Docker介绍及应用

**更改记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 更改时间 | 更改的章节 | 更改描述 | 更改人 |
| 2017-03-07 | 初稿 | 初稿 | 孙蒙佳 |
| 2017-03-08 | Mysql容器应用 | 新增 | 孙蒙佳 |
| 2017-03-09 | Nginx容器应用 | 新增 | 孙蒙佳 |
| 2017-03-15 | Oracle容器应用 | 新增 | 孙蒙佳 |

目录

[1 Docker介绍 4](#_Toc477352143)

[1.1 Docker简介 4](#_Toc477352144)

[1.1.1 什么是docker 4](#_Toc477352145)

[1.1.2 为什么要使用docker 4](#_Toc477352146)

[1.2 基本概念 5](#_Toc477352147)

[1.2.1 镜像 5](#_Toc477352148)

[1.2.2 容器 6](#_Toc477352149)

[1.2.3 仓库 6](#_Toc477352150)

[2 测试环境容器应用 8](#_Toc477352151)

[2.1 创建测试用容器 8](#_Toc477352152)

[2.1.1 基础镜像centos/jdk1.7\_as\_hornetq 8](#_Toc477352153)

[2.1.2 创建容器 8](#_Toc477352154)

[2.2 容器内操作 10](#_Toc477352155)

[2.2.1 启动aerospike 10](#_Toc477352156)

[2.2.2 启动hornetq 11](#_Toc477352157)

[2.2.3 测试环境配置举例 12](#_Toc477352158)

[2.3 退出容器 14](#_Toc477352159)

[2.4 连接容器 14](#_Toc477352160)

[2.5 删除容器 15](#_Toc477352161)

[3 Mysql容器应用 15](#_Toc477352162)

[3.1 Mysql镜像 15](#_Toc477352163)

[3.2 创建mysql容器 16](#_Toc477352164)

[3.3 Navicat连接mysql 17](#_Toc477352165)

[4 Oracle容器应用 17](#_Toc477352166)

[4.1 Oracle镜像 17](#_Toc477352167)

[4.2 创建oracle容器 18](#_Toc477352168)

[4.3 Navicat连接oracle 18](#_Toc477352169)

[5 Nginx容器应用 19](#_Toc477352170)

[5.1 Nginx镜像 19](#_Toc477352171)

[5.2 创建nginx镜像 19](#_Toc477352172)

[5.3 使用nginx容器做负载均衡 19](#_Toc477352173)

[6 容器互联 20](#_Toc477352174)

[6.1 Mysql和测试环境容器互联 20](#_Toc477352175)

[6.2 验证容器互联 21](#_Toc477352176)

[7 在图形化界面shipyard上进行操作 21](#_Toc477352177)

[7.1 查看镜像 22](#_Toc477352178)

[7.2 创建容器 22](#_Toc477352179)

[7.3 查看容器信息 22](#_Toc477352180)

[7.4 进入容器 23](#_Toc477352181)

[7.5 退出容器 24](#_Toc477352182)

# Docker介绍

## Docker简介

### 什么是docker

Docker是一个开源的引擎，可以轻松的为任何应用创建一个轻量级的、可移植的、自给自足的容器。开发者在笔记本上编译测试通过的容器可以批量地在生产环境中部署，包括VMs（虚拟机）、bare metal、OpenStack 集群和其他的基础应用平台。

Docker 使用 Google 公司推出的 Go 语言 进行开发实现，基于 Linux 内核的 cgroup，namespace，以及 AUFS 类的 Union FS 等技术，**对进程进行封装隔离**，属于**操作系统层面的虚拟化技术**。由于隔离的进程独立于宿主和其它的隔离的进程，因此也称其为容器。

Docker 在容器的基础上，进行了进一步的封装，从文件系统、网络互联到进程隔离等等，极大的简化了容器的创建和维护。使得 Docker 技术比虚拟机技术更为轻便、快捷。

### 为什么要使用docker

作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式相比具有众多的优势。

* **更高效的利用系统资源**

由于容器不需要进行硬件虚拟以及运行完整操作系统等额外开销，Docker 对系统资源的利用率更高。无论是应用执行速度、内存损耗或者文件存储速度，都要比传统虚拟机技术更高效。因此，相比虚拟机技术，一个相同配置的主机，往往可以运行更多数量的应用。

* **更快速的启动时间**

传统的虚拟机技术启动应用服务往往需要数分钟，而 Docker 容器应用，由于直接运行于宿主内核，无需启动完整的操作系统，因此可以做到秒级、甚至毫秒级的启动时间。大大的节约了开发、测试、部署的时间。

* **一致的运行环境**

开发过程中一个常见的问题是环境一致性问题。由于开发环境、测试环境、生产环境不一致，导致有些 bug 并未在开发过程中被发现。而 Docker 的镜像提供了除内核外完整的运行时环境，确保了应用运行环境一致性。

* **持续交付和部署**

对开发和运维人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。

使用 Docker 可以通过定制应用镜像来实现持续集成、持续交付、部署。开发人员可以通过 [Dockerfile](https://docs.docker.com/engine/reference/builder/) 来进行镜像构建，并结合 [持续集成](https://en.wikipedia.org/wiki/Continuous_integration) 系统进行集成测试，而运维人员则可以直接在生产环境中快速部署该镜像，甚至结合 [持续部署](https://en.wikipedia.org/wiki/Continuous_delivery) 系统进行自动部署。

而且使用 Dockerfile 使镜像构建透明化，不仅仅开发团队可以理解应用运行环境，也方便运维团队理解应用运行所需条件，帮助更好的生产环境中部署该镜像。

* **更轻松的迁移**

由于 Docker 确保了执行环境的一致性，使得应用的迁移更加容易。Docker 可以在很多平台上运行，无论是物理机、虚拟机、公有云、私有云，甚至是笔记本，其运行结果是一致的。因此用户可以很轻易的将在一个平台上运行的应用，迁移到另一个平台上，而不用担心运行环境的变化导致应用无法正常运行的情况。

* **更轻松的维护和扩展**

Docker 使用的分层存储以及镜像的技术，使得应用重复部分的复用更为容易，也使得应用的维护更新更加简单，基于基础镜像进一步扩展镜像也变得非常简单。此外，Docker 团队同各个开源项目团队一起维护了一大批高质量的[官方镜像](https://hub.docker.com/explore/)，既可以直接在生产环境使用，又可以作为基础进一步定制，大大的降低了应用服务的镜像制作成本。

