银河麒麟服务器操作系统 V4

docker 软件适配手册

天津麒麟信息技术有限公司

2019 年 6 月

目 录

1 概述 2
1.1 系统概述 2
1.2 环境概述 2
1.3 DOCKER 及相关软件简介 2
1.4 架构及原理 2
1.4.1 优点 4
2 使用方法 4
1) 安装 DOCKER 及相关软件 4
2) 设置开机自启动并启动 DOCKER-CE 4
3) DOCKER 容器使用 5
3.1 DOCKER 客户端 5
3.2 运行一个应用 5
3.3 查看应用容器 5
3.4 网络端口的快捷方式 6
3.5 查看应用程序日志 6
3.6 查看应用程序容器的进程 6
3.7 检查应用程序 7
3.8 停止应用容器 8
3.9 重启应用容器 8
3.10 移除应用容器 8
4) DOCKER 容器连接 8
1.1 网络端口映射 9
1.2 DOCKER 容器连接 10
1.2.1 容器命名 10
5) DOCKER 镜像使用 10
3.1 列出镜像列表 10
3.2 获取一个新的镜像 11
3.3 查找镜像 12
3.4 创建镜像 12
3.5 更新镜像 12

1 概述

1.1 系统概述

银河麒麟服务器操作系统主要面向军队综合电子信息系统、金融系统以及电力系统等国家关键行业的服务器应用领域，突出高安全性、高可用性、高效数据处理、虚拟化等关键技术优势，针对关键业务构建的丰富高效、安全可靠的功能特性，兼容适配长城、联想、浪潮、华为、曙光等国内主流厂商的服务器整机产品，以及达梦、金仓、神通、南大通用等主要国产数据库和中创、金蝶、东方通等国产中间件，满足虚拟化、云计算和大数据时代，服务器业务对操作系统在性能、安全性及可扩展性等方面的需求，是一款具有高安全、高可用、高可靠、高性能的自主可控服务器操作系统。

1.2 环境概述

服务器型号	长城信安擎天 DF720 服务器
CPU 类型	飞腾 2000+处理器
操作系统版本	Kylin-4.0.2-server-sp2-2000-19050910.Z1
内核版本	4.4.131
docker 版本	18.06.1~ce

