Lavatory Documentation

Gogo DevOps

Sep 30, 2019

Contents

CHAPTER 1

Contents

1.1 Getting Started

1.1.1 Authentication

Lavatory looks for several environment variables in order to authenticate:

`ARTIFACTORY_URL` - Base URL to use for Artifactory connections

`ARTIFACTORY_USERNAME` - Username to Artifactory

`ARTIFACTORY_PASSWORD` - Password for Artifactory

These will be loaded in at the beginning of a run and raise an exception if these environment variables are missing.

1.1.2 Purging Artifacts

1.1.2.1 Creating a Basic Policy

For this documentation lets assume a repository named `yum-local`. In a new directory, outside of Lavatory, create `yum_local.py`. This will be a retention policy that only impacts the `yum-local` repository.

In `yum_local.py` lets create a basic policy:

```
def purgelist(artifactory):
    """Policy to purge all artifacts older than 120 days"""
    purgable = artifactory.time_based_retention(keep_days=120)
    return purgable
```

The layout of the policy will look similar to

```
[root@localhost /]# tree path/
path
`-- to
```

(continues on next page)

(continued from previous page)

```
    `-- policies
        `-- yum_local.py
```

1.1.2.2 Running Lavatory

To test the policy you just created you can run `lavatory purge --policies-path=/path/to/policies --repo yum-local`

Below are all the options for the `purge` command:

```
$ lavatory purge --help
Usage: lavatory purge [OPTIONS]

  Deletes artifacts based on retention policies

Options:
  --policies-path TEXT      Path to extra policies directory
  --dryrun / --nodryrun     Dryrun does not delete any artifacts. On by
                            default
  --default / --no-default  If false, does not apply default policy
  --repo TEXT               Name of specific repository to run against. Can
                            use --repo multiple times. If not provided, uses
                            all repos.
  --help                    Show this message and exit.
```

If you want to run Lavatory against a specific repository, you can use `--repo <repo_name>`. You can specify `--repo` as multiple times to run against multiple repos. If `--repo` is not provided, Lavatory will run against all repos in Artifactory.

By default, Lavatory runs in drymode. Must include `--nodryrun` in order to actually delete Artifacts

1.1.3 CLI Help

You can run any Lavatory command with `--help` for assistance.

1.1.3.1 Verbosity

Adding `lavatory -v $command` will increase logging verbosity. You can add up to 5 `v` like `lavatory -vvvvv $command` for maximum verbosity.

1.2 Creating Retention Policies

Lavatory policies are implemented as Python plugins. Each policy is a `.py` file named after an Artifactory repository.

Each plugin represents one repository. The file name should match the repository name, replacing - with _.

For example, the repository `yum-local` should have a retention policy named `yum_local.py`

1.2.1 Anatomy of a Policy

Each policy needs to provide one function, `purgelist()`. This function takes one argument, `artifactory`, which is an instance of the `lavatory.utils.artifactory.Artifactory` class. This argument handles all communication with artifactory.

This function needs to return a list of artifacts to delete.

1.2.1.1 Docstring Description

The docstring following the function definition will be used as the policy description. This gets used in logging, as well as generating a list of all active policies.

1.2.1.2 Return Value

The return value of the policy should be a list of artifacts to delete. The artifacts are a dictionary that at minimum needs a `path` and `name` key. These keys are used by the delete function to remove the artifact.

`path`: path to artifact in the repository

`name`: Name of the artifact

Example Minimal Return:

```
[{ 'path': '222', 'name': 'Application-10.6.0-10.6.0.07-9cd3c33.iso'}]
```

This will delete artifact `<repo_name>/222/Application-10.6.0-10.6.0.07-9cd3c33.iso`

1.2.2 Policy Helpers

Below are some helper functions to assist in writing policies. These include easy ways to do time-based retention, count-based retention, or searching with AQL.