* **对比传统虚拟机总结**

|  |  |  |
| --- | --- | --- |
| 特性 | 容器 | 虚拟机 |
| 启动 | 秒级 | 分钟级 |
| 硬盘使用 | 一般为 MB | 一般为 GB |
| 性能 | 接近原生 | 弱于 |
| 系统支持量 | 单机支持上千个容器 | 一般几十个 |

## 基本概念

### 镜像

#### Docker镜像

我们都知道，操作系统分为内核和用户空间。对于 Linux 而言，内核启动后，会挂载 root 文件系统为其提供用户空间支持。而 Docker 镜像（Image），就相当于是一个 root 文件系统。比如官方镜像 ubuntu:14.04 就包含了完整的一套 Ubuntu 14.04 最小系统的 root 文件系统。

Docker 镜像是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的一些配置参数（如匿名卷、环境变量、用户等）。镜像不包含任何动态数据，其内容在构建之后也不会被改变。

#### 分层存储

因为镜像包含操作系统完整的 root 文件系统，其体积往往是庞大的，因此在 Docker 设计时，就充分利用[Union FS](https://en.wikipedia.org/wiki/Union_mount) 的技术，将其设计为分层存储的架构。所以严格来说，镜像并非是像一个 ISO 那样的打包文件，镜像只是一个虚拟的概念，其实际体现并非由一个文件组成，而是由**一组文件系统**组成，或者说，由多层文件系统联合组成。

镜像构建时，会一层层构建，前一层是后一层的基础。每一层构建完就不会再发生改变，后一层上的任何改变只发生在自己这一层。比如，删除前一层文件的操作，实际不是真的删除前一层的文件，而是仅在当前层标记为该文件已删除。在最终容器运行的时候，虽然不会看到这个文件，但是实际上该文件会一直跟随镜像。因此，在构建镜像的时候，需要额外小心，每一层尽量只包含该层需要添加的东西，任何额外的东西应该在该层构建结束前清理掉。

分层存储的特征还使得镜像的复用、定制变的更为容易。甚至可以用之前构建好的镜像作为基础层，然后进一步添加新的层，以定制自己所需的内容，构建新的镜像。

### 容器

镜像（Image）和容器（Container）的关系，就像是面向对象程序设计中的类和实例一样，镜像是静态的定义，容器是镜像运行时的实体。容器可以被创建、启动、停止、删除、暂停等。

容器的实质是进程，但与直接在宿主执行的进程不同，容器进程运行于属于自己的独立的命名空间。因此容器可以拥有自己的 root 文件系统、自己的网络配置、自己的进程空间，甚至自己的用户 ID 空间。**容器内的进程是运行在一个隔离的环境里**，使用起来，就好像是在一个独立于宿主的系统下操作一样。这种特性使得容器封装的应用比直接在宿主运行更加安全。也因为这种隔离的特性，很多人初学 Docker 时常常会把容器和虚拟机搞混。

镜像使用的是分层存储，容器也是如此。**每一个容器运行时，是以镜像为基础层，在其上创建一个当前容器的存储层**，我们可以称这个为容器运行时读写而准备的存储层为容器存储层。

容器存储层的生存周期和容器一样，容器消亡时，容器存储层也随之消亡。因此，**任何保存于容器存储层的信息都会随容器删除而丢失**。

按照 Docker 最佳实践的要求，容器不应该向其存储层内写入任何数据，容器存储层要保持无状态化。所有的文件写入操作，都应该使用 **数据卷**（Volume）、或者绑定宿主目录，在这些位置的读写会跳过容器存储层，直接对宿主(或网络存储)发生读写，其性能和稳定性更高。

数据卷的生存周期独立于容器，容器消亡，数据卷不会消亡。因此，使用数据卷后，容器可以随意删除、重新 run，数据却不会丢失。

### 仓库

#### Docker Registry

镜像构建完成后，可以很容易的在当前宿主上运行，但是，如果需要在其它服务器上使用这个镜像，我们就需要一个集中的存储、分发镜像的服务，Docker Registry 就是这样的服务。

一个 Docker Registry 中可以包含多个仓库（Repository）；每个仓库可以包含多个标签（Tag）；每个标签对应一个镜像。

通常，一个仓库会包含同一个软件不同版本的镜像，而标签就常用于对应该软件的各个版本。我们可以通过 <仓库名>:<标签> 的格式来指定具体是这个软件哪个版本的镜像。如果不给出标签，将以 latest 作为默认标签。

以 Ubuntu 镜像 为例，ubuntu 是仓库的名字，其内包含有不同的版本标签，如，14.04, 16.04。我们可以通过 ubuntu:14.04，或者 ubuntu:16.04 来具体指定所需哪个版本的镜像。如果忽略了标签，比如 ubuntu，那将视为 ubuntu:latest。

#### Docker Registry 公开服务

Docker Registry 公开服务是开放给用户使用、允许用户管理镜像的 Registry 服务。一般这类公开服务允许用户免费上传、下载公开的镜像，并可能提供收费服务供用户管理私有镜像。

最常使用的 Registry 公开服务是官方的 Docker Hub，这也是默认的 Registry，并拥有大量的高质量的官方镜像。

#### 私有 Docker Registry

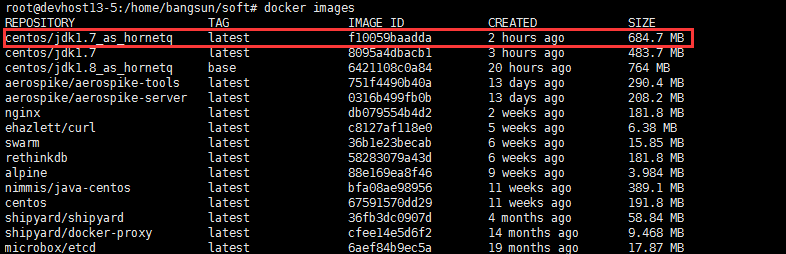
除了使用公开服务外，用户还可以在本地搭建私有 Docker Registry。Docker 官方提供了 Docker Registry 镜像，可以直接使用做为私有 Registry 服务。

# 测试环境容器应用

## 创建测试用容器

### 基础镜像centos/jdk1.7\_as\_hornetq

使用docker image命令查看镜像信息，看到一个名为centos/jdk1.7\_as\_hornetq的镜像，该镜像提供了centos系统，并已装好jdk1.7、aerospike、hornetq。



### 创建容器

使用**docker run -it centos/jdk1.7\_as\_hornetq**命令来创建容器，指定镜像为centos/jdk1.7\_as\_hornetq。容器创建成功后，直接进入容器终端



