Arvados - Feature #14291

[crunch-dispatch-cloud] AWS driver

10/03/2018 03:47 PM - Peter Amstutz

Status:	Resolved	**Start date:**	02/28/2019
Priority:	Normal	**Due date:**	
Assigned To:	Peter Amstutz	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	2019-03-13 Sprint		

Description

Subtasks:

Task # 14292: Review 14291-cdc-aws	**Resolved**

Related issues:

Related to Arvados - Story #14931: [arvados-dispatch-cloud] Configurable inst...	**Resolved**	**05/31/2019**
Related to Arvados - Story #13908: [Epic] Replace SLURM for cloud job schedul...	**Resolved**	
Related to Arvados - Bug #16270: "constraints not satisfiable by any configur...	**Resolved**	**04/01/2020**

Associated revisions

Revision af6447c8 - 03/08/2019 08:19 PM - Peter Amstutz

Merge branch '14291-cdc-aws' refs #14291

Arvados-DCO-1.1-Signed-off-by: Peter Amstutz <pamstutz@veritasgenetics.com>

History

#1 - 10/03/2018 03:47 PM - Peter Amstutz

- Status changed from New to In Progress

#2 - 10/03/2018 03:48 PM - Peter Amstutz

- Status changed from In Progress to New

- Assigned To set to Peter Amstutz

- Story points set to 2.0

#3 - 10/03/2018 04:12 PM - Peter Amstutz

- Assigned To deleted (Peter Amstutz)

- Target version changed from 2018-10-17 sprint to Arvados Future Sprints

#4 - 02/13/2019 04:36 PM - Tom Morris

- Target version changed from Arvados Future Sprints to 2019-02-27 Sprint

#5 - 02/13/2019 04:37 PM - Peter Amstutz

- Assigned To set to Peter Amstutz

#6 - 02/26/2019 08:20 PM - Peter Amstutz

- Status changed from New to In Progress

#7 - 02/27/2019 02:27 PM - Peter Amstutz

- Target version changed from 2019-02-27 Sprint to 2019-03-13 Sprint

#8 - 02/28/2019 04:44 PM - Peter Amstutz

14291-cdc-aws @ c82bd1f1c861beef92e317d5ce11136f4e4179a1

AWS driver with create, list, destroy, tag nodes, with tests. Supports attaching extra storage and spot instances. Uses official Amazon Go SDK instead of goamz. Added to crunch-dispatch-cloud drivers list.

A few things I encountered:

SSH keypair handling is a bit different. On Azure you provide the public key directly in the create machine request. On Amazon you need a separate import step, and then you refer to the public key by the name you give it. There's a limit to the number of keys you can import (5000). We could get clever with importing keys on the fly, or change the model so that instead of passing the ssh key in the create call, the ssh key configuration is driver specific and/or the key is passed in when the driver is initialized so it only has to be imported once.

Related, you can't specify the name of the user account like on Azure. The user account that gets the ssh key is baked into the image (I assume as part of the cloud-init configuration). On the images we use, I believe it is 'admin'. However I don't think this is a problem, we have "RemoteUser()" in the interface already, and AdminUser is already driver-specific.

On AWS, after a machine is terminated, it lingers in the instances list for a while (up to an hour, according to the docs). The worker will remember that the instance was in "shut down" state, but if the dispatcher restarts, those machines would be in "unknown" state. I don't know if we need to filter on VM state somewhere. The main thing we want to avoid is for it to think these machines are in "booting" state. I did notice that newly created machines get private IP addresses immediately, whereas terminated machines have a blank IP address.

Our InstanceType struct only has "Scratch", which is the number of bytes of scratch space used to make node size choices. Some instance types have scratch volumes by default, other instances require attached storage. I added an "AttachScratch" field to indicate when scratch space has to be explicitly requested, but this model could get more complicated.

#9 - 03/01/2019 07:33 PM - Tom Morris

- Release set to 15

#10 - 03/04/2019 05:59 PM - Tom Clegg

Peter Amstutz wrote:

> Uses official Amazon Go SDK instead of goamz.

Sigh of relief...

> SSH keypair handling is a bit different. On Azure you provide the public key directly in the create machine request. On Amazon you need a separate import step, and then you refer to the public key by the name you give it. There's a limit to the number of keys you can import (5000). We could get clever with importing keys on the fly, or change the model so that instead of passing the ssh key in the create call, the ssh key configuration is driver specific and/or the key is passed in when the driver is initialized so it only has to be imported once.