1.2.2.1 Time Based Retention

This policy will purge any artifact in the repository older than 120 days.

```
def purgelist(artifactory):
    """Policy to purge all artifacts older than 120 days"""
    purgable = artifactory.time_based_retention(keep_days=120)
    return purgable
```

`Artifactory.`**`time_based_retention`**(*keep_days=None*, *time_field='created'*, *item_type='file'*, *extra_aql=None*)

Retains artifacts based on number of days since creation.

extra_aql example: [{"@deployed": {"$match": "dev"}}, {"@deployed": {"$nmatch": "prod"}}] This would search for artifacts that were created after <keep_days> with property "deployed" equal to dev and not equal to prod.

Parameters

- **`keep_days`** (*`int`*) – Number of days to keep an artifact.
- **`time_field`** (*`str`*) – The field of time to look at (created, modified, stat.downloaded).
- **`item_type`** (*`str`*) – The item type to search for (file/folder/any).
- **`extra_aql`** (*`list`*) –

Returns List of artifacts matching retention policy

Return type list

1.2.2.2 Count Based Retention

This policy will retain the last 5 artifacts of each project in a repository.

```python
def purgelist(artifactory):
    """Policy to keep just the 5 most recent artifacts."""
    purgable = artifactory.count_based_retention(retention_count=5)
    return purgable
```

`Artifactory.`**`count_based_retention`**(*retention_count=None*, *project_depth=2*, *artifact_depth=3*, *item_type='folder'*, *extra_aql=None*)

Return all artifacts except the <count> most recent.

Parameters

- **`retention_count`** (*`int`*) – Number of artifacts to keep.
- **`project_depth`** (*`int`*) – how far down the Artifactory folder hierarchy to look for projects.
- **`artifact_depth`** (*`int`*) – how far down the Artifactory folder hierarchy to look for specific artifacts.
- **`item_type`** (*`str`*) – The item type to search for (file/folder/any).
- **`extra_aql`** (*`list`*) –

Returns List of all artifacts to delete.

Return type list

1.2.2.3 AQL Filtering

You can also use AQL to search for artifacts if you need more control than the count-based retention or time-based retention helps.

```
def purgelist(artifactory):
    """Policy to purge artifacts with deployed property of dev and not prod."""
    aql_terms = [{"@deployed": {"$match": "dev"}}, {"@deployed": {"$nmatch": "prod"}}]
    extra_fields = ['property.*']
    purgable = artifactory.filter(terms=aql_terms, fields=extra_fields, depth=None,␣
↪item_type="any")
    return purgable
```

All of the terms in `aql_terms` will be `ANDed` together and searched.

The above policy would use the below full AQL to search for artifacts.

```
items.find({"$and": [{"@deployed": {"$match": "dev"}},
            {"@deployed": {"$nmatch": "prod"}}, {"path": {"$nmatch": "*/repodata"}},
            {"repo": {"$eq": "yum-local"}}, {"type": {"$eq": "any"}}]}).include("stat",
↪ "property.*")
```

`Artifactory.`**`filter`**(*terms=None, depth=3, sort=None, offset=0, limit=0, fields=None, item_type='folder'*)

Get a subset of artifacts from the specified repo. This looks at the project level, but actually need to iterate lower at project level

This method does not use pagination. It assumes that this utility will be called on a repo sufficiently frequently that removing just the default n items will be enough.

Parameters

- **terms** (*list*) – an array of jql snippets that will be ANDed together
- **depth** (*int, optional*) – how far down the folder hierarchy to look
- **fields** (*list*) – Fields
- **sort** (*dict*) – How to sort Artifactory results
- **offset** (*int*) – how many items from the beginning of the list should be skipped (optional)
- **limit** (*int*) – the maximum number of entries to return (optional)
- **item_type** (*str*) – The item type to search for (file/folder/any).