#### 创建容器相关命令

* **创建基本容器**

docker run **-it** centos/jdk1.7\_as\_hornetq

-it 参数指定创建容器后进入控制台

* **创建容器，需开启aerospike**

docker run **--privileged -d** centos/jdk1.7\_as\_hornetq **/usr/sbin/init**

--privileged 参数指定root拥有真正的root权限。否则，容器内的root只是外部的一个普通用户权限

-d 参数 指定容器在后台运行

/usr/sbin/init 在容器内执行/usr/sbin/init命令

* **创建容器，需暴露容器内端口**

docker run **-p 5001:4001** -it centos/jdk1.7\_as\_hornetq

-p 参数指定容器暴露的端口，前面的端口为宿主机端口，后面的端口号是容器端口，外部可以根据暴露出来的端口连接到容器内部

* **创建容器，需挂载宿主目录**

docker run **-v /home/bangsun/solution/:/home/** centos/jdk1.7\_as\_hornetq

-v 参数指定容器挂载存储卷，:前面的路径是宿主机目录路径，后面是容器内目录路径

* **推荐用命令**

docker run **-v /home/bangsun/solution:/home/** **--privileged -d -p 10041:10041 -p 10001:10001** centos/jdk1.7\_as\_hornetq **/usr/sbin/init**

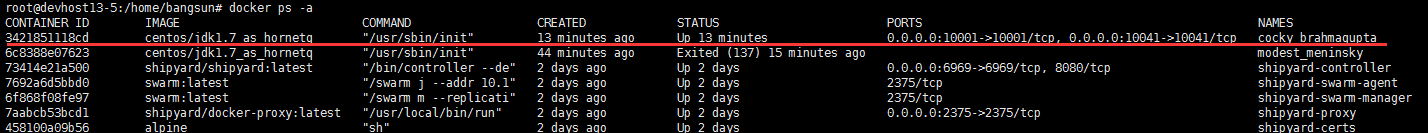
实际应用中，我们用这个命令创建测试用容器

#### 查看容器信息相关命令

* **查看已创建的容器**

docker ps -a

可以看到container id 为342开头的容器被创建，端口10001被映射到宿主机的10001端口，10041被映射到10041端口。即，访问宿主机的10001端口就相当于访问容器内的10001端口。



* **查看容器运行时详细信息**

docker inspect [容器id]





