Pulp - Issue #3541

Core should not add/remove content to a repository or create a repository_version without plugin input

04/02/2018 06:05 PM - amacdona@redhat.com

Status:	CLOSED - CURRENTRELEASE	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	bmbouter	**Estimated time:**	0:00 hour
Category:			
Sprint/Milestone:	3.0.0		
Severity:	2. Medium	**Groomed:**	Yes
Version:		**Sprint Candidate:**	No
Platform Release:		**Tags:**	
OS:		**Sprint:**	Sprint 61
Triaged:	Yes	**Quarter:**	

Description

General Problem:

Some plugins have validation requirements for the membership of content units in a repository. The validation required depends on the plugin and the content type. Currently add/remove is done with a POST to v3/repositories/1234/versions/. This does not involve the plugins at all, so there is no opportunity for plugins to create these validations.

Example Problem with Docker:

ManifestLists cannot be added unless the Manifests they refer to are also in the repository.
L1 is ManifestList, and it refers to (contains) 2 Manifests, M1, M2. The repository is considered corrupt if it contains L1, but not M1 and M2. M1 and M2 could be in the repository without L1.

Solutions:

Add a plugin opportunity to be involved

The RepositoryVersion.create() method will take a new, optional parameter called handler. A new object will be created in pulpcore.plugin.handlers.RepositoryVersionHandler.

```
class RepositoryVersionHandler:

    def add_content(self, qs_add_content, repo_version):
        pass

    def remove_content(self, qs_remove_content, repo_version):
        pass

    def validate(self, repo_version):
        # this method should only validate
        pass

    def repo_key_implementation(qs_add_content, repo_version, model_class, repo_unit_key):
        # the implementation of repo_key
        # not enabled by default
        # remove the content form repo_version when repo_unit_key has "another one" already in it.
```

Then a subclass would be:

```
class FileRepositoryVersionHandler(RepositoryVersionHandler):

    def __init__(custom_foo, custom_bar):
        # allow for customization here from params.
```

```
        pass

    def add_content(self, qs_add_content, repo_version):
        cls.repo_key_implementation(qs_add_content, repo_version, FileContent, 'relative_path')

# This effectively "enables" the repo_key functionality here instead of on the Content object.
```

The plugin will create an instance of their subclass, e.g. FileRepositoryVersionHandler, configured how they want, and then pass it to RepositoryVersion.create() like:

```
my_handler = FileRepositoryVersionHandler('a', 'b')
RepositoryVersion.create(handler=my_handler)
```

Providing a RepositoryVersionHandler is optional.

Integration with DeclarativeVersion

A new, optional parameter called handler will be added to DeclarativeVersion which will also accept and use the handler for various operations on the RepositoryVersion.

Related issues:

Related to Pulp - Task #3522: Plan Master/Detail Tasks	**CLOSED - WONTFIX**
Related to Python Support - Issue #3604: Bugs around creating content units	**MODIFIED**
Has duplicate Pulp - Issue #4740: Pulpcore doesn't provide a way to guarantee...	**CLOSED - DUPLICATE**
Blocked by Pulp - Story #5625: Typed Repositories	**CLOSED - CURRENTRELEASE**

Associated revisions

Revision 2f929910 - 11/12/2019 10:20 PM - bmbouter

Sync, Upload, Modify preserve uniqueness

Sync, Upload, and Modify now have added content with the same relative_path as existing content will remove the existing content.

Required PR: https://github.com/pulp/pulpcore/pull/369

https://pulp.plan.io/issues/3541 re #3541

Revision 16ec4589 - 11/13/2019 11:44 AM - ipanova@redhat.com

No duplicated content can be present in a repository version.

Required PR: https://github.com/pulp/pulpcore/pull/369

closes #3541

Revision 16ec4589 - 11/13/2019 11:44 AM - ipanova@redhat.com

No duplicated content can be present in a repository version.

Required PR: https://github.com/pulp/pulpcore/pull/369

closes #3541

Revision 09fe6589 - 11/13/2019 07:13 PM - bmbouter

Add Repository.finalize_new_version

This PR adds a central place for plugin writers to put validation and content modification code.

