Item 137
git_comments:

1. There may be a process reading from stdout/stderr, and if it exits, we will crash on a SIGPIPE when we try to write to them. By ignoring SIGPIPE, we can handle the EPIPE instead of crashing.

git_commits:

1. **summary:** YARN-8515. container-executor can crash with SIGPIPE after nodemanager restart. Contributed by Jim Brennan
   message: YARN-8515. container-executor can crash with SIGPIPE after nodemanager restart. Contributed by Jim Brennan (cherry picked from commit 17118f446c2387aa796849da8b69a845d9d307d3)

github_issues:

github_issues_comments:

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

1. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.
2. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.
   label: code-design
3. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.
4. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start

container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.

5. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.
6. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.
7. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.
8. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.
9. **summary:** container-executor can crash with SIGPIPE after nodemanager restart
   description: When running with docker on large clusters, we have noticed that sometimes docker containers are not removed - they remain in the exited state, and the corresponding container-executor is no longer running. Upon investigation, we noticed that this always seemed to happen after a nodemanager restart. The sequence leading to the stranded docker containers is: # Nodemanager restarts # Containers are recovered and then run for a while # Containers are killed for some (legitimate) reason # Container-executor exits without removing the docker container. After reproducing this on a test cluster, we found that the container-executor was exiting due to a SIGPIPE. What is happening is that the shell command executor that is used to start container-executor has threads reading from c-e's stdout and stderr. When the NM is restarted, these threads are killed. Then when the container-executor continues executing after the container exits with error, it tries to write to stderr (ERRORFILE) and gets a SIGPIPE. Since SIGPIPE is not handled, this crashes the container-executor before it can actually remove the docker container. We ran into this in branch 2.8. The way docker containers are removed has been completely redesigned in trunk, so I

don't think it will lead to this exact failure, but after an NM restart, potentially any write to stderr or stdout in the container-executor could cause it to crash.

jira_issues_comments:

