CloudSuite Investigation Phase I Report

Zeyuan Hu, iamzeyuanhu@utexas.edu

EID:zh4378 Spring 2018

Abstract

In this writeup, we investigate the working behavior of the CloudSuite benchmark [1].

1 Introduction

We use a machine that has 4 Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz processors and 4GB of memory. The machine runs Ubuntu 16.04.3 (kernel version 4.4.0-116-generic). CloudSuite benchmark contains a collection of benchmarks for different real-life scenarios[1]) (e.g., Data Analytics, Media Streaming, etc). We focus on the benchmark for web serving[2].

The benchmark contains four components: `web_server`(represents the web server), `memcached_server`(represents the memcached server), `db_server`(represents the database server, which runs MySQL), and `faban_client`(represents the faban client). We run all four components on the same host.

2 Benchmark Setup

We follow the instruction on the web serving page[3]. We first start the database server by first downloading the corresponding dockerfile:

```
docker pull cloudsuite/web-serving:db_server
```

Then, we start the database server:

```
docker run --security-opt seccomp:unconfined -dt --net=host --name=
    mysql_server cloudsuite/web-serving:db_server localhost
```

Similarly, we download the memcached server and the web server dockerfiles, and start them up:

[1] `http://cloudsuite.ch//pages/download/`
[2] `http://cloudsuite.ch//pages/benchmarks/webserving/`
[3] `http://cloudsuite.ch//pages/benchmarks/webserving/`

```
docker pull cloudsuite/web-serving:memcached_server
docker pull cloudsuite/web-serving:web_server

docker run --security-opt seccomp:unconfined -dt --net=host --name=
    memcache_server cloudsuite/web-serving:memcached_server
docker run --cap-add SYS_PTRACE --security-opt seccomp:unconfined -dt --net
    =host --name=web_server cloudsuite/web-serving:web_server /etc/
    bootstrap.sh
```

`--cap-add SYS_PTRACE` allows us to invoke `strace` command[4] inside the docker. Finally, we download the faban client and run the benchmark

```
docker pull cloudsuite/web-serving:faban_client
docker run --security-opt seccomp:unconfined --net=host --name=faban_client
     cloudsuite/web-serving:faban_client localhost
```

3 Experiments

We focus on the `web_server` component in the experiment. The `web_server` consists of elgg framework[5] (a web development framework), Nginx[6](a web server), and PHP-FPM[7](a PHP implementation of a variant of Common Gateway Interface(CGI)). The goal of our experiment is to understand the communication behavior between PHP-FPM and Nginx by answer the following questions.

1. **When running the benchmark, Nginx will create worker processes. How many worker processes will Nginx create during the benchmark?**

   We login to the `web_server` docker by executing `sudo docker exec -it \web_server bash` and then we issue `ps -fax` and the output is shown in Figure 1. As one can see, the Nginx creates four worker processes.

2. **Does Nginx create worker process on demand or create all of them before the test is running?**

[4] `http://www.skrenta.com/rt/man/strace.1.html`
[5] `https://elgg.org/`
[6] `http://nginx.org/`
[7] `https://php-fpm.org/`

```
root@kettle:/usr/share/nginx/html# ps -fax
  PID TTY      STAT   TIME COMMAND
  368 pts/1    Ss     0:00 bash
  390 pts/1    R+     0:00  \_ ps -fax
    1 pts/0    Ss     0:00 /bin/bash /etc/bootstrap.sh
   37 ?        Ss     0:00 php-fpm: master process (/etc/php5/fpm/php-fpm.conf)
  388 ?        S      0:00  \_ php-fpm: pool www
  389 ?        S      0:00  \_ php-fpm: pool www
   65 ?        Ss     0:00 nginx: master process /usr/sbin/nginx
   66 ?        S      0:00  \_ nginx: worker process
   67 ?        S      0:00  \_ nginx: worker process
   69 ?        S      0:00  \_ nginx: worker process
   70 ?        S      0:00  \_ nginx: worker process
   72 pts/0    S+     0:00 bash
```

Figure 1: Inside `web_server` docker

Nginx creates all the worker processes before the test is running. We can run `strace -p 65` and `65` is the PID of the Nginx master process and we see the following

```
root@kettle:/usr/share/nginx/html# strace -p 65
Process 65 attached
rt_sigsuspend([]
```

Master process is being suspended and wait for a signal to invoke signal handler or terminate process. In other words, even the test is running, the master process already finishes its job (i.e., spawning the worker processes) and thus, being suspended.

