏览器，打开 http://ip:port，进入 jenkins。

容器集群（构建 Docker Swarm 集群）

详见创建 Swarm

Demo：基于 CI、CD 的 Flask 项目

以上，我们搭建了基于 Docker 的 CI、CD 的基础服务，下面我们创建一个 Flask 项目的 Demo 来测试下：

- 在 gitlab 创建 python-hello-world 项目仓库
- clone 项目，本地开发
- 配置 jenkins
- 触发构建

Gitlab 上新建项目

在 gitlab 创建 python-hello-world 项目仓库。

本地开发

编写 Flask 应用

创建 app.py

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
    return "Hello World! <br/> version: v1   author: lain"
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000)
```

创建 requirements.txt

```
flask
```

编写 Dockerfile

```
From python:2.7
MAINTAINER lain
ADD . /app
WORKDIR ./app
RUN pip install -r requirements.txt
EXPOSE 5000
CMD ["python", "/app/hello.py"]
```

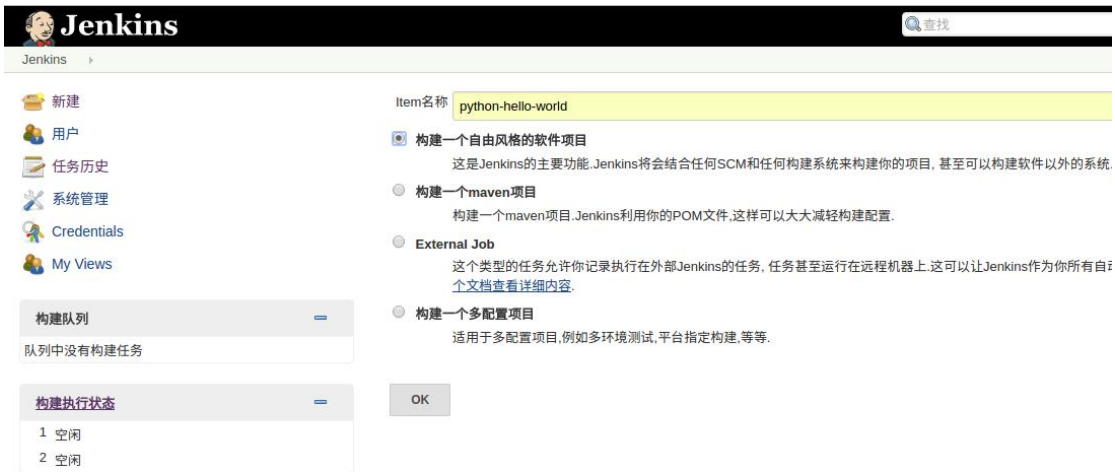
创建构建部署脚本 build_deploy.sh

