gordon-janitor

Release 0.0.1.dev7

Apr 05, 2023

Contents

Cloud DNS reconciliation - a service that checks cloud DNS records against a source of truth and submits corrections to gordon.

Release v0.0.1.dev7 (*What's new?*).

Warning: This is still in the planning phase and under active development. Gordon-Janitor should not be used in production, yet.

CHAPTER 1

Requirements

- Python 3.6

Support for other Python versions may be added in the future.

CHAPTER 2

Development

For development and running tests, your system must have all supported versions of Python installed. We suggest using pyenv.

2.1 Setup

```
$ git clone git@github.com:spotify/gordon-janitor.git && cd gordon-janitor
# make a virtualenv
(env) $ pip install -r dev-requirements.txt
```

2.2 Running tests

To run the entire test suite:

```
# outside of the virtualenv
# if tox is not yet installed
$ pip install tox
$ tox
```

If you want to run the test suite for a specific version of Python:

```
# outside of the virtualenv
$ tox -e py36
```

To run an individual test, call `pytest` directly:

```
# inside virtualenv
(env) $ pytest tests/test_foo.py
```

2.3 Build docs

To generate documentation:

```
(env) $ pip install -r docs-requirements.txt
(env) $ cd docs && make html  # builds HTML files into _build/html/
(env) $ cd _build/html
(env) $ python -m http.server $PORT
```

Then navigate to `localhost:$PORT`!

To watch for changes and automatically reload in the browser:

```
(env) $ cd docs
(env) $ make livehtml  # default port 8888
# to change port
(env) $ make livehtml PORT=8080
```

CHAPTER 3

Code of Conduct

This project adheres to the Open Code of Conduct. By participating, you are expected to honor this code.

CHAPTER 4

User's Guide

4.1 Configuring the Gordon Janitor Service

4.1.1 Example Configuration

An example of a `gordon-janitor.toml` file:

```
# Gordon Janitor Core Config
[core]
plugins = ["foo.plugin"]
debug = false

[core.logging]
level = "info"
handlers = ["syslog"]

# Plugin Config
["foo"]
# global config to the general "foo" package
bar = baz

["foo.plugin"]
# specific plugin config within "foo" package
baz = bla
```

You may choose to have a `gordon-janitor-user.toml` file for development. All tables are deep merged into `gordon-janitor.toml`, to limit the amount of config duplication needed. For example, you can override `core.debug` without having to redeclare which plugins you'd like to run.

```
[core]
debug = true

[core.logging]
```

(continues on next page)

(continued from previous page)

```
level = "debug"
handlers = ["stream"]
```

4.1.2 Supported Configuration

The following sections are supported:

core

plugins=LIST-OF-STRINGS
: Plugins that the Gordon Janitor service needs to load. If a plugin is not listed, the Janitor will skip it even if there's configuration.

  The strings must match the plugin's config key. See the plugin's documentation for config key names.

debug=true|false
: Whether or not to run the Gordon Janitor service in `debug` mode.

  If `true`, the Janitor will continue running even if installed & configured plugins can not be loaded. Plugin exceptions will be logged as warnings with tracebacks.

  If `false`, the Janitor will exit out if it can't load one or more plugins.

core.logging

level=info(default)|debug|warning|error|critical
: Any log level that is supported by the Python standard `logging` library.

handlers=LIST-OF-STRINGS
: `handlers` support any of the following handlers: `stream`, `syslog`, and `stackdriver`. Multiple handlers are supported. Defaults to `syslog` if none are defined.

  Note: If `stackdriver` is selected, `ulogger[stackdriver]` needs to be installed as its dependencies are not installed by default.

4.2 Gordon Janitor's Plugin System

Module for loading plugins distributed via third-party packages.

Plugin discovery is done via `entry_points` defined in a package's `setup.py`, registered under `'gordon.plugins'`. For example:

```
# setup.py
from setuptools import setup

setup(
    name=NAME,
    # snip
    entry_points={
        'gordon.plugins': [
```

(continues on next page)

(continued from previous page)

```
            'gcp.gpubsub = gordon_gcp.gpubsub:EventClient',
            'gcp.gce.a = gordon_gcp.gce.a:ReferenceSourceClient',
            'gcp.gce.b = gordon_gcp.gce.b:ReferenceSourceClient',
            'gcp.gdns = gordon_gcp.gdns:DNSProviderClient',
        ],
    },
    # snip
)
```

Plugins are initialized with any config defined in `gordon-user.toml` and `gordon.toml`. See *Configuring the Gordon Janitor Service* for more details.

Once a plugin is found, the loader looks up its configuration via the same key defined in its entry point, e.g. `gcp.gpubsub`.

The value of the entry point (e.g. `gordon_gcp.gpubsub:EventClient`) must point to a class. The plugin class is instantiated with its config.