- RepositoryVersion.__exit__ calls finalize_new_version
- Moves the /modify/ endpoint to pulpcore.plugin.actions as a mixin named ModifyRepositoryActionMixin.
- Renames Repository.repo_key to Repository.repo_key_fields.
- Remove the RemoveDuplicates stage for plugin writers to instead use Repository.finalize_new_version
- Remove the implicit usage of RemoveDuplicates from RepositoryVerison.add_content and RepositoryVersion.remove_content.
- Make the RemoveDuplicates available as pulpcore.plugin.repo_version_utils.remove_duplicates.

As an unrelated change, it also replaces the the >>> python codeblocks in various docblocks with python codeblock using :: and indention.

Required PR: https://github.com/pulp/pulp_file/pull/307

https://pulp.plan.io/issues/3541 re #3541

Revision 7645b4e8 - 11/13/2019 08:17 PM - bmbouter

Fixup release note and docstring

https://pulp.plan.io/issues/3541 closes #3541

History

#1 - 04/02/2018 08:07 PM - amacdona@redhat.com

- Related to Task #3522: Plan Master/Detail Tasks added

#2 - 04/04/2018 08:34 PM - milan

- Description updated

#3 - 04/10/2018 05:36 PM - bmbouter

Option 1 has a problem with it that there are some plugin types that don't need validation and now they would have to provide extra views, which for them just means more work.

Because of ^, I believe option 2 is ideal for the most number of plugin writers and users. Here's one description of what that solution could look like in practice:

```
class DockerManifestList(ContentUnit):

    ...

    @staticmethod
    def validate_content_in_repo_version(repo_version):
        """
        Validates a repository version's DockerManifestList units.

        This method specifically ensures that the repo has all the right DockerManifests that correspond
        with this DockerManifestList

        Args:
            repo_version (RepositoryVersion): The Repository version to validate

        Raises:
            django.core.exceptions.ValidationError: if the repository is invalid
        """
        # check all of the validation related DockerManifestList content
        if has_a_validation_problem:
            raise ValidationException('Repo foo has ManifestList Y but is missing Manifest Z')
```

What we can do then is add a validate method to RepositoryVersion itself that will call the validate method for each unique type of unit in the repo. If any of them fail allow the exception to be raised. This would look like:

```
class RepositoryVersion(Model):

    ...

    def validate(self):
        for type_name in all_unique_content_unit_types_in_self:
            type_cls = get_the_class_from_the_name(type_name) # Gets a reference to the class
            type_cls.validate_content_in_repo_version()
```

This could be called by anyone who wants to validate a RepositoryVersion so that is a new added value. One place specifically where that would be useful would be on the __exit__(...) context manager on RepositoryVerison.

The net effect would be that:

1. plugin writers can provide validation
2. Core can use that validation to validate repository versions
3. This provides multiple layers of validation so that plugin writer code can use these facilities to create value also.

#4 - 04/17/2018 04:32 PM - amacdona@redhat.com

- Triaged changed from No to Yes

#5 - 04/25/2018 10:36 PM - amacdona@redhat.com

- Related to Issue #3604: Bugs around creating content units added

#6 - 04/26/2019 12:42 AM - daviddavis

- Sprint/Milestone set to 3.0.0

#7 - 04/26/2019 12:42 AM - daviddavis

- Blocks Task #4028: Prevent duplicate files in repositories added

#8 - 04/26/2019 12:43 AM - daviddavis

- Has duplicate Issue #4740: Pulpcore doesn't provide a way to guarantee uniqueness in repo versions added

#9 - 04/26/2019 10:35 PM - bmbouter

- Tags deleted (Pulp 3)

#10 - 09/13/2019 05:12 PM - bmbouter

- Sprint/Milestone changed from 3.0.0 to 71

#11 - 09/13/2019 05:42 PM - bmbouter

- Sprint/Milestone changed from 71 to 3.0.0

#12 - 10/03/2019 06:08 PM - daviddavis

- Blocks deleted (Task #4028: Prevent duplicate files in repositories)

#13 - 10/09/2019 08:35 PM - bmbouter

- Subject changed from Core should not add/remove content to a repository without plugin input to Core should not add/remove content to a repository or create a repository_version without plugin input

- Description updated

#14 - 10/09/2019 08:44 PM - bmbouter

- Description updated

#15 - 10/09/2019 09:06 PM - bmbouter

With SomeContent.add_content and SomeContent.remove_content modifying the RepositoryVersion possible, each call to RepositoryVersion.add_content and RepositoryVersion.remove_content plugin writers need to be aware of this.