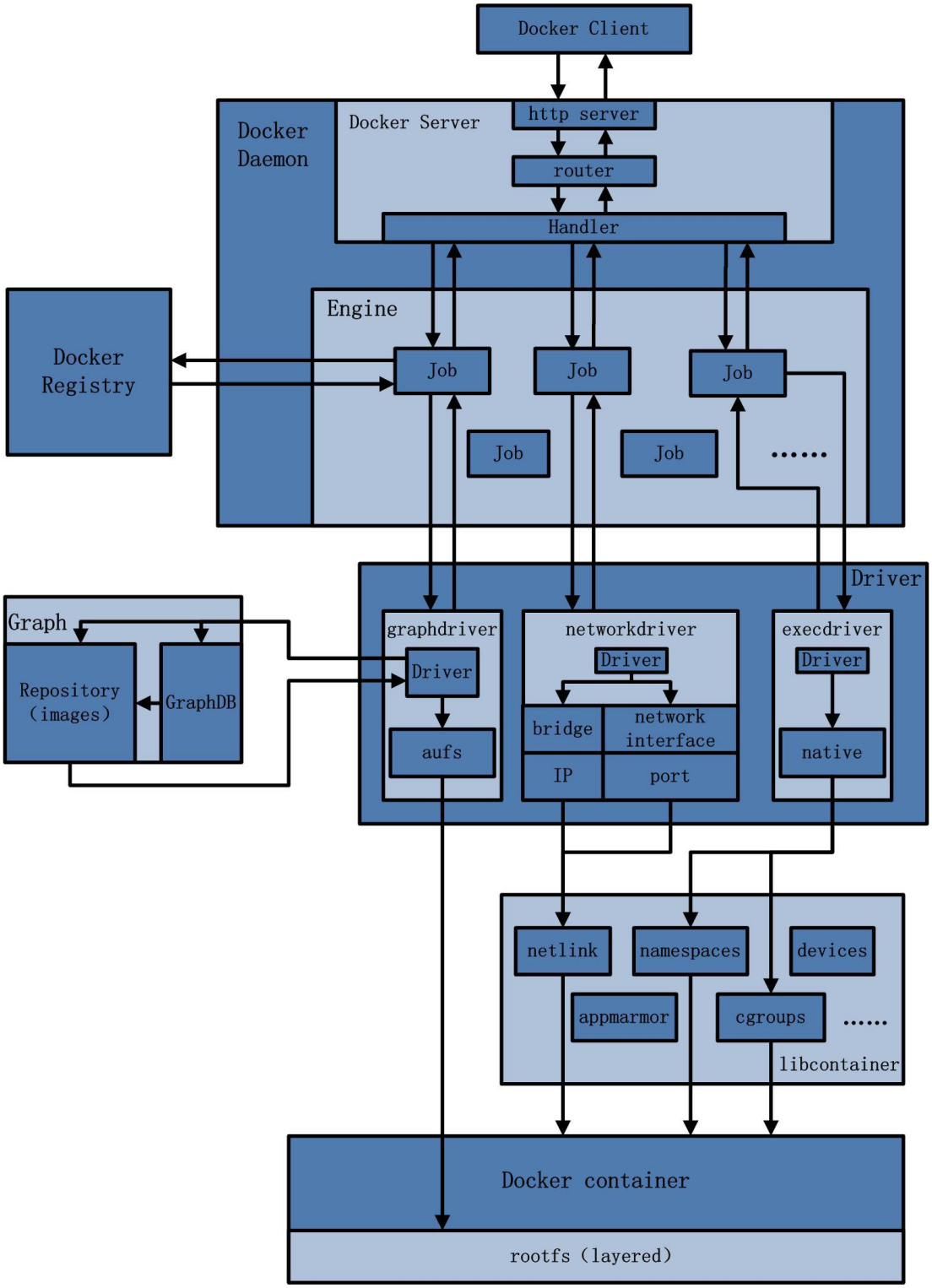
1.3 docker 及相关软件简介

Docker 是一个开源的应用容器引擎，基于 Go 语言并遵从 Apache2.0 协议开源。Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。

容器是完全使用沙箱机制，相互之间不会有任何接口，更重要的是容器性能开销极低。

1.4 架构及原理

架构如下图所示：



Docker 是一个 C/S 模式的架构，后端是一个松耦合架构。

- 用户使用 Docker Client 与 Docker Daemon 建立通信，并发送请求给后者。
- Docker Daemon 作为 Docker 架构中的主体部分，首先提供 Docker Server 的功能使其可以接受 Docker Client 的请求。
- Docker Engine 执行 Docker 内部的一系列工作，每一项工作都是以一个

Job 的形式的存在。

- Job 的运行过程中，当需要容器镜像时，则从 Docker Registry 中下载镜像，并通过镜像管理驱动 Graphdriver 将下载镜像以 Graph 的形式存储。
- 当需要为 Docker 创建网络环境时，通过网络管理驱动 Networkdriver 创建并配置 Docker 容器网络环境。
- 当需要限制 Docker 容器运行资源或执行用户指令等操作时，则通过 Execdriver 来完成。
- Libcontainer 是一项独立的容器管理包，Networkdriver 以及 Execdriver 都是通过 Libcontainer 来实现具体对容器进行的操作。

1.4.1 优点

1，简化程序

Docker 让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，便可以实现虚拟化。Docker 改变了虚拟化的方式，使开发者可以直接将自己的成果放入 Docker 中进行管理。方便快捷已经是 Docker 的最大优势，过去需要用数天乃至数周的 任务，在 Docker 容器的处理下，只需要数秒就能完成。

2、简化配置

Docker 镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

3、节省开支：

一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件，Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

2 使用方法

1) 安装 docker 及相关软件

```
[root@localhost ~]# apt-get install docker-ce docker-ce-cli containe
rd.io
```

2) 设置开机自启动并启动 Docker-ce

```
[root@localhost ~]#  sudo systemctl enable docker
[root@localhost ~]#  sudo systemctl start docker
```

3) Docker 容器使用

3.1 Docker 客户端

docker 客户端非常简单 ,我们可以直接输入 docker 命令来查看到 Docker 客户端的所有命令选项。

```
[root@localhost ~]# docker
```

可以通过命令 **docker command --help** 更深入的了解指定的 Docker 命令使用方法。

3.2 运行一个应用

使用 docker 构建一个数据库应用程序。

我们将在 docker 容器中运行一个 postgres 应用来运行一个数据库应用。

```
[root@localhost ~]# docker pull postgres  # 载入镜像
[root@localhost ~]# docker run \
          --name demo \
          --privileged=true \
          -e TZ='Asia/Shanghai' \
          -e POSTGRES_USER=koji \
          -e POSTGRES_DB=koji \
          -e PGDATA=/tmp/  \
          -v /root/data:/tmp/ \
          -p 5432:5432 \
          -v /root/pem/psql-run:/docker-entrypoint-initdb.d \
          -d \
          postgres
```

