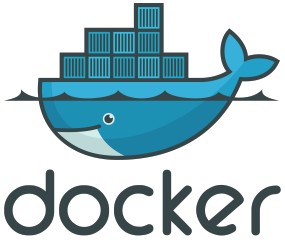
Docker 初级教程

Shen Hengheng

Institute of BigData, ShangRao
Institute of Computing Technology, Chinese Academy of Sciences



Contents

Deploying Your First Docker Container

任务：你将扮演 Jane 的角色，Jane 需要为正在使用的应用程序部署新的 Key-Value 存储数据库。经过讨论，决定使用 Redis，一个受欢迎的 KV 数据库。Jane 对 Redis 的部署方式并不熟悉，但听说 Docker 可以直接将服务部署到开发和生产中。

下面将讨论 Jane 将如何完成她的任务，并将 Redis 部署在 Docker 容器上用于生产和开发。

Running A Container

第一个任务是确定你要运行的 Redis 的 **Docker Image** 的名字。所有容器都是基于 Docker 镜像启动的。这些镜像包含启动过程所需的一切; 主机不需要进行任何配置或安装任何依赖。

Jane 可以在 `registry.hub.docker.com/` 上找到现有的图像，也可以使用命令 docker search <name> 进行查找。例如，要查找 Redis 的映像，可以使用

docker search redis

- docker run <options> <image-name>
- docker run -d <image-name>
  - **-d** 表示后台运行，如果没有-d 选项则表示该容器运行在前台。
  - **<image-name>:tag-name**
- example1: docker run -d redis
- example2: docker run -d redis:latest

Finding Running Containers

启动的容器在后台运行，利用 docker ps 命令列出所有正在运行的容器，正在运行的容器的镜像和运行时间。

- The command docker inspect <friendly-name|container-id> provides more details about a running container, such as IP address.
- exmaple: docker inspect bdfd1263a1fe
- The command docker logs <friendly-name|container-id> will display messages the container has written to standard error or standard out.
- exmaple: docker logs bdfd1263a1fe

Accessing Redis

- docker run -d –name redisHostPort -p 6379:6379 redis:latest
- By default, the port on the host is mapped to 0.0.0.0, which means all IP addresses. You can specify a particular IP address when you define the port mapping, for example, -p 127.0.0.1:6379:6379

The problem with running processes on a fixed port is that you can only run one instance.

- docker run -d –name redisDynamic -p 6379 redis:latest
- which port has been assigned is discovered via docker port redisDynamic 6379

Persisting Data

Note: the data stored keeps being removed when she deletes and re-creates a container. so needs the data be persisted and reused when you recreates a container.

Containers are designed to be stateless. Binding directories (also known as volumes) is done using the option -v <host-dir>:<container-dir>. When a directory is mounted, the files which exist in that directory on the host can be accessed by the container and any data changed/written to the directory inside the container will be stored on the host. This allows you to upgrade or change containers without losing your data.

- docker run -d –name redisMapped -v /opt/docker/data/redis:/data redis
- Docker allows you to **$PWD** as a placeholder for the current directory.
  - docker run -d –name redisMapped -v "$PWD/data":/data redis

Running A Container In The Foreground

Question: Jane wonders how containers work with foreground processes, such as ps or bash.

Answer: If Jane wanted to interact with the container (for example, to access a bash shell) she could include the options -it.

- The command docker run ubuntu ps launches an Ubuntu container and executes the command ps to view all the processes running in a container.
- Using docker run -it ubuntu bash allows Jane to get access to a bash shell inside of a container.

Contents

Create Dockerfile

任务：您将学习如何使用 Nginx 来创建一个用于运行静态 HTML 网站的 Docker 镜像。该教程将介绍如何构建一个运行 Nginx 和静态网站的 Docker 镜像。

- This base image is defined as an instruction in the Dockerfile. Docker Images are built based on the contents of a Dockerfile.
- The Dockerfile is a list of instructions describing how to deploy your application.
- example
  - FROM nginx:alpine
    COPY . /usr/share/nginx/html

Build Docker Image

The build command takes in some different parameters. The format is docker build -t <build-directory>. The -t parameter allows you to specify a friendly name for the image and a tag, commonly used as a version number. This allows you to track built images and be confident about which version is being started.

- build image from Dockerfile
  - docker build -t webserver-image:v1 .
- view a list of all the images on the host.
  - docker images

Run

When starting a container you need to give it permission and access to what it requires.