Is there an API for getting the fingerprint of an already-imported key by name? If so, we could name the key using its fingerprint, and check (once per driver per key) whether we need to import the key before using it in Create(). This would avoid some manual steps for the operator at install time and when changing the key.

(The existing code seems to be going in this direction, but needs a mutex on the *ec2InstanceSet, and shouldn't assume the key is the same in every call to Create.)

> Related, you can't specify the name of the user account like on Azure. The user account that gets the ssh key is baked into the image (I assume as part of the cloud-init configuration). On the images we use, I believe it is 'admin'. However I don't think this is a problem, we have "RemoteUser()" in the interface already, and AdminUser is already driver-specific.

Yes. Presumably we can't get the admin username from the Amazon API, so we'll just have to do the same thing as the Azure driver. Only the config docs will differ: Azure says "account we should create at runtime", EC2 will say "account you already baked into the image".

> On AWS, after a machine is terminated, it lingers in the instances list for a while (up to an hour, according to the docs). The worker will remember that the instance was in "shut down" state, but if the dispatcher restarts, those machines would be in "unknown" state. I don't know if we need to filter on VM state somewhere. The main thing we want to avoid is for it to think these machines are in "booting" state. I did notice that newly created machines get private IP addresses immediately, whereas terminated machines have a blank IP address.

Is it possible to see from the AWS response that the instance has been terminated? If so, it seems like the driver could safely drop it from the Instances() response entirely.

> Our InstanceType struct only has "Scratch", which is the number of bytes of scratch space used to make node size choices. Some instance types have scratch volumes by default, other instances require attached storage. I added an "AttachScratch" field to indicate when scratch space has to be explicitly requested, but this model could get more complicated.

(From discussion offline) We should have two separate size values, one for built-in scratch and one for external/attached scratch. The sum of these should be used when choosing an instance type; at least for now, the operator will need to either use an init script to combine them using devicemapper, or make sure one of them is zero.

I think it would be clearer to rename Scratch to ScratchBuiltin -- but for compatibility we (at least c-d-slurm) would need to continue accepting "Scratch" in configs. Perhaps "if scratch == 0 { scratch = builtin+attach }"?

Some initial comments after skimming the branch:

- SecurityGroupId → SecurityGroupIDs (I'm guessing restricting this to a single value would become an annoying problem sooner or later)
- SubnetId → SubnetID
- TAG_PREFIX → tagPrefix (ditto ARVADOS_DISPATCH_ID)
- EC2Interface looks like it shouldn't be exported
- The small amount of code here makes me very happy

#11 - 03/04/2019 11:16 PM - Peter Amstutz

Tom Clegg wrote:

> Is there an API for getting the fingerprint of an already-imported key by name? If so, we could name the key using its fingerprint, and check (once per driver per key) whether we need to import the key before using it in Create(). This would avoid some manual steps for the operator at install time and when changing the key.

So that's what I was alluding to in "we could get clever" but I feel like we should reconsider having "publicKey" in the interface for the Create() call. Since there is a limit to the number of keys you can import, I don't think the API should encourage a different key for every instance create (and if we are not going to do that, why bother?). Instead, it should be set it when the driver is initialized. On EC2 the driver initialization can still look up the key and/or do the key import for you so there's no additional work for the operator.

(Turns out yes, we can do a lookup by key fingerprint https://docs.aws.amazon.com/sdk-for-go/api/service/ec2/#EC2.DescribeKeyPairs)

> (From discussion offline) We should have two separate size values, one for built-in scratch and one for external/attached scratch. The sum of these should be used when choosing an instance type; at least for now, the operator will need to either use an init script to combine them using devicemapper, or make sure one of them is zero.
>
> I think it would be clearer to rename Scratch to ScratchBuiltin -- but for compatibility we (at least c-d-slurm) would need to continue accepting "Scratch" in configs. Perhaps "if scratch == 0 { scratch = builtin+attach }"?

The way it currently works for node manager, there is "disk" (provided by libcloud) and "scratch". If scratch > disk then it adds (scratch - disk) amount of attached storage. If we do something like, then for the builtin case Scratch BuiltinScratch, and for the attached disk case, Scratch is the amount to attach, and BuiltinScratch 0. This avoids backwards compatibility concerns. WDYT?