参数说明:

- **-d:**让容器在后台运行。
- **-P:**将容器内部使用的网络端口映射到我们使用的主机上。

3.3 查看应用容器

使用 docker ps 来查看我们正在运行的容器:

```
[root@localhost ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
 STATUS              PORTS                    NAMES
64b5f66eb029        demo-0823         "docker-entrypoint.s…"   9 months ago
Up 2 days           0.0.0.0:5432->5432/tcp   demo-0104
```

这里多了端口信息。

```
 PORTS
0.0.0.0:5432->5432/tcp
```

3.4 网络端口的快捷方式

通过 **docker ps** 命令可以查看到容器的端口映射，**docker** 还提供了另一个快捷方式 **docker port**，使用 **docker port** 可以查看指定 （ID 或者名字）容器的某个确定端口映射到宿主机的端口号。

上面我们创建的应用容器 ID 为 **64b5f66eb029** 名字为 demo-0104。

我可以使用 **docker port 64b5f66eb029** 或 **docker port demo-0104** 来查看容器端口的映射情况。

```
[root@localhost 2_demo]# docker port 64b5f66eb029
    5432/tcp -> 0.0.0.0:5432
[root@localhost 2_demo]# docker port demo-0104
    5432/tcp -> 0.0.0.0:5432
```

3.5 查看应用程序日志

docker logs [ID 或者名字] 可以查看容器内部的标准输出。

```
root@localhost:~# docker logs -f 64b5f66eb029
```

-f: 让 **docker logs** 像使用 **tail -f** 一样来输出容器内部的标准输出。

3.6 查看应用程序容器的进程

我们还可以使用 docker top 来查看容器内部运行的进程

```
[root@localhost 2_demo]# docker top demo-0104
    UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
    polkitd             3840                3825                0
Jun11               ?                   00:00:01            postgres
    polkitd             3900                3840                0
Jun11               ?                   00:00:00            postgres: checkpointer p
rocess
    polkitd             3901                3840                0
Jun11               ?                   00:00:01            postgres: writer process
    polkitd             3902                3840                0
Jun11               ?                   00:00:01            postgres: wal writer pro
```

```
cess
    polkitd             3903                3840                0
Jun11               ?                   00:00:01            postgres: autovacuum lau
ncher process
    polkitd             3904                3840                0
Jun11               ?                   00:00:02            postgres: stats collecto
r process
```

3.7 检查应用程序

使用 **docker inspect** 来查看 Docker 的底层信息。它会返回一个 JSON 文件记录着 Docker 容器的配置和状态信息。

```
[root@localhost 2_demo]# docker inspect demo-0104
    [
        {
            "Id": "64b5f66eb029f048c28b77b31e47eb8644dc59a84f0db86345c39ef9ae36bce5
",
            "Created": "2018-08-26T09:43:50.894829237Z",
            "Path": "docker-entrypoint.sh",
            "Args": [
                "postgres"
            ],
            "State": {
                "Status": "running",
                "Running": true,
                "Paused": false,
                "Restarting": false,
                "OOMKilled": false,
                "Dead": false,
                "Pid": 3840,
                "ExitCode": 0,
                "Error": "",
                "StartedAt": "2019-06-11T06:28:47.716331395Z",
                "FinishedAt": "2019-06-11T06:19:10.629685152Z"
            },
            "Image": "sha256:cb1efd8307d258d26134d69c12e0c72d031f04e3d8685475b3af747
e1bdfe6ed",
            "ResolvConfPath": "/var/lib/docker/containers/64b5f66eb029f048c28b77b31e
47eb8644dc59a84f0db86345c39ef9ae36bce5/resolv.conf",
            "HostnamePath": "/var/lib/docker/containers/64b5f66eb029f048c28b77b31e47
eb8644dc59a84f0db86345c39ef9ae36bce5/hostname",
            "HostsPath": "/var/lib/docker/containers/64b5f66eb029f048c28b77b31e47eb8
```

```
644dc59a84f0db86345c39ef9ae36bce5/hosts",
            "LogPath": "/var/lib/docker/containers/64b5f66eb029f048c28b77b31e47eb864
4dc59a84f0db86345c39ef9ae36bce5/64b5f66eb029f048c28b77b31e47eb8644dc59a84f0db86345c
39ef9ae36bce5-json.log",
    ……
```

