Pulp Container Support

Release 1.1.0.dev

Jan 22, 2020

Contents

The `pulp_container` plugin extends pulpcore to support hosting container images and container metadata, supporting `docker pull` and `podman pull`.

If you are just getting started, we recommend getting to know the *basic workflows*.

CHAPTER 1

Features

- *Mirror* container image repositories hosted on Docker-hub, Google Container Registry, Quay.io, etc
- *Create Versioned Repositories* so every operation is a restorable snapshot
- *Download content on-demand* when requested by clients to reduce disk space
- *Perform docker/podman pull* from a container distribution served by Pulp
- Curate container images by *whitelisting* what is mirrored from an external repository.
- Curate container images by creating repository versions with *a specific set* of images.
- De-duplication of all saved content

CHAPTER 2

How to use these docs

The documentation here should be considered **the primary documentation for managing container related content**. All relevent workflows are covered here, with references to some pulpcore supplemental docs. Users may also find pulpcore's conceptual docs useful.

This documentation falls into two main categories:

1. *Workflows* show the **major features** of the contaianer plugin, with links to reference docs.
2. REST API Docs are automatically generated and provide more detailed information for each **minor feature**, including all fields and options.

CHAPTER 3

Container Workflows

3.1 User Setup

3.1.1 Ansible Installer (Recommended)

We recommend that you install *pulpcore* and *pulp-container* together using the Ansible installer. If you install this way, pulpcore installation and all the following steps will be done for you.

3.1.2 Install `pulpcore`

Follow the installation instructions provided with pulpcore.

3.1.3 Install plugin

This document assumes that you have installed pulpcore into a the virtual environment `pulpvenv`.

Users should install from **either** PyPI or source.

From PyPI

```
sudo -u pulp -i
source ~/pulpvenv/bin/activate
pip install pulp-container
```

Install `pulp_container` from source

```
sudo -u pulp -i
source ~/pulpvenv/bin/activate
cd pulp_container
pip install -e .
```

3.1.4 Make and Run Migrations

```
django-admin migrate container
```

3.1.5 Configure Required Settings

The plugin expects to have defined additional settings. These settings are required if a user wants to use the token authentication while serving content, see *Registry Token Authentication*.

3.1.6 Run Services

```
django-admin runserver 24817
gunicorn pulpcore.content:server --bind 'localhost:24816' --worker-class 'aiohttp.
↪GunicornWebWorker' -w 2
sudo systemctl restart pulpcore-resource-manager
sudo systemctl restart pulpcore-worker@1
sudo systemctl restart pulpcore-worker@2
```

3.1.7 Enable OCI Container Image building

Pulp container plugin can be used to build an OCI format image from a Containerfile. The plugin uses buildah to build the container image. Buildah 1.11+ must be installed on the same machine that is running pulpcore-worker processes.

The systemd unit file for pulpcore-worker processes needs to add */usr/bin/* to the *PATH*. The user which pulpcore-worker runs as needs to be able to sudo without a password.

3.2 Workflows

If you have not yet installed the `pulp_container` plugin on your Pulp installation, please follow our *User Setup*. These documents will assume you have the environment installed and ready to go.

3.2.1 Recommended Tools

httpie: The REST API examples here use httpie to perform the requests. The `httpie` commands below assume that the user executing the commands has a `.netrc` file in the home directory. The `.netrc` should have the following configuration:

```
machine localhost
login admin
password admin
```

One should observe that `httpie` uses the configuration retrieved from `.netrc` by default. Due to this, a custom Authorization header is always overwritten by the Basic Authorization with the provided login and password. In order to send HTTP requests which contain JWT Authorization headers, ensure yourself that the plugin JWTAuth plugin was already installed.

If you configured the `admin` user with a different password, adjust the configuration accordingly. If you prefer to specify the username and password with each request, please see `httpie` documentation on how to do that.

jq: This documentation makes use of the jq library to parse the json received from requests, in order to get the unique urls generated when objects are created. To follow this documentation as-is please install the jq library with:

```
$ sudo dnf install jq
```

environtoment variables: To make these workflows copy/pastable, we make use of environment variables. The first variable to set is the hostname and port:

```
$ export BASE_ADDR=http://<hostname>:24817
```

3.2.2 Container Workflows

Synchronize a Repository

Users can populate their repositories with content from an external source like Docker Hub by syncing their repository.

Create a Repository

```
#!/usr/bin/env bash
export REPO_NAME=$(head /dev/urandom | tr -dc a-z | head -c5)

echo "Creating a new repository named $REPO_NAME."
export REPO_HREF=$(http POST $BASE_ADDR/pulp/api/v3/repositories/container/container/␣
↪name=$REPO_NAME \
  | jq -r '.pulp_href')

echo "Inspecting repository."
http $BASE_ADDR$REPO_HREF
```

Repository GET Response:

```
{
    "pulp_created": "2019-09-05T14:29:43.424822Z",
    "pulp_href": "/pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-
↪8434-0fe9d94c8053/",
    "latest_version_href": null,
    "versions_href": "/pulp/api/v3/repositories/container/container/ffcf03266-f0e4-
↪4497-8434-0fe9d94c8053/versions/",
    "description": null,
    "name": "codzo"
}
```

Reference (pulpcore): Repository API Usage

Create a Remote

Creating a remote object informs Pulp about an external content source. In this case, we will be using Docker Hub, but `pulp-container` remotes can be anything that implements the registry API, including *quay*, *google container registry*, or even another instance of Pulp.

Note: Container plugin supports both Docker and OCI media types.

