`docker` CLI & Dockerfile Cheat Sheet

RED HAT DEVELOPERS

Table of Contents

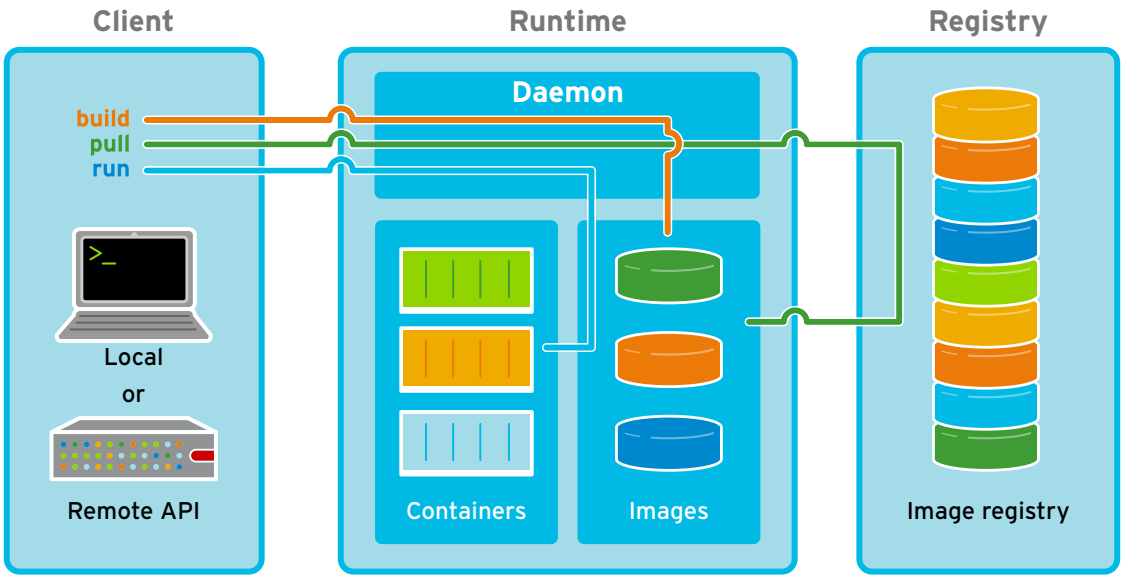
Introduction

Containers allow the packaging of your application (and everything that you need to run it) in a "container image". Inside a container you can include a base operating system, libraries, files and folders, environment variables, volume mount-points, and your application binaries.

A "container image" is a template for the execution of a container – It means that you can have multiple containers running from the same image, all sharing the same behavior, which promotes the scaling and distribution of the application. These images can be stored in a remote registry to ease the distribution.

Once a container is created, the execution is managed by the container runtime. You can interact with the container runtime through the "`docker`" command. The three primary components of a container architecture (client, runtime, & registry) are diagrammed below:

Container Architecture



RED HAT DEVELOPERS

1. docker CLI

1.1 Container Related Commands

`docker [CMD] [OPTS] [CONTAINER]`

Examples

All examples shown work in Red Hat Enterprise Linux

1. Run a container in interactive mode:

```
#Run a bash shell inside an image
$ docker run -it rhel7/rhel bash
#Check the release inside a container
[root@.../]# cat /etc/redhat-release
```

2. Run a container in detached mode:

```
$ docker run --name mywildfly -d -p 8080:8080 jboss/wildfly
```

3. Run a detached container in a previously created container network:

```
$ docker network create mynetwork
$ docker run --name mywildfly-net -d --net mynetwork \
      -p 8080:8080 jboss/wildfly
```

4. Run a detached container mounting a local folder inside the container:

```
$ docker run --name mywildfly-volume -d \
      -v myfolder/:/opt/jboss/wildfly/standalone/deployments/ \
      -p 8080:8080 jboss/wildflyjboss/wildfly
```

5. Follow the logs of a specific container:

```
$ docker logs -f mywildfly
$ docker logs -f [container-name|container-id]
```

6. List containers:

```
# List only active containers
$ docker ps
# List all containers
$ docker ps -a
```

7. Stop a container:

```
# Stop a container
$ docker stop [container-name|container-id]
# Stop a container (timeout = 1 second)
$ docker stop -t1
```

8. Remove a container:

```
# Remove a stopped container
$ docker rm [container-name|container-id]
# Force stop and remove a container
$ docker rm -f [container-name|container-id]
# Remove all containers
$ docker rm -f $(docker ps-aq)
# Remove all stopped containers
$ docker rm $(docker ps -q -f “status=exited”)
```

