Arvados - Bug #8799

[Node manager] nodes in slurm drained state are counted as "up" but not candidates for shut down

03/25/2016 02:23 PM - Peter Amstutz

Status:	Resolved	**Start date:**	04/06/2016
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2016-04-13 sprint		

Description

If node manager crashes or otherwise fails during node shutdown, a node can go into drained state; when node manager recovers the drained node is not a candidate for shutdown (because it's not "idle") but is still considered "up".

Node manager should not count these nodes as "up".

In addition, if a node is "drained" but not being actively shut down, node manager should either put it back into idle state, or go ahead and start a new shutdown actor.

Subtasks:

Task # 8890: Review 8799-make-drained-nodes-idle	**Resolved**
Task # 8889: Re-enable nodes in drained state	**Resolved**

Related issues:

Related to Arvados - Story #8000: [Node Manager] Shut down nodes in SLURM 'do...	**Resolved**

Associated revisions

Revision 92d7a504 - 04/10/2016 02:11 AM - Peter Amstutz

Merge branch '8799-make-drained-nodes-idle' closes #8799

History

#1 - 03/25/2016 02:42 PM - Peter Amstutz

- Subject changed from [Node manager] nodes in slurm drained state are not candidates for shut down to [Node manager] nodes in slurm drained state are counted as "up" but not candidates for shut down

- Description updated

#2 - 03/31/2016 02:08 PM - Nico César

I fixed this for now doing this:

```
scontrol update NodeName=compute10 State=idle
```

but the jobs get queued for a long time

#3 - 03/31/2016 04:55 PM - Brett Smith

- Target version set to 2016-04-13 sprint

#4 - 03/31/2016 04:56 PM - Brett Smith

Please talk to ops about their preferences about whether Node Manager tries to recover drained nodes (it does already have some logic to do this in cases when the wishlist flip-flops), or whether it should just shut them down and start fresh with new nodes. As far as I'm concerned, they can write the acceptable criteria for this story, at least as far as Node Manager's production behavior goes.

#5 - 04/01/2016 07:58 PM - Peter Amstutz

- Assigned To set to Peter Amstutz

#6 - 04/01/2016 09:10 PM - Peter Amstutz

From conversation in engineering: nodes which are "drained" but do not have an active ShutdownActor should be put back into "idle" state.

#7 - 04/06/2016 04:36 PM - Brett Smith

Reviewing 04354c9

rec.arvados_node["info"].get("slurm_state") in ("drng", "drain") - I hate to be the bearer of bad news, but with today's code this will never be true. crunch-dispatch is responsible for setting this field, and it only ever sets slurm_state to idle, alloc, or down. See the slurm_status method in crunch-dispatch.

I would be happy to see a story about getting that changed, but there was a series of tickets that got us to this behavior, so it would need discussion and I would be a little antsy about blocking this story on that.

One idea about an alternate approach: when we pair the Arvados node with a cloud node, if rec.cloud_node.id not in self.shutdowns, call reenable_node. reenable_node can ask SLURM what the real status is (the ShutdownActor already has code for this), and re-set it to idle accordingly.

In keeping with the spirit of other recent changes, does reenable_node need to re-raise fatal exceptions like ENOMEM? Would it maybe be best if it only caught and warned about subprocess.CalledProcessError? And if you adopt the alternate approach I suggested, I think you'll need to figure out an alternate retry strategy, too, since my idea unfortunately doesn't have that baked in.

self.daemon.shutdowns.get()[cloud_node.id] = self.node_shutdown - It would be nice to pass self.TIMEOUT to get() here.

Thanks.

#8 - 04/06/2016 05:31 PM - Brett Smith

Sorry, another stray thought popped up when I got away from my keyboard for a bit: ComputeNodeUpdateActor's real raison d'etre is to throttle API requests to the cloud when those start failing, to make sure we're a nicely behaved client. Right now it does that by just sleeping. If the cloud is misbehaving and there's a big backlog of cloud API requests, then calls to reenable_node can potentially wait a long time as ComputeNodeUpdateActor slowly works through that backlog. Basically, the problem of #7026 could affect SLURM re-enabling too. Maybe reenabling should stay out of ComputeNodeUpdateActor to avoid that backlog?