Note: Use whitelist_tags when a specific set of tags are needed to be mirrored instead of the whole repo.

```
#!/usr/bin/env bash
echo "Creating a remote that points to an external source of container images."
http POST $BASE_ADDR/pulp/api/v3/remotes/container/container/ \
    name='my-hello-repo' \
    url='https://registry-1.docker.io' \
    upstream_name='pulp/test-fixture-1'

echo "Export an environment variable for the new remote URI."
export REMOTE_HREF=$(http $BASE_ADDR/pulp/api/v3/remotes/container/container/ \
    | jq -r '.results[] | select(.name == "my-hello-repo") | .pulp_href')

echo "Inspecting new Remote."
http $BASE_ADDR$REMOTE_HREF
```

Remote GET Response:

```
{
    "pulp_created": "2019-09-05T14:29:44.267406Z",
    "pulp_href": "/pulp/api/v3/remotes/container/container/1cc699b7-24fd-4944-bde7-
↪86aed8ac12fa/",
    "pulp_last_updated": "2019-09-05T14:29:44.267428Z",
    "download_concurrency": 20,
    "name": "my-hello-repo",
    "policy": "immediate",
    "proxy_url": null,
    "ssl_ca_certificate": null,
    "ssl_client_certificate": null,
    "ssl_client_key": null,
    "ssl_validation": true,
    "upstream_name": "library/hello-world",
    "url": "https://registry-1.docker.io",
    "whitelist_tags": null
}
```

Reference: Container Remote Usage

Sync repository using a Remote

Use the remote object to kick off a synchronize task by specifying the repository to sync with. You are telling pulp to fetch content from the remote and add to the repository.

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

echo "Create a task to sync the repository using the remote."
export TASK_HREF=$(http POST $BASE_ADDR$REPO_HREF'sync/' remote=$REMOTE_HREF␣
↪mirror=False \
  | jq -r '.task')

# Poll the task (here we use a function defined in docs/_scripts/base.sh)
wait_until_task_finished $BASE_ADDR$TASK_HREF

# After the task is complete, it gives us a new repository version
echo "Set REPOVERSION_HREF from finished task."
export REPOVERSION_HREF=$(http $BASE_ADDR$TASK_HREF| jq -r '.created_resources | first
↪')

echo "Inspecting RepositoryVersion."
http $BASE_ADDR$REPOVERSION_HREF
```

Reference: Container Sync Usage

Repository Version GET Response (when complete):

```
{
    "pulp_created": "2019-09-05T14:29:45.563089Z",
    "pulp_href": "/pulp/api/v3/repositories/container/container/ffcf03266-f0e4-4497-
↪8434-0fe9d94c8053/versions/1/",
    "base_version": null,
    "content_summary": {
        "added": {
            "container.blob": {
                "count": 31,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-8434-
↪0fe9d94c8053/versions/1/"
            },
            "container.manifest": {
                "count": 21,
                "href": "/pulp/api/v3/content/container/manifests/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-8434-
↪0fe9d94c8053/versions/1/"
            },
            "container.tag": {
                "count": 8,
                "href": "/pulp/api/v3/content/container/tags/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-8434-
↪0fe9d94c8053/versions/1/"
            }
        },
        "present": {
            "container.blob": {
                "count": 31,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version=/
↪pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-8434-0fe9d94c8053/
↪versions/1/"
            },
            "container.manifest": {
                "count": 21,
                "href": "/pulp/api/v3/content/container/manifests/?repository_
↪version=/pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-8434-
↪0fe9d94c8053/versions/1/"
```

(continues on next page)

(continued from previous page)

```
            },
            "container.tag": {
                "count": 8,
                "href": "/pulp/api/v3/content/container/tags/?repository_version=/
↪pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-8434-0fe9d94c8053/
↪versions/1/"
            }
        },
        "removed": {}
    },
    "number": 1
}
```

Reference (pulpcore): Repository Version API Usage

Host and Consume a Container Repository

This section assumes that you have a repository with content in it. To do this, see the *Synchronize a Repository* documentation.

Create a Container Distribution to serve your Repository Version

Container Distributions can be used to serve the Container registry API containing the content in a repository's latest version or a specified repository version.

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

export DIST_NAME='testing-hello'
export DIST_BASE_PATH='test'

# Distributions are created asynchronously.
echo "Creating distribution \
  (name=$DIST_NAME, base_path=$DIST_BASE_PATH repository=$REPO_HREF)."
export TASK_HREF=$(http POST $BASE_ADDR/pulp/api/v3/distributions/container/container/
↪ \
  name=$DIST_NAME \
  base_path=$DIST_BASE_PATH \
  repository=$REPO_HREF | jq -r '.task')

# Poll the task (here we use a function defined in docs/_scripts/base.sh)
wait_until_task_finished $BASE_ADDR$TASK_HREF

echo "Setting DISTRIBUTION_HREF from the completed task."
# DISTRIBUTION_HREF is the pulp-api HREF, not the content app href
export DISTRIBUTION_HREF=$(http $BASE_ADDR$TASK_HREF | jq -r '.created_resources |␣
↪first')

echo "Inspecting Distribution."
http $BASE_ADDR$DISTRIBUTION_HREF
```

Response:

```
{
    "pulp_created": "2019-09-05T14:29:51.742086Z",
```

(continues on next page)

(continued from previous page)

```
    "pulp_href": "/pulp/api/v3/distributions/container/container/1b461dac-0839-4049-
↪aa8f-92f8e8f7f034/",
    "base_path": "test",
    "content_guard": null,
    "name": "testing-hello",
    "registry_path": "localhost:24816/test",
    "repository": "/pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-
↪8434-0fe9d94c8053/",
    "repository_version": null
}
```

Reference: Container Distribution Usage

Pull and run an image from Pulp

Once a distribution is configured to host a repository with Container images in it, that content can be consumed by container clients.