9. Execute a new process in an existing container:

```
# Execute and access bash inside a WildFly container
$ docker exec -it mywildfly bash
```

Command	Description
`daemon`	Run the persistent process that manages containers
`attach`	Attach to a running container to view its ongoing output or to control it interactively
`commit`	Create a new image from a container's changes
`cp`	Copy files/folders between a container and the local filesystem
`create`	Create a new container
`diff`	Inspect changes on a container's filesystem
`exec`	Run a command in a running container
`export`	Export the contents of a container's filesystem as a tar archive
`kill`	Kill a running container using SIGKILL or a specified signal
`logs`	Fetch the logs of a container
`pause`	Pause all processes within a container
`port`	List port mappings, or look up the public-facing port that is NAT-ed to the PRIVATE_PORT
`ps`	List containers
`rename`	Rename a container
`restart`	Restart a container
`rm`	Remove one or more containers
`run`	Run a command in a new container
`start`	Start one or more containers
`stats`	Display one or more containers' resource usage statistics
`stop`	Stop a container by sending SIGTERM then SIGKILL after a grace period
`top`	Display the running processes of a container
`unpause`	Unpause all processes within a container
`update`	Update configuration of one or more containers
`wait`	Block until a container stops, then print its exit code

1.2 Image Related Commands

`docker [CMD] [OPTS] [IMAGE]`

Examples

All examples shown work in Red Hat Enterprise Linux

1. Build an image using a Dockerfile:

```
#Build an image
$ docker build -t [username/]<image-name>[:tag] <dockerfile-path>
#Build an image called myimage using the Dockerfile in the same folder where the command was executed
$ docker build -t myimage:latest .
```

2. Check the history of an image:

```
# Check the history of the jboss/wildfly image
$ docker history jboss/wildfly
# Check the history of an image
$ docker history [username/]<image-name>[:tag]
```

3: List the images:

```
$ docker images
```

4: Remove an image from the local registry:

```
$ docker rmi [username/]<image-name>[:tag]
```

5. Tag an image:

```
# Creates an image called "myimage" with the tag "v1" for the image jboss/wildfly:latest
$ docker tag jboss/wildfly myimage:v1
# Creates a new image with the latest tag
$ docker tag <image-name> <new-image-name>
# Creates a new image specifying the "new tag" from an existing image and tag
$ docker tag <image-name>[:tag][username/] <new-image-name>.[:new-tag]
```

6. Exporting and importing an image to an external file:

```
# Export the image to an external file
$ docker save -o <filename>.tar
# Import an image from an external file
$ docker load -i <filename>.tar
```

7 Push an image to a registry:

```
$ docker push [registry/][username/]<image-name>[:tag]
```

Command	Description
`build`	Build images from a Dockerfile
`history`	Show the history of an image
`images`	List images
`import`	Create an empty filesystem image and import the contents of the tarball into it
`info`	Display system-wide information
`inspect`	Return low-level information on a container or image
`load`	Load an image from a tar archive or STDIN
`pull`	Pull an image or a repository from the registry
`push`	Push an image or a repository to the registry
`rmi`	Remove one or more images
`save`	Save one or more images to a tar archive (streamed to STDOUT by default)
`search`	Search one or more configured container registries for images
`tag`	Tag an image into a repository

1.3 Network related commands

`docker network [CMD] [OPTS]`

Command	Description
`connect`	Connects a container to a network
`create`	Creates a new network with the specified name
`disconnect`	Disconnects a container from a network
`inspect`	Displays detailed information on a network
`ls`	Lists all the networks created by the user
`rm`	Deletes one or more networks

1.4 Network related commands

`Default is https://index.docker.io/v1/`

Command	Description
`login`	Log in to a container registry server. If no server is specified then default is used
`logout`	Log out from a container registry server. If no server is specified then default is used

1.5 Volume related commands

`docker volume [CMD] [OPTS]`

Command	Description
`create`	Create a volume
`inspect`	Return low-level information on a volume
`ls`	Lists volumes
`rm`	Remove a volume

1.6 Related commands

Command	Description
`events`	Get real time events from the server
`inspect`	Show version information

2. Dockerfile

The Dockerfile provides the instructions to build a container image through the `` `docker build -t [username/]<image-name>[:tag] <dockerfile-path>` `` command. It starts from a previously existing Base image (through the FROM clause) followed by any other needed Dockerfile instructions.

This process is very similar to a compilation of a source code into a binary output, but in this case the output of the Dockerfile will be a container image.