For example, to open and bind to a network port on the host you need to provide the parameter -p <host-port>:<container-port>

- docker run -d -p 80:80 webserver-image:v1
- curl docker

Contents

Base Images

Docker images are built based on a Dockerfile. A Dockerfile defines all the steps required to create a Docker image with your application configured and ready to be run as a container. The image itself containers everything, from operating system to dependencies and configuration required to run your application.

All Docker images start from a base image. A base image is the same images from the Docker Registry which are used to start containers. Along with the image name, we can also include the image tag to indicate which particular version we want, by default, this is latest.

- FROM nginx:1.11-alpine
- FROM <image-name>:tag-name

Running Commands

The main commands two are COPY and RUN.

RUN <command> allows you to execute *any command* as you would at a command prompt, for example installing different application packages or running a build command. The results of the RUN are persisted to the image so it's important not to leave any unnecessary or temporary files on the disk as these will be included in the image.

COPY <src> <dest> allows you to copy files from the directory containing the Dockerfile to the container's image. This is extremely useful for source code and assets that you want to be deployed inside your container.

- COPY index.html /usr/share/nginx/html/index.html

Exposing Ports

Using the EXPOSE <port> command you tell Docker which ports should be open and can be bound too. You can define multiple ports on the single command, for example, EXPOSE 80 433 or EXPOSE 7000-8000

- We want our web server to be accessible via port 80, add the relevant EXPOSE line to the Dockerfile.
- Solution: EXPOSE 80

Default Commands

The *CMD* line in a Dockerfile defines the default command to run when a container is launched. If the command requires arguments then you need to use an array, for example ["cmd", "-a", "arga value", "-b", "argb-value"], which will be combined together and the command cmd -a arga value -b argb-value would be run.

- The command to run NGINX is nginx -g daemon off;. Set this as the default command in the Dockerfile.
- Solution: CMD ["nginx", "-g", "daemon off;"]

Building Containers

The build command takes in a directory containing the Dockerfile, executes the steps and stores the image in your local Docker Engine. If one fails because of an error then the build stops.

- docker build -t <image-name>:tag-name <directory of Dockerfile>
- docker build -t my-nginx-image:latest .

Launching New Image

- docker run -d -p 80:80 <image-id|friendly-tag-name>
- curl -i http://docker
- docker ps

Contents

Base Image

本场景继续探索如何构建 Docker 容器和自动化部署您的应用程序。这个案例将介绍如何在容器中部署一个 *Node.js* 应用程序。

The image for Node 7.0 is node:7-alpine. This is an Alpine-based build which is smaller and more streamlined than the official image.

In this case, an ideal directory would be /src/app as the environment user has read/write access to this directory.

Define a working directory using WORKDIR <directory> to ensure that all future commands are executed from the directory relative to our application.

- FROM node:7-alpine
- RUN mk

NPM Install

Install the dependencies required to run the application

- COPY package.json /src/app/package.json
  RUN npm install

Configuring Application

We can copy the entire directory where our Dockerfile is using COPY . <dest dir>.

Once the source code has been copied, the ports the application requires to be accessed is defined using EXPOSE <port>.

Finally, the application needs to be started. Once neat trick when using Node.js is to use the npm start command. This looks in the package.json file to know how to launch the application saving duplication of commands.

- COPY . /src/app
  EXPOSE 3000
  CMD ["npm", "start"]

Building Launching Container

To launch your application inside the container you first need to build an image.

- The command to build the image is
  - docker build -t my-nodejs-app .
- The command to launch the built image is
  - docker run -d –name my-running-app -p 3000:3000 my-nodejs-app

Environment Variables

Docker images should be designed that they can be transferred from one environment to the other without making any changes or requiring to be rebuilt. By following this pattern you can be confident that if it works in one environment, such as staging, then it will work in another, such as production.

With Docker, environment variables can be defined when you launch the container.

- -e <ENV_NAME>
- docker run -d –name my-production-running-app -e NODE_ENV=production -p 3000:3000 my-nodejs-app

Contents

OnBiuld

```
FROM node:7
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
ONBUILD COPY package.json /usr/src/app/
ONBUILD RUN npm install
ONBUILD COPY . /usr/src/app
CMD [ "npm", "start" ]
```

The result is that we can build this image but the application specific commands won't be executed until the built image is used as a base image. They'll then be executed as part of the base image's build.

Application Dockerfile

The create advantage of OnBuild images is that our Dockerfile is now much simpler and can be easily re-used across multiple projects without having to re-run the same steps improving build times.

- FROM node:7-onbuild EXPOSE 3000

Building Launching Container

- The command to build the image is
  - docker build -t my-nodejs-app .
- The command to launch the built image is
  - docker run -d –name my-running-app -p 3000:3000 my-nodejs-app