3. Will PHP-FPM create worker processes?

Yes. If we use `strace -f -q -s 100 -o app.trc -ff -p 37` [8], which is to trace system calls and signals and we see that

```
root@kettle:/usr/share/nginx/html# strace -f -s 100 -o app.trc -ff -p
    37
Process 37 attached
Process 2007 attached
Process 2008 attached
Process 2009 attached
```

[8] `-s 100` means the max string length for each line in `app.trc` is 100. Use larger value if you don't want the line get truncated (e.g., `2000`).

```
Process 2010 attached
Process 2011 attached
Process 2012 attached
Process 2013 attached
Process 2014 attached
Process 2015 attached
Process 2016 attached
Process 2017 attached
^CProcess 37 detached
Process 2017 detached
Process 2016 detached
```

and if we open the trace file (`app.trc.37`) of PHP-FPM master process (with PID 37), we can see the following

```
epoll_wait(9, 125b530, 1, 973) = -1 EINTR (Interrupted system call)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_DUMPED, si_pid=2000,
    si_status=SIGBUS, si_utime=1, si_stime=2} ---
write(7, "C", 1) = 1
rt_sigreturn() = -1 EINTR (Interrupted system call)
epoll_wait(9, {{EPOLLIN, {u32=15543904, u64=15543904}}}, 1, 432) = 1
read(5, "C", 1) = 1
wait4(-1, [{WIFSIGNALED(s) && WTERMSIG(s) == SIGBUS && WCOREDUMP(s)}],
     WNOHANG|WSTOPPED, NULL) = 2000
write(4, "[21-Apr-2018 15:39:51] WARNING: [pool www] child 2000 exited
     on signal 7 (SIGBUS - core dumped) afte"..., 130) = 130
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|
    SIGCHLD, child_tidptr=0x7f75e88eca10) = 2007
write(4, "[21-Apr-2018 15:39:51] NOTICE: [pool www] child 2007 started
    \n", 61) = 61
wait4(-1, 0x7ffe87eab74c, WNOHANG|WSTOPPED, NULL) = 0
read(5, 0x7ffe87eab83f, 1) = -1 EAGAIN (Resource temporarily
    unavailable)
```

```
epoll_wait(9, {}, 1, 432) = 0
epoll_wait(9, 125b530, 1, 1000) = -1 EINTR (Interrupted system call)
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_DUMPED, si_pid=2001,
    si_status=SIGBUS, si_utime=0, si_stime=0} ---
write(7, "C", 1) = 1
rt_sigreturn() = -1 EINTR (Interrupted system call)
epoll_wait(9, {{EPOLLIN, {u32=15543904, u64=15543904}}}, 1, 274) = 1
...
```

`clone` indicates that PHP-FPM master process clones a child process and the master process then wait for the child process. `epoll_wait` waits for an I/O event on an epoll file descriptor. In this case, when child exits, the master process will read from file descriptor and record child status.