Returns List of artifacts returned from query

Return type list

1.3 Example Policies

These are example policies for different retention use-cases

1.3.1 Keep last 120 days of artifacts

```
def purgelist(artifactory):
    """Policy to purge all artifacts older than 120 days"""
    purgable = artifactory.time_based_retention(keep_days=120)
    return purgable
```

1.3.2 Keep artifacts downloaded in the last 60 days

```
def purgelist(artifactory):
    """Policy to purge all artifacts not downloaded in last 60 days"""
    purgeable = artifactory.time_based_retention(keep_days=60, time_field='stat.
↪downloaded')
    return purgeable
```

1.3.3 Keep 5 most recent artifacts

```
def purgelist(artifactory):
    """Policy to keep just the 5 most recent artifacts."""
    purgeable = artifactory.count_based_retention(retention_count=5)
    return purgeable
```

1.3.4 Keep artifacts with specific properties

```
def purgelist(artifactory):
    """Policy to purge artifacts with deployed property of dev and not prod."""
    aql_terms = [{"@deployed": {"$match": "dev"}}, {"@deployed": {"$nmatch": "prod"}}]
    extra_fields = ['property.*']
    purgeable = artifactory.filter(terms=aql_terms, fields=extra_fields, depth=None,␣
↪item_type="any")
    return purgeable
```

1.3.5 Keep all artifacts

```
def purgelist(artifactory):
    """Keep artifacts indefinitely."""
    return []
```

1.3.6 Move artifacts to a different repo after 3 days.

```python
def purgelist(artifactory):
    """Moves artifacts to yum-local after 3 days."""
    movable = artifactory.time_based_retention(keep_days=3)
    artifactory.move_artifacts(artifacts=movable, dest_repository='yum-local')
    return []
```

1.3.7 More complicated examples

```python
def purgelist(artifactory):
    """Purges artifacts that have not been downloaded in the last month,
    That do not have a build.correlation_id,
    and are not in the */latest path."""

    docker_terms = [ { "stat.downloaded": { "$before": "1mo" }},
                    { "@build.correlation_ids": { "$nmatch": "*" }},
                    { "name": { "$match": "manifest.json" }},
                    { "path": { "$nmatch": "*/latest" }}
                ]
    purgeable = artifactory.filter(terms=docker_terms, depth=None, item_type="file")

    return purgeable
```

```python
def purgelist(artifactory):
    """If deployed to prod, keep artifact forever,
    if deployed to stage, keep 30 days,
    if deployed to dev, keep 21 days,
    if never deployed, keep 30 days."""

    not_deployed = [ { "@deployed": { "$nmatch": "*" }}]

    only_dev =  [ { "@deployed": { "$match": "*dev*"} },
                { "@deployed": {"$nmatch": "*prod*"} },
                { "@deployed": { "$nmatch": "*stage*"} }
                ]

    only_stage =  [ { "@deployed": { "$match": "*stage*"} },
                    { "@deployed": {"$nmatch": "*prod*"} },
                ]

    undeployed_purgeable = artifactory.time_based_retention(keep_days=30, extra_
↪aql=not_deployed)
    only_dev_purgeable = artifactory.time_based_retention(keep_days=21, extra_
↪aql=only_dev)
    only_stage_purgeable = artifactory.time_based_retention(keep_days=30, extra_
↪aql=only_dev)

    all_purgeable = undeployed_purgeable + only_dev_purgeable + only_stage_purgeable
    return all_purgeable
```

1.4 How To Create Releases

1.4.1 Setup

Add the following to ~/.pypirc file

```
[distutils]
index-servers =
    pypi

[pypi]
repository = https://pypi.python.org/pypi
username = username
password = xxxyyyzzz
```

1.4.2 Upload Release

When releasing a new version, the following needs to occur:

1. Ensure all test via tox pass
2. Add version Tag

   ```
   git tag -a v#.#.#
   git push --tags
   ```

3. Generate and upload the package

   ```
   python3 setup.py bdist_wheel upload -r pypi
   ```

1.5 src

1.5.1 lavatory package

1.5.1.1 Subpackages

1.5.1.1.1 lavatory.commands package

Submodules

lavatory.commands.policies module

List policies and descriptions

lavatory.commands.policies.**get_description**(*plugin_source*, *repository*)

Given a repository and plugin source, gets policy description.