Podman

```
$ podman pull localhost:24816/foo
```

If SSL has not been setup for your Pulp, configure podman to work with the insecure registry:

Edit the file `/etc/containers/registries.conf.` and add:

```
[registries.insecure]
registries = ['localhost:24816']
```

More info: https://www.projectatomic.io/blog/2018/05/podman-tls/

Docker

If SSL has not been setup for your Pulp, configure docker to work with the insecure registry:

Edit the file `/etc/docker/daemon.json` and add:

```
{
    "insecure-registries" : ["localhost:24816"]

}
```

More info: https://docs.docker.com/registry/insecure/#deploy-a-plain-http-registry

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

CONTAINER_TAG='manifest_a'

echo "Setting REGISTRY_PATH, which can be used directly with the Docker Client."
export REGISTRY_PATH=$(http $BASE_ADDR$DISTRIBUTION_HREF | jq -r '.registry_path')

echo "Next we pull the image from pulp and run it."
echo "$REGISTRY_PATH:$CONTAINER_TAG"
sudo docker run $REGISTRY_PATH:$CONTAINER_TAG
```

Docker Output:

```
Unable to find image 'localhost:24816/test:latest' locally
Trying to pull repository localhost:24816/test ...
sha256:451ce787d12369c5df2a32c85e5a03d52cbcef6eb3586dd03075f3034f10adcd: Pulling from␣
↪localhost:24816/test
1b930d010525: Pull complete
Digest: sha256:451ce787d12369c5df2a32c85e5a03d52cbcef6eb3586dd03075f3034f10adcd
Status: Downloaded newer image for localhost:24816/test:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Listing Repositories

A registry may contain several repositories which hold collections of multiple images. Each repository is identified by its unique name. The list of names of all distributed repositories is made available through the `_catalog` endpoint.

For instance, let's assume that a new distribution of a repository with the name `bar` was recently created. Its name is now possible to fetch from the list of names of distributed repositories:

```
http :24816/v2/_catalog

HTTP/1.1 200 OK
Content-Length: 25
Content-Type: application/json; charset=utf-8
Date: Wed, 22 Jan 2020 09:27:16 GMT
Docker-Distribution-API-Version: registry/2.0
Server: Python/3.7 aiohttp/3.6.2

{
    "repositories": [
        "foo",
        "bar"
    ]
}
```

Note: For the sake of simplicity, there is missing a part that requires a user to authenticate via a Bearer token. The

token authentication is enabled by default and does not come pre-configured out of the box. An administrator needs to to set up the environment in advance to enable users to consume content with an authorized access. Lean more at *Registry Token Authentication*.

Manage Container Content in a Repository

There are multiple ways that users can manage Container content in repositories:

1. *Tag* or *Untag* Manifests in a repository.
2. Recursively *add* or *remove* Container content.
3. Copy *tags* or *manifests* from source repository.

Each of these workflows kicks off a task, and when the task is complete, a new repository version will have been created.

Tagging

Images are described by manifests. The procedure of an image tagging is related to manifests because of that. In pulp, it is required to specify a digest of a manifest in order to create a tag for the corresponding image.

Below is provided an example on how to tag an image within a repository. First, a digest of an existing manifest is selected. Then, a custom tag is applied to the corresponding manifest.

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

export TAG_NAME='custom_tag'
export MANIFEST_DIGEST=
↪'sha256:21e3caae28758329318c8a868a80daa37ad8851705155fc28767852c73d36af5'

echo "Tagging the manifest."
export TASK_URL=$(http POST $BASE_ADDR$REPO_HREF'tag/' \
  repository=$REPO_HREF tag=$TAG_NAME digest=$MANIFEST_DIGEST \
  | jq -r '.task')

wait_until_task_finished $BASE_ADDR$TASK_URL

echo "Getting a reference to a newly created tag."
export CREATED_TAG=$(http $BASE_ADDR$TASK_URL \
  | jq -r '.created_resources | .[] | select(test("content"))')

echo "Display properties of the created tag."
http $BASE_ADDR$CREATED_TAG
```

A new distribution can be created to include the newly created tag. This allows clients to pull the image with the applied tag.

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

export TAG_NAME='custom_tag'

export DIST_NAME='testing-tagging'
export DIST_BASE_PATH='tag'

echo "Publishing the latest repository."
```

(continues on next page)

(continued from previous page)

```
export TASK_URL=$(http POST $BASE_ADDR/pulp/api/v3/distributions/container/container/␣
↪\
  name=$DIST_NAME base_path=$DIST_BASE_PATH repository=$REPO_HREF \
  | jq -r '.task')

wait_until_task_finished $BASE_ADDR$TASK_URL

export DISTRIBUTION_HREF=$(http $BASE_ADDR$TASK_URL \
  | jq -r '.created_resources | first')
export REGISTRY_PATH=$(http $BASE_ADDR$DISTRIBUTION_HREF \
  | jq -r '.registry_path')

echo "Pulling ${REGISTRY_PATH}:${TAG_NAME}."
docker run $REGISTRY_PATH:$TAG_NAME
```

Each tag has to be unique within a repository to prevent ambiguity. When a user is trying to tag an image with a same name but with a different digest, the tag associated with the old manifest is going to be eliminated in a new repository version.

Reference: Container Tagging Usage

Untagging

An untagging is an inverse operation to the tagging. To remove a tag applied to an image, it is required to issue the following calls.