A plugin will not have access to another plugin's configuration. For example, the `gcp.gpusub` will not have access to the configuration for `gcp.gdns`.

See *Gordon Janitor's Plugin System* for details on how to write a plugin for Gordon.

`gordon.plugins_loader.`**`load_plugins`**(*config*, *plugin_kwargs*)

Discover and instantiate plugins.

Parameters

- **`config`** (*dict*) – loaded configuration for the Gordon service.
- **`plugin_kwargs`** (*dict*) – keyword arguments to give to plugins during instantiation.

Returns list of names of plugins, list of instantiated plugin objects, and any errors encountered while loading/instantiating plugins. A tuple of three empty lists is returned if there are no plugins found or activated in gordon config.

Return type Tuple of 3 lists

4.2.1 Writing a Plugin

Todo: Add documentation once interfaces are firmed up

4.3 Plugin Interfaces

Interface definitions for Gordon Janitor Plugins.

Please see *Gordon Janitor's Plugin System* for more information on writing a plugin for the Gordon Janitor service.

`interface` `gordon_janitor.interfaces.`**`IAuthority`**(*config*, *rrset_channel*, *metrics=None*)

Scan source of truth(s) of hosts and emit messages to Reconciler.

The purpose of this client is to consult a source of truth, for example, the list instances APIs in Google Compute Engine or AWS EC2, or consulting one's own database of hosts. A message per DNS zone with every instance record (per service owner's own requirements) will then be put onto the `rrset_channel` queue for a Reconciler to - you guessed it - reconcile.

Parameters

- **config** (*dict*) – Authority-specific configuration.
- **rrset_channel** (*asyncio.Queue*) – queue to put record set messages for later validation.
- **metrics** (*obj*) – Optional object to emit Authority-specific metrics.

run()
Start plugin in the main event loop.

Once required work is all processed, *cleanup()* needs to be called.

cleanup()
Cleanup once plugin-specific work is cleanup.

Cleanup work might include allowing outstanding asynchronous Python tasks to finish, cancelling them if they extend beyond a desired timeout, and/or closing HTTP sessions.

interface gordon_janitor.interfaces.**IGenericPlugin**(*config*, ***plugin_kwargs*)
Do not implement this interface directly.

Use *IAuthority*, *IReconciler*, or *IPublisher* instead.

Parameters

- **config** (*dict*) – Plugin-specific configuration.
- **plugin_kwargs** (*dict*) – Plugin-specific keyword arguments. See specific interface declarations.

run()
Start plugin in the main event loop.

Once required work is all processed, *cleanup()* needs to be called.

cleanup()
Cleanup once plugin-specific work is cleanup.

Cleanup work might include allowing outstanding asynchronous Python tasks to finish, cancelling them if they extend beyond a desired timeout, and/or closing HTTP sessions.

interface gordon_janitor.interfaces.**IPublisher**(*config*, *changes_channel*, *metrics=None*)
Publish change messages to the pub/sub Gordon consumes.

Clients that implement *IPublisher* will consume from the `changes_channel` queue and publish the message to the configured pub/sub for which Gordon subscribes.

Parameters

- **config** (*dict*) – Publisher-specific configuration.
- **changes_channel** (*asyncio.Queue*) – queue to consume the corrective messages needing to be published.
- **metrics** (*obj*) – Optional object to emit Publisher-specific metrics.

run()
Start plugin in the main event loop.

Once required work is all processed, *cleanup()* needs to be called.

cleanup()
Cleanup once plugin-specific work is cleanup.

Cleanup work might include allowing outstanding asynchronous Python tasks to finish, cancelling them if they extend beyond a desired timeout, and/or closing HTTP sessions.

interface gordon_janitor.interfaces.**IReconciler**(*config*, *rrset_channel*, *changes_channel*, *metrics=None*)

Validate current records in DNS against desired Authority.

Clients that implement *IReconciler* will create a change message for the configured *IPublisher* client plugin to consume if there is a discrepancy between records in DNS and the desired state.

Once validation is done, the *IReconciler* client will need to emit a None message to the changes_channel queue, signalling to an *IPublisher* client to publish the message to a pub/sub to which Gordon subscribes.

Parameters

- **config** (*dict*) – Reconciler-specific configuration.
- **rrset_channel** (*asyncio.Queue*) – queue from which to consume record set messages to validate.
- **changes_channel** (*asyncio.Queue*) – queue to publish corrective messages to be published.
- **metrics** (*obj*) – Optional object to emit Reconciler-specific metrics.

run()

Start plugin in the main event loop.

Once required work is all processed, *cleanup()* needs to be called.

cleanup()

Cleanup once plugin-specific work is cleanup.

Cleanup work might include allowing outstanding asynchronous Python tasks to finish, cancelling them if they extend beyond a desired timeout, and/or closing HTTP sessions.

