Subject: CUDA support inside containers
Posted by abufrejoval on Sat, 12 Nov 2016 13:33:32 GMT
View Forum Message <> Reply to Message

I'm trying to set up a playground for machine learning engineers based on a beefy dual socket Xeon E5 with plenty of RAM and a Tesla GPU from Nvidia (pure compute GPU, no video-out).

CUDA seems to manage multi-tasking well enough, as long as limited resources such as GPU RAM are not exhausted, multiple applications run happily side-by-side.

The main issue I'm trying to solve is that most machine learning application stacks come with distinct userlands: There is Ubuntu, CentOS or plain Docker in all kinds of versions and I'd like them to co-exist happily without re-installation or exclusivity (PCI-passthrough to VM) and that's after all what container virtualization was designed to do, right?

While I've seen reports that with Docker CUDA workloads are possible, I'd always rather run Docker inside an OpenVZ container and I'd also rather give the guys the IaaS experience they are used to. They are also likely to do development work inside there and that's where Docker starts to become cumbersome.

Problem is that this new generic system resource, the GPU, today isn't quite treated like CPU, RAM or storage by OpenVZ: There is no built-in redirection layer for GPUs (BTW: How would that look with AMD APUs?).

The CUDA software evidently needs access to /dev/nvidia* to get things done and inside a container that currently seems a no-no.

Since this is a rather generic issue going forward: Any ideas how you'll want to implement that?

And is there a dirty hack which could be done to make this possible in the mean-time?

Security isn't an issue in this context: They are all friends in this case. But of course, security and strict resource allocation would be required for the production variant going forward.

Subject: Re: CUDA support inside containers
Posted by khorenko on Mon, 14 Nov 2016 18:04:35 GMT
View Forum Message <> Reply to Message

Hi,

we tried this several years ago and it seemed to work with no additional development from our side,
so you can just try and tell us if it works now or not.

The basic steps were:
1) install both on the Node and inside Containers
   a) nVidia drivers for CUDA support

b) CUDA compiler kit providing nvcc compiler
(there were some linker problems, but they were resolvable)

2) provide Containers access to nvidia0, nvidia1, nvidiactl devices

3) run a cuda-enabled program inside a Container

That's it.

Hope that helps.

Subject: Re: CUDA support inside containers
Posted by abufrejoval on Mon, 14 Nov 2016 22:28:20 GMT
View Forum Message <> Reply to Message

Yes, that certainly helps!

It was item 2 (device access) that I was particularly struggling with and I searched forth and back in the OpenVZ 7 PDF documents for any hint as to know this could be done. We never did device access during almost eight years of intense OpenVZ usage...

Turns out 'vztctl set --devnodes' didn't make it into the print version, nor did the rich set of examples or the detailled discussion as to what is going on behind the scenes when you do that sort of thing

I scanned some stuff I had found on how Docker handles this (https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker-p lugin) and I was afraid device sharing would require shared hard links.

Turns out it's so much easier on the mature container platform

Thanks a lot for your help!

Subject: Re: CUDA support inside containers
Posted by abufrejoval on Tue, 15 Nov 2016 00:38:11 GMT
View Forum Message <> Reply to Message

Sorry to say it's not quite solved yet...

I can now install CUDA, compile all the sample code etc, but once I try to start up any CUDA executable I'm running into run-time check issues with CUDA: First order of business is to check if the CUDA kernel module is loaded and it seems containers can't see /proc/modules.

Dunno if there is any way to tell CUDA to stop checking, for now I'll have to insert a 'sleep <some-hours>'

Subject: Re: CUDA support inside containers
Posted by khorenko on Tue, 15 Nov 2016 05:45:35 GMT
View Forum Message <> Reply to Message

A quick question: and is the module really loaded on the Hardware Node?

Please try to run same CUDA program on the host first to be sure all modules are really got loaded, and after that try it inside the Container.

Subject: Re: CUDA support inside containers
Posted by khorenko on Tue, 15 Nov 2016 07:58:09 GMT
View Forum Message <> Reply to Message

About vzctl "--devnodes" option. Yes, you are right, it's missing in docs, sorry.
i've filed a bug for that: https://bugs.openvz.org/browse/OVZ-6831

In the meanwhile please use vzctl man page option description - it is there:

```
   Device access management

   --devnodes device:r|w|rw|none
      Give  access  (r  - read, w - write, rw - read/write, none - no
      access) to special file /dev/device from CT.
```

Subject: Re: CUDA support inside containers
Posted by abufrejoval on Wed, 16 Nov 2016 02:37:23 GMT
View Forum Message <> Reply to Message

Yes, the module is loaded and working on the hardware node: I'm testing in parallel doing strace via the CUDA device info utililty outside and inside to see where things start going wrong.

One of the first things the Nvidia runtime is doing is to check for the presence of the nvidia* loadable modules scanning /proc/modules and that file is always empty inside containers as far as Internet searches tell me. It then goes and tries to load a matching nvidia0 module which neither exists nor would make sense inside containers, I guess.

I couldn't actually find any information on how the GPU code is represented to a user process and perhaps there are good reasons details are somewhat scarce as process isolation and security are somewhat lower on the priority list for CUDA. I also wonder how process scheduling is done for GPU side processes and how that is coordinated with CPU side processes. I don't wonder too much, though, because I can only imagine that being ugly.

It generally seems to work, as I've run quite a few CUDA code samples in parallel without any evident problems: Resource contention could become an issue, given that GPU VRAM tends to be far more limited and then lack virtual memory management. Could be useful to manage CUDA memory limits on containers eventually.