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

export TAG_NAME='custom_tag'

echo "Untagging a manifest which is labeled with ${TAG_NAME}"
export TASK_URL=$(http POST $BASE_ADDR$REPO_HREF'untag/' \
  repository=$REPO_HREF tag=$TAG_NAME \
  | jq -r '.task')

wait_until_task_finished $BASE_ADDR$TASK_URL

echo "Getting a reference to all removed tags."
export REPO_VERSION=$(http $BASE_ADDR$TASK_URL \
  | jq -r '.created_resources | first')
export REMOVED_TAGS=$(http $BASE_ADDR$REPO_VERSION \
  | jq -r '.content_summary | .removed | ."container.tag" | .href')

echo "List removed tags from the latest repository version."
http $BASE_ADDR$REMOVED_TAGS
```

Pulp will create a new repository version which will not contain the corresponding tag. The removed tag however still persists in a database. When a client tries to untag an image that was already untagged, a new repository version is created as well.

Reference: Container Untagging Usage

Recursively Add Content to a Repository

Any Container content can be added to a repository version with the recursive-add endpoint. Here, "recursive" means that the content will be added, as well as all related content.

Relations:

- Adding a **tag** will also add the tagged manifest and its related content.
- Adding a **manifest** (manifest list) will also add related manifests and their related content.
- Adding a **manifest** (not manifest list) will also add related blobs.

Note: Because tag names are unique within a repository version, adding a tag with a duplicate name will first remove the existing tag (non-recursively).

Begin by following the *Synchronize* workflow to start with a repository that has some content in it.

Next create a new repository that we can add content to.

```
#!/usr/bin/env bash
export DEST_REPO_NAME=$(head /dev/urandom | tr -dc a-z | head -c5)

echo "Create a second repository so we can add content to it."
export DEST_REPO_HREF=$(http POST $BASE_ADDR/pulp/api/v3/repositories/container/
↪container/ name=$DEST_REPO_NAME \
  | jq -r '.pulp_href')

echo "Inspect repository."
http $BASE_ADDR$DEST_REPO_HREF
```

Now we recursively add a tag to the destination repository.

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

echo "Retrieve the href of Tag manifest_a in the synced repository."
export TAG_HREF=$(http $BASE_ADDR'/pulp/api/v3/content/container/tags/?repository_
↪version='$REPOVERSION_HREF'&name=manifest_a' \
  | jq -r '.results | first | .pulp_href')

echo "Create a task to recursively add a tag to the repo."
export TASK_HREF=$(http POST $BASE_ADDR$REPO_HREF'add/' \
  content_units:="[\"$TAG_HREF\"]" \
  | jq -r '.task')

# Poll the task (here we use a function defined in docs/_scripts/base.sh)
wait_until_task_finished $BASE_ADDR$TASK_HREF

# After the task is complete, it gives us a new repository version
export ADDED_VERSION=$(http $BASE_ADDR$TASK_HREF| jq -r '.created_resources | first')

echo "Inspect RepositoryVersion."
http $BASE_ADDR$ADDED_VERSION
```

We have added our single tag, as well as the content necessary for that tag to function correctly when pulled by a client.

New Repository Version:

```
{
    "pulp_created": "2019-09-05T19:04:06.152589Z",
    "pulp_href": "/pulp/api/v3/repositories/container/container/ce642635-dd9b-423f-
↪82c4-86a150b9f5fe/versions/10/",
    "base_version": null,
    "content_summary": {
        "added": {
            "container.tag": {
                "count": 1,
                "href": "/pulp/api/v3/content/container/tags/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/ce642635-dd9b-423f-82c4-
↪86a150b9f5fe/versions/10/"
            }
        },
        "present": {
            "container.blob": {
                "count": 20,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version=/
↪pulp/api/v3/repositories/container/container/ce642635-dd9b-423f-82c4-86a150b9f5fe/
↪versions/10/"
            },
            "container.manifest": {
                "count": 10,
                "href": "/pulp/api/v3/content/container/manifests/?repository_
↪version=/pulp/api/v3/repositories/container/container/ce642635-dd9b-423f-82c4-
↪86a150b9f5fe/versions/10/"
            },
            "container.tag": {
                "count": 1,
                "href": "/pulp/api/v3/content/container/tags/?repository_version=/
↪pulp/api/v3/repositories/container/container/ce642635-dd9b-423f-82c4-86a150b9f5fe/
↪versions/10/"
            }
        },
        "removed": {
            "container.tag": {
                "count": 1,
                "href": "/pulp/api/v3/content/container/tags/?repository_version_
↪removed=/pulp/api/v3/repositories/container/container/ce642635-dd9b-423f-82c4-
↪86a150b9f5fe/versions/10/"
            }
        }
    },
    "number": 10
}
```


Note: Directly adding a manifest that happens to be tagged in another repo will **not** include its tags.

Reference: Container Recursive Add Usage

Recursively Remove Content from a Repository

Any Container content can be removed from a repository version with the recursive-remove endpoint. Recursive remove is symmetrical with recursive add, meaning that performing a recursive-add and a recursive-remove back-to-back with the same content will result in the original content set. If other operations (i.e. tagging) are done between

recursive-add and recursive remove, they can break the symmetry.

Removing a tag also removes the tagged_manifest and its related content, which is **new behavior with Pulp 3**. If you just want to remove the tag, but not the related content, use the *untagging workflow*.

Recursive remove **does not** remove content that is related to content that will stay in the repository. For example, if a manifest is tagged, the manifest cannot be removed from the repository– instead the tag should be removed.

See *relations*

Continuing from the *recursive add workflow*, we can remove the tag and the related content that is no longer needed.

