Docker tutorial

Ivan Murashko

Contents

Introduction

There are several examples of docker usage. They are collected in one place mainly for future references.

The source code for examples can be found in the article git repository [1] in the folder **dockertutorial/src**.

1 Base commands

1.1 Simple program run

You can run a command (**uname -a**) with

```
$ docker run ubuntu uname -a
```

The container **ubuntu:latest** will be used in the case.

The command execution status can be viewed with

```
$ docker ps -a
CONTAINER ID   IMAGE   COMMAND     CREATED         ...
b5e8d82cce29   ubuntu  "uname -a"  5 seconds ago   ...
```

If you run the docker with –**rm** flag then the status info will not be stored.

1.2 Interactive session

You can run an interactive shell with

```
$ docker run -it ubuntu /bin/bash
```

where **-it** means –**interactive** –**tty**, **ubuntu** the latest ubuntu image and **/bin/bash** - the command to be start

1.3 Docker as a daemon

First of all run the docker in interactive mode and as daemon

```
$ docker run -itd ubuntu
```

possible output

```
649dae02de59ea3eb065a40b1248b2d322986e563ab12af3126fa4bb4710008a
```

Check the docker run in daemon mode with

```
$ docker ps
```

output:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND     ...
649dae02de59        ubuntu              "/bin/bash" ...
```

Execute **ls /var** command in the run docker

```
$ docker exec -it 649dae02de59 ls /var
backups  cache  lib  local  lock  log  mail  opt  run  spool  tmp
```

Stop it with

```
$ docker stop 649dae02de59
649dae02de59
```

Check the result

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND   ...
```

1.4 Network daemon

You can run a nginx web server with the following command

```
$ docker run -p 8080:80 -d nginx
```

This will map docker container port 80 to the host port 8080 or in other words make the nginx server available on the host machine via port 8080:

```
$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Server: nginx/1.17.4
Date: Sat, 28 Sep 2019 18:44:45 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 24 Sep 2019 14:49:10 GMT
Connection: close
ETag: "5d8a2ce6-264"
Accept-Ranges: bytes

Connection closed by foreign host.
```

1.5 Stop all

You can stop all containers with

```
$ docker container stop $(docker container ls -aq)
```

1.6 Cleanup

The following command will remove everything

```
$ docker system prune -a
WARNING! This will remove:
        - all stopped containers
        - all networks not used by at least one container
        - all images without at least one container associated to them
        - all build cache
Are you sure you want to continue? [y/N] y
Deleted Containers:
b5e8d82cce2942a24c709b630ff4e0dd705b89d78f2777065446ce97cf152cab
...
Total reclaimed space: 6.113GB
```

2 Creating docker images

2.1 Build

In the example we will create a docker image that will help us to compile and run code that uses thrift protocol [2]. The necessary libs for us are C++ and PHP.

The Dockerfile can be found in the article git repository [1] in the folder **dockertutorial/src/thrift**.

```
FROM ubuntu:18.04

# Don't interact during installation
# This is espesially important for tz packet installation (php deps)
ENV DEBIAN_FRONTEND=noninteractive

# Allow composer run from root user (default user in docker)
# The composer is required by php lib installation
ENV COMPOSER_ALLOW_SUPERUSER 1

# Setup thrift version be used
ENV THRIFT_VERSION v0.12.0
ENV THRIFT_SRC https://github.com/apache/thrift/archive/${THRIFT_VERSION}.tar.gz



# The script install and setup all deps + build thrift
RUN buildDeps=" \
                automake \
                bison \
                curl \
                flex \
                g++ \
                php7.2-dev \
                php7.2-xml \
                git \
                composer \
                libboost-dev \
                libboost-filesystem-dev \
                libboost-program-options-dev \
                libboost-system-dev \
                libboost-test-dev \
                libevent-dev \
                libssl-dev \
                libtool \
                make \
                pkg-config \
        "; \
        apt-get update && \
    apt-get install -y --no-install-recommends $buildDeps && \
    rm -rf /var/lib/apt/lists/* \
```

```
        && curl -k -sSL "${THRIFT_SRC}" -o thrift.tar.gz \
        && mkdir -p /usr/src/thrift \
        && tar zxf thrift.tar.gz -C /usr/src/thrift --strip-components=1 \
        && rm thrift.tar.gz \
        && cd /usr/src/thrift \
        && ./bootstrap.sh \
        && PHP_PREFIX=/usr/lib/php/Thrift ./configure \
        && make \
        && make install \
        && cd / \
        && rm -rf /usr/src/thrift \
        && rm -rf /var/cache/apt/* \
        && rm -rf /var/lib/apt/lists/* \
        && rm -rf /tmp/* \
        && rm -rf /var/tmp/*

# By default start up apache in the foreground,
# override with /bin/bash for interative.
CMD [ "thrift" ]
```

You can compile it with

```
$ cd src/thrift/
$ docker build -t thriftbuilder .
...
Successfully tagged thriftbuilder:latest
$
```

You can look at the image with

```
$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED              SIZE
thriftbuilder       latest              be1ccc48fb53        About a
minute ago
```

You can test the result with

```
$ docker run --rm thriftbuilder  thrift --version
Thrift version 0.12.0
```

The –**rm** option is used to be sure that the stopped container was removed.