1. Here is an example case that we saw: Docker ps info for this container: {noformat} 968e4a1a0fca 90188f3d752e "bash /grid/4/tmp/..." 6 days ago Exited (143) 6 days ago container_e07_1528760012992_2875921_01_000069 {noformat} NM Log with some added info from Docker container and journalctl to show where the docker container started/exited: {noformat} 2018-06-27 16:32:48,779 [IPC Server handler 9 on 8041] INFO containermanager.ContainerManagerImpl: Start request for container_e07_1528760012992_2875921_01_000069 by user p_condor 2018-06-27 16:32:48,782 [AsyncDispatcher event handler] INFO application.ApplicationImpl: Adding container_e07_1528760012992_2875921_01_000069 to application application_1528760012992_2875921 2018-06-27 16:32:48,783 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from NEW to LOCALIZING 2018-06-27 16:32:48,783 [AsyncDispatcher event handler] INFO yarn.YarnShuffleService: Initializing container container_e07_1528760012992_2875921_01_000069 2018-06-27 16:32:48,786 [AsyncDispatcher event handler] INFO localizer.ResourceLocalizationService: Created localizer for container_e07_1528760012992_2875921_01_000069 2018-06-27 16:32:48,786 [LocalizerRunner for container_e07_1528760012992_2875921_01_000069] INFO localizer.ResourceLocalizationService: Writing credentials to the nmPrivate file /grid/4/tmp/yarn-local/nmPrivate/container_e07_1528760012992_2875921_01_000069.tokens. Credentials list: 2018-06-27 16:32:52,654 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from LOCALIZING to LOCALIZED 2018-06-27 16:32:52,684 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from LOCALIZED to RUNNING 2018-06-27 16:32:52,684 [AsyncDispatcher event handler] INFO monitor.ContainersMonitorImpl: Starting resource-monitoring for container_e07_1528760012992_2875921_01_000069 2018-06-27 16:32:53.345 Docker container started 2018-06-27 16:32:54,429 [Container Monitor] DEBUG ContainersMonitorImpl.audit: Memory usage of ProcessTree 103072 for container-id container_e07_1528760012992_2875921_01_000069: 132.5 MB of 3 GB physical memory used; 4.3 GB of 6.3 GB virtual memory used 2018-06-27 16:33:25,422 [main] INFO nodemanager.NodeManager: STARTUP_MSG: /** STARTUP_MSG: Starting NodeManager STARTUP_MSG: user = mapred STARTUP_MSG: host = gsbl607n22.blue.ygrid.yahoo.com/10.213.59.232 STARTUP_MSG: args = [] STARTUP_MSG: version = 2.8.3.2.1806111934 2018-06-27 16:33:31,140 [main] INFO containermanager.ContainerManagerImpl: Recovering container_e07_1528760012992_2875921_01_000069 in state LAUNCHED with exit code -1000 2018-06-27 16:33:31,140 [main] INFO application.ApplicationImpl: Adding container_e07_1528760012992_2875921_01_000069 to application application_1528760012992_2875921 2018-06-27 16:33:32,771 [main] INFO containermanager.ContainerManagerImpl: Waiting for containers: 2018-06-27 16:33:33,280 [main] INFO containermanager.ContainerManagerImpl: Waiting for containers: 2018-06-27 16:33:33,178 [main] INFO containermanager.ContainerManagerImpl: Waiting for containers: 2018-06-27 16:33:33,776 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from NEW to LOCALIZING 2018-06-27 16:33:34,393 [AsyncDispatcher event handler] INFO yarn.YarnShuffleService: Initializing container container_e07_1528760012992_2875921_01_000069 2018-06-27 16:33:34,433 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from LOCALIZING to LOCALIZED 2018-06-27 16:33:34,461 [ContainersLauncher #23] INFO nodemanager.ContainerExecutor: Reacquiring container_e07_1528760012992_2875921_01_000069 with pid 103072 2018-06-27 16:33:34,463 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from LOCALIZED to RUNNING 2018-06-27 16:33:34,482 [AsyncDispatcher event handler] INFO monitor.ContainersMonitorImpl: Starting resource-monitoring for container_e07_1528760012992_2875921_01_000069 2018-06-27 16:33:35,304 [main] INFO nodemanager.NodeStatusUpdaterImpl: Sending out 598 NM container statuses: 2018-06-27 16:33:35,356 [main] INFO nodemanager.NodeStatusUpdaterImpl: Registering with RM using containers 2018-06-27 16:33:35,902 [Container Monitor] DEBUG ContainersMonitorImpl.audit: Memory usage of ProcessTree 103072 for container-id container_e07_1528760012992_2875921_01_000069: 1.1 GB of 3 GB physical memory used; 4.5 GB of 6.3 GB virtual memory used 2018-06-27 16:34:22,480 [Container Monitor] DEBUG ContainersMonitorImpl.audit: Memory usage of ProcessTree 103072 for container-id container_e07_1528760012992_2875921_01_000069: 1.1 GB of 3 GB physical memory used; 4.5 GB of 6.3 GB virtual memory us 2018-06-27 16:34:25,738 [IPC Server handler 6 on 8041] INFO containermanager.ContainerManagerImpl: Stopping container with container Id: container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:25,739 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from RUNNING to KILLING 2018-06-27 16:34:25,739 [AsyncDispatcher event handler] INFO launcher.ContainerLaunch: Cleaning up container container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26.276 Docker container exited Jun 27 16:34:26 gsbl607n22.blue.ygrid.yahoo.com dockerd-current[17973]: time="2018-06-27T16:34:26.275701908Z" level=error msg="containerd: deleting container" error="exit status 1: \"container 968e4a1a0fcafa7e01a6ab688a86a86615369b644061e90939f60542 Jun 27 16:34:28 gsbl607n22.blue.ygrid.yahoo.com dockerd-current[17973]: time="2018-06-27T16:34:28.470794342Z" level=warning msg="968e4a1a0fcafa7e01a6ab688a86a86615369b644061e90939f60542468dcb1b cleanup: failed to unmount secrets: invalid argument" 2018-06-27 16:34:26,516 [Container Monitor] DEBUG ContainersMonitorImpl.audit: Memory usage of ProcessTree 103072 for container-id container_e07_1528760012992_2875921_01_000069: -1B of 3 GB physical memory used; -1B of 6.3 GB virtual memory used 2018-06-27 16:34:26,769 [ContainersLauncher #23] INFO nodemanager.ContainerExecutor: container_e07_1528760012992_2875921_01_000069 was deactivated 2018-06-27 16:34:26,870 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from KILLING to EXITED_WITH_FAILURE 2018-06-27