4.4 API Reference

4.4.1 main

Main module to run the Gordon Janitor service.

The service expects a gordon-janitor.toml and/or a gordon-janitor-user.toml file for configuration in the current working directory, or in a provided root directory.

Any configuration defined in gordon-janitor-user.toml overwrites those in gordon-janitor.toml.

Example:

```
$ python gordon_janitor/main.py
$ python gordon_janitor/main.py -c /etc/default/
$ python gordon_janitor/main.py --config-root /etc/default/
```

gordon_janitor.main.**setup**(*config_root=''*)

Service configuration and logging setup.

Configuration defined in gordon-janitor-user.toml will overwrite gordon-janitor.toml.

Parameters **config_root** (*str*) – where configuration should load from, defaults to current working directory.

Returns A dict for Gordon service configuration

gordon_janitor.main.**setup**(*config_root=''*)

Service configuration and logging setup.

Configuration defined in `gordon-janitor-user.toml` will overwrite `gordon-janitor.toml`.

Parameters **config_root** (*str*) – where configuration should load from, defaults to current working directory.

Returns A dict for Gordon service configuration

4.4.2 plugins_loader

Module for loading plugins distributed via third-party packages.

Plugin discovery is done via `entry_points` defined in a package's `setup.py`, registered under `'gordon.plugins'`. For example:

```
# setup.py
from setuptools import setup

setup(
    name=NAME,
    # snip
    entry_points={
        'gordon.plugins': [
            'gcp.gpubsub = gordon_gcp.gpubsub:EventClient',
            'gcp.gce.a = gordon_gcp.gce.a:ReferenceSourceClient',
            'gcp.gce.b = gordon_gcp.gce.b:ReferenceSourceClient',
            'gcp.gdns = gordon_gcp.gdns:DNSProviderClient',
        ],
    },
    # snip
)
```

Plugins are initialized with any config defined in `gordon-user.toml` and `gordon.toml`. See *Configuring the Gordon Janitor Service* for more details.

Once a plugin is found, the loader looks up its configuration via the same key defined in its entry point, e.g. `gcp.gpubsub`.

The value of the entry point (e.g. `gordon_gcp.gpubsub:EventClient`) must point to a class. The plugin class is instantiated with its config.

A plugin will not have access to another plugin's configuration. For example, the `gcp.gpusub` will not have access to the configuration for `gcp.gdns`.

See *Gordon Janitor's Plugin System* for details on how to write a plugin for Gordon.

gordon.plugins_loader.**load_plugins**(*config*, *plugin_kwargs*)

Discover and instantiate plugins.

Parameters

- **config** (*dict*) – loaded configuration for the Gordon service.
- **plugin_kwargs** (*dict*) – keyword arguments to give to plugins during instantiation.

Returns list of names of plugins, list of instantiated plugin objects, and any errors encountered while loading/instantiating plugins. A tuple of three empty lists is returned if there are no plugins found or activated in gordon config.

Return type Tuple of 3 lists

gordon.plugins_loader.**load_plugins**(*config*, *plugin_kwargs*)
Discover and instantiate plugins.

Parameters

- **config** (*dict*) – loaded configuration for the Gordon service.
- **plugin_kwargs** (*dict*) – keyword arguments to give to plugins during instantiation.

Returns list of names of plugins, list of instantiated plugin objects, and any errors encountered while loading/instantiating plugins. A tuple of three empty lists is returned if there are no plugins found or activated in gordon config.

Return type Tuple of 3 lists

CHAPTER 5

Project Information

5.1 License and Credits

`gordon-janitor` is licensed under the Apache 2.0 license. The full license text can be also found in the source code repository.

5.2 How to Contribute

Every open source project lives from the generous help by contributors that sacrifice their time and `gordon-janitor` is no different.

This project adheres to the Open Code of Conduct. By participating, you are expected to honor this code. If the core project maintainers/owners feel that this Code of Conduct has been violated, we reserve the right to take appropriate action, including but not limited to: private or public reprimand; temporary or permanent ban from the project; request for public apology.

5.2.1 Communication/Support

Feel free to drop by the Spotify FOSS Slack organization in the #gordon channel.

5.2.2 Contributor Guidelines/Requirements

Contributors should expect a response within one week of an issue being opened or a pull request being submitted. More time should be allowed around holidays. Feel free to ping your issue or PR if you have not heard a timely response.

Submitting Bugs

Before submitting, users/contributors should do the following:

- **Basic troubleshooting:**
  - Make sure you're on the latest supported version. The problem may be solved already in a later release.
  - Try older versions. If you're on the latest version, try rolling back a few minor versions. This will help maintainers narrow down the issue.
  - Try the same for dependency versions - up/downgrading versions.
- Search the project's issues to make sure it's not already known, or if there is already an outstanding pull request to fix it.
- If you don't find a pre-existing issue, check the discussion on Slack. There may be some discussion history, and if not, you can ask for help in figuring out if it's a bug or not.

