jctanner collectionversion search_vector should not include every tag ever (#1277 … History

10 contributors

419 lines (322 sloc) | 14.1 KB

```python
from logging import getLogger

from django.conf import settings
from django.db import models
from django.db.models import UniqueConstraint, Q
from django.contrib.postgres import fields as psql_fields
from django.contrib.postgres import search as psql_search
from django_lifecycle import AFTER_UPDATE, BEFORE_UPDATE, hook

from pulpcore.plugin.models import (
    BaseModel,
    Content,
    Remote,
    Repository,
    RepositoryVersion,
    Distribution,
    SigningService,
    Task,
    EncryptedTextField,
)
from .downloaders import AnsibleDownloaderFactory


log = getLogger(__name__)


class Role(Content):
    """
    A content type representing a Role.
    """

    TYPE = "role"

    namespace = models.CharField(max_length=64)
    name = models.CharField(max_length=64)
    version = models.CharField(max_length=128)

    @property
    def relative_path(self):
        """
        Return the relative path of the ContentArtifact.
        """
        return self.contentartifact_set.get().relative_path

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"
        unique_together = ("version", "name", "namespace")


class Collection(BaseModel):
    """A model representing a Collection."""

    namespace = models.CharField(max_length=64, editable=False)
    name = models.CharField(max_length=64, editable=False)

    def __str__(self):
        """Return a representation."""
        return f"<{self.__class__.__name__}: {self.namespace}.{self.name}>"

    class Meta:
        unique_together = ("namespace", "name")


class CollectionImport(models.Model):
    """A model representing a collection import task details."""

    task = models.OneToOneField(
        Task, on_delete=models.CASCADE, editable=False, related_name="+", primary_key=True
    )
    messages = models.JSONField(default=list, editable=False)

    class Meta:
        ordering = ["task__pulp_created"]

    def add_log_record(self, log_record):
        """
        Records a single log message but does not save the CollectionImport object.

        Args:
            log_record(logging.LogRecord): The logging record to record on messages.

        """
        self.messages.append(
            {"message": log_record.msg, "level": log_record.levelname, "time": log_record.created}
        )


class Tag(BaseModel):
    """A model representing a Tag.

    Fields:

        name (models.CharField): The Tag's name.
    """

    name = models.CharField(max_length=64, unique=True, editable=False)

    def __str__(self):
        """Returns tag name."""
        return self.name


class CollectionVersion(Content):
    """
    A content type representing a CollectionVersion.

    This model is primarily designed to adhere to the data format for Collection content. That spe
    is here: https://docs.ansible.com/ansible/devel/dev_guide/collections_galaxy_meta.html

    Fields:

        authors (psql_fields.ArrayField): A list of the CollectionVersion content's authors.
        contents (models.JSONField): A JSON field with data about the contents.
        dependencies (models.JSONField): A dict declaring Collections that this collection
            requires to be installed for it to be usable.
        description (models.TextField): A short summary description of the collection.
        docs_blob (models.JSONField): A JSON field holding the various documentation blobs in
            the collection.
        manifest (models.JSONField): A JSON field holding MANIFEST.json data.
        files (models.JSONField): A JSON field holding FILES.json data.
        documentation (models.CharField): The URL to any online docs.
        homepage (models.CharField): The URL to the homepage of the collection/project.
        issues (models.CharField): The URL to the collection issue tracker.
        license (psql_fields.ArrayField): A list of licenses for content inside of a collection.
        name (models.CharField): The name of the collection.
        namespace (models.CharField): The namespace of the collection.
        repository (models.CharField): The URL of the originating SCM repository.
        version (models.CharField): The version of the collection.
        requires_ansible (models.CharField): The version of Ansible required to use the collection
        is_highest (models.BooleanField): Indicates that the version is the highest one
            in the collection. Import and sync workflows update this field, which then
            triggers the database to [re]build the search_vector.

    Relations:

        collection (models.ForeignKey): Reference to a collection model.
        tag (models.ManyToManyField): A symmetric reference to the Tag objects.