16:34:26,872 [DeletionService #1] INFO nodemanager.LinuxContainerExecutor: Deleting absolute path : /grid/3/tmp/yarn-local/usercache/p_condor/appcache/application_1528760012992_2875921/container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26,872 [DeletionService #3] INFO nodemanager.LinuxContainerExecutor: Deleting absolute path : /grid/5/tmp/yarn-local/usercache/p_condor/appcache/application_1528760012992_2875921/container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26,872 [DeletionService #0] INFO nodemanager.LinuxContainerExecutor: Deleting absolute path : /grid/2/tmp/yarn-local/usercache/p_condor/appcache/application_1528760012992_2875921/container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26,873 [AsyncDispatcher event handler] INFO container.ContainerImpl: Container container_e07_1528760012992_2875921_01_000069 transitioned from EXITED_WITH_FAILURE to DONE 2018-06-27 16:34:26,873 [AsyncDispatcher event handler] INFO application.ApplicationImpl: Removing container_e07_1528760012992_2875921_01_000069 from application application_1528760012992_2875921 2018-06-27 16:34:26,873 [AsyncDispatcher event handler] INFO monitor.ContainersMonitorImpl: Stopping resource-monitoring for container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26,873 [AsyncDispatcher event handler] INFO logaggregation.AppLogAggregatorImpl: Considering container container_e07_1528760012992_2875921_01_000069 for log-aggregation 2018-06-27 16:34:26,873 [AsyncDispatcher event handler] INFO containermanager.AuxServices: Got event CONTAINER_STOP for appId application_1528760012992_2875921 2018-06-27 16:34:26,873 [AsyncDispatcher event handler] INFO yarn.YarnShuffleService: Stopping container container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26,875 [DeletionService #0] INFO nodemanager.LinuxContainerExecutor: Deleting absolute path : /grid/4/tmp/yarn-local/usercache/p_condor/appcache/application_1528760012992_2875921/container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26,875 [DeletionService #1] INFO nodemanager.LinuxContainerExecutor: Deleting absolute path : /grid/1/tmp/yarn-local/usercache/p_condor/appcache/application_1528760012992_2875921/container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:26,876 [AsyncDispatcher event handler] WARN containermanager.ContainerManagerImpl: couldn't find container container_e07_1528760012992_2875921_01_000069 while processing FINISH_CONTAINERS event 2018-06-27 16:34:26,877 [DeletionService #1] INFO nodemanager.LinuxContainerExecutor: Deleting absolute path : /grid/0/tmp/yarn-local/usercache/p_condor/appcache/application_1528760012992_2875921/container_e07_1528760012992_2875921_01_000069 2018-06-27 16:34:30,557 [Node Status Updater] INFO nodemanager.NodeStatusUpdaterImpl: Removed completed containers from NM context: [container_e07_1528760012992_2875921_01_000069] {noformat} Docker state info: {noformat} "State": { "Status": "exited", "Running": false, "Paused": false, "Restarting": false, "OOMKilled": false, "Dead": false, "Pid": 0, "ExitCode": 143, "Error": "", "StartedAt": "2018-06-27T16:32:53.345303052Z", "FinishedAt": "2018-06-27T16:34:26.276037261Z" }, {noformat}