3.8 停止应用容器

```
docker stop demo-0104
```

3.9 重启应用容器

已经停止的容器，我们可以使用命令 docker start 来启动。

```
docker start demo-0104
```

docker ps -l 查询最后一次创建的容器：

```
[root@localhost 2_demo]# docker ps -l
    CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
    0341a255b414        centos              "/bin/bash"         6 weeks ago
Created                                 zealous_payne
```

正在运行的容器，我们可以使用 docker restart 命令来重启

3.10 移除应用容器

我们可以使用 docker rm 命令来删除不需要的容器

```
[root@localhost 2_demo]# docker rm demo-0104
```

删除容器时，容器必须是停止状态，否则会报如下错误

```
[root@localhost 2_demo]# docker rm demo-0104
    Error response from daemon: You cannot remove a running container 64b5f66eb029f0
48c28b77b31e47eb8644dc59a84f0db86345c39ef9ae36bce5. Stop the container before attem
pting removal or force remove
```

4) Docker 容器连接

前面我们实现了通过网络端口来访问运行在 docker 容器内的服务。下面我们来实现通过端口连接到一个 docker 容器

1.1 网络端口映射

我们创建了一个 python 应用的容器。

```
[root@localhost 2_demo]# docker run -d -P postgres
47020d64305fbc1b055b94ddad90c4fef41a7f3937c66e1a10c075fb5544f216
```

另外，我们可以指定容器绑定的网络地址，比如绑定 127.0.0.1。

我们使用 **-P** 参数创建一个容器，使用 **docker ps** 可以看到容器端口 5432 绑定主机端口 32768。

```
[root@localhost 2_demo]# docker port modest_poincare
5432/tcp -> 0.0.0.0:32768
```

我们也可以使用 **-p** 标识来指定容器端口绑定到主机端口。

两种方式的区别是:

- **-P :**是容器内部端口**随机**映射到主机的高端口。
- **-p :** 是容器内部端口绑定到**指定**的主机端口。

```
[root@localhost 2_demo]# docker run -d -p 127.0.0.1:5000:5432 postgres
5727839cc310b43abba20d79204c16fd3f2d2077cad8bd94e84cae5958b2e4c1
[root@localhost 2_demo]# docker port zen_hugle
5432/tcp -> 127.0.0.1:5000
```

另外，我们可以指定容器绑定的网络地址，比如绑定 127.0.0.1。

```
[root@localhost 2_demo]# docker run -d -p 127.0.0.1:5001:5432 postgres
9caceea75cab6e1b1f02b76dacbda9f3fd68da36cb1d8f0319b6f1f1518984b7
[root@localhost 2_demo]# docker port vibrant_mayer
5432/tcp -> 127.0.0.1:5001
```

这样我们就可以通过访问 127.0.0.1:5001 来访问容器的 5432 端口。

上面的例子中，默认都是绑定 tcp 端口，如果要绑定 UDP 端口，可以在端口后面加上 **/udp**。

```
[root@localhost 2_demo]# docker run -d -p 127.0.0.1:5000:5432/udp postgres
41bb1046c6b13c575140dfbf4096e982c2c547c3e3c42e683aa89efe9bb754c3
[root@localhost 2_demo]# docker port zealous_wescoff
5432/udp -> 127.0.0.1:5000
```

1.2 Docker 容器连接

端口映射并不是唯一把 docker 连接到另一个容器的方法。

docker 有一个连接系统允许将多个容器连接在一起，共享连接信息。

docker 连接会创建一个父子关系，其中父容器可以看到子容器的信息。

1.2.1 容器命名

当我们创建一个容器的时候，docker 会自动对它进行命名。另外，我们也可以使用 **--name** 标识来命名容器，例如：

```
[root@localhost 2_demo]# docker run -d -P --name kylin postgres
dd4f4c33f1d459a754f1da242176f521eec6364376dd2fe7d6f9f9647ee97dfe
```

我们可以使用 **docker ps** 命令来查看容器名称。

```
[root@localhost 2_demo]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
 STATUS              PORTS                     NAMES
dd4f4c33f1d4        postgres            "docker-entrypoint.s…"   57 seconds ago
  Up 56 seconds       0.0.0.0:32769->5432/tcp   kylin
64b5f66eb029        demo-0823         "docker-entrypoint.s…"   9 months ago
Up 2 days           0.0.0.0:5432->5432/tcp    demo-0104
```