#9 - 04/06/2016 07:35 PM - Peter Amstutz

Now 241ef75

New/modified behaviors:

- Arvados nodes which are reported as "down" are not counted as "up"
- Arvados nodes which are reported as "down" and don't have an active ShutdownActor are resumed with slurm.ComputeNodeMonitorActor.resume_node()
- Nodes that sinfo reports are in slurm end states (down/failed/drain) are shutdown eligible
- Unhandled exceptions in shutdown actor cause the shutdown to be canceled instead of just finished

#10 - 04/08/2016 06:39 PM - Brett Smith

Reviewing e13c868

I actually have the most comments about the tests, partly because I'm not sure they're demonstrating the desired behavior:

- test_shutdown_down_node and test_no_shutdown_drain_node have the exact same code after setting the mock's return value, so they seem to be asserting that they have the same behavior? It might be that the code is fine but a "reason not eligible for shutdown" string is truthy and passes assertTrue. Maybe prefer assertIs(True, actual) and assertIsNot(True, actual)?
- I don't understand why the "get SLURM state" subprocesses are repeated three times. Even looking at the code, I'm a little fuzzy how it gets to three calls. It also seems a little brittle to have the tests care about exactly how many times we check the current SLURM state, unless I'm missing some reason why that's important for correct functionality. Would it be a little less implementation-dependent to just check for the presence or absence of the update call?
- test_resume_node_exception sets both a mock's side_effect and return_value. I opened an interpreter to double-check that side_effect takes precedence. A comment about why we set return_value, or removing it, might help readability.

Assuming that the first point there doesn't reveal a bug, the code itself looks fine. There are a couple of places where I had to go checking *other* code to convince myself the behavior was right, so I might humbly suggest comments could be helpful for future readers there:

- rec.arvados_node["info"].get("slurm_state") == "down" - This relies on crunch-dispatch's state translation code, as discussed earlier. I wonder if a warning that it *will* break if that changes is prudent.

- The fact that you iterate on cloud_nodes and not arvados_nodes in _nodes_down is critical, because in_state('down') would return True for unpaired Arvados nodes.

Thanks.

#11 - 04/08/2016 07:36 PM - Brett Smith

Peter Amstutz wrote:

> - Nodes that sinfo reports are in slurm end states (down/failed/drain) are shutdown eligible

So you're doing #8000 in this branch too? That's fine, just making sure we're keeping track of things.

#12 - 04/08/2016 09:33 PM - Peter Amstutz

Brett Smith wrote:

> Reviewing e13c868
>
> I actually have the most comments about the tests, partly because I'm not sure they're demonstrating the desired behavior:
>
> - test_shutdown_down_node and test_no_shutdown_drain_node have the exact same code after setting the mock's return value, so they seem to be asserting that they have the same behavior? It might be that the code is fine but a "reason not eligible for shutdown" string is truthy and passes assertTrue. Maybe prefer assertIs(True, actual) and assertIsNot(True, actual)?

You're absolutely right, that is what was happeing. Fixing that showed a flaw in the logic, which I also fixed.

- I don't understand why the "get SLURM state" subprocesses are repeated three times. Even looking at the code, I'm a little fuzzy how it gets to three calls. It also seems a little brittle to have the tests care about exactly how many times we check the current SLURM state, unless I'm missing some reason why that's important for correct functionality. Would it be a little less implementation-dependent to just check for the presence or absence of the update call?

ComputeNodeMonitorActor schedules two initial calls to consider_shutdown() (one immediate one in the constructor, another at the "bootup fail timeout" in on_start(). The 3rd call is from resume_node(). However, you're right this is a bit noisy and brittle for what the test is checking for, so following your suggestion I made the check simpler.

- test_resume_node_exception sets both a mock's side_effect and return_value. I opened an interpreter to double-check that side_effect takes precedence. A comment about why we set return_value, or removing it, might help readability.

I think that was a typo. Removed.

> Assuming that the first point there doesn't reveal a bug, the code itself looks fine. There are a couple of places where I had to go checking *other* code to convince myself the behavior was right, so I might humbly suggest comments could be helpful for future readers there:

It did, and thanks for pointing it out.

- rec.arvados_node["info"].get("slurm_state") == "down" - This relies on crunch-dispatch's state translation code, as discussed earlier. I wonder if a warning that it *will* break if that changes is prudent.

I added a comment, and tweaked the logic to say not is ("idle", "alloc") which should have the same effect but is a bit less likely to break if we decide to distinguish between additional states.

- The fact that you iterate on cloud_nodes and not arvados_nodes in _nodes_down is critical, because in_state('down') would return True for unpaired Arvados nodes.

Also added a comment.

> So you're doing #8000 in this branch too? That's fine, just making sure we're keeping track of things.

That's right. Especially since crunch-dispatch smooshes "drain" and "down" into a single "down" state I decided I really needed to address both cases. I forgot we had a ticket for that behavior already. We can double check that #8000 is addressed and probably close it when this branch merges.

#13 - 04/08/2016 09:52 PM - Brett Smith

ba53423 looks good to me. Thanks.

#14 - 04/10/2016 02:15 AM - Peter Amstutz

- % Done changed from 50 to 100

- Status changed from New to Resolved

Applied in changeset arvados|commit:92d7a504629d848927d5334b0a801cd3993a585b.