Parameters

- **plugin_source** (*PluginBase*) – The source of plugins from PluginBase.
- **repository** (*str*) – The name fo the repository to get policy description.

Returns A dictionary of repo name and policy description

Return type dict

lavatory.commands.purge module

Purges artifacts.

`lavatory.commands.purge.`**`apply_purge_policies`**(*selected_repos*, *policies_path=None*, *dryrun=True*, *default=True*)

Sets up the plugins to find purgable artifacts and delete them.

Parameters

- **`selected_repos`** (*`list`*) – List of repos to run against.
- **`policies_path`** (*`str`*) – Path to extra policies
- **`dryrun`** (*`bool`*) – If true, will not actually delete artifacts.
- **`default`** (*`bool`*) – If true, applies default policy to repos with no specific policy.

`lavatory.commands.purge.`**`generate_purge_report`**(*purged_repos*, *before_purge_data*)

Generates a performance report based on deleted artifacts.

Parameters

- **`purged_repos`** (*`list`*) – List of repos that had policy applied.
- **`before_purge_data`** (*`dict`*) – Data on the state of Artifactory before purged artifacts

lavatory.commands.stats module

Statistics of the repo.

Module contents

1.5.1.1.2 lavatory.policies package

Submodules

lavatory.policies.default module

`lavatory.policies.default.`**`purgelist`**(*artifactory*)

Default Policy. Keeps the last 5 artifacts from each project

Module contents

1.5.1.1.3 lavatory.utils package

Submodules

lavatory.utils.artifactory module

Artifactory purger module.

class lavatory.utils.artifactory.**Artifactory**(*repo_name=None*)

Bases: `object`

Artifactory purger class.

count_based_retention(*retention_count=None*, *project_depth=2*, *artifact_depth=3*, *item_type='folder'*, *extra_aql=None*)

Return all artifacts except the <count> most recent.

Parameters

- **retention_count** (*int*) – Number of artifacts to keep.
- **project_depth** (*int*) – how far down the Artifactory folder hierarchy to look for projects.
- **artifact_depth** (*int*) – how far down the Artifactory folder hierarchy to look for specific artifacts.
- **item_type** (*str*) – The item type to search for (file/folder/any).
- **extra_aql** (*list*) –

Returns List of all artifacts to delete.

Return type list

filter(*terms=None*, *depth=3*, *sort=None*, *offset=0*, *limit=0*, *fields=None*, *item_type='folder'*)

Get a subset of artifacts from the specified repo. This looks at the project level, but actually need to iterate lower at project level

This method does not use pagination. It assumes that this utility will be called on a repo sufficiently frequently that removing just the default n items will be enough.

Parameters

- **terms** (*list*) – an array of jql snippets that will be ANDed together
- **depth** (*int, optional*) – how far down the folder hierarchy to look
- **fields** (*list*) – Fields
- **sort** (*dict*) – How to sort Artifactory results
- **offset** (*int*) – how many items from the beginning of the list should be skipped (optional)
- **limit** (*int*) – the maximum number of entries to return (optional)
- **item_type** (*str*) – The item type to search for (file/folder/any).

Returns List of artifacts returned from query

Return type list

get_all_repo_artifacts(*depth=None*, *item_type='file'*, *with_properties=True*)

returns all artifacts in a repo with metadata

Parameters

- **depth** (*int*) – How far down Artifactory folder to look. None will go to bottom of folder.

- **item_type** (*str*) – The item type to search for (file/folder/any).
- **with_properties** (*bool*) – Include artifact properties or not.

Returns List of all artifacts in a repository.

Return type list

get_artifact_properties (*artifact*)
Given an artifact, queries for properties from artifact URL

Parameters **artifact** (*dict*) – Dictionary of artifact info. Needs artifact['name'] and ['path'].

Returns Dictionary of all properties on specific artifact

Return type dict

move_artifacts (*artifacts=None*, *dest_repository=None*)
Moves a list of artifacts to dest_repository.