* **查看容器IP地址**

docker inspect [容器id]|grep IPAddress

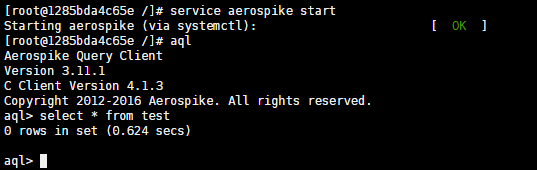


## 容器内操作

容器内的操作就跟在centos操作系统上的相同。

### 启动aerospike

[root@1285bda4c65e /]# **service aerospike start**



aerospike配置文件所在路径：/etc/aerospike/aerospike.conf

默认端口：3000

注意：容器内启动aerospike需指定—privileged以及执行/usr/sbin/init

启动命令参考2.1.2.1相关命令章节

### 启动hornetq

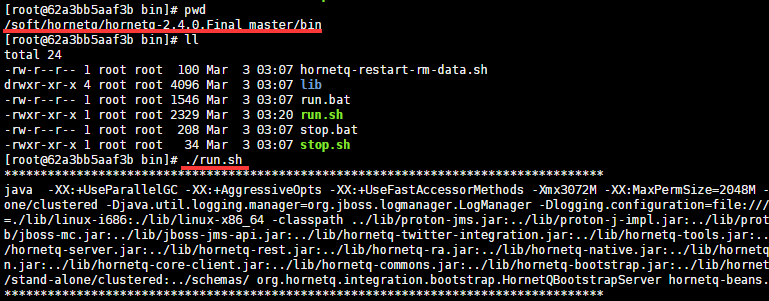
Hornetq：172.17.0.2:5548,172.17.0.2:5547

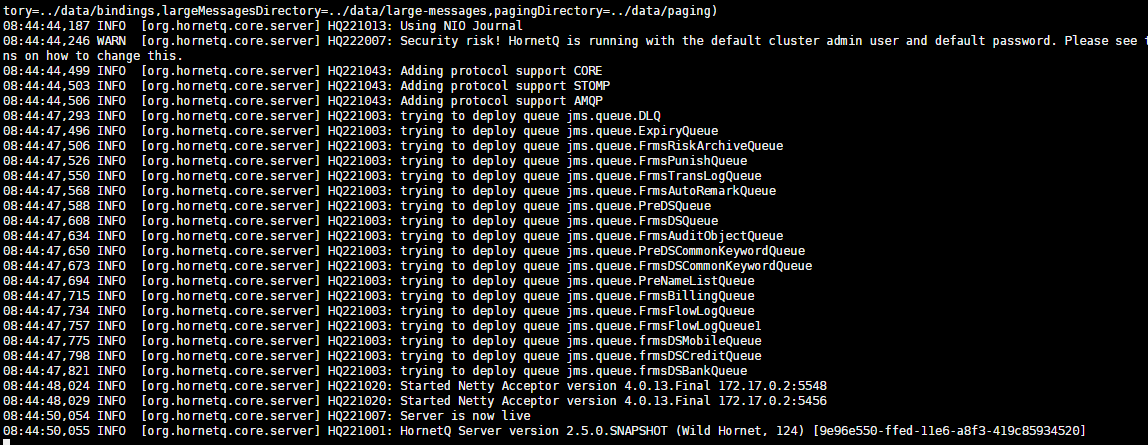
Jconsole：4001,4002

* **hornetq主**

执行命令路径：/soft/hornetq/hornetq-2.4.0.Final\_master/bin

[root@62a3bb5aaf3b bin]# **./run.sh**

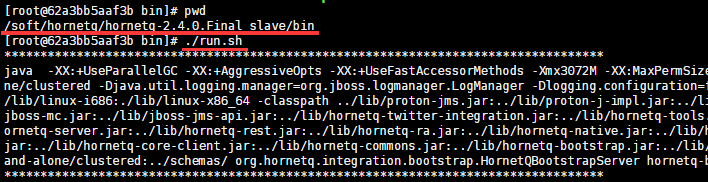


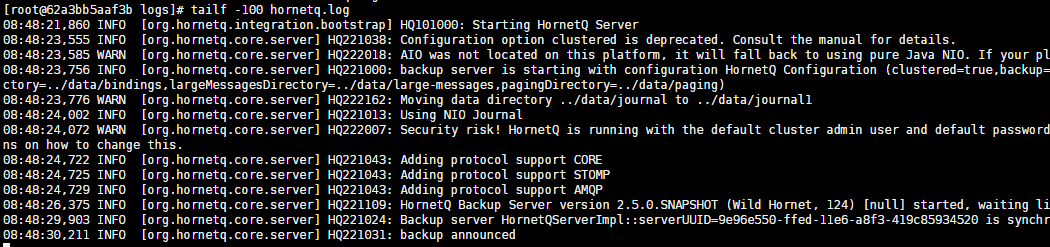


* **Hornetq备**

执行命令路径：/soft/hornetq/hornetq-2.4.0.Final\_slave/bin

[root@62a3bb5aaf3b bin]# **./run.sh**





### 测试环境配置举例

本文以电子银行为例进行说明。

* **创建容器**

docker run -v /home/bangsun/solution:/home/ --privileged -d -p 10041:10041 -p 10001:10001 centos/jdk1.7\_as\_hornetq /usr/sbin/init

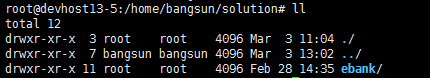
-p参数暴露10041端口和10001端口，用于外部连接。

-v参数挂载/home/bangsun/solution目录到容器内的/home目录。

/usr/sbin/init 在容器内执行/usr/sbin/init命令。

* **查看目录是否挂载成功**

宿主机/home/bangsun/solution目录下有一个ebank文件夹，若挂载成功，容器/home目录下也应该有该文件夹



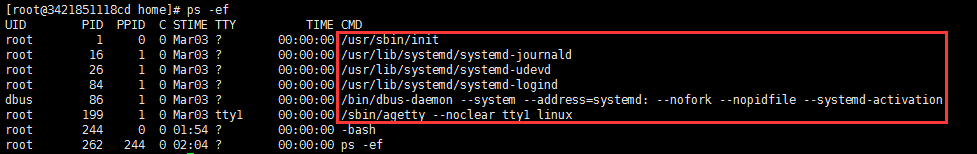
容器内查看：/home目录下已成功加载



注：宿主机的/home/bangsun/solution/ebank目录中放置了电子银行测试环境的所有包。挂载文件内容，宿主机和容器均可以修改。它的生命周期独立于容器，docker不会在容器删除后自动删除挂载目录中的数据。

* **查看/usr/sbin/init命令是否执行成功**

Ps –ef查看后台进程，可以看到如下进程，说明/usr/sbin/init命令执行成功



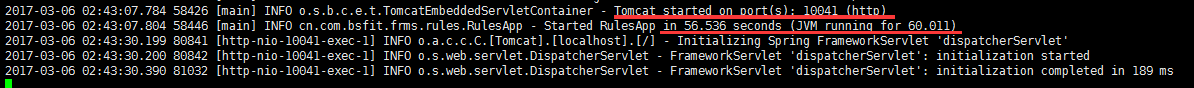
* **电子银行测试环境部署**

环境部署所在路径：/home/ebank

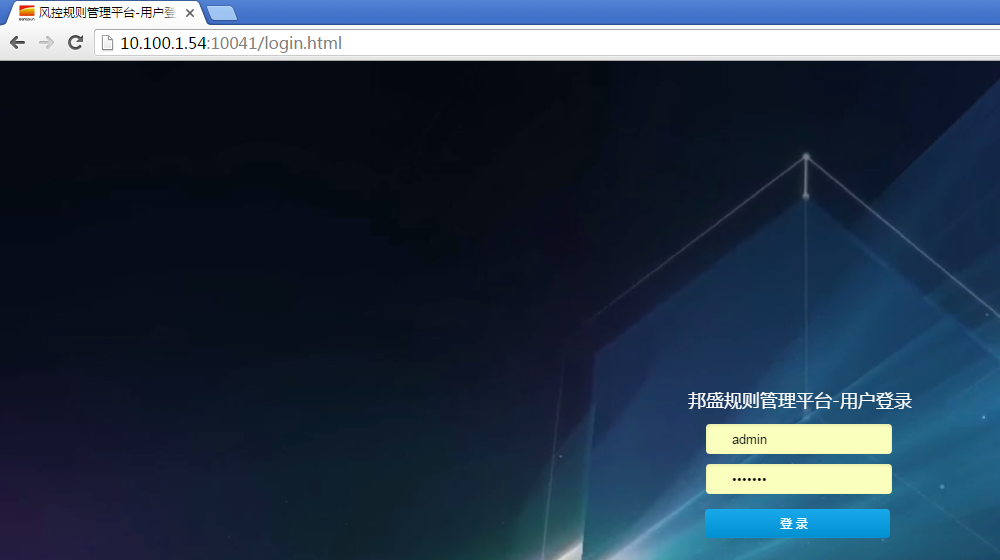
查看容器ip得到该容器内IP地址为：172.17.0.2（详情查看2.1.2.2章节）。

相应的修改各组件配置文件中关于ip的配置后启动即可。

* **启动规格平台后访问**



规则平台启动成功后，端口10041启动连接。根据创建容器时-p 10041:10041命令可知，容器内部的10041端口暴露在宿主机的10041端口，那么我们通过宿主机的10041端口访问。可正常访问：