#12 - 03/05/2019 04:27 PM - Tom Clegg

Peter Amstutz wrote:

> So that's what I was alluding to in "we could get clever" but I feel like we should reconsider having "publicKey" in the interface for the Create() call. Since there is a limit to the number of keys you can import, I don't think the API should encourage a different key for every instance create (and if we are not going to do that, why bother?). Instead, it should be set it when the driver is initialized. On EC2 the driver initialization can still look up the key and/or do the key import for you so there's no additional work for the operator.

I think a better way to discourage the dispatcher from using too many different keys is to document that in the driver interface. The AWS driver might also log a warning if the number of stored keys seems too high.

Setting the key during driver initialization would create a new obstacle to on-the-fly key rotation (which we hope to do) without actually limiting the number of keys added -- e.g., "generate new key at startup" would still be just as easy and dangerous.

> (Turns out yes, we can do a lookup by key fingerprint https://docs.aws.amazon.com/sdk-for-go/api/service/ec2/#EC2.DescribeKeyPairs)

That's a bonus, although I suppose we should still name them something like "arvados-dispatch-cloud {fingerprint}" for uniqueness, and (only because it's so easy) check the fingerprint rather than trust "lookup by fingerprint" to be infallible.

> > I think it would be clearer to rename Scratch to ScratchBuiltin -- but for compatibility we (at least c-d-slurm) would need to continue accepting "Scratch" in configs. Perhaps "if scratch == 0 { scratch = builtin+attach }"?
>
> The way it currently works for node manager, there is "disk" (provided by libcloud) and "scratch". If scratch > disk then it adds (scratch - disk) amount of attached storage. If we do something like, then for the builtin case Scratch BuiltinScratch, and for the attached disk case, Scratch is the amount to attach, and BuiltinScratch 0. This avoids backwards compatibility concerns. WDYT?

Yes - it seems like the only question is whether to also accept explicit builtin+attached sizes. For example

- use "Scratch" as the number to use when choosing node types (if not given, use BuiltinScratch+AttachScratch)
- use "AttachScratch" to create/attach additional space when creating instances (if not given, use Scratch−BuiltinScratch)
- accept "BuiltinScratch" in config (default zero)

#13 - 03/06/2019 08:55 PM - Peter Amstutz

14291-cdc-aws @ 3246e20c201aedc9fd448ab99d7d0950654e218b

https://ci.curoverse.com/view/Developer/job/developer-run-tests/1102/

- Tweak symbol names
- Import public keys as needed
- Now there is "Scratch" "IncludedScratch" and "AddedScratch", if a value is missing, it is checked/set at config load time

#14 - 03/06/2019 10:13 PM - Tom Clegg

This seems like Scratch<Added+Included is an error but Scratch>Added+Included is automatically corrected by increasing Added, which seems arbitrary. Perhaps the second case should be "if Added==0 { Added=Scratch-Included }" so both kinds of mismatch are errors, and only 0/omitted values ever get automatically corrected? Also, it might be slightly clearer to make the 2nd and 3rd cases "else if" since it's impossible to match more than one of the conditions.

```
                        if t.Scratch == 0 {
                                t.Scratch = t.IncludedScratch + t.AddedScratch
                        }
                        if (t.Scratch - t.IncludedScratch) > t.AddedScratch {
                                t.AddedScratch = t.Scratch - t.IncludedScratch
                        }
                        if t.Scratch != (t.IncludedScratch + t.AddedScratch) {
                                return fmt.Errorf("%v: Scratch != (IncludedScratch + AddedScratch)", t.Name)
                        }
```

Amazon says VolumeSize is given in GiB, so I suspect this should be x>>30 instead of x/1000000000 (or, if the intent is to add 7%, it should be done in a more obvious way):

```
                                VolumeSize:          aws.Int64((int64(instanceType.AddedScratch) / 1000000000
) + 1),
```

Also, I'd be inclined to round up a bit more carefully -- maybe (x + 1<<30 - 1)>>30 -- so that requesting exactly 5GiB gets you exactly 5GiB.

I tried to figure out from docs whether we're double-encoding the userdata.