I looked at the Stages API, and I believe it's safe because the last thing it does is call RepositoryVersion.add_content for the batch here and then requests a new batch. Similarly in RepositoryVersion.remove_content is called here before a new batch is requested.

#16 - 10/09/2019 09:09 PM - bmbouter

- Description updated

#17 - 10/09/2019 09:10 PM - bmbouter

- Description updated

#18 - 10/10/2019 10:08 PM - bmbouter

- Sprint set to Sprint 60

#19 - 10/11/2019 03:22 PM - ttereshc

- Groomed changed from No to Yes

#20 - 10/14/2019 10:32 PM - gmbnomis

I am very sorry to be late again with my comment, but, looking at the current proposal, it feels very much like a distributed implementation of a plugin specific add/remove endpoint (the former option 1). What I mean by this: If we implement creating a repository version using a plugin endpoint, we would customize the same operations (add, remove, and validation), but not per content type, but per plugin.

I think that doing it on plugin level would allow us to follow a more "holistic" approach:

1. The validation as proposed can only validate content types that are present, but not content type that belong to a plugin but are not there. If, for example, a plugin has a content type that needs to be in a (non-empty) repo version exactly once (say a file with external signatures), it would be harder to implement a check for the "content is missing" case. Basically, all other content types would have to check the presence of the potentially missing content type.

2. Since a plugin knows exactly which content types it manages, it is easier to take a "holistic" more optimized approach and

- only check content types that are necessary
- reuse database query results
- query multiple content types at once (e.g. using prefetching)

3. If add_content/remove_content implement additional functionality, it is easier to support additional parameters if the endpoint belongs to the plugin. For example, whether to remove RPMs based on ModuleMD removal could be a flag to the add/remove POST (I have no idea whether this makes sense, but you hopefully get the idea why additional plugin specific parameters might be sensible)

4. I don't know if repositories containing content of multiple plugins is still a thing, But if so, validation as proposed would always re-validate the entire content across all content types even if the added/removed content only belongs to a single plugin.

Of course, there are drawbacks to a plugin specific endpoint:
1. As already said, plugins will need to do additional work even in the "no customization required" case. (But this is the case for other entities already)
2. One cannot add/remove content from multiple plugins in a single add/remove operation (and thus, in a single repo version).

I reckon that 2. is the more important point here. If we regard this as a key feature, the "hooks per content type" approach may be better suited. Otherwise, the per plugin approach looks more flexible and easier to optimize in the future (IMHO).

Regardless on how we decide, I have two suggestions to the approach described in the issue above:

- I have the strong feeling that putting add_content, remove_content and validate_repo_version hooks into Content is overloading the Content classes. Content should be pretty basic and should not have intricate knowledge of RepoVersion. I think we need to find a way to put this functionality somewhere else and link it to the respective Content class/type.
- The recursive call_plugin_hook parameter feels error prone. What if we split up the methods? In this case, we would have additional "raw" content add/remove methods in RepositoryVersion that call no hooks and are to be used by content add/remove hooks.

#21 - 10/15/2019 09:46 PM - bmbouter

gmbnomis wrote:

> I am very sorry to be late again with my comment, but, looking at the current proposal, it feels very much like a distributed implementation of a plugin specific add/remove endpoint (the former option 1). What I mean by this: If we implement creating a repository version using a plugin endpoint, we would customize the same operations (add, remove, and validation), but not per content type, but per plugin.
>
> I think that doing it on plugin level would allow us to follow a more "holistic" approach:
>
> 1. The validation as proposed can only validate content types that are present, but not content type that belong to a plugin but are not there. If, for example, a plugin has a content type that needs to be in a (non-empty) repo version exactly once (say a file with external signatures), it would be harder to implement a check for the "content is missing" case. Basically, all other content types would have to check the presence of the potentially missing content type.

I agree doing it at the plugin level could be better than the content-by-content level. The case that a whole type is missing motivates that well.