Portal后台管理系统启动和运行的方式跟规则平台相同，就不再重复解释。

## 退出容器

可以通过logout或exit退出容器





## 连接容器

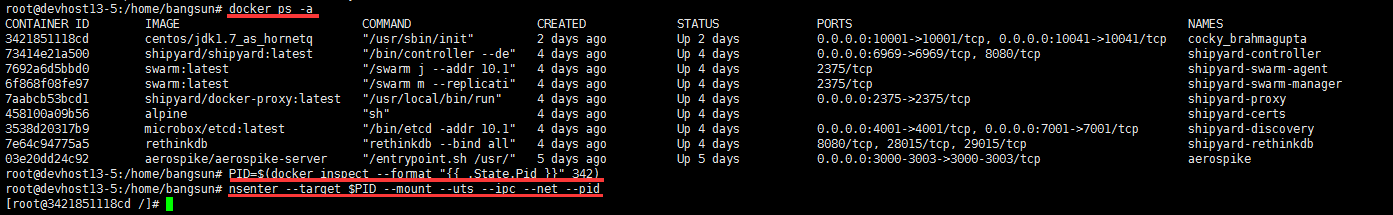
* **查看容器**

docker ps -a

* **使用nsenter进入容器**

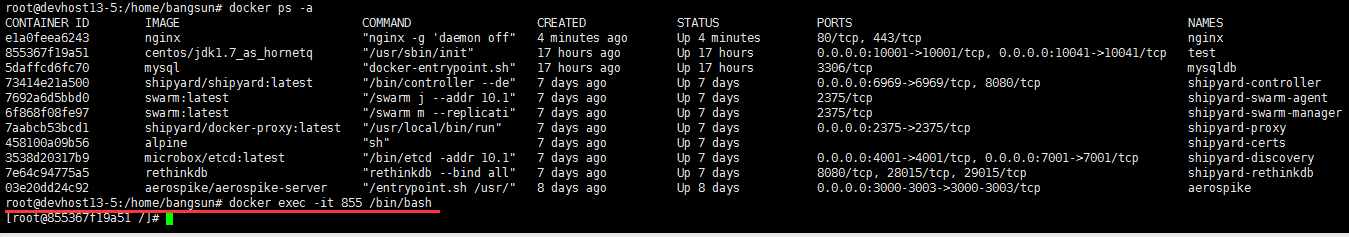
PID=$(docker inspect --format "{{ .State.Pid }}" [容器id])

nsenter --target $PID --mount --uts --ipc --net --pid



* **使用exec进入容器**

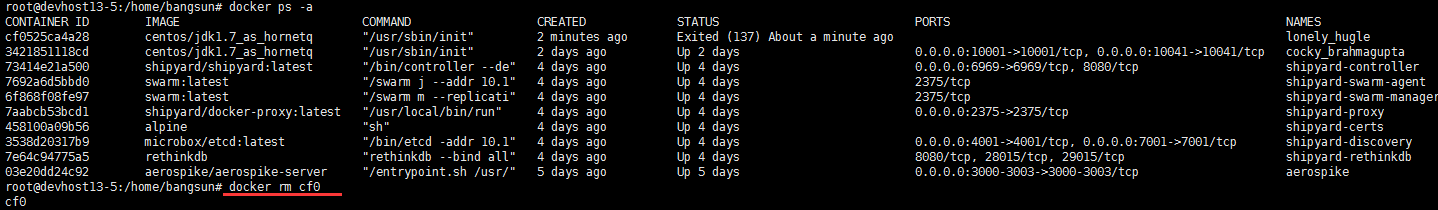
docker exec -it [容器id] /bin/bash



## 删除容器

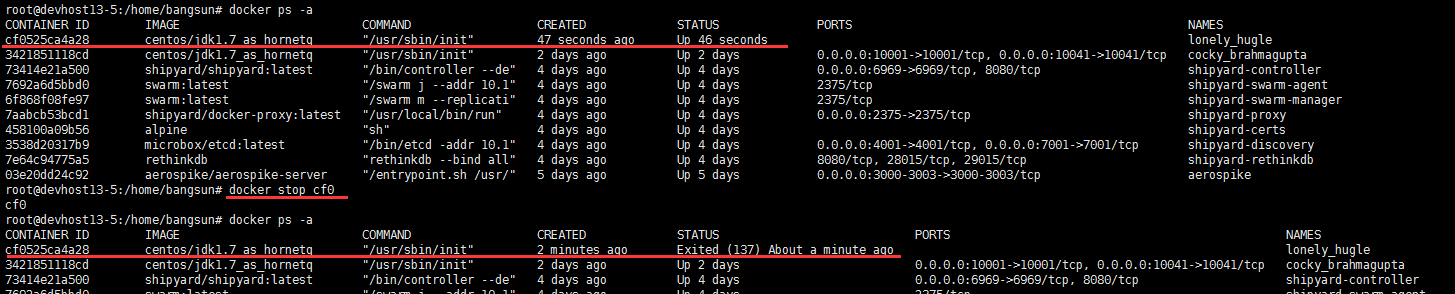
* **删除停止状态的容器**

docker rm [容器id]



* **停止容器**

docker stop [容器id]



如果要删除一个运行中的容器，可以添加 -f 参数。Docker 会发送 SIGKILL 信号给容器。

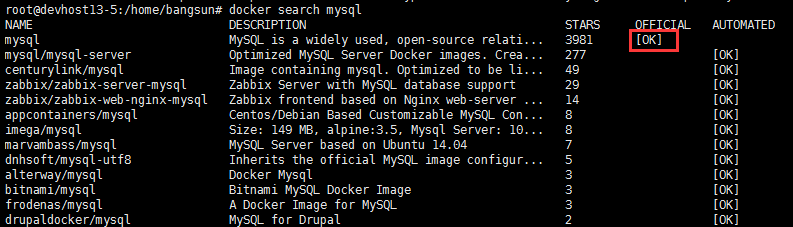
# Mysql容器应用

## Mysql镜像

这里介绍了mysql官方镜像的查找和下载方法，如本地已有该镜像，不需要重复下载。

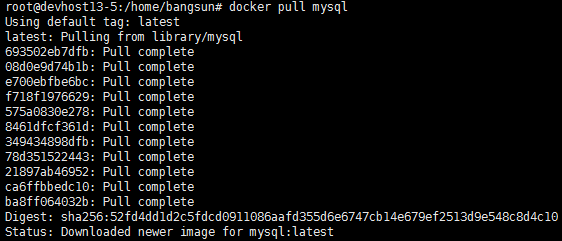
* **查找镜像**

通过docker search mysql可查看仓库中的可用镜像，有official标签的为官方镜像



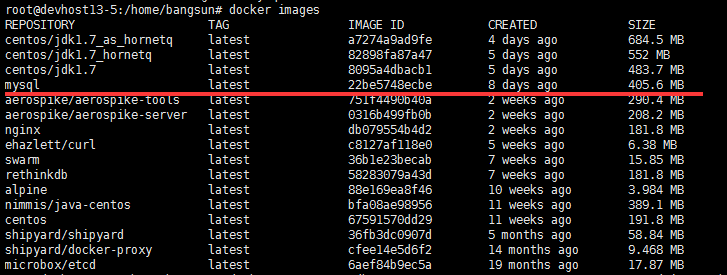
* **下载镜像**

通过docker pull mysql下载镜像，不指定标签的话默认下载latest



* **查看镜像**

下载结束后，可通过docker images查看到

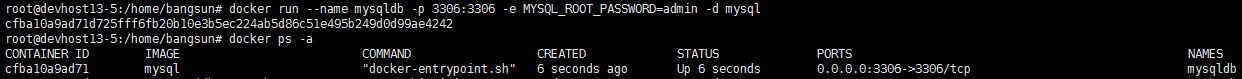


## 创建mysql容器

* **创建临时mysql容器**

docker run --name mysqldb -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=admin -d mysql