2. **body:** I have been able to repro this reliably on a test cluster. Repro steps are: # Start sleep job with a lot of mappers sleeping for 50 seconds # on one worker node, kill NM after a set of containers starts # restart the NM # On the gw, kill the application (before the current containers finish) This will leave the containers on the node where the nodemanager was restarted in the exited state. container-executor is not cleaning up the docker containers. Here is an strace of one of the container-executors when the application is killed: {noformat} -bash-4.2$ sudo strace -s 4096 -f -p 7176 strace: Process 7176 attached read(3, "143\n", 4096) = 4 close(3) = 0 wait4(7566, [\{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 7566 --- SIGCHLD \{si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=7566, si_uid=0, si_status=0, si_utime=1, si_stime=0} --- munmap(0x7f233bfa4000, 4096) = 0 write(2, "Docker container exit code was not zero: 143\n", 45) = -1 EPIPE (Broken pipe) --- SIGPIPE \{si_signo=SIGPIPE, si_code=SI_USER, si_pid=7176, si_uid=0} --- +++ killed by SIGPIPE +++ {noformat} The problem is that when container-executor is started by the NM using the priviledged operation executor, it attaches stream readers to stdout and stderr. When we restart the NM, these threads are killed. Then when the application is killed, it kills the running containers and container-executor returns from waiting for the docker container. When it tries to write an error message to stderr, it generates a SIGPIPE signal, because the other end of the pipe has been killed. Since we are not handling that signal, container-executor crashes and we never remove the docker container. I have verified that if I change container-executor to ignore SIGPIPE, the problem does not occur.
   label: code-design
3. Submitted patch for trunk.
4. | (x) *{color:red}-1 overall{color}* | \\ \\ || Vote || Subsystem || Runtime || Comment || | {color:blue}0{color} | {color:blue} reexec {color} | {color:blue} 0m 35s{color} | {color:blue} Docker mode activated. {color} | || || || || {color:brown} Prechecks {color} || | {color:green}+1{color} | {color:green} @author {color} | {color:green} 0m 0s{color} | {color:green} The patch does not contain any @author tags. {color} | | {color:red}-1{color} | {color:red} test4tests {color} | {color:red} 0m 0s{color} | {color:red} The patch doesn't appear to include any new or modified tests. Please justify why no new tests are needed for this patch. Also please list what manual steps were performed to verify this patch. {color} | || || || || {color:brown} trunk Compile Tests {color} || | {color:green}+1{color} | {color:green} mvninstall {color} | {color:green} 27m 38s{color} | {color:green} trunk passed {color} | | {color:green}+1{color} | {color:green} compile {color} | {color:green} 1m 1s{color} | {color:green} trunk passed {color} | | {color:green}+1{color} | {color:green} mvnsite {color} | {color:green} 0m 37s{color} | {color:green} trunk passed {color} | | {color:green}+1{color} | {color:green} shadedclient {color} | {color:green} 40m 28s{color} | {color:green} branch has no errors when building and testing our client artifacts. {color} | || || || || {color:brown} Patch Compile Tests {color} || | {color:green}+1{color} | {color:green} mvninstall {color} | {color:green} 0m 36s{color} | {color:green} the patch passed {color} | | {color:green}+1{color} | {color:green} compile {color} | {color:green} 0m 56s{color} | {color:green} the patch passed {color} | | {color:green}+1{color} | {color:green} cc {color} | {color:green} 0m 56s{color} | {color:green} the patch passed {color} | | {color:green}+1{color} | {color:green} javac {color} | {color:green} 0m 56s{color} | {color:green} the patch passed {color} | | {color:green}+1{color} | {color:green} mvnsite {color} | {color:green} 0m 34s{color} | {color:green} the patch passed {color} | | {color:green}+1{color} | {color:green} whitespace {color} | {color:green} 0m 1s{color} | {color:green} The patch has no whitespace issues. {color} | | {color:green}+1{color} | {color:green} shadedclient {color} | {color:green} 12m 27s{color} | {color:green} patch has no errors when building and testing our client artifacts. {color} | || || || || {color:brown} Other Tests {color} || | {color:red}-1{color} | {color:red} unit {color} | {color:red} 18m 4s{color} | {color:red} hadoop-yarn-server-nodemanager in the patch failed. {color} | | {color:green}+1{color} | {color:green} asflicense {color} |

{color:green} 0m 23s{color} | {color:green} The patch does not generate ASF License warnings. {color} | | {color:black} {color} | {color:black} {color} | {color:black} 74m 30s{color} | {color:black} {color} | \\ \\ || Subsystem || Report/Notes || | Docker | Client=17.05.0-ce Server=17.05.0-ce Image:yetus/hadoop:abb62dd | | JIRA Issue | YARN-8515 | | JIRA Patch URL | https://issues.apache.org/jira/secure/attachment/12931329/YARN-8515.001.patch | | Optional Tests | asflicense compile cc mvnsite javac unit | | uname | Linux c76b663d45d3 3.13.0-153-generic #203-Ubuntu SMP Thu Jun 14 08:52:28 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux | | Build tool | maven | | Personality | /testptch/patchprocess/precommit/personality/provided.sh | | git revision | trunk / b37074b | | maven | version: Apache Maven 3.3.9 | | Default Java | 1.8.0_171 | | unit | https://builds.apache.org/job/PreCommit-YARN-Build/21222/artifact/out/patch-unit-hadoop-yarn-project_hadoop-yarn_hadoop-yarn-server_hadoop-yarn-server-nodemanager.txt | | Test Results | https://builds.apache.org/job/PreCommit-YARN-Build/21222/testReport/ | | Max. process+thread count | 301 (vs. ulimit of 10000) | | modules | C: hadoop-yarn-project/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager U: hadoop-yarn-project/hadoop-yarn/hadoop-yarn-server/hadoop-yarn-server-nodemanager | | Console output | https://builds.apache.org/job/PreCommit-YARN-Build/21222/console | | Powered by | Apache Yetus 0.8.0-SNAPSHOT http://yetus.apache.org | This message was automatically generated.

5. The unit test failure is YARN-8518. Might want to wait for that one to go through before we continue with this one, just to see that test-container-executor succeeds. I tested this manually, running several test jobs and restarting the NM while jobs were running. Because trunk has [~shanekumpf@gmail.com]'s docker life-cycle changes, I don't see the same failure I saw on branch 2.8, but the patch does not introduce any new problems that I can see.
6. Thanks for the patch! +1 lgtm. I'll commit this tomorrow if there are no objections.
7. Thanks, Jim! I committed this to trunk, branch-3.1, branch-3.0, branch-2, branch-2.9, and branch-2.8.