> 2. Since a plugin knows exactly which content types it manages, it is easier to take a "holistic" more optimized approach and
>
> - only check content types that are necessary
> - reuse database query results
> - query multiple content types at once (e.g. using prefetching)

Agreed. I like the "plugin check" replacing the "type-by-type check"

> 3. If add_content/remove_content implement additional functionality, it is easier to support additional parameters if the endpoint belongs to the plugin. For example, whether to remove RPMs based on ModuleMD removal could be a flag to the add/remove POST (I have no idea whether this makes sense, but you hopefully get the idea why additional plugin specific parameters might be sensible)

I agree. For the copy case this makes perfect sense. The challenge is that these checks also need to be at the add_content and remove_content API itself so cases like sync can automatically handle them as well. Having the additional call to the "holistic" plugin check in RepositoryVersion.add_content and RepositoryVersion.remove_content is easy enough, but passing extra parameters in there isn't. What do you think about the need for this to be included for all RepositoryVersion manipulations, not just those driven by plugin endpoints?

> 4. I don't know if repositories containing content of multiple plugins is still a thing, But if so, validation as proposed would always re-validate the entire content across all content types even if the added/removed content only belongs to a single plugin.

I believe dalley has suggested we adopt master/detail repositories and add the validation there, and treat them similar to how other plugins require a

plugin writer to make it's Detail instance.

> Of course, there are drawbacks to a plugin specific endpoint:
> 1. As already said, plugins will need to do additional work even in the "no customization required" case. (But this is the case for other entities already)

I'm still concerned about this not being inclusive of all RepositoryVersion manipulations.

> 2. One cannot add/remove content from multiple plugins in a single add/remove operation (and thus, in a single repo version).
>
> I reckon that 2. is the more important point here. If we regard this as a key feature, the "hooks per content type" approach may be better suited. Otherwise, the per plugin approach looks more flexible and easier to optimize in the future (IMHO).
>
> Regardless on how we decide, I have two suggestions to the approach described in the issue above:
>
> - I have the strong feeling that putting add_content, remove_content and validate_repo_version hooks into Content is overloading the Content classes. Content should be pretty basic and should not have intricate knowledge of RepoVersion. I think we need to find a way to put this functionality somewhere else and link it to the respective Content class/type.

+1 to this approach, let's adjust the plan

> - The recursive call_plugin_hook parameter feels error prone. What if we split up the methods? In this case, we would have additional "raw" content add/remove methods in RepositoryVersion that call no hooks and are to be used by content add/remove hooks.

I didn't quiet follow this last part.

#22 - 10/16/2019 12:45 PM - daviddavis

> The recursive call_plugin_hook parameter feels error prone. What if we split up the methods? In this case, we would have additional "raw" content add/remove methods in RepositoryVersion that call no hooks and are to be used by content add/remove hooks.

I agree with this. I think that we're overloading add_content and remove_content and I also worry about these functions being called recursively. Where would the content add/remove hooks be called from though?

#23 - 10/16/2019 08:54 PM - bmbouter

We have several plugin writer conventions about which python modules they will place various objects. We could do one more of those and have the plugin writer write a MyPluginRepositoryVersionHandler. This would be in plugin_name.app.handlers. It could have the add_content and remove_content, and validate interfaces there.

Core would provide pulpcore.plugin.handlers.RepositoryVersionHandler and plugin writers would subclass it. The core defines:

```
class RepositoryVersionHandler:

    @clsmethod
    def add_content(cls, qs_add_content, repo_version):
        pass

    @clsmethod
    def remove_content(cls, qs_remove_content, repo_version):
        pass

    @clsmethod
    def validate(cls, repo_version):
        pass

    def repo_key_implementation(qs_add_content, repo_version, model_class, repo_unit_key):
        # the implementation of repo_key
        # not enabled by default
        # remove the content form repo_version when repo_unit_key has "another one" already in it.
```

Then a subclass would be:

```
class FileRepositoryVersionHandler(RepositoryVersionHandler):

    @clsmethod
    def add_content(cls, qs_add_content, repo_version):
        cls.repo_key_implementation(qs_add_content, repo_version, FileContent, 'relative_path')
        # This effectively "enables" the repo_key functionality here instead of on the Content object.
```

Core would load this and know that for any Content type provided by this plugin, this is the Handler to call. The RepositoryVersion.add_content() and RepositoryVersion.remove_content() would call these handles for any content in the queryset it is adding/removing. There would be 1 call per plugin so it could do it holistically.