    """

    TYPE = "collection_version"

    # Data Fields
    authors = psql_fields.ArrayField(models.CharField(max_length=64), default=list, editable=False
    contents = models.JSONField(default=list, editable=False)
    dependencies = models.JSONField(default=dict, editable=False)
    description = models.TextField(default="", blank=True, editable=False)
    docs_blob = models.JSONField(default=dict, editable=False)
    manifest = models.JSONField(default=dict, editable=False)
    files = models.JSONField(default=dict, editable=False)
    documentation = models.CharField(default="", blank=True, max_length=2000, editable=False)
    homepage = models.CharField(default="", blank=True, max_length=2000, editable=False)
    issues = models.CharField(default="", blank=True, max_length=2000, editable=False)
    license = psql_fields.ArrayField(models.CharField(max_length=32), default=list, editable=False
    name = models.CharField(max_length=64, editable=False)
    namespace = models.CharField(max_length=64, editable=False)
    repository = models.CharField(default="", blank=True, max_length=2000, editable=False)
    version = models.CharField(max_length=128, editable=False)
    requires_ansible = models.CharField(null=True, max_length=255)

    is_highest = models.BooleanField(editable=False, default=False)

    # Foreign Key Fields
    collection = models.ForeignKey(
        Collection, on_delete=models.PROTECT, related_name="versions", editable=False
    )
    tags = models.ManyToManyField(Tag, editable=False)

    # Search Fields
    #   This field is populated by a trigger setup in the database by
    #   a migration file. The trigger only runs when the table is
    #   updated. CollectionVersions are INSERT'ed into the table, so
    #   the search_vector does not get populated at initial creation
    #   time. In the import or sync workflows, is_highest gets toggled
    #   back and forth, which causes an UPDATE operation and then the
    #   search_vector is built.
    search_vector = psql_search.SearchVectorField(default="")
```

```python

    @property
    def relative_path(self):
        """
        Return the relative path for the ContentArtifact.
        """
        return "{namespace}-{name}-{version}.tar.gz".format(
            namespace=self.namespace, name=self.name, version=self.version
        )

    def __str__(self):
        """Return a representation."""
        return f"<{self.__class__.__name__}: {self.namespace}.{self.name} {self.version}>"

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"
        unique_together = ("namespace", "name", "version")
        constraints = [
            UniqueConstraint(
                fields=("collection", "is_highest"),
                name="unique_is_highest",
                condition=Q(is_highest=True),
            )
        ]


class CollectionVersionSignature(Content):
    """
    A content type representing a signature that is attached to a content unit.

    Fields:
        data (models.BinaryField): A signature, base64 encoded. # Not sure if it is base64 encoded
        digest (models.CharField): A signature sha256 digest.
        pubkey_fingerprint (models.CharField): A fingerprint of the public key used.

    Relations:
        signed_collection (models.ForeignKey): A collection version this signature is relevant to.
        signing_service (models.ForeignKey): An optional signing service used for creation.
    """

    PROTECTED_FROM_RECLAIM = False
    TYPE = "collection_signature"

    signed_collection = models.ForeignKey(
        CollectionVersion, on_delete=models.CASCADE, related_name="signatures"
    )
    data = models.TextField()
    digest = models.CharField(max_length=64)
    pubkey_fingerprint = models.CharField(max_length=64)
    signing_service = models.ForeignKey(
        SigningService, on_delete=models.SET_NULL, related_name="signatures", null=True
    )

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"
        unique_together = ("pubkey_fingerprint", "signed_collection")


class DownloadLog(BaseModel):
    """
    A download log for content units by user, IP and org_id.
    """

    content_unit = models.ForeignKey(
        Content, on_delete=models.CASCADE, related_name="download_logs"
    )
    user = models.ForeignKey(
        settings.AUTH_USER_MODEL,
        on_delete=models.CASCADE,
        null=True,
        related_name="download_logs",
    )
    ip = models.GenericIPAddressField()
    extra_data = models.JSONField(null=True)
    user_agent = models.TextField()
    repository = models.ForeignKey(
        Repository, on_delete=models.CASCADE, related_name="download_logs"
    )
    repository_version = models.ForeignKey(
        RepositoryVersion, null=True, on_delete=models.SET_NULL, related_name="download_logs"
    )


class RoleRemote(Remote):
    """
    A Remote for Ansible content.