Somehow I doubt the GPU uses paged virtual memory for its own code but I see DMA access from the GPU to normal 'CPU' memory mention in documents and that reminds me of some of the earliest classical security exploits on mainframes, where virtual channel programs would happily deliver you the clear text password file, virtual memory and OS security successfully hid from you.

I guess GPU memory would eventually be MMAPed to CPU address space and at least provide process isolation for CPU access via PTEs. Hopefully the GPU code then doesn't start copying data across process boundaries of distinct CUDA users, neither for CPU side memory nor for GPU side memory. Could be interesting for people trying to attack usage of CUDA for Blockchain type cryptography.

Long term I'm just hoping AMD style HSA will become the norm and GPU code will run in normally managed virtual HBM2 memory.

Short term I guess I need to find some way to either disable the CUDA sanity check or fake a /proc/modules file which makes the CUDA runtime happy enough.

Subject: Re: CUDA support inside containers
Posted by khorenko on Wed, 16 Nov 2016 06:02:08 GMT
View Forum Message <> Reply to Message

May be "./NVIDIA-Linux-x86_64-XXX.run --no-kernel-module" option helps you?

https://www.qubes-os.org/doc/install-nvidia-driver/

Subject: Re: CUDA support inside containers
Posted by abufrejoval on Fri, 18 Nov 2016 03:37:08 GMT
View Forum Message <> Reply to Message

khorenko wrote on Wed, 16 November 2016 07:02May be "./NVIDIA-Linux-x86_64-XXX.run --no-kernel-module" option helps you?

https://www.qubes-os.org/doc/install-nvidia-driver/

First of all: Thanks for sticking with me! You guys give incredible support!

I'm guessing you are referring to this article: http://sqream.com/setting-cuda-linux-containers-2/ ..and the fact that there are pre-built docker containers actually available from Nvidia for CUDA.

I had been using the 'native' installers instead of the run file, so *.rpm for my CentoS7 container and *.deb for the Ubuntu one, which is why I didn't find that option right away.

If you use the run file, you can extract separate installers for the driver, the toolkit and the examples and there I can install along the lines of your recommendation and the article above.

But to make a long story short: LXC containers, even the so called unprivileged ones, get to see astonishing things from the host, including full dmesg info and--yes--/proc/modules, /proc/devices and I don't know what else...

That seems to make CUDA possible inside LXC (and Docker), while OpenVZ evidently hides much more, but either way, I have so far not been able to reproduce the results of the article above.

My guess is that the behaviour of the CUDA runtime has changed over time and that the 0.8 release may be behaving differently. My understanding is that it should really just use the device nodes to interact with the driver but from the system call traces I can see that it tries "to make things easy" for the user and will load the nvidia modules at application startup, if the user failed to do that himself (or automate that).

And all this logic which I can follow in the system call traces (attached) fails on OpenVZ, because the /proc file system misses information.

But it also fails on the plain CentOS7 machine with LXC containers using the same 0.8 software release, only it fails when it attempts to open the /dev/nvidia-uvm device node inside the container (outside works).

File permissions are alright but I guess LXC finally intervenes at this point.

I'll check with Docker on the CentOS control system next...

Having trouble attaching, will try in separate post!

Subject: Re: CUDA support inside containers
Posted by abufrejoval on Fri, 18 Nov 2016 04:10:43 GMT

View Forum Message <> Reply to Message

...any attempt to attach a file (5k zip or 15k text) will result in a "file attachment too big..." error: Should I actually paste into a response? It's very ugly (and a little big)

