ABOUT BEN CANE

Benjamin Cane is a systems architect in the financial services industry. He writes about Linux systems administration on his blog and has recently published his first book, Red Hat Enterprise Linux Troubleshooting Guide.

Adjusting Linux Kernel Parameters with Docker Compose

Posted by: Ben Cane in Linux December 20th, 2017

Docker Compose is a great utility for anyone developing Dockerized applications. It's a tool that I personally use daily. Recently I came across yet another powerful feature of Docker Compose: the ability to change Linux Kernel Parameters.

In today's article, we will explore how to use this often overlooked but "useful when you need it" feature of Docker Compose.

To get started however, let's first create a Redis service using Docker Compose.

Starting with a Simple Redis Service

Docker Compose is a tool that allows users to create Dockerized services with a simple YAML file. To get a better understanding of how this works, let's go ahead and create an example

```
docker-compose.yml
```

.

```
1 version: '3'
2 services:
3   redis:
4     image: redis:latest
```

In the above example, we have a single "service" named

```
redis
```

. This service definition is how Docker Compose allows users to define different services. Docker Compose will take these services and run them within Docker containers.

In the example above, the

```
redis
```

service is provided by a container using the

```
redis:latest
```

image. To start this service, we can simply execute the

```
docker-compose
```

command followed by the


```
03 Creating rediscomposeexample_redis_1 ...
```

```
Creating rediscomposeexample_redis_1 ... done
Attaching to rediscomposeexample_redis_1
redis_1  | 1:C 12 Oct 03:57:02.915 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
redis_1  | 1:M 12 Oct 03:57:02.937 # WARNING: The TCP backlog setting of 511 cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower value of 128.
redis_1  | 1:M 12 Oct 03:57:02.937 # Server started, Redis version 3.2.6
redis_1  | 1:M 12 Oct 03:57:02.939 # WARNING you have Transparent Huge Pages (THP) support enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is disabled.
redis_1  | 1:M 12 Oct 03:57:02.940 * The server is now ready to accept connections on port 6379
```

In the above output, we can see that the

`docker-compose up`

command created a single container, named

`rediscomposeexample_redis_1`

.

As far as creating a single Redis container with Docker Compose, that is it. With only a few lines within a

`docker-compose.yml`

file and a single command, we have a running Redis container. However, this example doesn't show the power of Docker Compose.

To get a better idea of how Docker Compose is useful, let's add another container into the mix.

Multi-service Docker Compose

Since our first example used a Redis service, let's make our next container a bit more interesting. We will go ahead and add a Redis Commander service to our

`docker-compose.yml`

file.

Redis Commander is an application that allows users to explore a Redis instance through a browser. What this means is not only do we have to have an instance of Redis Commander, we also need to be able to connect it to our

`redis`

service.

```
version: '3'
services:
  redis:
    image: redis:latest
  redis-commander:
    image: tenstartups/redis-commander
    command: --redis-host redis
    ports:
      - 8081:8081
```

In the above example, we added another service named

`redis-commander`

. This service will launch a container using the

`tenstartups/redis-commander`

image.

At this point, the

`redis-commander`

service would not be able to connect to the

`redis`

service. For these services to communicate, we need to define the dependency within the

`docker-compose.yml`

file.

We can do this by adding the

`depends_on`

key shown below.

```
    ports:
      - 8081:8081
```

With our two services linked, let's go ahead and execute another

```
docker-compose up
```

.

```
$ docker-compose up
Creating network "rediscomposeexample_default" with the default driver
Creating rediscomposeexample_redis_1 ...
Creating rediscomposeexample_redis_1 ... done
Creating rediscomposeexample_redis-commander_1 ...
Creating rediscomposeexample_redis-commander_1 ... done
Attaching to rediscomposeexample_redis_1, rediscomposeexample_redis-commander_1
redis_1            | 1:C 16 Oct 20:56:51.613 # Warning: no config file specified, using the default
config. In order to specify a config file use redis-server /path/to/redis.conf
redis_1            | 1:M 16 Oct 20:56:51.618 # WARNING: The TCP backlog setting of 511 cannot be enforced
because /proc/sys/net/core/somaxconn is set to the lower value of 128.
redis_1            | 1:M 16 Oct 20:56:51.618 # Server started, Redis version 3.2.6
redis_1            | 1:M 16 Oct 20:56:51.619 # WARNING you have Transparent Huge Pages (THP) support
enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run
the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your
/etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is
disabled.
redis_1            | 1:M 16 Oct 20:56:51.619 * The server is now ready to accept connections on port 6379
redis-commander_1  | No Save: true
redis-commander_1  | listening on  0.0.0.0 : 8081
redis-commander_1  | Redis Connection redis:6379 Using Redis DB #0
```

In the above example, we can see there are two Docker containers running: a

```
redis
```

container as well as a

```
redis-commander
```

container.

This is a much better example of why Docker Compose is a powerful tool. In this example, we quickly specified multiple services within a single YAML file. We then turn those services into linked Docker containers with a single command.

We can also copy this

```
docker-compose.yml
```

file to any server and start up these same services. So not only is Docker Compose useful for local development, it can also be useful for deployment.

!Sign up for a free Codeship Account

Modifying Kernel Parameters with Docker Compose

Now that we have a better idea of what Docker Composes is and how it works, let's get to an uncommon feature: using Docker Compose to change Linux Kernel Parameters of our services.