--name指定容器名称，-e MYSQL\_ROOT\_PASSWORD指定mysql的root密码



* **创建mysql容器，数据存放在本地目录，持久化数据**

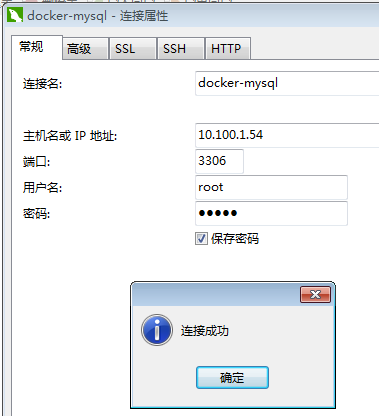
docker run --name mysqldb -p 3306:3306 -e MYSQL\_ROOT\_PASSWORD=admin -v /home/bangsun/DB/mysql/data:/var/lib/mysql -d mysql

-v /home/bangsun/DB/mysql/data:/var/lib/mysql将主机当前目录下的/home/bangsun/mysql/data挂载到容器的/var/lib/mysql

通过挂载主机目录的方式，可以持久化数据，就算mysql容器被删除，数据文件仍旧存在。

## Navicat连接mysql

通过navicat连接到mysql数据库。通过宿主机10.100.1.54的3306端口连接。

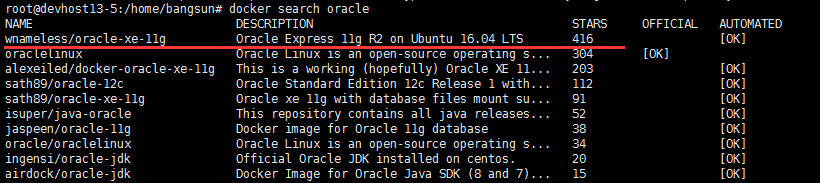


# Oracle容器应用

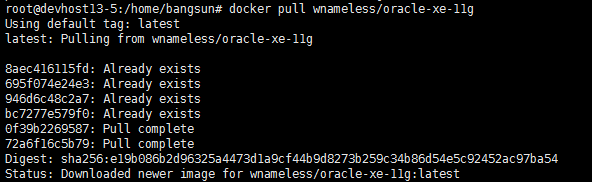
## Oracle镜像

这里介绍了oracle镜像的查找和下载方法，如本地已有该镜像，不需要重复下载。

* **下载wnameless的oralce镜像**



docker pull wnameless/oracle-xe-11g

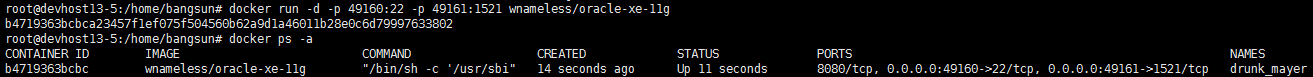


## 创建oracle容器

* **创建oracle容器**

docker run -d -p 49160:22 -p 49161:1521 wnameless/oracle-xe-11g

运行，并开放 49160 和 49161 端口，分别对应 22 端口和 Oracle 端口（SSH 和 oracle 数据库）



* **数据库信息**

hostname: localhost

port: 49161

sid: xe

username: system

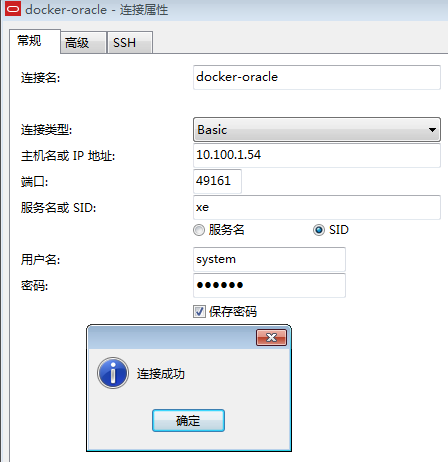
password: oracle

SYSTEM和SYS的初始密码都为 oracle

Container SSH 的 root 密码为admin

## Navicat连接oracle

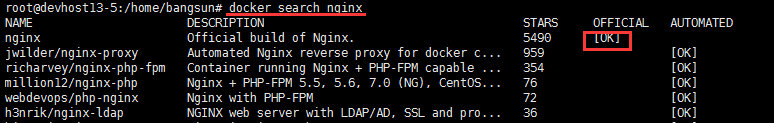
通过navicat连接到oracle数据库。通过宿主机10.100.1.54的49161端口来进行连接。



# Nginx容器应用

## Nginx镜像

* **下载官方nginx镜像**

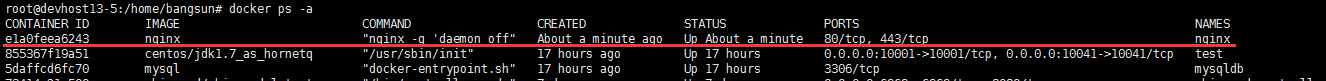




## 创建nginx镜像

* **创建nginx容器**

docker run --name nginx -d nginx



## 使用nginx容器做负载均衡

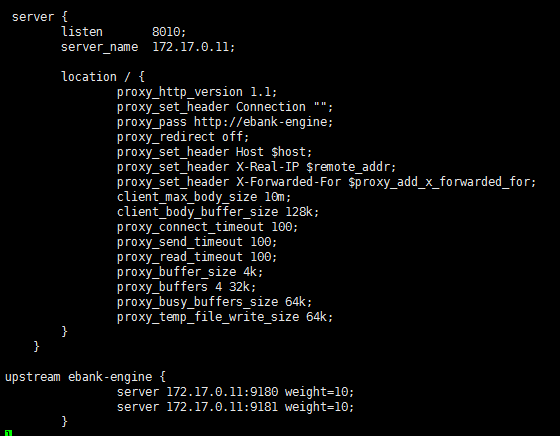
* **修改宿主机上的nginx配置**

以电子银行的引擎为例，修改nginx配置文件的方式如下：

修改宿主机上的配置文件vi /home/bangsun/soft/nginx/nginx.conf



添加相应的配置信息，举例如下：



* **将宿主机上的配置拷贝到nginx容器**

docker cp /home/bangsun/soft/nginx/nginx.conf [容器id]:/etc/nginx/nginx.conf



* **重启nginx容器，配置生效**

关闭

docker stop [容器id]