sorry, then here it goes (deviceQuery utility fails inside the container while attempts load the nvida kernel modules, since it couldn't find them in /proc/modules)

```
execve("./deviceQuery", ["./deviceQuery"], [/* 19 vars */]) = 0
brk(0)                                  = 0xcd1000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb1b000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/usr/local/cuda-8.0/lib64/tls/x86_64/librt.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT
(No such file or directory)
```

```
stat("/usr/local/cuda-8.0/lib64/tls/x86_64", 0x7ffe4d063450) = -1 ENOENT (No such file or
directory)
open("/usr/local/cuda-8.0/lib64/tls/librt.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such
file or directory)
stat("/usr/local/cuda-8.0/lib64/tls", 0x7ffe4d063450) = -1 ENOENT (No such file or directory)
open("/usr/local/cuda-8.0/lib64/x86_64/librt.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No
such file or directory)
stat("/usr/local/cuda-8.0/lib64/x86_64", 0x7ffe4d063450) = -1 ENOENT (No such file or directory)
open("/usr/local/cuda-8.0/lib64/librt.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file
or directory)
stat("/usr/local/cuda-8.0/lib64", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=93681, ...}) = 0
mmap(NULL, 93681, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f125fb04000
close(3)                                = 0
open("/lib64/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300\"\0\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=44096, ...}) = 0
mmap(NULL, 2128952, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125f6f3000
mprotect(0x7f125f6fa000, 2093056, PROT_NONE) = 0
mmap(0x7f125f8f9000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6000) = 0x7f125f8f9000
close(3)                                = 0
open("/usr/local/cuda-8.0/lib64/libpthread.so.0", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No
such file or directory)
open("/lib64/libpthread.so.0", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240l\0\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=142304, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb03000
mmap(NULL, 2208864, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125f4d7000
mprotect(0x7f125f4ed000, 2097152, PROT_NONE) = 0
mmap(0x7f125f6ed000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16000) = 0x7f125f6ed000
mmap(0x7f125f6ef000, 13408, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f125f6ef000
close(3)                                = 0
open("/usr/local/cuda-8.0/lib64/libdl.so.2", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file
or directory)
open("/lib64/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\320\16\0\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=19520, ...}) = 0
mmap(NULL, 2109744, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125f2d3000
mprotect(0x7f125f2d6000, 2093056, PROT_NONE) = 0
mmap(0x7f125f4d5000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x2000) = 0x7f125f4d5000
close(3)                                = 0
open("/usr/local/cuda-8.0/lib64/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No
such file or directory)
open("/lib64/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20\265\5\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=995840, ...}) = 0
mmap(NULL, 3175456, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125efcb000
mprotect(0x7f125f0b4000, 2097152, PROT_NONE) = 0
mmap(0x7f125f2b4000, 40960, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xe9000) = 0x7f125f2b4000
mmap(0x7f125f2be000, 82976, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f125f2be000
close(3)                                = 0
open("/usr/local/cuda-8.0/lib64/libm.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file
or directory)
open("/lib64/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260T\0\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1141560, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb02000
mmap(NULL, 3150168, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125ecc9000
mprotect(0x7f125edca000, 2093056, PROT_NONE) = 0
mmap(0x7f125efc9000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x100000) = 0x7f125efc9000
close(3)                                = 0
open("/usr/local/cuda-8.0/lib64/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such
file or directory)
open("/lib64/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\360*\0\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=88720, ...}) = 0
mmap(NULL, 2184192, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125eab3000
mprotect(0x7f125eac8000, 2093056, PROT_NONE) = 0
mmap(0x7f125ecc7000, 8192, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x14000) = 0x7f125ecc7000
close(3)                                = 0
open("/usr/local/cuda-8.0/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file
or directory)
open("/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \34\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2112384, ...}) = 0
mmap(NULL, 3936832, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125e6f1000
mprotect(0x7f125e8a8000, 2097152, PROT_NONE) = 0
mmap(0x7f125eaa8000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b7000) = 0x7f125eaa8000
mmap(0x7f125eaae000, 16960, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f125eaae000
close(3)                                = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb01000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb00000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fafe000
arch_prctl(ARCH_SET_FS, 0x7f125fafe740) = 0
mprotect(0x7f125eaa8000, 16384, PROT_READ) = 0
mprotect(0x7f125ecc7000, 4096, PROT_READ) = 0
mprotect(0x7f125efc9000, 4096, PROT_READ) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fafd000
mprotect(0x7f125f2b4000, 32768, PROT_READ) = 0
mprotect(0x7f125f4d5000, 4096, PROT_READ) = 0
mprotect(0x7f125f6ed000, 4096, PROT_READ) = 0
mprotect(0x7f125f8f9000, 4096, PROT_READ) = 0
mprotect(0x667000, 12288, PROT_READ)    = 0
mprotect(0x7f125fb1c000, 4096, PROT_READ) = 0
munmap(0x7f125fb04000, 93681)           = 0
set_tid_address(0x7f125fafea10)         = 16610
set_robust_list(0x7f125fafea20, 24)     = 0
rt_sigaction(SIGRTMIN, {0x7f125f4dd780, [], SA_RESTORER|SA_SIGINFO, 0x7f125f4e6100},
NULL, 8) = 0
rt_sigaction(SIGRT_1, {0x7f125f4dd810, [], SA_RESTORER|SA_RESTART|SA_SIGINFO,
0x7f125f4e6100}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
getrlimit(RLIMIT_STACK, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
brk(0)                                  = 0xcd1000
brk(0xcf2000)                           = 0xcf2000
brk(0)                                  = 0xcf2000
futex(0x66abb8, FUTEX_WAKE_PRIVATE, 2147483647) = 0
futex(0x7f125f2d096c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
futex(0x7f125f2d0978, FUTEX_WAKE_PRIVATE, 2147483647) = 0