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

echo "Create a task to recursively remove the same tag to the repo."
export TASK_HREF=$(http POST $BASE_ADDR$REPO_HREF'remove/' \
  content_units:="[\"$TAG_HREF\"]" \
  | jq -r '.task')

# Poll the task (here we use a function defined in docs/_scripts/base.sh)
wait_until_task_finished $BASE_ADDR$TASK_HREF

# After the task is complete, it gives us a new repository version
export REMOVED_VERSION=$(http $BASE_ADDR$TASK_HREF | jq -r '.created_resources | first
↪')

echo "Inspect RepositoryVersion."
http $BASE_ADDR$REMOVED_VERSION
```

Now we can see that the tag and related content that was added has now been removed, resulting in an empty repository.

New Repository Version:

```
{
    "pulp_created": "2019-09-10T13:25:44.078017Z",
    "pulp_href": "/pulp/api/v3/repositories/container/container/c2f67416-7200-4dcc-
↪9868-f320431aae20/versions/2/",
    "base_version": null,
    "content_summary": {
        "added": {},
        "present": {},
        "removed": {
            "container.blob": {
                "count": 20,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version_
↪removed=/pulp/api/v3/repositories/container/container/c2f67416-7200-4dcc-9868-
↪f320431aae20/versions/2/"
            },
            "container.manifest": {
                "count": 10,
                "href": "/pulp/api/v3/content/container/manifests/?repository_version_
↪removed=/pulp/api/v3/repositories/container/container/c2f67416-7200-4dcc-9868-
↪f320431aae20/versions/2/"
            },
            "container.tag": {
                "count": 1,
                "href": "/pulp/api/v3/content/container/tags/?repository_version_
↪removed=/pulp/api/v3/repositories/container/container/c2f67416-7200-4dcc-9868-
↪f320431aae20/versions/2/"
            }
```

(continues on next page)

(continued from previous page)

```
        }
    },
    "number": 2
}
```

Note: Users can remove all content from the repo by specifying '*' in the content_units

Reference: Container Recursive Remove Usage

Recursively Copy Tags from a Source Repository

Tags in one repository can be copied to another repository using the tag copy endpoint.

When no names are specified, all tags are recursively copied. If names are specified, only the matching tags are recursively copied.

If tag names being copied already exist in the destination repository, the conflicting tags are removed from the destination repository and the new tags are added. This action is not recursive, no manifests or blobs are removed.

Again we start with a new destination repository.

```
#!/usr/bin/env bash
export DEST_REPO_NAME=$(head /dev/urandom | tr -dc a-z | head -c5)

echo "Create a second repository so we can add content to it."
export DEST_REPO_HREF=$(http POST $BASE_ADDR/pulp/api/v3/repositories/container/
↪container/ name=$DEST_REPO_NAME \
  | jq -r '.pulp_href')

echo "Inspect repository."
http $BASE_ADDR$DEST_REPO_HREF
```

With copy (contrasted to recursive add) we do not need to retrieve the href of the tag. Rather, we can specify the tag by source repository and name.

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

echo "Create a task to copy a tag to the repo."
export TASK_HREF=$(http POST $BASE_ADDR$REPO_HREF'copy_tags/' \
  source_repository=$REPO_HREF \
  names:="[\"manifest_a\"]" \
  | jq -r '.task')

# Poll the task (here we use a function defined in docs/_scripts/base.sh)
wait_until_task_finished $BASE_ADDR$TASK_HREF

# After the task is complete, it gives us a new repository version
export TAG_COPY_VERSION=$(http $BASE_ADDR$TASK_HREF | jq -r '.created_resources |␣
↪first')

echo "Inspect RepositoryVersion."
http $BASE_ADDR$TAG_COPY_VERSION
```

New Repository Version:

```
{
    "pulp_created": "2019-09-10T13:42:12.572859Z",
    "pulp_href": "/pulp/api/v3/repositories/container/container/2b1c6d76-c369-4f31-
↪8eb8-9d5d92bb2346/versions/1/",
    "base_version": null,
    "content_summary": {
        "added": {
            "container.blob": {
                "count": 20,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/2b1c6d76-c369-4f31-8eb8-
↪9d5d92bb2346/versions/1/"
            },
            "container.manifest": {
                "count": 10,
                "href": "/pulp/api/v3/content/container/manifests/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/2b1c6d76-c369-4f31-8eb8-
↪9d5d92bb2346/versions/1/"
            },
            "container.tag": {
                "count": 1,
                "href": "/pulp/api/v3/content/container/tags/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/2b1c6d76-c369-4f31-8eb8-
↪9d5d92bb2346/versions/1/"
            }
        },
        "present": {
            "container.blob": {
                "count": 20,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version=/
↪pulp/api/v3/repositories/container/container/2b1c6d76-c369-4f31-8eb8-9d5d92bb2346/
↪versions/1/"
            },
            "container.manifest": {
                "count": 10,
                "href": "/pulp/api/v3/content/container/manifests/?repository_
↪version=/pulp/api/v3/repositories/container/container/2b1c6d76-c369-4f31-8eb8-
↪9d5d92bb2346/versions/1/"
            },
            "container.tag": {
                "count": 1,
                "href": "/pulp/api/v3/content/container/tags/?repository_version=/
↪pulp/api/v3/repositories/container/container/2b1c6d76-c369-4f31-8eb8-9d5d92bb2346/
↪versions/1/"
            }
        },
        "removed": {}
    },
    "number": 1
}
```


Reference: Container Copy Tags Usage

Recursively Copy Manifests from a Source Repository

Manifests in one repository can be copied to another repository using the manifest copy endpoint.

If digests are specified, only the manifests (and their recursively related content) will be added.

If media_types are specified, only manifests matching that media type (and their recursively related content) will be added. This allows users to copy only manifest lists, for example.

