Arvados - Bug #8374

[KEEPSTORE] detection of new drives

02/04/2016 03:39 PM - Nico César

Status:	Closed	**Start date:**	02/04/2016
Priority:	Normal	**Due date:**	
Assigned To:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			

Description

Once the keepstore service is running we need to be able to "refresh" new additions of drives to serve. Currently the only option is to restart the service, killing all serving connections.

This could be done manually with a SIGUSR1 signal or an automatic poll every 10 seconds that re-checks for new directories. Or both.

Related issues:

Related to Arvados - Feature #16048: Automatically reload/restart services on...	**Resolved**	**04/14/2020**

History

#1 - 03/07/2016 10:22 PM - Tom Clegg

It might be feasible to implement a more general feature: "update configuration without service interruption". This would allow you to add new blob storage volumes, for example, not just auto-discovered local disk volumes.

- SO_REUSEPORT on linux >= 3.9 lets two programs (with the same EUID) listen on the same interface + port.
- on SIGTERM:
  1. detach from pgrp (this might require a fork during startup?)
  2. keep listening to our network socket long enough for our replacement process to start up (configurable interval like 10 seconds? detecting when the new process is ready seems sketchy)
  3. stop listening on our port, but keep working on any connections we've already accepted
  4. exit after all active requests have finished

One potential problem with this is that we'd have two processes running at once, both thinking they can use -max-buffers worth of RAM.

#2 - 03/08/2016 02:41 PM - Joshua Randall

@tomclegg, I think the procedure you suggest is generally good but the part about using a timeout feels like a hack and would almost certainly end up resulting in unintentional downtime (for example, if the new service does not successfully start up for any reason).

haproxy implements live reconfiguration for a network service without any loss of availability and without any hacks like waiting 10s or detecting when the new server is ready. The way they handle the handoff could easily be implemented for keepstore.

The requisite functionality would be:
- keepstore would handle a "finish" signal (different from "terminate" / SIGTERM) - haproxy uses SIGUSR1 which seems appropriate. Upon receiving SIGUSR1, the keepstore would being a graceful shutdown, unbinding the port for listening but continuing to process any open connections until they close. Once all connections are closed, exit.
- a new flag that you can pass to keepstore on startup to ask it to signal the old keepstore once it is ready (haproxy uses "-sf <pid>" which stands for "send finish"). Once the new keepstore knows that it is ready to handle new connections, it signals the existing running keepstore to gracefully shutdown.

Note that you don't actually **require** SO_REUSEPORT in order to do this handoff, if you can accept a very very small amount of downtime during the listening port handoff (according to section 4 of the haproxy manual: http://www.haproxy.org/download/1.7/doc/management.txt, they would expect 1 missed connection attempt per restart per 10000 connection attempts per second). Without SO_REUSEPORT, two additional signal handlers are implemented by haproxy: SIGTTOU (to temporarily unbind the listening ports so that another process can attempt to startup) and SIGTTIN (to rebind the ports and continue as if nothing had happened).

Even with SO_REUSEPORT, it is possible that packets will be very rarely missed (but most clients will retry) if the SYN comes through at exactly the right (wrong) time. However, it seems very unlikely that a keepstore would be subject to the number of connections per second that are required in order to start to see these problems.

I think this would be great functionality to have!

#3 - 03/08/2016 03:07 PM - Tom Clegg

Another approach without "process manipulation vs. threads" potholes:

1. Avoid killing requests already in progress

- after receiving SIGTERM and closing the TCP listener, wait for all current requests to finish before exiting

2. Avoid service interruption during restart

- add -reuseport command line flag -- see https://github.com/kavu/go_reuseport
- supervisor/sysadmin can start the new process (and wait for it to say "listening at ...") before sending TERM to the old process.

Even though (afaik) runit doesn't have a built-in feature for "start new before killing old", I think it's better to leave the process-juggling up to the daemon supervisor rather than build it into keepstore itself.

#4 - 03/08/2016 03:33 PM - Tom Clegg

Joshua Randall wrote:

> - keepstore would handle a "finish" signal (different from "terminate" / SIGTERM) - haproxy uses SIGUSR1 which seems appropriate. Upon receiving SIGUSR1, the keepstore would being a graceful shutdown, unbinding the port for listening but continuing to process any open connections until they close. Once all connections are closed, exit.

Hm. I was thinking SIGTERM should do this. Is there any reason why someone would want SIGTERM to do an *ungraceful* shutdown?

> - a new flag that you can pass to keepstore on startup to ask it to signal the old keepstore once it is ready (haproxy uses "-sf <pid>" which stands for "send finish"). Once the new keepstore knows that it is ready to handle new connections, it signals the existing running keepstore to gracefully shutdown.

This certainly sounds better than grepping a log file for the "listening" message, or "wait 10 seconds and hope for the best".

-send-term=PID?

> Without SO_REUSEPORT, two additional signal handlers are implemented by haproxy: SIGTTOU (to temporarily unbind the listening ports so that another process can attempt to startup) and SIGTTIN (to rebind the ports and continue as if nothing had happened).

I think it would be feasible to support SIGTTOU/SIGTTIN too, but it might be a low priority if SO_REUSEPORT works.

> Even with SO_REUSEPORT, it is possible that packets will be very rarely missed (but most clients will retry) if the SYN comes through at exactly the right (wrong) time. However, it seems very unlikely that a keepstore would be subject to the number of connections per second that are required in order to start to see these problems.

Interesting. If Linux doesn't correctly handle a race between "incoming connection" and "unbind", I don't suppose there is anything we can do about it (at least at this level). And solving 99.99% of the problem seems OK.

#5 - 03/09/2016 02:43 PM - Peter Amstutz

Another means to achieve a zero-gap process restart is to do a file descriptor handoff of the listen socket, so the new process accepts connections on the existing file descriptor instead of having to re-open the port with SO_REUSEADDR. There's a couple ways to do this:

- fork()/exec() the new process with the FD remaining open in the child process.
- Set up a unix domain socket between the old server and new server and use sendmsg(2) to send the file descriptor. See "File descriptor passing" on this page:

http://www.thomasstover.com/uds.html

#6 - 01/18/2020 12:49 AM - Peter Amstutz

- Related to Feature #16048: Automatically reload/restart services on configuration change added

#7 - 01/18/2020 12:50 AM - Peter Amstutz

- Status changed from New to Closed