Pulp - Story #2656

As a plugin developer, I can declare what platform plugin API version(s) I support.

03/22/2017 06:14 PM - semyers

Status:	CLOSED - WORKSFORME	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0:00 hour
Sprint/Milestone:	3.0.0		
Platform Release:		**Tags:**	
Groomed:	No	**Sprint:**	
Sprint Candidate:	No	**Quarter:**	

Description

misa brought this up in a recent conversation, which was a desire for a plugin to be able to declare that it does not work unless the platform plugin API version is greater or less than a given version.

Functionally speaking, I think this means that a plugin *should* to be able declare a minimum-supported platform plugin API version and a maximum-supported plugin API version.

Comparator support could also be useful, so that for the minimum supported version a plugin developer could specify that the platform version must be "greater than" *or* "greater than or equal" to the platform plugin API version and for the maximum supported version a plugin develop could specify "less than" *or* "less than or equal to" a given platform plugin API version. Optionally, a simpler approach would be to declare that the minimum supported version evaluated by platform will always be evaluated as "greater than or equal" and the maximum supported version will always be evaluated as "less than".

So, for example, we introduce a new feature in 3.2 that results in a corresponding y-version bump of the plugin API version. Any plugin that requires that new feature to run should be able to report the minimum plugin API version required for that plugin to function.

Another example is if we alter a feature in 4.0, so that a plugin depending on that feature no longer works with 4.0. That plugin should be able to indicate that its maximum plugin API version is any version less than the corresponding platform plugin API version.

At the moment, I believe that the plugin API will be versioned separately from the platform code versioning. If we decide not to do that, then any mention of plugin API versioning should be changed to refer directly to the platform version.

This story is not concerned with what platform does, when loading plugins, if a plugin reports that it is incompatible with the Pulp platform that is loading it. We can file that story if this one is accepted, since there's minimal value in filing such a story if we don't first supply an interface for plugins to declare their compatibility.

History

#1 - 03/22/2017 08:04 PM - bmbouter

We should definitely do this. Regarding plugin api versioning, we need to version the plugin api separately from the platform code to allow user facing semver to not interfere with plugin developer facing semver.

#2 - 03/22/2017 08:09 PM - bmbouter

To groom this I think we need a specific proposal. I'm not sure how platform would discover the interface that the plugin writer implements. Several possibilities include:

- Accessible through the plugin entry point and accessed by some naming convention.
- Use Django's discovery mechanisms to get a reference to the plugin's interface implementation.

Also is this interface a function, method, or class?

#3 - 07/14/2017 02:59 PM - amacdona@redhat.com

- Tags Pulp 3 Plugin Writer Alpha added

#4 - 07/17/2017 04:42 PM - bmbouter

The current thinking is that this is already handled through dependency resolution. Any plugin can declare a dependency like: pulpcore-plugin~=x.y which will prevent a backwards incompatible version from ever being installed. My concern is that pip wouldn't respect this in all of the necessary

situations.

#5 - 07/17/2017 04:44 PM - bizhang

There is definitely going to be documentation on how a plugin writer should declare pulpcore-plugin dependency, and what they can expect from that. There is some of this up already: http://docs.pulpproject.org/en/3.0/nightly/contributing/3.0-development/plugin-layout.html#packaging

#6 - 07/17/2017 05:48 PM - lchimonji10

pip will upgrade or downgrade packages as necessary to make them comply with version specifiers. Here's an example with ~=:

```
#!/usr/bin/env bash

python3 -m venv env
source env/bin/activate
pip install --upgrade pip
pip install requests==2.18.1
pip freeze | grep requests # requests==2.18.1

cat >requirements.txt <
```

pip will also upgrade or downgrade packages as necessary for version specifiers like < and >=. For more information on version specifiers, see PEP 440.

#7 - 07/17/2017 06:44 PM - bizhang

- Tags Pulp 3 added

- Tags deleted (Pulp 3 Plugin Writer Alpha)

As discussed on IRC this is a general pip issue when it comes do dependency resolution. If a plugin is dependent on pulpcore-plugin~=1.8 and a user runs pip install --upgrade pulpcore-plugin and pulls down pulpcore-plugin==2.0 the plugin would stop working.

This is not something that needs to be resolved for the alpha so I am removing that tag.
There is currently work being done on pip that will add dependency resolution: https://github.com/pypa/pip/issues/988#issuecomment-308969728

We can reevaluate this for pulp3 at a later date

#8 - 12/05/2018 02:31 PM - amacdona@redhat.com

- Status changed from NEW to CLOSED - WORKSFORME

Using standard pip depsolving seems to be working for us at the moment. I don't think a user would explicitly upgrade/install pulpcore-plugin, instead it should be upgraded and installed when upgrading/installing the plugin. Please reopen if there are problems.

#9 - 04/25/2019 06:46 PM - daviddavis

- Sprint/Milestone set to 3.0.0

#10 - 04/26/2019 10:38 PM - bmbouter

- Tags deleted (Pulp 3)