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

echo "Create a task to copy all manifests from source to destination repo."
export TASK_HREF=$(http POST $BASE_ADDR$REPO_HREF'copy_manifests/' \
  source_repository=$REPO_HREF \
  | jq -r '.task')

# Poll the task (here we use a function defined in docs/_scripts/base.sh)
wait_until_task_finished $BASE_ADDR$TASK_HREF

# After the task is complete, it gives us a new repository version
export MANIFEST_COPY_VERSION=$(http $BASE_ADDR$TASK_HREF | jq -r '.created_resources
↪| first')

echo "Inspect RepositoryVersion."
http $BASE_ADDR$MANIFEST_COPY_VERSION
```

New Repository Version:

```
{
    "pulp_created": "2019-09-20T13:53:04.907351Z",
    "pulp_href": "/pulp/api/v3/repositories/container/container/70450dfb-ae46-4061-
↪84e3-97eb71cf9414/versions/2/",
    "base_version": null,
    "content_summary": {
        "added": {
            "container.blob": {
                "count": 31,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/70450dfb-ae46-4061-84e3-
↪97eb71cf9414/versions/2/"
            },
            "container.manifest": {
                "count": 21,
                "href": "/pulp/api/v3/content/container/manifests/?repository_version_
↪added=/pulp/api/v3/repositories/container/container/70450dfb-ae46-4061-84e3-
↪97eb71cf9414/versions/2/"
            }
        },
        "present": {
            "container.blob": {
                "count": 31,
                "href": "/pulp/api/v3/content/container/blobs/?repository_version=/
↪pulp/api/v3/repositories/container/container/70450dfb-ae46-4061-84e3-97eb71cf9414/
↪versions/2/"
            },
            "container.manifest": {
                "count": 21,
                "href": "/pulp/api/v3/content/container/manifests/?repository_
↪version=/pulp/api/v3/repositories/container/container/70450dfb-ae46-4061-84e3-
↪97eb71cf9414/versions/2/"
            }
        },
```

(continues on next page)

(continued from previous page)

```
        "removed": {}
    },
    "number": 2
}
```

Reference: Container Copy Manifests Usage

Build an OCI image from a Containerfile

Warning: All container build APIs are tech preview in Pulp Container 1.1. Backwards compatibility when upgrading is not guaranteed.

This feature may not be available in all deployments due to permission problems. *buildah* needs to also be installed. The user running the pulp worker process needs to be able to use *sudo* without a password, though this limitation should be removed in the near future.

Users can add new images to a container repository by uploading a Containerfile. The syntax for Containerfile is the same as for a Dockerfile. The same REST API endpoint also accepts a JSON string that maps artifacts in Pulp to a filename. Any artifacts passed in are available inside the build container at */pulp_working_directory*.

Create a Repository

```
#!/usr/bin/env bash
export REPO_NAME=$(head /dev/urandom | tr -dc a-z | head -c5)

echo "Creating a new repository named $REPO_NAME."
export REPO_HREF=$(http POST $BASE_ADDR/pulp/api/v3/repositories/container/container/␣
↪name=$REPO_NAME \
  | jq -r '.pulp_href')

echo "Inspecting repository."
http $BASE_ADDR$REPO_HREF
```

Repository GET Response:

```
{
    "pulp_created": "2019-09-05T14:29:43.424822Z",
    "pulp_href": "/pulp/api/v3/repositories/container/container/fcf03266-f0e4-4497-
↪8434-0fe9d94c8053/",
    "latest_version_href": null,
    "versions_href": "/pulp/api/v3/repositories/container/container/ffcf03266-f0e4-
↪4497-8434-0fe9d94c8053/versions/",
    "description": null,
    "name": "codzo"
}
```

Create an Artifact

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

echo "Create a text file and upload it to Pulp"

echo 'Hello world!' > example.txt

export ARTIFACT_HREF=$(http --form POST http://localhost/pulp/api/v3/artifacts/ \
    file@./example.txt \
    | jq -r '.pulp_href')

echo "Inspecting new artifact."
http $BASE_ADDR$ARTIFACT_HREF
```

Artifact GET Response:

```
{
    "pulp_created": "2019-05-16T20:07:48.066089Z",
    "pulp_href": "/pulp/api/v3/artifacts/cff8078a-826f-4f7e-930d-422c2f134a07/",
    "file": "artifact/97/
↪144ab16c9aa0e6072d471d6aebe7c21083e21359137e676445bfeb4051ba25",
    "md5": "5148c996f375ed5aab94ef6993df90a0",
    "sha1": "a7bd2bcaf1d68505f3e8b2cfe3505d01b31db306",
    "sha224": "18a167922b68a3fb8f2d9a71fa78f9776f5402dce4b3d97d5cea2559",
    "sha256": "97144ab16c9aa0e6072d471d6aebe7c21083e21359137e676445bfeb4051ba25",
    "sha384":
↪"4cd006bfac7f2e41baa8c411536579b134daeb3ad666310d21463f384a7020360703fc5538b4eca724033498d514e144
↪",
    "sha512":
↪"e1aae6bbc6fd24cf890b82ffa824629518e6e93935935a0b7c008fbd9fa59f08aa32a7d8580b31a65b21caa0f48e737d8e
↪",
    "size": 11
}
```

Reference (pulpcore): Artifact API Usage

Create a Containerfile

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

echo "Create a Containerfile that expects foo/bar/example.txt inside /pulp_working_
↪directory."

echo 'FROM centos:7

# Copy a file using RUN statement (absolute path required)
RUN cp /pulp_working_directory/foo/bar/example.txt /

# Copy a file using COPY statement (relative path can be used)
COPY foo/bar/example.txt /inside-image.txt

# Print the content of the file when the container starts
CMD ["cat", "/inside-image.txt"]' >> Containerfile
```

Build an OCI image

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

echo "Create a task that will build a container image from a Containerfile."

export TASK_HREF=$(http --form POST :$REPO_HREF'build_image/' containerfile@./
↪Containerfile \
artifacts="{\"$ARTIFACT_HREF\": \"foo/bar/example.txt\"}"  | jq -r '.task')

# Poll the task (here we use a function defined in docs/_scripts/base.sh)
wait_until_task_finished $BASE_ADDR$TASK_HREF

# After the task is complete, it gives us a new repository version
echo "Set REPOVERSION_HREF from finished task."
export REPOVERSION_HREF=$(http $BASE_ADDR$TASK_HREF| jq -r '.created_resources | first
↪')

echo "Inspecting RepositoryVersion."
http $BASE_ADDR$REPOVERSION_HREF
```

Registry Token Authentication

Pulp registry supports the token authentication. This enables users to pull content with an authorized access. A token server grants access based on the user's privileges and current scope.