futex(0x7f125f4d60d0, FUTEX_WAKE_PRIVATE, 2147483647) = 0
fstat(1, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb1a000
open("/usr/local/cuda-8.0/lib64/libcuda.so.1", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such
file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=93681, ...}) = 0
mmap(NULL, 93681, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f125fae6000
close(3)                                = 0
open("/lib64/libcuda.so.1", O_RDONLY|O_CLOEXEC) = 3
```

```
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300+\t\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=8222824, ...}) = 0
mmap(NULL, 10370056, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0)
= 0x7f125dd0d000
mprotect(0x7f125e3ca000, 2093056, PROT_NONE) = 0
mmap(0x7f125e5c9000, 1163264, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x6bc000) = 0x7f125e5c9000
mmap(0x7f125e6e5000, 48136, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f125e6e5000
close(3)                                = 0
open(" /usr/local/cuda-8.0/lib64/libnvidia-fatbinaryloader.so.367.4 8 ", O_RDONLY|O_CLOEXEC)
= -1 ENOENT (No such file or directory)
open("/lib64/libnvidia-fatbinaryloader.so.367.48", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200W\0\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=319912, ...}) = 0
mmap(NULL, 2416520, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f125dabf000
mprotect(0x7f125db03000, 2097152, PROT_NONE) = 0
mmap(0x7f125dd03000, 40960, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x44000) = 0x7f125dd03000
close(3)                                = 0
stat("/etc/sysconfig/64bit_strstr_via_64bit_strstr_sse2_unaligned ", 0x7ffe4d059170) = -1
ENOENT (No such file or directory)
stat("/etc/sysconfig/64bit_strstr_via_64bit_strstr_sse2_unaligned ", 0x7ffe4d059170) = -1
ENOENT (No such file or directory)
sched_get_priority_max(SCHED_RR)        = 99
sched_get_priority_min(SCHED_RR)        = 1
munmap(0x7f125fae6000, 93681)           = 0
clock_gettime(CLOCK_MONOTONIC_RAW, {22945, 334245287}) = 0
open("/proc/sys/vm/mmap_min_addr", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb19000
read(3, "4096\n", 1024)                 = 5
close(3)                                = 0
munmap(0x7f125fb19000, 4096)            = 0
open("/sys/devices/system/cpu/online", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such file or
directory)
open("/proc/stat", O_RDONLY|O_CLOEXEC)  = 3
read(3, "cpu  165242 0 39288 54855748 329"..., 8192) = 959
close(3)                                = 0
statfs("/dev/shm/", {f_type=0x1021994, f_bsize=4096, f_blocks=65536, f_bfree=65536,
f_bavail=65536, f_files=65536, f_ffree=65535, f_fsid={0, 0}, f_namelen=255, f_frsize=4096}) = 0
futex(0x7f125f8fa330, FUTEX_WAKE_PRIVATE, 2147483647) = 0
open("/dev/shm/cuda_injection_path_shm", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1
ENOENT (No such file or directory)
open("/root/.nv/nvidia-application-profile-globals-rc", O_RDONLY) = -1 ENOENT (No such file or
directory)
```

```
open("/root/.nv/nvidia-application-profiles-rc", O_RDONLY) = -1 ENOENT (No such file or
directory)
open("/root/.nv/nvidia-application-profiles-rc.d", O_RDONLY) = -1 ENOENT (No such file or
directory)
open("/etc/nvidia/nvidia-application-profiles-rc", O_RDONLY) = -1 ENOENT (No such file or
directory)
open("/etc/nvidia/nvidia-application-profiles-rc.d/", O_RDONLY) = -1 ENOENT (No such file or
directory)
open("/usr/share/nvidia/nvidia-application-profiles-367.48-rc ", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=5225, ...}) = 0
fstat(3, {st_mode=S_IFREG|0444, st_size=5225, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb19000
read(3, "# Application profiles for the N"..., 4096) = 4096
read(3, "tern\" : { \"feature\" : \"procname\""..., 4096) = 1129
close(3)                                = 0
munmap(0x7f125fb19000, 4096)            = 0
open("/usr/share/nvidia/nvidia-application-profiles-rc", O_RDONLY) = -1 ENOENT (No such file or
directory)
readlink("/proc/16610/exe", "/home/thomas/NVIDIA_CUDA-8.0_Sam"..., 4095) = 72
geteuid()                               = 0
socket(PF_LOCAL, SOCK_SEQPACKET|SOCK_CLOEXEC, 0) = 3
setsockopt(3, SOL_SOCKET, SO_PASSCRED, [1], 4) = 0
connect(3, {sa_family=AF_LOCAL, sun_path="/tmp/nvidia-mps/control"}, 26) = -1 ENOENT (No
such file or directory)
close(3)                                = 0
lstat("/proc", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
lstat("/proc/self", {st_mode=S_IFLNK|S_ISVTX|0777, st_size=0, ...}) = 0
readlink("/proc/self", "16610", 4095)   = 5
lstat("/proc/16610", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
lstat("/proc/16610/exe", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/proc/16610/exe", "/home/thomas/NVIDIA_CUDA-8.0_Sam"..., 4095) = 72
lstat("/home", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
lstat("/home/thomas", {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0
lstat("/home/thomas/NVIDIA_CUDA-8.0_Samples", {st_mode=S_IFDIR|0775, st_size=4096, ...}) =
0
lstat("/home/thomas/NVIDIA_CUDA-8.0_Samples/1_Utilities", {st_mode=S_IFDIR|0755,
st_size=4096, ...}) = 0
lstat(" /home/thomas/NVIDIA_CUDA-8.0_Samples/1_Utilities/deviceQuery ",
{st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
lstat(" /home/thomas/NVIDIA_CUDA-8.0_Samples/1_Utilities/deviceQuery /deviceQuery ",
{st_mode=S_IFREG|0775, st_size=578786, ...}) = 0
open("/proc/modules", O_RDONLY)         = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb19000
read(3, "", 1024)                       = 0
close(3)                                = 0
```

```
munmap(0x7f125fb19000, 4096)           = 0
stat("/sys/bus/pci/devices", 0x7ffe4d0582f0) = -1 ENOENT (No such file or directory)
geteuid()                              = 0
open("/proc/sys/kernel/modprobe", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f125fb19000
read(3, "/sbin/modprobe\n", 1024)       = 15
read(3, "", 1024)                       = 0
close(3)                               = 0
munmap(0x7f125fb19000, 4096)           = 0
stat("/sbin/modprobe", {st_mode=S_IFREG|0755, st_size=150664, ...}) = 0
clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD,
child_tidptr=0x7f125fafea10) = 16611
wait4(16611, [{WIFEXITED(s) && WEXITSTATUS(s) == 1}], 0, NULL) = 16611
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=16611, si_status=1,
si_utime=0, si_stime=0} ---
stat("/usr/bin/nvidia-modprobe", {st_mode=S_IFREG|S_ISUID|0755, st_size=29384, ...}) = 0
geteuid()                              = 0
ioctl(-1, FIGETBSZ, 0)                 = -1 EBADF (Bad file descriptor)
munmap(0x7f125dd0d000, 10370056)       = 0
munmap(0x7f125dabf000, 2416520)        = 0
futex(0x66b260, FUTEX_WAKE_PRIVATE, 2147483647) = 0
write(1, "./deviceQuery Starting...\n\n CUDA"..., 155) = 155
exit_group(1)                          = ?
+++ exited with 1 +++
```