5) Docker 镜像使用

当运行容器时，使用的镜像如果在本地中不存在，docker 就会自动从 docker 镜像仓库中下载，默认是从 Docker Hub 公共镜像源下载。

下面我们来学习：

- 1、管理和使用本地 Docker 主机镜像
- 2、创建镜像

3.1 列出镜像列表

我们可以使用 **docker images** 来列出本地主机上的镜像。

```
[root@localhost 2_demo]# docker images
```

```
REPOSITORY              TAG                 IMAGE ID            CREATED
SIZE
centos                  latest              9f38484d220f        3 months ago
202MB
httpd                   latest              ef1dc54703e2        5 months ago
132MB
centos                  <none>              1e1148e4cc2c        6 months ago
202MB
demo-0823             latest              cb1efd8307d2        9 months ago        26
9MB
postgres                latest              45e33d1af449        9 months ago
228MB
google/nodejs-runtime   latest              bfadf5bbacf1        3 years ago
88.2MB
progrium/stress         latest              db646a8f4087        4 years ago
282MB
```

各个选项说明:

- **REPOSITORY**：表示镜像的仓库源
- **TAG**：镜像的标签
- **IMAGE ID**：镜像 ID
- **CREATED**：镜像创建时间
- **SIZE**：镜像大小

同一仓库源可以有多个 TAG，代表这个仓库源的不同个版本.

3.2 获取一个新的镜像

当我们在本地主机上使用一个不存在的镜像时 Docker 就会自动下载这个镜像。如果我们想预先下载这个镜像，我们可以使用 docker pull 命令来下载它。

```
[root@localhost 2_demo]# docker pull ubuntu:16.04
16.04: Pulling from library/ubuntu
9ff7e2e5f967: Pull complete
59856638ac9f: Pull complete
6f317d6d954b: Pull complete
a9dde5e2a643: Pull complete
Digest: sha256:cad5e101ab30bb7f7698b277dd49090f520fe063335643990ce8fbd15ff920ef
Status: Downloaded newer image for ubuntu:16.04
```

下载完成后，我们可以直接使用这个镜像来运行容器。

3.3 查找镜像

```
root@localhost:~$  docker search httpd
```

NAME:镜像仓库源的名称

DESCRIPTION:镜像的描述

OFFICIAL:是否 docker 官方发布

3.4 创建镜像

当我们从 docker 镜像仓库中下载的镜像不能满足我们的需求时，我们可以通过以下两种方式对镜像进行更改。

- 1.从已经创建的容器中更新镜像，并且提交这个镜像
- 2.使用 Dockerfile 指令来创建一个新的镜像

3.5 更新镜像

更新镜像之前，我们需要使用镜像来创建一个容器。

```
[root@localhost 2_demo]# docker run -it  ubuntu:16.04
root@84b9fd344162:/#
```

在运行的容器内使用 apt-get update 命令进行更新。

在完成操作之后，输入 exit 命令来退出这个容器。

此时 ID 为 84b9fd344162 的容器，是按我们的需求更改的容器。我们可以通过命令 docker commit 来提交容器副本。

```
[root@localhost 2_demo]# docker commit -m="has update" -a="kylin" 84b9fd344162 kylin/ubuntu:v2
sha256:f8d8389ebaa7e642f87983ee8264c569aa8ddf451ee1fc62476e346965e91616
```

各个参数说明：

- **-m:**提交的描述信息
- **-a:**指定镜像作者
- 84b9fd344162：容器 ID
- **kylin/ubuntu:v2:**指定要创建的目标镜像名

我们可以使用 **docker images** 命令来查看我们的新镜像 **kylin/ubuntu:v2**：

```
[root@localhost 2_demo]# docker images
REPOSITORY              TAG                 IMAGE ID            CREATED
```

```
  SIZE
  kylin/ubuntu            v2                  f8d8389ebaa7        35 seconds ago
  119MB
  ubuntu                  16.04               2a697363a870        4 weeks ago
  119MB
  centos                  latest              9f38484d220f        3 months ago
  202MB
  httpd                   latest              ef1dc54703e2        5 months ago
  132MB
  centos                  <none>              1e1148e4cc2c        6 months ago
  202MB
  demo-0823             latest              cb1efd8307d2        9 months ago
269MB
  postgres                latest              45e33d1af449        9 months ago
  228MB
  google/nodejs-runtime   latest              bfadf5bbacf1        3 years ago
  88.2MB
  progrium/stress         latest              db646a8f4087        4 years ago
  282MB
```

使用我们的新镜像 **kylin/ubuntu** 来启动一个容器

```
  [root@localhost 2_demo]# docker run -t -i kylin/ubuntu:v2 /bin/bash
  root@b7e292b88a90:/#
```