Case Study: Resolving frozen processes in kubernetes

See an up to date, HTML version of this case study on andrewhowden.com

Skills

- Understanding of Linux, syscalls, POSIX filsystem semantics, network filesystems
- Ability to research gaps in knowledge to debug systems further
- Ability to reason through implementations of unknown filesystems (At the time, NFS was largely unknown)
- Ability to pcap and verify the transit of NFS RPC messages.

Problem

In previous work it was determined that there were outstanding issues with the kernel implementation in NFSv4 and a change was made to shift from the NFSv4 protocol to NFSv3. At the same a few PHP workloads started becoming "stuck" during their application bootstrap.

Colleagues were able to refine the issue down to a single workload that would reliably break during the first requests to a newly created version of that workload.

Investigation

The processes were easily identifiable from the host `pid` namespace[1] as they were a series of php-fpm worker processes under a single fpm manager process, all of which were in D[2] state. Given this, it was easiest to use `nsenter --all --target ${PID}`[3].

From there, the goal is to determine exactly what the process was doing when it got stuck. To that end I executed `sudo strace -f -s 8096 -p ${PID}`[4] to listen to what syscalls that process was making to the kernel, hopefully including the last syscall with arguments that was executed. Unfortunately this did not work; rather, `strace` got stuck and could not be terminated with `SIGINT` (`ctrl + c`) nor `SIGKILL`. That reduced the problem down to something in "kernel land". The process did eventually quit, which likely meant that it was some process getting stuck in the kernel rather than something fundamentally broken.

Any issues in the kernel are generally well documented in `dmesg`[5]. Within dmesg there were a stack of messages that looked like:

```
[927063.519245] lockd: server 10.1.84.31 not responding, still trying
[927077.175116] lockd: server 10.1.84.31 OK
```

As well some:

```
[  +0.000062] nlmclnt_lock: VFS is out of sync with lock manager!
[Aug26 15:05] Process 14437(php-fpm) has RLIMIT_CORE set to 1
```

`lockd`[6] is documented in `man`[7], which indicated:

> The rpc.lockd program starts the NFS lock manager (NLM) on kernels that don't start it automatically. However, since most kernels do start it automatically, rpc.lockd. is usually not required. Even so, running it anyway is harmless.

At the time, it wasn't clear whether NFS could manage locks. Its a network filesystem, and they sometimes come with less than well known limitations such as the inability to `flock`. It wasn't clear whether this was likely to be an network problem, NFS problem or ZFS problem.

Shortly thereafter it was determined that the problem workload and the NFS server were indeed on different servers:

```
$ kubectl describe pod php-fpm-${REDACTED} -n ${REDACTED}
Name:           php-fpm-${REDACTED}
Namespace:      ${REDACTED}
Priority:       0
Node:           ${REDACTED}

$ kubectl get nodes -o wide | grep  ${IP}
${REDACTED}        Ready    worker   8d      v1.15.1   ${REDACTED}    <none>
Ubuntu 19.04   5.0.0-1011-aws   docker://18.9.5
```

The pattern was identified across several nodes (correlated by the logs in dmesg), as well as with several upstream NFS servers. That meant either that there was a systemic network problem, or that there was some problem with all NFS upstreams. Both were possible as there had been both network changes and changes to the upstream file storage structure.

Eventually the issue was narrowed down specifically to the following syscall pattern via `strace`:

```
open("/srv/path/to/file/that/would/be/written.js", O_RDONLY) = 11
fstat(11, {st_mode=S_IFREG|0666, st_size=1252708, ...}) = 0
lseek(11, 0, SEEK_CUR)                  = 0
flock(11, LOCK_EX
```

As well, only some `flock` calls were problematic:

```
$ cat strace.log | grep 'flock'
flock(11, LOCK_SH)                      = 0
flock(11, LOCK_UN)                      = 0
flock(11, LOCK_SH)                      = 0
flock(11, LOCK_UN)                      = 0
flock(11, LOCK_EX
```

That made the problem visible at "only exclusive flock calls" and allowed a minimal test case reproducible just by using the `flock`[8] command.

At this point, it was determined the problem was likely due to inability for NFS to acquire locks. However, it was not yet clear why this would be the case — NFS could clearly communicate for other file traffic.

Further reading about NFS showed that NFS handles the locks process with a separate binary and communication layer (the aforementioned `lockd` service) as well as another service to manage communication between nodes (`rpc.statd`). A short investigation later showed some servers with:

```
rpc-statd.service - NFS status monitor for NFSv2/3 locking.
   Loaded: loaded (/lib/systemd/system/rpc-statd.service; disabled; vendor preset:
enabled)
   Active: inactive (dead)
```

The service was enabled after verifying this would have no negative consequences for production.

Evaluation

At the time of writing, it is believed that `rpc.statd` functions both in a client and a server component. So, nodes that had workloads that were client workloads as well as having workloads that exposed an NFS server would work by chance when the node started `rpc.statd` with the NFS client.

However, nodes which had fewer workloads and no NFS clients would not start `rpc.statd` automatically and correspondingly could not fulfil the NFS lock contract.

The solution was to start this service everywhere as well as add it to the infrastructure as code definition.

References

1. http://man7.org/linux/man-pages/man7/pid_namespaces.7.html .
2. https://linux.die.net/man/1/ps .
3. http://man7.org/linux/man-pages/man1/nsenter.1.html .
4. http://man7.org/linux/man-pages/man1/strace.1.html .

5. http://man7.org/linux/man-pages/man1/dmesg.1.html .
6. https://linux.die.net/man/8/rpc.lockd .
7. https://linux.die.net/man/1/man .
8. https://linux.die.net/man/2/flock .