Subject: Re: CUDA support inside containers
Posted by khorenko on Fri, 18 Nov 2016 18:07:02 GMT

View Forum Message <> Reply to Message

Ok, so what do we have at the moment (i did not try anything of this, so base on your words only and thus correct me if i miss something):
there are 2 ways of installation
1) via .run file which probably requires more steps but potentially allows to disable check for kernel module and thus allows to run CUDA program right now in OpenVZ Containers with no additional development

2) there is another way of CUDA software installation (btw, what's it? just rpm -ihv of some rpms? Which ones?)
It's easier, but it results in CUDA attempts to check for kernel module and fails dues to empty /proc/modules and inability to load module (the latter cannot be changed, otherwise anyone can load "bad" kernel module and crash the Hardware Node)

So i'm not against making things easier and don't see serious reasons why not to show /proc/modules content inside Container.

Do you have a great will to implement this and send the patch?

P.S. btw, try to workaround this check in order to understand how many of them are ahead: just save the output of /proc/modules on the host, copy the file inside the Container and mount the file onto /proc/modules file inside the Container.

```
# echo asdf > aaa
# cat aaa
asdf
# mount --bind aaa /proc/modules
# cat /proc/modules
asdf
```

Subject: Re: CUDA support inside containers
Posted by abufrejoval on Mon, 21 Nov 2016 01:11:11 GMT

View Forum Message <> Reply to Message

I've spent the week-end digging deeper

I think we can safely put aside the installation option disucssion: How you install CUDA inside the container doesn't really seem to matter.

The problem is with the behaviour of the CUDA run-time libraries and I have found no way to influence that anywhere in the documentation.
It's possible that this behaviour changed with the CUDA releases, but I couldn't downgrade on my primary system as that uses a GTX 1070, which requires CUDA 0.8.

The secondary has a GTX 980ti, so there I may be able to go back to CUDA 0.7, should that be necessary for verfication.

As I described I am using the secondary system for baseline comparison and it running a plain CentOS 7 instead.

(Turns out important, as neither LXC nor Docker will run on OpenVZ7, but that's another issue...)

I'm always running the very same samples outside the container on the host and inside the container to compare and using strace as the only means to follow what's going on.

I can see the CUDA runtime going through the /proc and /sys filesystem extensively (/proc/modules is just the first of many files inspected) and once it's happy to find all modules loaded it will open the CUDA devices ('/dev/nvidia-uvm' first), mmap them and follow up with some ioctl(): Obviously I can't see the memory channel interaction with the device.

Inside my Ubuntu based LXC container CUDA applications succeed with all their /procFS searches, are happy with the loaded modules (so they won't try to load them themselves) but then fail to open '/dev/nvidia-uvm' with 'operation not permitted', even if the device files inside the container have world RW permissions. All of that happens as part of the very first CUDA API call ('cudaGetDeviceCount()') and cannot be influenced by compile or startup options.

Looking closer at the LXC configuration I got the impression that UID mapping for non-privileged LXC containers is somewhat inbred into Ubuntu and that compatibility with CentOS is poor (there are UID/GID mapping extensions to 'adduser' RHEL doesn't know about). RHEL depreciated LXC almost immediately after RHEL 7 came out (bastards!) and it's showing (can't decide if they are just lazy or they try to hurt Ubuntu).

I then tried with Docker, not the RHEL supplied one, but the newest 1.12 from Docker, because that's what Nvidia worked with when they created their sample images.

Of course it took me a while to get the storage setup working with that one...

Those two sample Docker images from the Nvidia repository seemed to work: Unfortunately they are so minimal I couldn't even run an strace inside, so again I cannot really compare against outsie or LXC.

And then they stopped working today, evidently because the Docker image got updated to a slightly newer CUDA runtime than the host...

I'm now setting up an Ubuntu host, to try to get an LXC baseline working and because that seems to be somewhat more popular among the AI research crowd.

However that's not what I want to use going forward: I've been a huge OpenVZ fan for ten years now (bet the company on it, too) and not eager to relearn an Ubuntu userland either.

So I hope I can count on you guys making CUDA work going forward, because currently CUDA is the undisputed king in AI accelleration and that's why its support is a make or break issue.

Ah, yes I did bind mount a couple of files copies from the host /proc to inside the container in an attempt to make the run-time happy, but I had to stop when I couldn't find a way to create (or overlay) the /proc/driver directly: ProcFS won't allow me to create directories.

Somehow I haven't found a way yet to overlay a sparse directory tree where any file or directoy not present in the overlay will be used from the underlying FS. I thought that was supported already, but perhaps that's just wishful thinking (also a bit of a nightmare).

Subject: Re: CUDA support inside containers
Posted by abufrejoval on Mon, 21 Nov 2016 03:55:29 GMT

View Forum Message <> Reply to Message

As reported under the LXC thread https://forum.openvz.org/index.php?t=tree&goto=52646&amp;S=8435426c35e817f11e7afea78c459581#msg_52646 CUDA containerization works with LXC and Docker on Ubuntu.