2.2 Cleanup

To remove the image you can use

```
$ docker image rm thriftbuilder
```

If you got an error during the removal:

```
Error response from daemon: conflict: unable to remove repository
reference "thriftbuilder" (must force) - container dc0836d6f65f is
using its referenced image be1ccc48fb53
```

then try to remove the stopped container before

```
$ docker container rm dc0836d6f65f
```

TBD

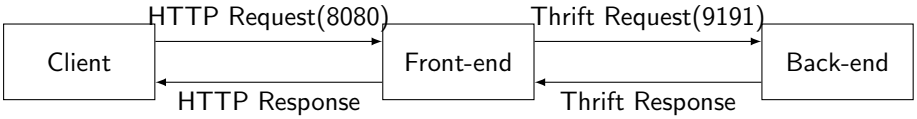

Figure 1: Applications

3 Apps in docker

I am going to create a small application that consists of 2 parts. The first one is front-end written in php that communicates with a back-end server written in C++. The communication is done via thrift protocol [2]. The front-end server as well as back-end server are run in different container and all communication is done via the network.

The client (see 1) is run on a host and communicates with front-end run in a docker container via HTTP protocol, port 8080 is used. The front-end communicates with a back-end application that is run on another docker container. The communication is done via 9191 port.

3.1 Build and run back-end (C++) application

We are going to create a separate docker image for C++ application. The image Dockerfile can be found in the article git repository [1] in the folder **dockertutorial/src/cpp**.

```
FROM thriftbuilder
RUN mkdir -p /usr/src/cpp
COPY ./cpp /usr/src/cpp/
COPY ./thrift/proto.thrift /usr/src/cpp/proto.thrift
RUN cd /usr/src/cpp \
   && make all
CMD /usr/src/cpp/server
```

As you can see it is based on the thriftbuilder image created before.

The daemon can be built with the following command run from **./src** folder:

```
$ docker build -t cppapp -f cpp/Dockerfile .
```

It also builds the application inside the docker. It can be run as follows

```
$ docker run --rm -p 9191:9191 -d cppapp
```

The application code can be found in **dockertutorial/src/cpp** folder:

```
#include "Time.h"
#include <thrift/protocol/TBinaryProtocol.h>
#include <thrift/server/TSimpleServer.h>
#include <thrift/transport/TBufferTransports.h>
#include <thrift/transport/TServerSocket.h>
#include <string>
#include <ctime>
```

```
using namespace ::apache::thrift;
using namespace ::apache::thrift::protocol;
using namespace ::apache::thrift::transport;
using namespace ::apache::thrift::server;

using namespace ::proto;

class TimeHandler : virtual public TimeIf {
public:
  TimeHandler() {
  }

  /**
   * Retrives the time
   */
  void getInfo(std::string &_return) {
    std::time_t now = std::time(NULL);
    _return = std::ctime(&now);
  }
};

int main(int argc, char **argv) {
  int port = 9191;
  ::apache::thrift::stdcxx::shared_ptr<TimeHandler>
    handler(new TimeHandler());
  ::apache::thrift::stdcxx::shared_ptr<TProcessor>
    processor(new TimeProcessor(handler));
  ::apache::thrift::stdcxx::shared_ptr<TServerTransport>
    serverTransport(new TServerSocket(port));
  ::apache::thrift::stdcxx::shared_ptr<TTransportFactory>
    transportFactory(new TBufferedTransportFactory());
  ::apache::thrift::stdcxx::shared_ptr<TProtocolFactory>
    protocolFactory(new TBinaryProtocolFactory());

  TSimpleServer server(processor, serverTransport,
                       transportFactory,
                       protocolFactory);
  server.serve();
  return 0;
}
```