If we look back at the output from our above example when we started the

```
redis
```

service, we can see a couple of warnings. One of those warnings is in regards to the

```
somaxconn
```

setting.

```
redis_1            | 1:M 16 Oct 20:56:51.618 # WARNING: The TCP backlog setting of 511 cannot be enforced
because /proc/sys/net/core/somaxconn is set to the lower value of 128.
```

The

```
somaxconn
```

parameter is a Linux Kernel Parameter that specifies the maximum backlogged TCP/IP sockets. This parameter is a setting in Linux that by default is set to

```
128
```

. This means that the kernel will only allow

```
128
```

connections to be "backlogged" at a time.

For highly accessed services like Redis, this may not be enough. The error above shows that Redis was attempting to use a value of

value is the default of

```
128
```

, Redis was not able to use a value of

```
511
```

.

In a non-containerized world, changing this kernel parameter would be as simple as placing the following into the

```
/etc/sysctl.conf
```

file.

```
1 net.core.somaxconn=1024
```

After adding the above, you would simply execute

```
sysctl -p
```

. However, in the container world this is a bit different.

We could perform the same tasks via a

```
Dockerfile
```

. However in our example, we are using public images from DockerHub. A simpler approach would be to specify this setting within the

```
docker-compose.yml
```

file itself.

To do this, we simply need to add the

```
sysctl
```

key as shown below.

```
01 version: '3'
02 services:
03   redis:
04     image: redis:latest
05     sysctls:
06       net.core.somaxconn: 1024
07   redis-commander:
08     image: tenstartups/redis-commander
09     command: --redis-host redis
10     depends_on:
11       - redis
12     ports:
13       - 8081:8081
```

In the above, we added two simple lines. These two lines will set the

```
net.core.somaxconn
```

parameter to

```
1024
```

. This is well above the

```
511
```

value Redis was trying to use.

Let's see what happens if we once again execute a

```
docker-compose up
```

command.

```
1 $ docker-compose up redis
2 Recreating rediscomposeexample_redis_1 ...
3 Recreating rediscomposeexample_redis_1 ... done
4 Attaching to rediscomposeexample_redis_1
5 redis_1            | 1:C 16 Oct 21:27:24.423 # Warning: no config file specified, using the default
  config. In order to specify a config file use redis-server /path/to/redis.conf
6 redis_1            | 1:M 16 Oct 21:27:24.426 # Server started, Redis version 3.2.6
7 redis_1            | 1:M 16 Oct 21:27:24.427 # WARNING you have Transparent Huge Pages (THP) support
  enabled in your kernel. This will create latency and memory usage issues with Redis. To fix this issue run
  the command 'echo never > /sys/kernel/mm/transparent_hugepage/enabled' as root, and add it to your
  /etc/rc.local in order to retain the setting after a reboot. Redis must be restarted after THP is
  disabled.
8 redis_1            | 1:M 16 Oct 21:27:24.427 * DB loaded from disk: 0.000 seconds
9 redis_1            | 1:M 16 Oct 21:27:24.427 * The server is now ready to accept connections on port 6379
```

to the command. This is a way of telling Docker Compose to only bring up the

```
redis
```

service.

If we look at the above, we can see that the warning around the

```
somaxconn
```

value is gone. This means with two simple lines, we were successful in changing the Redis container's

```
somaxconn
```

kernel parameter.

Summary

In today's articl, we went through a little refresher on Docker Compose. We explored how it is useful for launching multiple connected containers. Finally, we also learned an often overlooked feature, one we can use to easily change the Linux Kernel Parameters within our containers.

Published on System Code Geeks with permission by Ben Cane, partner at our SCG program. See the original article here: Adjusting Linux Kernel Parameters with Docker Compose

Opinions expressed by System Code Geeks contributors are their own.

Tagged with: DOCKER

Leave a Reply

1 Comment on "Adjusting Linux Kernel Parameters with Docker Compose"

Notify of new follow-up comments Email ›

Sort by: newest|oldest|most voted

Fred

Excellent, thanks. I was just about to containerized Redis!

Guest

0 REPLY

3 days 17 hours ago

KNOWLEDGE BASE

Courses

Minibooks

THE CODE GEEKS NETWORK

.NET Code Geeks

Java Code Geeks

System Code Geeks

Web Code Geeks

HALL OF FAME

Apache Configuration Tutorial (CentOS / Ubuntu Linux)

Apache enable SSL / TLS tutorial

Apache mod_rewrite example: Redirecting and rewriting URLs

Apache name-based Virtual Host Configuration Example (and setting bandwidth usage limits)

Apache URL rewrite example: Rewriting and redirecting URLs with mod_rewrite

How to install the Apache web server (CentOS / Ubuntu Linux Installation)

Linux sed Examples

Nginx Configuration Guide (Ubuntu 12.04 LTS)

Nginx SSL configuration guide (Ubuntu 12.04 LTS)

Nginx Websockets proxying guide (Ubuntu 12.04 LTS)

ABOUT SYSTEM CODE GEEKS

SCGs (System Code Geeks) is an independent online community focuse ultimate Operating System developers resource center; targeted at the architect, technical team lead (senior developer), project manager and alike. SCGs serve the OS developer, OS engineer and DevOps communi news written by domain experts, articles, tutorials, reviews, announcem snippets and open source projects.

DISCLAIMER

All trademarks and registered trademarks appearing on System Code G property of their respective owners.