Providing a RepositoryVersionHandler is optional. Plugins that don't need it won't have to specify it.

#24 - 10/17/2019 01:08 PM - daviddavis

This sounds great. +1 from me.

I wonder if the add_content could handle https://pulp.plan.io/issues/5567#note-8 by default too?

#25 - 10/17/2019 01:17 PM - gmbnomis

bmbouter wrote:

> We have several plugin writer conventions about which python modules they will place various objects. We could do one more of those and have the plugin writer write a MyPluginRepositoryVersionHandler. This would be in plugin_name.app.handlers. It could have the add_content and remove_content, and validate interfaces there.

I like the idea of a handler. However, this scheme does not allow to customize the handler using parameters (as discussed above).

Perhaps we could make the handler instance-based instead of class-based. In this scheme, the plugin would prepare a handler instance (if it needs one) and pass it to RepositoryVersion as a parameter.

For sync, the handler instance would be an additional optional parameter to DeclarativeVersion.
For the add/remove endpoint (which would be specific to the plugin), the creation of the handler needs to happen in the respective task (which could pass parameters from the add/remove call to the handler)

One important use case for the handler parameters could be to have different behavior of the handler during sync. (For example auto-generating/auto-removing some additional content may only be desired on the add/remove endpoint, but not for sync)

> Core would load this and know that for any Content type provided by this plugin, this is the Handler to call. The RepositoryVersion.add_content() and RepositoryVersion.remove_content() would call these handles for any content in the queryset it is adding/removing. There would be 1 call per plugin so it could do it holistically.

The instance based handler is definitely more work for the plugin writer (because it is not automatically picked up like your proposed scheme and we have more boilerplate code for the endpoint). OTOH, I think the instance based approach is more flexible.

#26 - 10/17/2019 08:47 PM - bmbouter

- Description updated

Edits to incorporate the feedback from gmbnomis that seems clear we should do.

#27 - 10/17/2019 08:50 PM - bmbouter

gmbnomis I wrote out what I think is the design you were mentioning. I think it's a great idea because it removes much of the guesswork about "which content" or "which plugin" to call for various operations. I just did an audit of this type of usage, and plugin code has all the opportunities to create the instances customized in various ways.

#28 - 10/17/2019 09:55 PM - ttereshc

Thanks for the discussion and re-write of the issue, it looks good to me.
+1 for the approach with handler
+1 to allow customization in plugin handler's *init*
+1 to keep core endpoint for repo version creation and not to force plugins to define their own.

#29 - 10/18/2019 10:51 AM - gmbnomis

ttereshc wrote:

> +1 to keep core endpoint for repo version creation and not to force plugins to define their own.

as discussed on IRC, I think that a plugin with a custom handler needs to have an endpoint of its own. Then, the plugin has a add/remove task of its own where it can instantiate the handler.

I don't know how to get a custom handler instance into the generic task at
https://github.com/pulp/pulpcore/blob/065364029f21f7aefcda44ce601fcd7b86cf10ed/pulpcore/app/tasks/repository.py#L97.

#30 - 10/25/2019 09:46 AM - rchan

- Sprint changed from Sprint 60 to Sprint 61

#31 - 10/26/2019 09:43 PM - dalley

- Blocked by Story #5625: Typed Repositories added

#32 - 11/07/2019 07:37 PM - dalley

- Status changed from NEW to ASSIGNED

- Assignee set to bmbouter

#33 - 11/08/2019 06:58 PM - bmbouter

Initial PRs handling sync and upload use cases are here:

https://github.com/pulp/pulp_file/pull/307/files
https://github.com/pulp/pulpcore/pull/369/files

#34 - 11/12/2019 11:09 PM - bmbouter

- Status changed from ASSIGNED to POST

#35 - 11/13/2019 08:43 PM - bmbouter

- Status changed from POST to MODIFIED

Applied in changeset pulpcore|7645b4e84c7fae14c49b12ac0c40e6c85b093779.

#36 - 12/06/2019 05:55 PM - ipanova@redhat.com

Applied in changeset pulp_container:16ec458969a7598e46cbb3a8c2e6100ae5842158.

#37 - 12/13/2019 06:30 PM - bmbouter

- Status changed from MODIFIED to CLOSED - CURRENTRELEASE