I'd be happy to help you making it work with my favorite container OS

Subject: OverlayFS (was Re: CUDA support inside containers)
Posted by abufrejoval on Mon, 21 Nov 2016 12:18:14 GMT
View Forum Message <> Reply to Message

I guess OverlayFS is what I need to selectively replicate /proc and /sys contents from the host to make the CUDA runtime happy.

I remember reading about it on lwn.net and it looks like it's actually been backported from 3.18 to RHEL/CentOS/(OpenVZ?)7 3.10 kernels to support Docker, but it seems it's kernel only and missing support from the userland tools ("mount: unknown file system type 'overlayfs'...").

It's quite maddenning, because device access to /dev/nvidia* from inside the container in general seems to be supported in OpenVZ: I get a 'proper' "invalid argument" when doing a 'cat /dev/nvidia-uvm' and not the dreaded "permission denied" I get from LXC on the native CentOS.

Subject: Re: OverlayFS (was Re: CUDA support inside containers)
Posted by khorenko on Mon, 21 Nov 2016 16:11:14 GMT
View Forum Message <> Reply to Message

abufrejoval wrote on Mon, 21 November 2016 15:18I guess OverlayFS is what I need to selectively replicate /proc and /sys contents from the host to make the CUDA runtime happy.

I remember reading about it on lwn.net and it looks like it's actually been backported from 3.18 to RHEL/CentOS/(OpenVZ?)7 3.10 kernels to support Docker, but it seems it's kernel only and missing support from the userland tools ("mount: unknown file system type 'overlayfs'...").

It's quite maddenning, because device access to /dev/nvidia* from inside the container in general seems to be supported in OpenVZ: I get a 'proper' "invalid argument" when doing a 'cat /dev/nvidia-uvm' and not the dreaded "permission denied" I get from LXC on the native CentOS.

You can use overlayfs inside a Virtuozzo 7/OpenVZ 7 Container, all you need is to load the appropriate kernel module on the host.

Did you try to get further with mounting /proc/modules inside a Container with the file with content from the host's /proc/modules?

Subject: Re: OverlayFS (was Re: CUDA support inside containers)
Posted by abufrejoval on Tue, 22 Nov 2016 08:42:43 GMT
View Forum Message <> Reply to Message

khorenko wrote on Mon, 21 November 2016 17:11abufrejoval wrote on Mon, 21 November 2016 15:18I guess OverlayFS is what I need to selectively replicate /proc and /sys contents from the host to make the CUDA runtime happy.

I remember reading about it on lwn.net and it looks like it's actually been backported from 3.18 to RHEL/CentOS/(OpenVZ?)7 3.10 kernels to support Docker, but it seems it's kernel only and missing support from the userland tools ("mount: unknown file system type 'overlayfs'...").

It's quite maddenning, because device access to /dev/nvidia* from inside the container in general seems to be supported in OpenVZ: I get a 'proper' "invalid argument" when doing a 'cat /dev/nvidia-uvm' and not the dreaded "permission denied" I get from LXC on the native CentOS.

You can use overlayfs inside a Virtuozzo 7/OpenVZ 7 Container, all you need is to load the appropriate kernel module on the host.

Did you try to get further with mounting /proc/modules inside a Container with the file with content from the host's /proc/modules?

I write too much, that's why the answers got lost

Yes, I did try mounting /proc/modules via a copied file from the host which I then bind mounted as you suggested.

And then the runtime library just wanted the *next* file which wasn't there. I copied that, too but eventually I got stuck at /proc/devices, which is a *directory* on the host but an empty *file* on the guest: I couldn't bind mount a directory over the empty file (nor could I delete the empty file from the guest's procfs).

That's why I thought I might potentially get there using the overlayfs, which really seems to support all kinds of dirty tricks.

And there the trouble is, that the Virtuozzo 3.10. kernel supports the overlayfs functionality via its sys-call interface thanks to Docker running so much quicker with it. But the actual user land tool 'mount' doesn't understand the -t overlayfs parameter: I'd have to go and get one from e.g. a more up-to-date Fedora and statically compile that against a matching c-library etc.

In short words: All kinds of trouble when ideally Nvidia should offer a run-time library option, which doesn't do all these 'convenience' checks.

I've sent a request to Nvidia accordingly and I'm hoping for them to fix the issue at the source.

Of course somewhere within Virtuozzo there must be a table which decides which elements in /proc and /sys are visible to guests and which need translation (e.g. UID or PID mapping).

I should be able to patch that code and build a 'matching' CUDA kernel, just to see if that eventually solves the problem, too.

But I'd invest that effort only, if I could be sure that CUDA enabled Docker workloads also run on both the host and inside OpenVZ containers, because that would make OpenVZ feature complete with regards to the environment I need to build. That requires support for the current docker-engine 1.12.1 on both sides and evidently Docker and OpenVZ don't get along as well as I had hoped any more. Some tests using older Docker variants had looked rather promising early this year, but Nvidia has built a docker-plugin, which requires 1.10 or greater.

Essentially I want to support two major 'client' workloads: CUDA enabled Docker images and CUDA enabled 'IaaS' container.
Ubuntu delivery both, but with--well Ubuntu and LXC both of which require significant relearning and additional risks.

I really don't want to go down that road, but at the moment I have no choice.

Subject: Re: OverlayFS (was Re: CUDA support inside containers)
Posted by khorenko on Wed, 23 Nov 2016 15:57:35 GMT

View Forum Message <> Reply to Message

Hi,

sorry for the delay, just a quick note:
i've file a dev task for making CUDA management in VZ Containers easier:
https://bugs.openvz.org/browse/OVZ-6834

and the first issue fixed: we've allowed to show content of /proc/modules inside Containers.