启动

docker start [容器id]

# 容器互联

容器的连接（linking）系统是除了端口映射外，另一种跟容器中应用交互的方式。

该系统会在源和接收容器之间创建一个隧道，接收容器可以看到源容器指定的信息。

## Mysql和测试环境容器互联

* **创建一个名为mysqldb的mysql容器**

docker run --name mysqldb -e MYSQL\_ROOT\_PASSWORD=admin -d mysql

在启动 mysql 容器的时候并没有使用 -p 和 -P 标记，从而避免了暴露数据库端口到外部网络上。

* **创建测试环境容器，并将它连接到mysqldb容器**

docker run -v /home/bangsun/solution:/home/ --privileged -d -p 10041:10041 -p 10001:10001 --name test **--link mysqldb:tomysql** centos/jdk1.7\_as\_hornetq /usr/sbin/init

此时，mysqldb 容器和 test 容器建立互联关系。

--link 参数的格式为 --link name:alias，其中 name 是要链接的容器的名称，alias 是这个连接的别名。

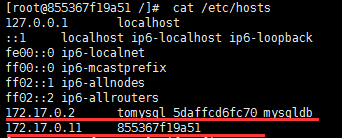
## 验证容器互联

Docker在两个互联的容器之间创建了一个安全隧道，而且不用映射它们的端口到宿主主机上，避免暴露数据库端口到外部网络上。

* **连接进入测试环境容器内**

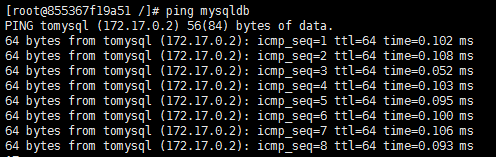
连接进入容器内部的命令详见2.4章节

* **查看hosts文件**



这里有两个hosts信息，第一个是mysqldb容器的IP、连接名、id和主机名，第二个是mysqldb容器，test容器用自己的id作为默认主机名。