The feature is enabled by default. However, it is possible to disable it from the settings by declaring `TOKEN_AUTH_DISABLED=True`.

The token authentication requires users also to define the following settings:

- **A fully qualified domain name of a token server with an associated port number**. The token server is responsible for generating Bearer tokens. Append the constant `TOKEN_SERVER` to the settings file `pulp_container/app/settings.py`.
- **A token signature algorithm**. A particular signature algorithm can be chosen only from the list of supported algorithms. Pulp uses exclusively asymmetric cryptography to sign and validate tokens. Therefore, it is possible only to choose from the algorithms, such as ES256, RS256, or PS256. Append the the constant `TOKEN_SIGNATURE_ALGORITHM` with a selected algorithm to the settings file.
- **Paths to secure keys**. These keys are going to be used for a signing and validation of tokens. Remember that the keys have to be specified in the **PEM format**. To generate keys, one could use the openssl utility. In the following example, the utility is used to generate keys with the algorithm ES256.

  1. Generate a private key:

     ```
     $ openssl ecparam -genkey -name prime256v1 -noout -out /tmp/private_key.pem
     ```

  2. Check if the generated private key has the proposed permissions:
     - mode: 600
     - owner: pulp (the account that pulp runs under)
     - group: pulp (the group of the account that pulp runs under)
  3. Generate a public key out of the private key:

```
$ openssl ec -in /tmp/private_key.pem -pubout -out /tmp/public_key.pem
```

4. Check if the generated public key has the proposed permissions:
   - mode: 644
   - owner: pulp (the account that pulp runs under)
   - group: pulp (the group of the account that pulp runs under)

Below is provided an example of the settings file:

```
TOKEN_SERVER = "http://localhost:24816/token"
TOKEN_SIGNATURE_ALGORITHM = 'ES256'
PUBLIC_KEY_PATH = '/tmp/public_key.pem'
PRIVATE_KEY_PATH = '/tmp/private_key.pem'
```

To learn more about Pulp settings, take a look at Configuration.

Restart Pulp services in order to reload the updated settings. Pulp will fetch a domain for the token server and will initialize all handlers according to that. Check if the token authentication was successfully configured by initiating the following set of commands in your environment:

```
$ http 'http://localhost:24816/v2/'

HTTP/1.1 401 Access to the requested resource is not authorized. A provided Bearer␣
↪token is invalid.
Content-Length: 92
Content-Type: text/plain; charset=utf-8
Date: Mon, 14 Oct 2019 16:46:48 GMT
Docker-Distribution-API-Version: registry/2.0
Server: Python/3.7 aiohttp/3.6.1
Www-Authenticate: Bearer realm="http://localhost:24816/token",service="localhost:24816
↪"

401: Access to the requested resource is not authorized. A provided Bearer token is␣
↪invalid.
```

Send a request to a specified realm:

```
$ http 'http://localhost:24816/token?service=localhost:24816'

HTTP/1.1 200 OK
Content-Length: 566
Content-Type: application/json; charset=utf-8
Date: Mon, 14 Oct 2019 16:47:33 GMT
Server: Python/3.7 aiohttp/3.6.1

{
    "expires_in": 300,
    "issued_at": "2019-10-14T16:47:33.107118Z",
    "token":
↪"eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiIsImtpZCI6IkhBM1Q6SVlSUjpHUTNUOklPTEM6TVE0RzpFT0xDOkdGUVQ6QVpUR
↪eyJhY2Nlc3MiOlt7InR5cGUiOiIiLCJuYW1lIjoiIiwiYWN0aW9ucyI6W119XSwiYXVkIjoibG9jYWxob3N0OjI0ODE2IiwiZXh
↪pirj8yhbjYnldxmZ-jIZ72VJrzxkAnwLXLu1ND9QAL-kl3gZrvPbp98w2xdhEoQ_
↪7WEka4veb6uU5ZzmD87X1Q"
}
```

Use the generated token to access the root again:

```
$ http 'localhost:24816/v2/' --auth-type=jwt --auth=
↪"eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NiIsImtpZCI6IkhBM1Q6SVlSUjpHUTNUOklPTEM6TVE0RzpFT0xDOkdGUVQ6QVpUR
↪eyJhY2Nlc3MiOlt7InR5cGUiOiIiLCJuYW1lIjoiIiwiYWN0aW9ucyI6W119XSwiYXVkIjoibG9jYWxob3N0OjI0ODE2IiwiZXh
↪pirj8yhbjYnldxmZ-jIZ72VJrzxkAnwLXLu1ND9QAL-kl3gZrvPbp98w2xdhEoQ_
↪7WEka4veb6uU5ZzmD87X1Q"

HTTP/1.1 200 OK
Content-Length: 2
Content-Type: application/json; charset=utf-8
Date: Mon, 14 Oct 2019 16:50:26 GMT
Docker-Distribution-API-Version: registry/2.0
Server: Python/3.7 aiohttp/3.6.1

{}
```

After performing multiple HTTP requests, the root responded with a default value `{}`. Received token can be used to access all endpoints within the requested scope too.

Regular container engines, like docker, or podman, can take advantage of the token authentication. The authentication is handled by the engines as shown before.

```
podman pull localhost:24816/foo/bar
```

3.3 REST API

Pulpcore Reference: pulpcore REST documentation.