According to AWS docs

- "If you are using an AWS SDK or command line tool, Base64-encoding is performed for you, and you can load the text from a file. Otherwise, you must provide Base64-encoded text."

According to AWS SDK docs

- "If you are using a command line tool, base64-encoding is performed for you, and you can load the text from a file. Otherwise, you must provide base64-encoded text."

So... probably not? I guess the Go SDK docs should be more convincing.

Remove debug printf:

```
        log.Printf("terminating %v", *inst.instance.InstanceId)
```

This should use c.Logf():

```
                log.Printf("%v %v %v", i.String(), i.Address(), tg)
```

Instead of copying & pasting the testKey from Azure, both could use the test key & helper func from the dispatcher test package: pk, _ := test.LoadTestKey(c, "test/sshkey_dispatch").

I wonder if it's worth making the volume type configurable right away, just because it's easy? gp2 seems like an obvious default but st1 is half the price if the workload is HDD-suitable.

Should check the error returned by ImportKeyPair. If it's annoying to classify the error as "already exists" vs. "actual problem", maybe after any error we could try retrieving the key by name, and ignore the error if that works? I think with the current setup we'd just see a later error like "can't create instance because specified key doesn't exist" -- I'd like to make sure we also get a clue about why the import didn't work.

Is there a story behind this automatic tag? I see that nodemanager does this too, but is it needed/used by anything? Given the existence of the arvados-dispatch-id tag, this one seems superfluous.

```
                &ec2.Tag{
                        Key:   aws.String("arvados-class"),
                        Value: aws.String("dynamic-compute"),
                },
```

#15 - 03/07/2019 03:42 PM - Peter Amstutz

$GOPATH/bin/govendor list -v +unused +missing +external

```
  m github.com/aws/aws-sdk-go/internal/sdkio
    ├──  v  git.curoverse.com/arvados.git/vendor/github.com/aws/aws-sdk-go/aws
    ├──  vt git.curoverse.com/arvados.git/vendor/github.com/aws/aws-sdk-go/aws/request
    └──  vt git.curoverse.com/arvados.git/vendor/github.com/aws/aws-sdk-go/aws/signer/v4

$ $GOPATH/bin/govendor add -v +missing +external && $GOPATH/bin/govendor remove +unused
Error: Package "github.com/aws/aws-sdk-go/internal/sdkio" not a go package or not in GOPATH.

$ go get github.com/aws/aws-sdk-go/internal/sdkio

$ $GOPATH/bin/govendor add -v +missing +external && $GOPATH/bin/govendor remove +unused

$ $GOPATH/bin/govendor list -v +unused +missing +external

$ arvbox start test --only lib/cloud/ec2

            ********** Running lib/dispatchcloud install **********

/var/lib/arvados/test/GOPATH/src/git.curoverse.com/arvados.git/vendor/github.com/aws/aws-sdk-go/aws/types.go:7
:2: use of internal package not allowed

      ********** !!!!!! lib/dispatchcloud install FAILED !!!!!! **********

Failures (3):
Fail: cmd/arvados-server install (1s)
Fail: lib/dispatchcloud install (0s)
Fail: services/crunch-dispatch-slurm install (0s)

$ arvbox start test --skip-install --only lib/cloud/ec2

All test suites passed.
```

#16 - 03/07/2019 05:04 PM - Peter Amstutz

Seems like I messed up vendor.json at some point by using "govendor add" on individual dependencies. Possibly by adding "github.com/aws/aws-sdk-go" as a "tree" dependency. Here's how I fixed it:

1. cleaned up the GOPATH and vendor/ dirs
2. reverted vendor.json to master
3. ran "go get git.curoverse.com/arvados.git/lib/cloud/ec2"
4. ran "$GOPATH/bin/govendor add -v +missing +external"

Seems to be working now: https://ci.curoverse.com/view/Developer/job/developer-run-tests-remainder/1145/

#17 - 03/07/2019 09:50 PM - Peter Amstutz

Tom Clegg wrote:

> This seems like Scratch<Added+Included is an error but Scratch>Added+Included is automatically corrected by increasing Added, which seems arbitrary. Perhaps the second case should be "if Added==0 { Added=Scratch-Included }" so both kinds of mismatch are errors, and only 0/omitted values ever get automatically corrected? Also, it might be slightly clearer to make the 2nd and 3rd cases "else if" since it's impossible to match more than one of the conditions.