3.2 Build and run front-end (php) application

We are going to create a separate image for front-end application. The image will be based on the thriftbuilder image created before. It will install and setup apache web server to work with php content. The setup instruction was taken from [3]. The image Dockerfile can be found in the article git repository [1] in the folder **dockertutorial/src/php**.

```
FROM thriftbuilder
```

```
ENV DEBIAN_FRONTEND=noninteractive

RUN deps=" \
                apache2 \
                libapache2-mod-php7.2 \
        "; \
    apt-get update && \
    apt-get install -y --no-install-recommends $deps && \
    rm -rf /var/lib/apt/lists/* \
        && cd / \
        && rm -rf /usr/src/thrift \
        && rm -rf /var/cache/apt/* \
        && rm -rf /var/lib/apt/lists/* \
        && rm -rf /tmp/* \
        && rm -rf /var/tmp/*

# Enable apache mods.
RUN a2enmod php7.2
RUN a2enmod rewrite

# Update the PHP.ini file, enable <? ?> tags and quieten logging.
RUN sed -i "s/short_open_tag = Off/short_open_tag = On/" \
/etc/php/7.2/apache2/php.ini
RUN sed -i "s/error_reporting = .*$/error_reporting = E_ERROR | E_WARNING | E_PARSE/" \
/etc/php/7.2/apache2/php.ini

# Manually set up the apache environment variables
ENV APACHE_RUN_USER www-data
ENV APACHE_RUN_GROUP www-data
ENV APACHE_LOG_DIR /var/log/apache2
ENV APACHE_LOCK_DIR /var/lock/apache2
ENV APACHE_PID_FILE /var/run/apache2.pid

# Expose apache.
EXPOSE 80

# Copy this repo into place.
COPY ./php /var/www/site
COPY ./thrift/proto.thrift /var/www/site

# Generate the required thrift files
RUN \
        cd /var/www/site && \
    thrift -r --gen php proto.thrift

# Update the default apache site with the config we created.
ADD ./apache2/apache-config.conf /etc/apache2/sites-enabled/000-default.conf

# By default start up apache in the foreground,
# override with /bin/bash for interative.
CMD /usr/sbin/apache2ctl -D FOREGROUND
```

The daemon can be built with the following command run from **./src** folder:

```
$ docker build -t phpapp -f php/Dockerfile .
```

It also builds the application inside the docker. It can be run as follows

```
$ docker run --rm -p 8080:80 -d phpapp
```

The application code can be found in **dockertutorial/src/php** folder:

```
<?php
error_reporting(E_ALL);

$GEN_DIR = __DIR__ . "/gen-php/";

require_once '/usr/lib/php/Thrift/ClassLoader/ThriftClassLoader.php';
use Thrift\ClassLoader\ThriftClassLoader;

$loader = new ThriftClassLoader();
$loader->registerNamespace('Thrift', '/usr/lib/php');
$loader->registerNamespace('proto', $GEN_DIR);
$loader->registerDefinition('proto', $GEN_DIR);
$loader->register();

use \Thrift\Protocol\TBinaryProtocol;
use \Thrift\Transport\TSocket;
use \Thrift\Transport\TBufferedTransport;
use \Thrift\Exception\TException;

try {
  $socket = new TSocket('172.17.0.1', 9191);
  $transport = new TBufferedTransport($socket, 1024, 1024);
  $transport->open();
  $protocol = new TBinaryProtocol($transport);
  $client = new \proto\TimeClient($protocol);
  $nowtime = $client->getInfo();
  print "Current time from back-end: $nowtime\n";
} catch (TException $tx) {
  print 'TException: '.$tx->getMessage()."\n";
}
?>
```

3.3 Test

First of all start the required containers if them have not been started before

```
$ docker run --rm -p 9191:9191 -d cppapp
b6ae2f7687b56d9d68028a38aa5f3357cdf096cfa0ead83768e17b4d270edf23
$ docker run --rm -p 8080:80 -d phpapp
d8d3ea9415ac614df623c30abf70c48030f03c81ae922e541085cfb082d4e4c4
```

The font-end can be tested via telnet as follow

```
$ telnet localhost 8080
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 03 Oct 2019 14:33:00 GMT
Server: Apache/2.4.29 (Ubuntu)
Content-Length: 54
Connection: close
Content-Type: text/html; charset=UTF-8

Current time from back-end: Thu Oct  3 14:33:00 2019

Connection closed by foreign host.
```

4 Appendix

4.1 MacOS setup

For Mac OS you might want to install `colima`, for instance from MacPorts

```
$ sudo port install colima
```

You have to start colima to be able to use docker on MacOS:

```
$ colima start
INFO[0000] starting colima
INFO[0000] runtime: docker
INFO[0001] creating and starting ...                     context=vm
INFO[0002] downloading disk image ...                    context=vm
INFO[0026] provisioning ...                              context=docker
INFO[0027] starting ...                                  context=docker
INFO[0027] done
```

After that you can run your docker commands

```
$ docker run ubuntu uname -a
Linux 45d5eb771176 6.8.0-50-generic #51-Ubuntu SMP
↪  PREEMPT_DYNAMIC Sat Nov  9 18:03:35 UTC 2024 aarch64 aarch64
↪  aarch64 GNU/Linux
```

Don't forget to stop colima as soon as you finish your work

```
$ colima stop
```

References

[1] Ivan Murashko. Articles git repository, 2019.

[2] Facebook. Apache thrift.

[3] Dan Pupius. Apache and php on docker.