Kernel 3.10.0-327.36.1.vz7.20.2 should appear tomorrow at
https://download.openvz.org/virtuozzo/factory/x86_64/os/Pack ages/v/

Subject: Re: OverlayFS (was Re: CUDA support inside containers)
Posted by abufrejoval on Wed, 23 Nov 2016 19:17:43 GMT

View Forum Message <> Reply to Message

khorenko wrote on Wed, 23 November 2016 16:57Hi,

sorry for the delay, just a quick note:
i've file a dev task for making CUDA management in VZ Containers easier:
https://bugs.openvz.org/browse/OVZ-6834

and the first issue fixed: we've allowed to show content of /proc/modules inside Containers.

Kernel 3.10.0-327.36.1.vz7.20.2 should appear tomorrow at
https://download.openvz.org/virtuozzo/factory/x86_64/os/Pack ages/v/

Ooops, didn't expect you to react so fast or I would have prepared better...

Unfortunately I haven't found a way to attach strace logs into my messages (any file is always too big), so you'll have to do with manual edits and this being terribly long...

Here are the /proc and /sys access I've been able to find from a *successful* execution of the basic deviceQuery utility on the host.
I have not check (yet) which of those would work inside the container because they are being reflected/translated.

It starts with a lot of dynamic loading stuff, which I'll leave out mostly

```
execve("./deviceQuery", ["./deviceQuery"], [/* 18 vars */]) = 0
brk(0)                                  = 0x1601000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdaba38000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=120462, ...}) = 0
mmap(NULL, 120462, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fcdaba1a000
close(3)                                = 0
open("/lib64/librt.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300\"\0\0\0\0\0\0 "..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=44096, ...}) = 0
```

... //lots more libraries

```
stat("/etc/sysconfig/64bit_strstr_via_64bit_strstr_sse2_unaligned ", 0x7fffbdd53980) = -1 ENOENT (No such file or directory)
stat("/etc/sysconfig/64bit_strstr_via_64bit_strstr_sse2_unaligned ", 0x7fffbdd53980) = -1 ENOENT (No such file or directory)
sched_get_priority_max(SCHED_RR)        = 99
sched_get_priority_min(SCHED_RR)        = 1
munmap(0x7fcdab9f6000, 120462)          = 0
clock_gettime(CLOCK_MONOTONIC_RAW, {19326, 808928358}) = 0
open("/proc/sys/vm/mmap_min_addr", O_RDONLY) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdaba36000
read(3, "4096\n", 1024)                 = 5
close(3)                                = 0
munmap(0x7fcdaba36000, 4096)            = 0
```

// I haven't checked this one, I just noticed that there was also /sys access but currently this seems to be the only one

```
open("/sys/devices/system/cpu/online", O_RDONLY|O_CLOEXEC) = 3

read(3, "0-23\n", 8192)                 = 5
close(3)                                = 0
statfs("/dev/shm/", {f_type=0x1021994, f_bsize=4096, f_blocks=16493769, f_bfree=16493590,
f_bavail=16493590, f_files=16493769, f_ffree=16493759, f_fsid={0, 0}, f_namelen=255, f_frsize=4096}) = 0
futex(0x7fcdab817330, FUTEX_WAKE_PRIVATE, 2147483647) = 0
open("/dev/shm/cuda_injection_path_shm", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)
```

... // some configuration file access stuff

// this stuff I believe is working

```
readlink("/proc/201468/exe", "/home/thomas/NVIDIA_CUDA-8.0_Sam"..., 4095) = 72
geteuid()                        = 0
socket(PF_LOCAL, SOCK_SEQPACKET|SOCK_CLOEXEC, 0) = 3
setsockopt(3, SOL_SOCKET, SO_PASSCRED, [1], 4) = 0
connect(3, {sa_family=AF_LOCAL, sun_path="/tmp/nvidia-mps/control"}, 26) = -1 ENOENT (No such file or directory)
close(3)                         = 0
lstat("/proc", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
lstat("/proc/self", {st_mode=S_IFLNK|S_ISVTX|0777, st_size=0, ...}) = 0
readlink("/proc/self", "201468", 4095)  = 6
lstat("/proc/201468", {st_mode=S_IFDIR|0555, st_size=0, ...}) = 0
lstat("/proc/201468/exe", {st_mode=S_IFLNK|0777, st_size=0, ...}) = 0
readlink("/proc/201468/exe", "/home/thomas/NVIDIA_CUDA-8.0_Sam"..., 4095) = 72
lstat("/home", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
lstat("/home/thomas", {st_mode=S_IFDIR|0700, st_size=4096, ...}) = 0
lstat("/home/thomas/NVIDIA_CUDA-8.0_Samples", {st_mode=S_IFDIR|0775, st_size=4096, ...}) = 0
lstat("/home/thomas/NVIDIA_CUDA-8.0_Samples/1_Utilities", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
lstat(" /home/thomas/NVIDIA_CUDA-8.0_Samples/1_Utilities/deviceQuery ", {st_mode=S_IFDIR|0755, st_size=4096, ...}) = 0
lstat(" /home/thomas/NVIDIA_CUDA-8.0_Samples/1_Utilities/deviceQuery /deviceQuery ", {st_mode=S_IFREG|0775, st_size=582882, ...}) = 0
```

//this would be where things start failing inside a container

```
open("/proc/modules", O_RDONLY)        = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdaba36000
read(3, "arc4 12608 0 - Live 0xffffffffa1"..., 1024) = 1024
close(3)                         = 0
munmap(0x7fcdaba36000, 4096)           = 0
```