    """

    TYPE = "role"

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"


class CollectionRemote(Remote):
    """
    A Remote for Collection content.
    """

    TYPE = "collection"

    requirements_file = models.TextField(null=True)
    auth_url = models.CharField(null=True, max_length=255)
    token = EncryptedTextField(null=True)
    sync_dependencies = models.BooleanField(default=True)
    signed_only = models.BooleanField(default=False)

    @property
    def download_factory(self):
        """
        Return the DownloaderFactory which can be used to generate asyncio capable downloaders.

        Upon first access, the DownloaderFactory is instantiated and saved internally.

        Plugin writers are expected to override when additional configuration of the
        DownloaderFactory is needed.

        Returns:
            DownloadFactory: The instantiated DownloaderFactory to be used by
                get_downloader()

        """
        try:
            return self._download_factory
        except AttributeError:
            self._download_factory = AnsibleDownloaderFactory(self)
            return self._download_factory

    @hook(
        AFTER_UPDATE,
        when_any=["url", "requirements_file", "sync_dependencies", "signed_only"],
        has_changed=True,
    )
    def _reset_repository_last_synced_metadata_time(self):
        AnsibleRepository.objects.filter(
            remote_id=self.pk, last_synced_metadata_time__isnull=False
        ).update(last_synced_metadata_time=None)

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"


class GitRemote(Remote):
    """
    A Remote for Collection content hosted in Git repositories.
    """

    TYPE = "git"

    metadata_only = models.BooleanField(default=False)
    git_ref = models.TextField()

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"


class AnsibleCollectionDeprecated(Content):
    """
    A model that represents if a Collection is `deprecated` for a given RepositoryVersion.
    """

    TYPE = "collection_deprecation"

    namespace = models.CharField(max_length=64, editable=False)
    name = models.CharField(max_length=64, editable=False)

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"
        unique_together = ("namespace", "name")


class AnsibleRepository(Repository):
    """
    Repository for "ansible" content.

    Fields:

        last_synced_metadata_time (models.DateTimeField): Last synced metadata time.
    """

    TYPE = "ansible"
    CONTENT_TYPES = [
        Role,
        CollectionVersion,
        AnsibleCollectionDeprecated,
        CollectionVersionSignature,
    ]
    REMOTE_TYPES = [RoleRemote, CollectionRemote]

    last_synced_metadata_time = models.DateTimeField(null=True)
    gpgkey = models.TextField(null=True)

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"

        permissions = (("modify_ansible_repo_content", "Can modify ansible repository content"),)

    def finalize_new_version(self, new_version):
        """Finalize repo version."""
        removed_collection_versions = new_version.removed(
            base_version=new_version.base_version
        ).filter(pulp_type=CollectionVersion.get_pulp_type())

        # Remove any deprecated and signature content associated with the removed collection
        # versions
        for version in removed_collection_versions:
            version = version.cast()

            signatures = new_version.get_content(
                content_qs=CollectionVersionSignature.objects.filter(signed_collection=version)
            )
            new_version.remove_content(signatures)

            other_collection_versions = new_version.get_content(
                content_qs=CollectionVersion.objects.filter(collection=version.collection)
            )

            # AnsibleCollectionDeprecated applies to all collection versions in a repository,
            # so only remove it if there are no more collection versions for the specified
            # collection present.
            if not other_collection_versions.exists():
                deprecations = new_version.get_content(
                    content_qs=AnsibleCollectionDeprecated.objects.filter(
                        namespace=version.namespace, name=version.name
                    )
                )

                new_version.remove_content(deprecations)

    @hook(BEFORE_UPDATE, when="remote", has_changed=True)
    def _reset_repository_last_synced_metadata_time(self):
        self.last_synced_metadata_time = None


class AnsibleDistribution(Distribution):
    """
    A Distribution for Ansible content.
    """

    TYPE = "ansible"

    class Meta:
        default_related_name = "%(app_label)s_%(model_name)s"
```