Example Dockerfile

This example creates a custom WildFly container with a custom administrative user. It also exposes the administrative port 9990 and binds the administrative interface publicly through the parameter 'bmanagement'.

```
# Use the existing WildFly image
FROM jboss/wildfly

# Add an administrative user
RUN /opt/jboss/wildfly/bin/add-user.sh admin Admin#70365 --silent

#Expose the administrative port
EXPOSE 8080 9990

#Bind the WildFly management to all IP addresses
CMD ["/opt/jboss/wildfly/bin/standalong.sh", "-b", "0.0.0.0",
"-bmanagement", "0.0.0.0"]
```

Using the example Dockerfile

```
# Build the WildFly image
$ docker build -t mywildfly .

#Run a WildFly server
$ docker run -it -p 8080:8080 -p 9990:9990 mywildfly

#Access the WildFly administrative console and log in with the credentials admin/Admin#70635
open http://<docker-daemon-ip>:9990 in a browser
```

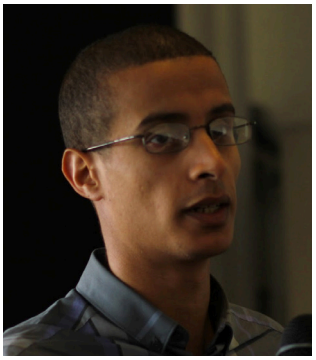
Dockerfile instruction arguments

Command	Description
FROM	Sets the base image for subsequent
MAINTAINER	Sets the author field of the generated images
RUN	Execute commands in a new layer on top of the current image and commit the results
CMD	Allowed only once (if many then last one takes effect)
LABEL	Adds metadata to an image
EXPOSE	Informs container runtime that the container listens on the specified network ports at runtime
ENV	Sets an environment variable
ADD	Copy new files, directories, or remote file URLs from into the filesystem of the container
COPY	Copy new files or directories into the filesystem of the container
ENTRYPOINT	Allows you to configure a container that will run as an executable
VOLUME	Creates a mount point and marks it as holding externally mounted volumes from native host or other containers
USER	Sets the username or UID to use when running the image
WORKDIR	Sets the working directory for any RUN, CMD, ENTRYPOINT, COPY, and ADD commands
ARG	Defines a variable that users can pass at build-time to the builder using --build-arg
ONBUILD	Adds an instruction to be executed later, when the image is used as the base for another build
STOPSIGNAL	Sets the system call signal that will be sent to the container to exit

Example: Running a web server container

```
$ mkdir -p www/                                  # Create a directory (if it doesn't already exist)

$ echo "Server is up" > www/index.html           # Make a text file to serve later

$ docker run -d \                                # Run process in a container as a daemon
  -p 8000:8000 \                                 # Map port 8000 in container to 8000 on host
  --name=pythonweb \                             # Name the container "pythonweb"
  -v `pwd`/www:/var/www/html \                   # Map container html to host www directory
  -w /var/www/html \                             # Set working directory to /var/www/html
  rhel7/rhel \                                   # Choose the rhel7/rhel directory
  /bin/python \                                  # Run the Python command for
  -m SimpleHTTPServer 8000                       #   a simple web server listening to port 8000

$ curl <container-daemon-ip>:8000                # Check that the server is working

$ docker ps                                      # See that the container is running
$ docker inspect pythonweb | less                # Inspect the container
$ docker exec -it pythonweb bash                 # Open the running container and look inside
```

About the authors



Bachir Chihani, Ph.D. holds an engineering degree from Ecole Superieure d'Informatique (Algeria) as well as a PhD degree in Computer Science from Telecom SudParis (France). Bachir has worked as a data engineer, software engineer, and research engineer for many years. Previously, he worked as a network engineer and got a CCNA Cisco-certification. Bachir has been programming for many years in Scala/Spark, Java EE, Android and Go. He has a keen interest in Open Source technologies particularly in the fields of Automation, Distributed Computing and Software/System Design and he likes sharing his experience through blogging.

Bachir authored many research papers in the field of Context-Awareness and reviewed many papers for International conferences. He also served as a technical reviewer for many books including Spring Boot in Action (Manning, 2016) and Unified Log Processing (Manning, 2016).

Rafael Benevides is a Director of Developer Experience at Red Hat. In his current role he helps developers worldwide to be more effective in software development, and he also promotes tools and practices that help them to be more productive. He worked in several fields including application architecture and design. Besides that, he is a member of Apache DeltaSpike PMC - a Duke's Choice Award winner project. And a speaker in conferences like JUDCon, TDC, JavaOne and Devoxx

Twitter: @rafabene

Linkdeln: https://www.linkedin.com/in/rafaelbenevides

www.rafabene.com.