4. How does Nginx setup the communication channel with PHP-FPM?

We use the strace again `strace -f -s 100 -o nginx.trc -p 69` and we do `grep socket nginx.trc` and `grep connect nginx.trc` and we see the following

```
root@kettle:/usr/share/nginx/html# grep socket nginx.trc
69 socket(PF_LOCAL, SOCK_STREAM, 0) = 15
69 socket(PF_LOCAL, SOCK_STREAM, 0) = 15
69 socket(PF_LOCAL, SOCK_STREAM, 0) = 15
69 socket(PF_LOCAL, SOCK_STREAM, 0) = 15
69 socket(PF_LOCAL, SOCK_STREAM, 0) = 15
69 socket(PF_LOCAL, SOCK_STREAM, 0) = 15

root@kettle:/usr/share/nginx/html# grep connect nginx.trc
69 connect(15, {sa_family=AF_LOCAL, sun_path="/var/run/php5-fpm.sock
    "}, 110) = 0
69 connect(15, {sa_family=AF_LOCAL, sun_path="/var/run/php5-fpm.sock
    "}, 110) = 0
69 connect(15, {sa_family=AF_LOCAL, sun_path="/var/run/php5-fpm.sock
    "}, 110) = 0
```

```
69 connect(15, {sa_family=AF_LOCAL, sun_path="/var/run/php5-fpm.sock
   "}, 110) = 0
69 connect(15, {sa_family=AF_LOCAL, sun_path="/var/run/php5-fpm.sock
   "}, 110) = 0
69 connect(15, {sa_family=AF_LOCAL, sun_path="/var/run/php5-fpm.sock
   "}, 110) = 0
```

Thus, we can say that Nginx communicates with PHP-FPM using sockets.

5. **Does the channel setup when a new worker process is created? Or when the worker process wants to talk to PHP-FPM?**

   The channel sets up when the worker process want to talk to PHP-FPM. For each worker process, we issue `strace -f -s 100 -o nginx.trc -p <worker_pid>` and we can see some worker performs `epoll_wait` all the time while other workers are talking with PHP-FPM:

```
root@kettle:/usr/share/nginx/html# strace -f -s 100 -p 66
Process 66 attached
epoll_wait(13, {}, 512, 500) = 0
epoll_wait(13, {}, 512, 500) = 0
epoll_wait(13, {}, 512, 500) = 0
epoll_wait(13, {}, 512, 500) = 0
epoll_wait(13, {}, 512, 500) = 0

root@kettle:/usr/share/nginx/html# strace -f -s 100 -p 70
Process 70 attached
epoll_wait(19, {{EPOLLIN, {u32=2707665296, u64=140164670402960}}},
   512, -1) = 1
accept4(10, {sa_family=AF_INET, sin_port=htons(34074), sin_addr=
   inet_addr("127.0.0.1")}, [16], SOCK_NONBLOCK) = 12
epoll_ctl(19, EPOLL_CTL_ADD, 12, {EPOLLIN|EPOLLET, {u32=2707665872,
   u64=140164670403536}}) = 0
epoll_wait(19, {{EPOLLIN, {u32=2707665872, u64=140164670403536}}},
   512, 60000) = 1
```

```
recvfrom(12, "HEAD / HTTP/1.1\r\nHost: localhost:8080\r\nUser-Agent:
    curl/7.47.0\r\nAccept: */*\r\n\r\n", 1024, 0, NULL, NULL) = 79
stat("/usr/share/nginx/html/elgg/", {st_mode=S_IFDIR|0755, st_size
    =4096, ...}) = 0
stat("/usr/share/nginx/html/elgg/", {st_mode=S_IFDIR|0755, st_size
    =4096, ...}) = 0
stat("/usr/share/nginx/html/elgg/index.php", {st_mode=S_IFREG|0664,
    st_size=516, ...}) = 0
epoll_ctl(19, EPOLL_CTL_MOD, 12, {EPOLLIN|EPOLLOUT|EPOLLET, {u32
    =2707665872, u64=140164670403536}}) = 0
getsockname(12, {sa_family=AF_INET, sin_port=htons(8080), sin_addr=
    inet_addr("127.0.0.1")}, [16]) = 0
socket(PF_LOCAL, SOCK_STREAM, 0) = 14
ioctl(14, FIONBIO, [1]) = 0
...
```

Thus, the connection is established when the worker process wants to talk to PHP-FPM.

References

[1] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *SIGPLAN Not.*, vol. 47, pp. 37–48, Mar. 2012.