也可以在测试环境容器内部ping mysqldb，可以ping通，说明和mysqldb互通。



注：可以链接多个父容器到子容器，比如可以链接多个 test 到 mysqldb 容器上。

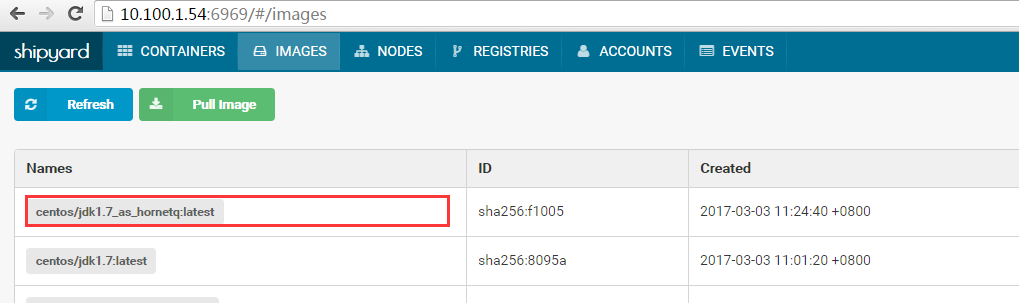
# 在图形化界面shipyard上进行操作

shipyard图形化管理平台

地址：10.100.1.54:6969

用户名/密码：admin/shipyard

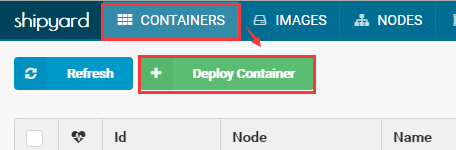
## 查看镜像



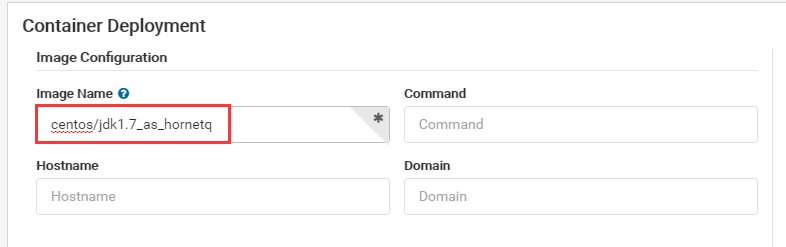
## 创建容器

Shipyard上创建容器：

1、containers(容器)页面点击deploy container

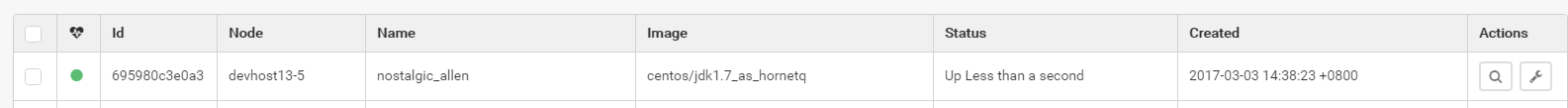


2、填写镜像名称后点击deploy



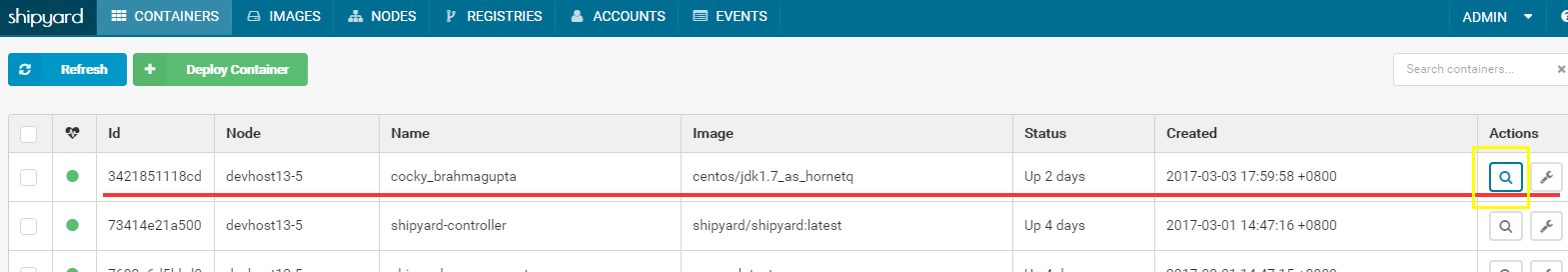


3、回到container页面，可查看到新建的容器



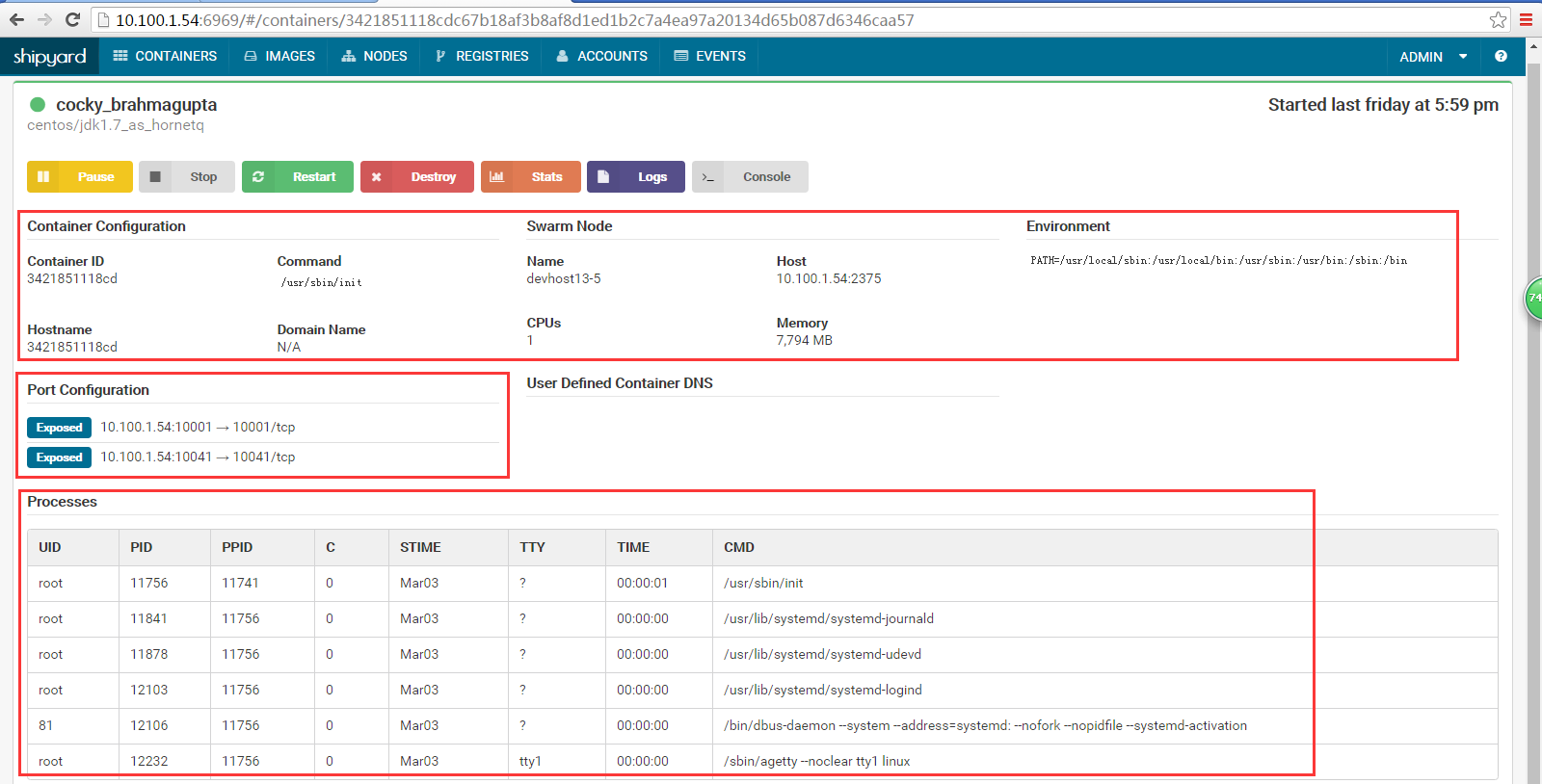
## 查看容器信息

1. Containers页面，选中要查看的容器，点击右侧的放大镜按钮



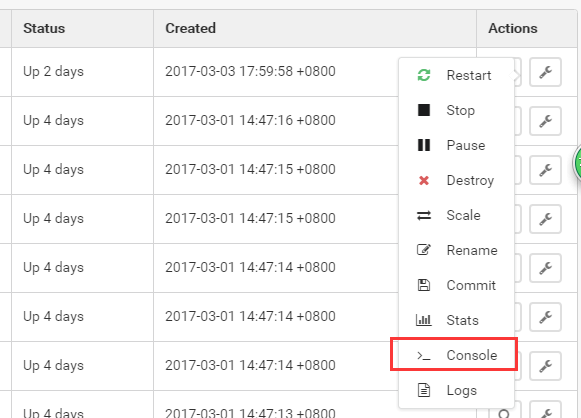
1. 页面跳转到详细信息页

展示了容器的配置信息，端口信息，容器内进行等信息。

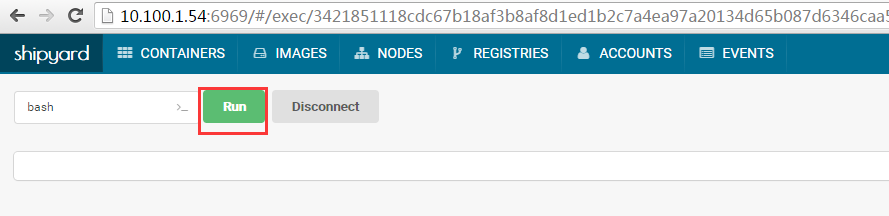


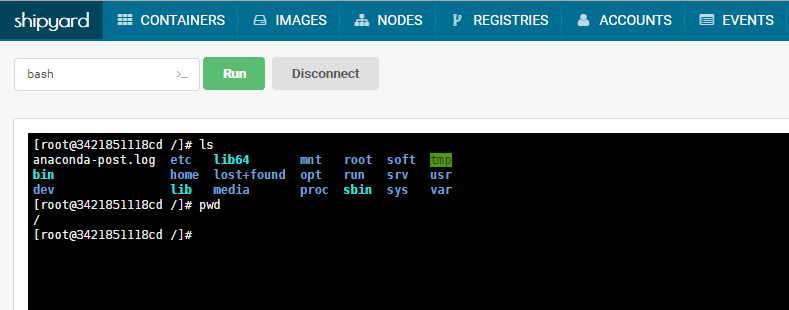
## 进入容器

1. containers页面选中容器后，点击右侧的工具按钮，选择console按钮



1. 点击绿色的run按钮即可





## 退出容器

点击disconnect按钮