Parameters

- **artifacts** (*list*) – List of artifacts to move.
- **dest_repository** (*str*) – The name of the destination repo.

purge (*dry_run*, *artifacts*)
Purge artifacts from the specified repo.

Parameters

- **dry_run** (*bool*) – Dry run mode True/False
- **artifacts** (*list*) – Artifacts.

Returns Count purged.

Return type purged (int)

repos (*repo_type='local'*)
Return a dictionary of repos with basic info about each.

Parameters **repo_type** (*str*) – Type of repository to list. (local/virtual/cache/any)

Returns Dictionary of repos.

Return type repos (dict)

time_based_retention (*keep_days=None*, *time_field='created'*, *item_type='file'*, *extra_aql=None*)
Retains artifacts based on number of days since creation.

extra_aql example: [{"@deployed": {"$match": "dev"}}, {"@deployed": {"$nmatch": "prod"}}] This would search for artifacts that were created after <keep_days> with property "deployed" equal to dev and not equal to prod.

Parameters

- **keep_days** (*int*) – Number of days to keep an artifact.
- **time_field** (*str*) – The field of time to look at (created, modified, stat.downloaded).
- **item_type** (*str*) – The item type to search for (file/folder/any).
- **extra_aql** (*list*) –

Returns List of artifacts matching retention policy

Return type list

lavatory.utils.get_artifactory_info module

Helper method for getting artifactory information.

`lavatory.utils.get_artifactory_info.`**`get_artifactory_info`**(*repo_names=None*, *repo_type='local'*)

Get storage info from Artifactory.

Parameters

- **`repo_names`** (*`tuple, optional`*) – Name of artifactory repo.
- **`repo_type`** (*`str`*) – Type of artifactory repo.

Returns Dictionary of repo data. storage_info (dict): Storage information api call.

Return type keys (dict, optional)

`lavatory.utils.get_artifactory_info.`**`get_repos`**(*repo_names=None*, *repo_type='local'*)

`lavatory.utils.get_artifactory_info.`**`get_storage`**(*repo_names=None*, *repo_type=None*)

lavatory.utils.performance module

Performance comparison

`lavatory.utils.performance.`**`get_percentage`**(*old*, *new*)

Gets percentage from old and new values

Parameters

- **`old`** (*`num`*) – old value
- **`new`** (*`num`*) – new value

Returns Percentage, or zero if none

Return type number

`lavatory.utils.performance.`**`get_performance_report`**(*repo_name*, *old_info*, *new_info*)

compares retention policy performance, showing old amount of space and new.

Parameters

- **`repo_name`** (*`str`*) – Name of repository
- **`old_info`** (*`dict`*) – Metadata of repository before run
- **`new_info`** (*`dict`*) – Metadata of repository after run

lavatory.utils.setup_pluginbase module

`lavatory.utils.setup_pluginbase.`**`get_directory_path`**(*directory*)

Gets policy from plugin_source.

Parameters **`directory`** (*`Path`*) – Directory path

Returns The full expanded directory path

Return type full_path (Path)

`lavatory.utils.setup_pluginbase.`**`get_policy`**(*plugin_source*, *repository*, *default=True*)

Gets policy from plugin_source.

Parameters

- **`plugin_source`** (*`PluginBase`*) – the plugin source from loading plugin_base.
- **`repository`** (*`string`*) – Name of repository.
- **`default`** (*`bool`*) – If to load the default policy.

Returns The policy python module.