Now fixes up whichever one is missing, although if you have "Scratch" and both "Added" and "Included" are zero, it will prefer to set Added==Scratch.

> Amazon says VolumeSize is given in GiB, so I suspect this should be x>>30 instead of x/1000000000 (or, if the intent is to add 7%, it should be done in a more obvious way):
>
> Also, I'd be inclined to round up a bit more carefully -- maybe (x + 1<<30 - 1)>>30 -- so that requesting exactly 5GiB gets you exactly 5GiB.

Done.

> I tried to figure out from docs whether we're double-encoding the userdata.
>
> According to AWS docs
>
> - "If you are using an AWS SDK or command line tool, Base64-encoding is performed for you, and you can load the text from a file. Otherwise, you must provide Base64-encoded text."
>
> According to AWS SDK docs

- "If you are using a command line tool, base64-encoding is performed for you, and you can load the text from a file. Otherwise, you must provide base64-encoded text."

So... probably not? I guess the Go SDK docs should be more convincing.

It successfully runs the UserData script to create /var/run/test-file (verified by logging into the node) so I'm pretty confident it is doing it correctly, whatever it is doing.

> I wonder if it's worth making the volume type configurable right away, just because it's easy? gp2 seems like an obvious default but st1 is half the price if the workload is HDD-suitable.

Ideally this would go on InstanceType? I added EbsVolumeType but that applies to all instances.

> Should check the error returned by ImportKeyPair. If it's annoying to classify the error as "already exists" vs. "actual problem", maybe after any error we could try retrieving the key by name, and ignore the error if that works? I think with the current setup we'd just see a later error like "can't create instance because specified key doesn't exist" -- I'd like to make sure we also get a clue about why the import didn't work.

Yea, I had assumed everything was fine and it was an idempotent API call, but actually I just wasn't checking errors.

After a few loops around the garden path, I figured out how to generate the right kind of key fingerprints that AWS expects so now we can look it up to get the right name, and only try to import it if not found.

> Is there a story behind this automatic tag? I see that nodemanager does this too, but is it needed/used by anything? Given the existence of the arvados-dispatch-id tag, this one seems superfluous.
> [...]

Yes, the security policy on 4xphq disallows node manager's credentials from deleting anything not tagged "arvados-class: dynamic-compute". It took me and Ward a couple hours to figure out why I couldn't destroy nodes.

This actually raises 2 issues.

- We don't have a configuration option to add arbitrary user tags, so there's nowhere to put "arvados-class: dynamic-compute"
- Arbitrary user tags are currently given a prefix by Azure and EC2 drivers. However, in discussions with Nico, I also learned some customers have cloud policies that require labeling machines with their own tags. Prefixes interferes with this (as well as the arvados-class example above).

So on this ticket or a follow on ticket, we need to adjust how tagging works.

#18 - 03/07/2019 10:05 PM - Peter Amstutz

14291-cdc-aws 4cb692fff4953a4a69ea2fb50ec29a8f5e9c9951

https://ci.curoverse.com/view/Developer/job/developer-run-tests/1110/

#19 - 03/08/2019 12:24 AM - Tom Clegg

- Related to Story #14931: [arvados-dispatch-cloud] Configurable instance tags added

#20 - 03/08/2019 12:24 AM - Tom Clegg

Added #14931 for tagging.

#21 - 03/08/2019 07:42 PM - Tom Clegg

EbsVolumeType → EBSVolumeType

LGTM.

I think the current tagging situation is fine for dev/test/pilot purposes.

Should we update the Azure driver to error out if AddedScratch>0, and change our existing Azure dev configs (Scratch → IncludedScratch)? If we leave it as "assume you meant Included because we haven't implemented Added", it'll get confusing when we do implement Added>0: existing configs will mysteriously start buying extra scratch.

#22 - 03/08/2019 07:43 PM - Tom Clegg

- Related to Story #13908: [Epic] Replace SLURM for cloud job scheduling/dispatching added

#23 - 03/08/2019 09:30 PM - Peter Amstutz

- Status changed from In Progress to Resolved

#24 - 04/01/2020 08:39 PM - Tom Clegg

- Related to Bug #16270: "constraints not satisfiable by any configured instance type " added