What to include in a bug report:

- What version of Python is being used? i.e. 2.7.13, 3.6.2, PyPy 2.0
- What operating system are you on? i.e. Ubuntu 14.04, RHEL 7.4
- What version(s) of the software are you using?
- How can the developers recreate the bug? Steps to reproduce or a simple base case that causes the bug is extremely helpful.

Contributing Patches

No contribution is too small. We welcome fixes for typos and grammar bloopers just as much as feature additions and fixes for code bloopers!

- Check the outstanding issues and pull requests first to see if development is not already being done for what you which to change/add/fix.
- If an issue has the `available` label on it, it's up for grabs for anyone to work on. If you wish to work on it, just comment on the ticket so we can remove the `available` label.
- Do not break backwards compatibility.
- Once any feedback is addressed, please comment on the pull request with a short note, so we know that you're done.
- Write good commit messages.

Workflow

- This project follows the gitflow branching model. Please name your branch accordingly.
- Always make a new branch for your work, no matter how small. Name the branch a short clue to the problem you're trying to fix or feature you're adding.
- Ideally, a branch should map to a pull request. It is possible to have multiple pull requests on one branch, but is discouraged for simplicity.
- Do not submit unrelated changes on the same branch/pull request.

- Multiple commits on a branch/pull request is fine, but all should be atomic, and relevant to the goal of the PR. Code changes for a bug fix, plus additional tests (or fixes to tests) and documentation should all be in one commit.
- Pull requests should be rebased off of master.
- To finish and merge a release branch, project maintainers should first create a PR to merge the branch into `develop`. Then, they should merge the release branch into `master` locally and push to master afterwards.
- Bugfixes meant for a specific release branch should be merged into that branch through PRs.

Code

- See docs on how to setup your environment for development.
- Code should follow the Google Python Style Guide.
- **Documentation is not optional.**
  - Docstrings are required for public API functions, methods, etc. Any additions/changes to the API functions should be noted in their docstrings (i.e. "added in 2.5")
  - If it's a new feature, or a big change to a current feature, consider adding additional prose documentation, including useful code snippets.
- **Tests aren't optional.**
  - Any bug fix should have a test case that invokes the bug.
  - Any new feature should have test coverage hitting at least $PERCENTAGE.
  - Make sure your tests pass on our CI. You will not get any feedback until it's green, unless you ask for help.
  - Write asserts as "expected == actual" to avoid any confusion.
  - Add good docstrings for test cases.

Github Labels

The super secret decoder ring for the labels applied to issues and pull requests.

Triage Status

- `needs triaging`: a new issue or pull request that needs to be triaged by the goalie
- `no repro`: a filed (closed) bug that can not be reproduced - issue can be reopened and commented upon for more information
- `won't fix`: a filed issue deemed not relevant to the project or otherwise already answered elsewhere (i.e. questions that were answered via linking to documentation or stack overflow, or is about GCP products/something we don't own)
- `duplicate`: a duplicate issue or pull request
- `waiting for author`: issue/PR has questions or requests feedback, and is awaiting the other for a response/update

Development Status

To be prefixed with `Status:`, e.g. `Status:  abandoned`.

- `abandoned`: issue or PR is stale or otherwise abandoned
- `available`: bug/feature has been confirmed, and is available for anyone to work on (but won't be worked on by maintainers)
- `blocked`: issue/PR is blocked (reason should be commented)
- `completed`: issue has been addressed (PR should be linked)
- `wip`: issue is currently being worked on
- `on hold`: issue/PR has development on it, but is currently on hold (reason should be commented)
- `pending`: the issue has been triaged, and is pending prioritization for development by maintainers
- `review needed`: awaiting a review from project maintainers

Types

To be prefixed with `Type:` e.g. `Type:  bug`.

- `bug`: a bug confirmed via triage
- `feature`: a feature request/idea/proposal
- `improvement`: an improvement on existing features
- `maintenance`: a task for required maintenance (e.g. update a dependency for security patches)
- `extension`: issues, feature requests, or PRs that support other services/libraries separate from core

5.2.3 Local Development Environment

TODO

5.3 Changelog

5.3.1 0.0.1.dev7 (2018-11-15)

Adds

End of run log message, metric

Fixes

- Correctly handle plugin loader errors
- Deep merge user config file when used

5.3.2 0.0.1.dev6 (2018-09-07)

Changes

Bump Gordon core version requirement

5.3.3 0.0.1.dev5 (2018-08-09)

Changes

Import plugins_loader from Gordon to obtain metrics client

5.3.4 0.0.1.dev3 (2018-03-22)

Changes

Rename interface methods.

5.3.5 0.0.1.dev1 (2018-02-27)

Changes

Initial development release.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index