3.3.1 Pulp Container Endpoints

REST API Reference container REST documentation

3.4 Changelog

3.4.1 1.0.0 (2019-12-12)

Features

- As a user, I can remove all repository container content with ["*"] #5756
- Enable users to disable the token authentication from the settings #5796
- As a user I can manage images in OCI format. #5816

Bugfixes

- Allow users to provide fully qualified domain name of a token server with an associated port number #5779

Improved Documentation

- Add note about access permissions for private and public keys #5778

Misc

- #4592, #5701, #5757, #5780, #5830

3.4.2 1.0.0rc1 (2019-11-18)

Features

- No duplicated content can be present in a repository version. #3541
- Convert manifests of the format schema 2 to schema 1 #4244
- Add support for pulling content using token authentication #4938
- Store whitelisted tags in a list instead of CSV string #5515
- Make repositories "typed". Repositories now live at a detail endpoint. Sync is performed by POSTing to {repo_href}/sync/ remote={remote_href}. #5625
- Added v2s2 to v2s1 converter. #5635

Bugfixes

- Fix using specified proxy for downloads. #5637

Improved Documentation

- Change the prefix of Pulp services from pulp-* to pulpcore-* #4554

Deprecations and Removals

- Change *_type* to *pulp_type* #5454
- Change *_id*, *_created*, *_last_updated*, *_href* to *pulp_id*, *pulp_created*, *pulp_last_updated*, *pulp_href* #5457
- Remove "_" from *_versions_href*, *_latest_version_href* #5548
- Removing base field: *_type* . #5550
- Sync is no longer available at the {remote_href}/sync/ repository={repo_href} endpoint. Instead, use POST {repo_href}/sync/ remote={remote_href}.

  Creating / listing / editing / deleting Container repositories is now performed on /pulp/api/v3/repositories/container/container/ instead of /pulp/api/v3/repositories/. Only Container content can be present in a Container repository, and only a Container repository can hold Container content. #5625

Misc

- #3308, #5580, #5690

3.4.3 4.0.0b7 (2019-10-02)

Bugfixes

- Fix a bug that allowed arbitrary url prefixes for custom endpoints. #5486
- Add Docker-Distribution-API-Version header among response headers. #5527

Misc

- #5470

3.4.4 4.0.0b6 (2019-09-05)

Features

- Add endpoint to recursively copy manifests from a source repository to a destination repository. #3403
- Add endpoint to recursively add docker content to a repository. #3405
- As a user I can sync from a docker repo published by Pulp2/Pulp3. #4737
- Add support for tagging and untagging manifests via an additional endpoint #4934
- Add endpoint for copying all tags from a source repository, or specific tags by name. #4947
- Add ability to filter Manifests and ManifestTags by media_type and digest #5033
- Add ability to filter Manifests, ManifestTags and Blobs by multiple media_types #5157
- Add endpoint to recursively remove docker content from a repository. #5179

Bugfixes

- Allow Accept header to send multiple values. #5211
- Populate ManifestListManifest thru table during sync. #5235
- Fixed a problem where repeated syncs created invalid orphaned tags. #5252

Misc

- #4681, #5213, #5218

3.4.5 4.0.0b5 (2019-07-04)

Bugfixes

- Add 'Docker-Content-Digest' header to the response headers. #4646
- Allow docker remote whitelist_tags to be unset to null. #5017
- Remove schema1 manifest signature when calculating its digest. #5037

Improved Documentation

- Switch to using towncrier for better release notes. #4875
- Add an example to the whitelist_tag help text #4994
- Add list of features to the docker landing page. #5030

Misc

- #4572, #4994, #5014

4.0.0b4

- Enable sync from registries that use basic auth or have private repos
- Enable on_demand or streamed sync
- Enable whitelist tags specification when syncing
- Compatibility with pulpcore-plugin-0.1.0rc2

Comprehensive list of changes and bugfixes for beta 4.

4.0.0b3

- Enable sync from gcr and quay registries
- Enable support to handle pagination for tags/list endpoint during sync
- Enable support to manage a docker image that has manifest schema v2s1
- Enable docker distribution to serve directly latest repository version

Comprehensive list of changes and bugfixes for beta 3.

4.0.0b2

- Compatibility with pulpcore-plugin-0.1.0rc1
- Performance improvements and bug fixes
- Add support for syncing repo with foreign layers
- Change sync pipeline to use Futures to handle nested content
- Make Docker distributions asyncronous
- Add support to create publication directly

4.0.0b1

- Add support for basic sync of a docker repo form a V2Registry
- Add support for docker/podman pull from a docker distbution served by Pulp

3.5 Contributing

To contribute to the `pulp_container` package follow this process:

1. Clone the GitHub repo
2. Make a change
3. Make sure all tests passed
4. Add a file into CHANGES folder (Changelog update).
5. Commit changes to own `pulp_container` clone
6. Make pull request from github page for your clone against master branch

3.5.1 Changelog update

The CHANGES.rst file is managed using the towncrier tool and all non trivial changes must be accompanied by a news entry.

To add an entry to the news file, you first need an issue in pulp.plan.io describing the change you want to make. Once you have an issue, take its number and create a file inside of the `CHANGES/` directory named after that issue number with an extension of .feature, .bugfix, .doc, .removal, or .misc. So if your issue is 3543 and it fixes a bug, you would create the file `CHANGES/3543.bugfix`.

PRs can span multiple categories by creating multiple files (for instance, if you added a feature and deprecated an old feature at the same time, you would create CHANGES/NNNN.feature and CHANGES/NNNN.removal). Likewise if a PR touches multiple issues/PRs you may create a file for each of them with the exact same contents and Towncrier will deduplicate them.

The contents of this file are reStructuredText formatted text that will be used as the content of the news file entry. You do not need to reference the issue or PR numbers here as towncrier will automatically add a reference to all of the affected issues when rendering the news file.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search