```
#!/bin/bash
#build in jenkins
# docker 私有仓库的地址
REG_URL=10.0.0.4:15000
# swarm 集群 manage 节点的地址
SWARM_MANAGE_URL=tcp://10.0.0.2:2375
# service 名称
SERVICE_NAME=$JOB_NAME
# 根据时间生成 tag
TAG=$REG_URL/$JOB_NAME:`date +%y%m%d-%H-%M`
# 使用项目目录下的 Dockerfile 文件打包
docker build -t  $TAG  $WORKSPACE/.
```

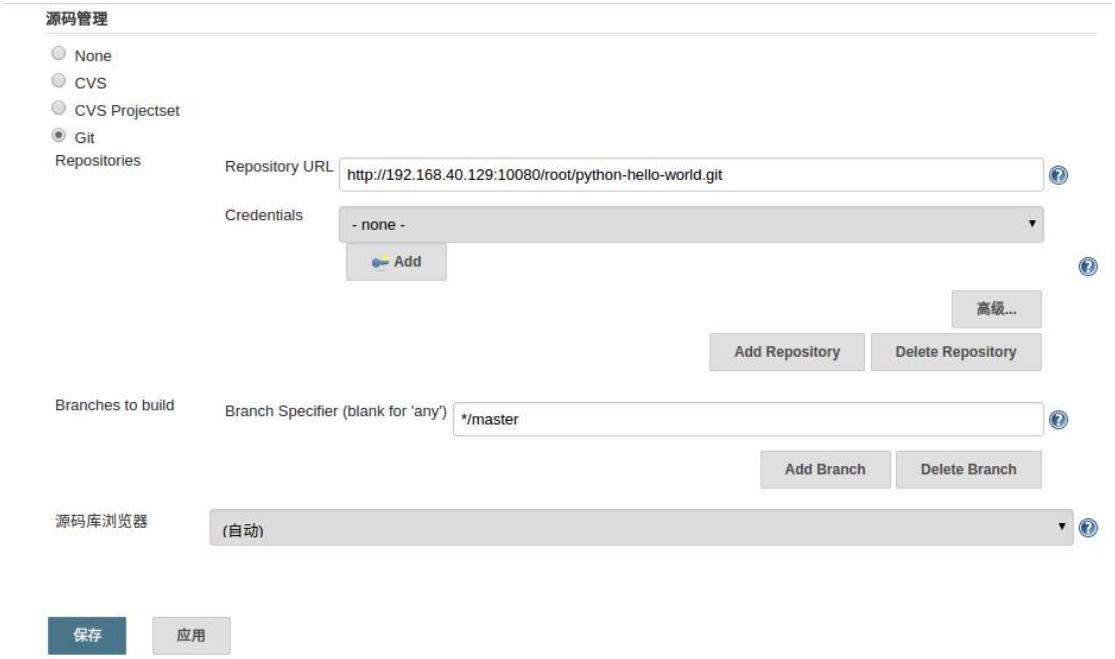
```
docker push   $TAG
docker rmi $TAG
# 创建或更新服务
if docker -H $SWARM_MANAGE_URL service ls | grep -i $SERVICE_NAME; then
       docker -H $SWARM_MANAGE_URL service update  $SERVICE_NAME --image $TAG
else
       docker -H $SWARM_MANAGE_URL service create --name $SERVICE_NAME --replicas 3 -p 15000:5000 $TAG
Fi
```

配置 Jenkins

在 Jenkins 上创建一个 python-hello-world 项目，选择自由风格即可：

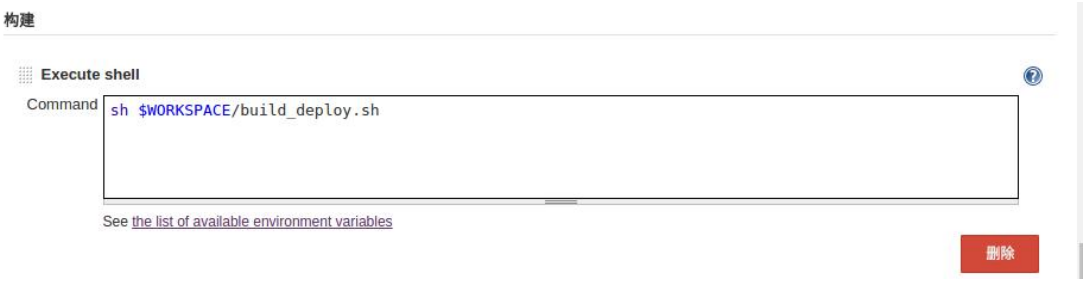


设置 git:



设置构建触发器，这里设置每分钟拉取一次，也可设置 gitlab hook：

设置构建需要执行的脚本，这里设置成 build_deploy.sh:



触发 CI、CD

推送到代码到 gitlab 的 master 分支，然后可在 jenkins console 查看到构建集成和部署的信息。

```
Successfully installed Jinja2-2.8 MarkupSafe-0.23 Werkzeug-0.11.11 click-6.6 flask-0.11.1 itsdangerous-0.24
[91mYou are using pip version 8.1.2, however version 9.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
[0m ---> b3abf3f55f6c
Removing intermediate container 7cb13d944ed2
Step 6 : EXPOSE 5000
 ---> Running in 6d3141eb9752
 ---> 892407930e3b
Removing intermediate container 6d3141eb9752
Step 7 : CMD python /app/hello.py
 ---> Running in f29cb8dd57be
 ---> e0afe0be0eac
Removing intermediate container f29cb8dd57be
Successfully built e0afe0be0eac
The push refers to a repository [10.0.0.4:5000/python-hello-world]
230952785ef3: Preparing
30a7e9b72a1f: Preparing
79ae05a40127: Preparing
160abe12c7c6: Preparing
58f7a855e280: Preparing
787c930753b4: Preparing
9f17712cba0b: Preparing
223c0d04a137: Preparing
fe4c16cbf7a4: Preparing
787c930753b4: Waiting
9f17712cba0b: Waiting
223c0d04a137: Waiting
fe4c16cbf7a4: Waiting
30a7e9b72a1f: Pushed
79ae05a40127: Pushed
230952785ef3: Pushed
58f7a855e280: Pushed
160abe12c7c6: Pushed
223c0d04a137: Pushed
fe4c16cbf7a4: Pushed
9f17712cba0b: Pushed
787c930753b4: Pushed
161118-07-46: digest: sha256:f473d34889b0bbb8e7cbd19cfe7ab7c51027a7e0c7b2af0b1e1e0c44d771d811 size: 2220
Untagged: 10.0.0.4:5000/python-hello-world:161118-07-46
Untagged: 10.0.0.4:5000/python-hello-world@sha256:f473d34889b0bbb8e7cbd19cfe7ab7c51027a7e0c7b2af0b1e1e0c44d771d811
Deleted: sha256:e0afe0be0eac86d317d03aa1f105f268ed114f830b91e1384349cb4305f7bcf0
Deleted: sha256:892407930e3b500ad79c728f89c5d6110652578d0635bf1a6de245d134734ea0
Deleted: sha256:b3abf3f55f6ccdf4d6a9c6119ebb556a8a5388f0813e8361ebf7da77ea2e35ed
Deleted: sha256:0d607ce2febc8654a07d2d326195abeeb63974b666c67c93d81b0e60bb510733
1phb6uxst5qfq68kdg0sr5ojg
Finished: SUCCESS
```

访问测试

在 swarm 集群主节点上查看当前 service：