// inside a container this is a zero size file not a directory

```
open("/proc/devices", O_RDONLY)        = 3
fstat(3, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
```

```
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fcdaba36000
read(3, "Character devices:\n  1 mem\n  2 p"..., 1024) = 652
close(3)                                = 0
munmap(0x7fcdaba36000, 4096)            = 0
```

// that should work, because I declared those devices as visible (can I can even open them)

```
stat("/dev/nvidia-uvm", {st_mode=S_IFCHR|0666, st_rdev=makedev(243, 0), ...}) = 0
stat("/dev/nvidia-uvm-tools", {st_mode=S_IFCHR|0666, st_rdev=makedev(243, 1), ...}) = 0
open("/dev/nvidia-uvm", O_RDWR|O_CLOEXEC) = 3

fcntl(3, F_GETFD)                       = 0x1 (flags FD_CLOEXEC)
ioctl(3, FIBMAP, 0x7fffbdd531c0)        = 0
clock_gettime(CLOCK_MONOTONIC_RAW, {19326, 812779374}) = 0
ioctl(3, 0x27, 0x7fffbdd531e0)          = 0
ioctl(3, 0x7ff, 0x7fffbdd531e0)         = 0
mmap(NULL, 135168, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0)
= 0x7fcdab9f3000
open("/proc/modules", O_RDONLY)         = 4
fstat(4, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fcdaba36000
read(4, "arc4 12608 0 - Live 0xffffffffa1"..., 1024) = 1024
read(4, "13a0000\ntun 27141 1 - Live 0xfff"..., 1024) = 1024
read(4, "persistent_data, Live 0xffffffff"..., 1024) = 1024
close(4)                                = 0
munmap(0x7fcdaba36000, 4096)            = 0
```

// and here the bind mount fails because I have a zero size file in /proc that blocks me from duplicating the directory tree below

```
open("/proc/driver/nvidia/params", O_RDONLY) = 4
fstat(4, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fcdaba36000
read(4, "Mobile: 4294967295\nResmanDebugLe"..., 1024) = 441
close(4)                                = 0
munmap(0x7fcdaba36000, 4096)            = 0
stat("/dev/nvidiactl", {st_mode=S_IFCHR|0666, st_rdev=makedev(195, 255), ...}) = 0
open("/dev/nvidiactl", O_RDWR)          = 4
fcntl(4, F_SETFD, FD_CLOEXEC)           = 0
ioctl(4, 0xc04846d2, 0x7fffbdd53d80)    = 0
ioctl(4, 0xc00446ca, 0x7fcdaa60c060)    = 0
ioctl(4, 0xca0046c8, 0x7fcdaa60b660)    = 0
ioctl(4, 0xc020462b, 0x7fffbdd53dd0)    = 0
ioctl(4, 0xc020462a, 0x7fffbdd53ba0)    = 0
ioctl(4, 0xc020462a, 0x7fffbdd53ba0)    = 0
```

```
... // lots of stuff going on

// here are some /proc access which I believe you support already including PID translation

getrlimit(RLIMIT_AS, {rlim_cur=RLIM64_INFINITY, rlim_max=RLIM64_INFINITY}) = 0
open("/proc/self/maps", O_RDONLY)       = 8
fstat(8, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdaba36000
read(8, "00400000-00469000 r-xp 00000000 "..., 1024) = 1024
close(8)                                = 0
munmap(0x7fcdaba36000, 4096)            = 0
mmap(0x200000000, 4297064448, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x200000000
open("/proc/self/maps", O_RDONLY)       = 8
fstat(8, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fcdaba36000
read(8, "00400000-00469000 r-xp 00000000 "..., 1024) = 1024
close(8)                                = 0
munmap(0x7fcdaba36000, 4096)            = 0
mmap(0x10000000000, 4294967296, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x10000000000
stat("/proc/201468/ns/pid", {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
stat("/proc/201468/ns/pid", {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
socket(PF_LOCAL, SOCK_SEQPACKET|SOCK_CLOEXEC, 0) = 8
unlink("")                              = -1 ENOENT (No such file or directory)
bind(8, {sa_family=AF_LOCAL, sun_path=@"cuda-uvmfd-4026531836-201468"}, 32) = 0
listen(8, 128)                          = 0
mmap(NULL, 8392704, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7fcda31d9000
mprotect(0x7fcda31d9000, 4096, PROT_NONE) = 0
clone(child_stack=0x7fcda39d8fb0, flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, parent_tidptr=0x7fcda39d99d0, tls=0x7fcda39d9700, child_tidptr=0x7fcda39d99d0) = 201470
futex(0x160b9c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x66c260, FUTEX_WAKE_PRIVATE, 2147483647) = 0
```

Subject: Re: OverlayFS (was Re: CUDA support inside containers)
Posted by abufrejoval on Mon, 28 Nov 2016 14:34:11 GMT

View Forum Message <> Reply to Message

Short version:
displaying /proc/modules content inside containers isn't enough to enable CUDA applications, '/proc/devices' and '/proc/driver/nvidia/parms' will also be required, '/sys/devices/cpu/online' is being checked but isn't fatal and may need adjustments for resource management.

Longer version on JIRA:
I don't know if you wanted me to take the discussion and follow-up to JIRA...

So after you didn't react to my last post with the extra logs I signed up for JIRA and posted inside and outside logs for comparison and additional fixes.

Thanks you for your efforts!