Return type policy (func)

`lavatory.utils.setup_pluginbase.`**`setup_pluginbase`**(*extra_policies_path=None*)

Sets up plugin base with default path and provided path

Parameters **`extra_policies_path`** (*`str`*) – Extra path to find plugins in

Returns PluginBase PluginSource for finding plugins

Return type PluginSource

Module contents

1.5.1.2 Submodules

1.5.1.3 lavatory.consts module

1.5.1.4 lavatory.credentials module

`lavatory.credentials.`**`load_credentials`**()

1.5.1.5 lavatory.exceptions module

Lavatory related custom exceptions

`exception` `lavatory.exceptions.`**`InvalidPoliciesDirectory`**

Bases: *`lavatory.exceptions.LavatoryError`*

Extra policies directory is invalid or missing

`exception` `lavatory.exceptions.`**`LavatoryError`**

Bases: `Exception`

Lavatory related error

`exception` `lavatory.exceptions.`**`MissingEnvironmentVariable`**(*missing_var*)

Bases: *`lavatory.exceptions.LavatoryError`*

Required environment variable is missing

1.5.1.6 Module contents

CHAPTER 2

Lavatory

Tooling to define repository specific retention policies in Artifactory. Allows highly customizable retention policies via Python plugins.

See Lavatory Documentation for the full docs.

2.1 Requirements

- Python 3.5+
- Artifactory user with API permissions

2.2 Authentication

This tool looks for 3 enviroment variables in order to authenticate:

`ARTIFACTORY_URL` - Base URL to use for Artifactory connections

`ARTIFACTORY_USERNAME` - Username to Artifactory

`ARTIFACTORY_PASSWORD` - Password for Artifactory

These will be loaded in at the beginning of a run and raise an exception if missing.

2.3 Installing

From pypi:

`pip install lavatory`

Or install directly from the code:

```
git clone https://github.com/gogoair/lavatory
cd lavatory
pip install -U .
```

2.4 Running

```
$ lavatory --help
Usage: lavatory [OPTIONS] COMMAND [ARGS]...

  Lavatory is a tool for managing Artifactory Retention Policies.

Options:
  -v, --verbose  Increases logging level.
  --help         Show this message and exit.

Commands:
  purge  Deletes artifacts based on retention policies.
  stats    Get statistics of a repo.
  version  Print version information.
```

2.4.1 Purging Artifacts

`lavatory purge --policies-path=/path/to/policies`

```
$ lavatory purge --help
Usage: lavatory purge [OPTIONS]

  Deletes artifacts based on retention policies.

Options:
  --policies-path TEXT            Path to extra policies directory.
  --dryrun / --nodryrun           Dryrun does not delete any artifacts.
                                  [default: True]
  --default / --no-default        Applies default retention policy.  [default:
                                  True]
  --repo TEXT                     Name of specific repository to run against.
                                  Can use --repo multiple times. If not
                                  provided, uses all repos.
  --repo-type [local|virtual|cache|any]
                                  The types of repositories to search for.
                                  [default: local]
  --help                          Show this message and exit.
```

If you want to run Lavatory against a specific repository, you can use `--repo <repo_name>`. You can specify `--repo` as multiple times to run against multiple repos. If `--repo` is not provided, Lavatory will run against all repos in Artifactory.

2.4.2 Getting Statistics

`lavatory stats --repo test-local`

```
$ lavatory stats --help
Usage: lavatory stats [OPTIONS]

  Get statistics of a repo.

Options:
  --repo TEXT               Name of specific repository to run against. Can
                            use --repo multiple times. If not provided, uses
                            all repos.
  --help       Show this message and exit.
```

2.5 Policies

See the Creating Retention Policies docs for more details on how to create and use retention policies with Lavatory.

2.5.1 Listing Policies

Lavatory looks at a policy functions docstring in order to get a description. You can list all repos and a description of the policy that would apply to them with the `lavatory policies` command.

```
$ lavatory policies --help
Usage: lavatory policies [OPTIONS]

  Prints out a JSON list of all repos and policy descriptions.

Options:
  --policies-path TEXT            Path to extra policies directory.
  --repo TEXT                     Name of specific repository to run against.
                                  Can use --repo multiple times. If not
                                  provided, uses all repos.
  --repo-type [local|virtual|cache|any]
                                  The types of repositories to search for.
                                  [default: local]
  --help                          Show this message and exit.
```

2.6 Testing

```
pip install -r requirements-dev.txt
tox
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index