```
ubuntu@vm10-0-0-2:~$ sudo docker service ls
ID            NAME                MODE        REPLICAS  IMAGE                                                COMMAND
dz303gknpcna  python-hello-world  replicated  3/3       10.0.0.4:5000/python-hello-world:161118-09-57
```

查看当前 service 的容器状态：

```
ubuntu@vm10-0-0-2:~$ sudo docker service ps python-hello-world
ID                         NAME                  IMAGE                                                NODE              DESIRED STATE  CURRENT STATE           ERROR
2044ymgycep90hbijsnbmqc66  python-hello-world.1  10.0.0.4:5000/python-hello-world:161118-09-57  vm10-0-0-4.ksc.com  Running        Running 39 seconds ago
360hyq4fmiyth1dfbj9n7kado  python-hello-world.2  10.0.0.4:5000/python-hello-world:161118-09-57  vm10-0-0-3.ksc.com  Running        Running 39 seconds ago
5vumn1ewp2mlyilqmsa3v6xsb  python-hello-world.3  10.0.0.4:5000/python-hello-world:161118-09-57  vm10-0-0-2.ksc.com  Running        Running 39 seconds ago
```

访问浏览器，服务已经部署到 swarm 集群：

```
HTTP/1.0 200 OK
Content-Length: 45
Content-Type: text/html; charset=utf-8
Date: Fri, 18 Nov 2016 08:06:32 GMT
Server: Werkzeug/0.11.11 Python/2.7.12

Hello World! <br/> version: v1   author: lain
```

更新代码版本，并推送到代码仓库 gilab：

```
from flask import Flask
```

```
app = Flask(__name__)
@app.route('/')
def hello_world():
    return "Hello World! <br/> version: v2   author: lain"
if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000)
```

等待 2 分钟，可以在 jenkins 查看到新的构建是否成功。
在 master 节点查看 service 的容器状态：

```
ubuntu@vm10-0-0-2:~$ sudo docker service ps python-hello-world
ID                  NAME                    IMAGE                                                  NODE              DESIRED STATE  CURRENT STATE            ERROR
1sja6shb3b2g7w5drsxhgslu1  python-hello-world.1      10.0.0.4:5000/python-hello-world:161118-10-01  vm10-0-0-2.ksc.com  Running        Running 26 seconds ago
2044yngycep90hb1jsnbmqc66   \_ python-hello-world.1  10.0.0.4:5000/python-hello-world:161118-09-57  vm10-0-0-4.ksc.com  Shutdown       Shutdown 26 seconds ago
60nzmn5h7yo6t5lvto5to99v8  python-hello-world.2      10.0.0.4:5000/python-hello-world:161118-10-01  vm10-0-0-4.ksc.com  Running        Running 14 seconds ago
360hy64fm1yth1dfbj9n7kado   \_ python-hello-world.2  10.0.0.4:5000/python-hello-world:161118-09-57  vm10-0-0-3.ksc.com  Shutdown       Shutdown 14 seconds ago
5i35th0kvhvqoqstodo13bfes  python-hello-world.3      10.0.0.4:5000/python-hello-world:161118-10-01  vm10-0-0-3.ksc.com  Running        Running 2 seconds ago
5vumn1ewp2mly1iqmsa3v6xsb   \_ python-hello-world.3  10.0.0.4:5000/python-hello-world:161118-09-57  vm10-0-0-2.ksc.com  Shutdown       Shutdown 3 seconds ago
```

访问浏览器：

```
HTTP/1.0 200 OK
Content-Length: 45
Content-Type: text/html; charset=utf-8
Date: Fri, 18 Nov 2016 10:03:15 GMT
Server: Werkzeug/0.11.11 Python/2.7.12

Hello World! <br/> version: v2   author: lain
```