gordon-dns-gcp

Release 0.0.1.dev38

Apr 05, 2023

Contents

Event-driven Cloud DNS and DNS Reconciliation Services

Google Cloud Platform (GCP) plugins for gordon, an open-source, event-driven service for 3rd party DNS providers, and for gordon-janitor, an open-source service that checks Cloud DNS records against a source of truth (e.g. Compute Engine) and submits corrections to gordon via Google Pubsub.

The `gordon-gcp` plugins add optional support for the following:

Service:

- Consuming events from Google Cloud Pub/Sub
- Reading from Google Compute Engine for record information
- Creating, updating, and deleting records within Google Cloud DNS

Janitor:

- Reading instance lists from Google Compute Engine
- Comparing record sets from Google Compute Engine and Google Cloud DNS
- Publishing any required DNS changes to Google Cloud Pub/Sub

Release v0.0.1.dev38 (*What's new?*).

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

CHAPTER 1

Requirements

- Python 3.6
- Google Cloud Platform account
- Service account JSON key that has relevant access (i.e. read and/or write) to the plugin service you want to use (e.g. Google Cloud DNS, Pub/Sub, or Compute Engine). See Google's documentation on how to create a key.

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-gcp.git && cd gordon-gcp
# 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 Service Configuration

Configuring Google Cloud Platform *Plugins* for the gordon service.

4.1.1 Example Gordon Configuration with GCP Plugin

```
# Gordon Core Config
[core]
[core.logging]
level = "debug"
handlers = ["syslog"]

# Plugin Config
[gcp]
project = "my-gcp-project"
dns_zone = "example.com."
default_zone_prefix = "production"

[gcp.enricher]
keyfile = "/path/to/keyfiles/compute.json"

[gcp.event_consumer]
keyfile = "/path/to/keyfiles/pubsub.json"
topic = "my-dns-changes"
subscription = "gordon-consumer"
max_messages = 25
max_msg_age = 300

[gcp.publisher]
project = "my-dns-project"
keyfile = "/path/to/keyfiles/dns.json"
```

```
publish_wait_timeout = 10
default_ttl = 300
```

4.1.2 Plugin Configuration

Attention: Configuration defined for a specific provider (`event_consumer`, `enricher`, `publisher`) will overwrite values of the same keys defined under *gcp*, then inherit the rest.

Attention: A specific provider does **not** have access to configuration for the other individual providers.

Note: Any configuration key/value listed here may also be used in the specific plugin configuration. Values set in a plugin-specific config section will overwrite what's set in this general `[gcp]` section.

[gcp]

project="STR"

Required: Google Project ID which hosts the relevant GCP services (e.g. Cloud DNS, Pub/Sub, Compute Engine).

To learn more about GCP projects, please see Google's docs on creating & managing projects.

dns_zone="STR"

Required: DNS zone to administer. Must be a fully-qualified domain name (FQDN), ending in `.`, e.g. `example.com.`. If it's a reverse zone, it must be in the form 'A.B.in-addr.arpa.'. This setting must be either in this section, or in both the `[gcp.event_consumer]` and `[gcp.publisher]` sections.

Note: this is separate from Google's 'managed zone' names. Google uses custom string names with specific requirements for storing records. Gordon requires that managed zone names be based on DNS names. For all domains, remove the trailing dot and replace all other dots with dashes. For reverse records, then use only the two most significant octets, prepended with 'reverse-'. (E.g. `foo.bar.com.` -> `foo-bar-com` and `0.168.192.in-addr.arpa.` -> `reverse-168-192`.)

keyfile="/path/to/keyfile.json"

Optional: Path to the Service Account JSON keyfile to use while authenticating against Google APIs. If not provided the default Service Account will be used instead.

While one global key for all plugins is supported, it's advised to create a key per plugin with only the permissions it requires. To setup a service account, follow Google's docs on creating & managing service account keys.

default_zone_prefix="STR"

Optional: Prefix associated with Google managed zone names, prepended with a '-' to the generated name. For example prefix "production" will produced a managed zone name of "production-example-com" for the "example.com." DNS zone.

[gcp.event_consumer]

All configuration options above in the general `[gcp]` may be used here. Additional Google Pub/Sub Consumer-related configuration options are:

topic="STR"
: *Required*: A topic to which the Event Consumer client must subscribe.

  For more information on Google Pub/Sub topics, please see Google's docs on managing topics.

subscription="STR"
: *Required*: A subscription to the `topic` from which the Event Consumer client will pull.

  For more information on Google Pub/Sub subscriptions, please see Google's docs on managing subscriptions.

max_messages=INT
: *Optional*: Number of Pub/Sub messages to process at a time. Defaults to 25.

max_msg_age=INT
: *Optional*: Discard incoming messages older than this many seconds. Defaults to 300.

`[gcp.enricher]`

All configuration options above in the general `[gcp]` may be used here. If `dns_zone` isn't present here, it must be in `[gcp]`.

`[gcp.publisher]`

All configuration options above in the general `[gcp]` may be used here. If `dns_zone` isn't present here, it must be in `[gcp]`. Additional Google Cloud DNS configuration options are:

default_ttl=INT
: *Required*: The default TTL in seconds. This will be used if the publisher receives a record set to be published that does not yet have the TTL set. Must be greater than 4.

publish_wait_timeout=INT|FLOAT
: *Optional*: Timeout in seconds for waiting for confirmation that changes have been successfully completed within Google Cloud DNS. Default is 60 seconds.

api_version="STR"
: *Optional*: API version for both the changes endpoint and the resource records endpoint.

4.2 Janitor Configuration

Configuring Google Cloud Platform *Plugins* for the gordon-janitor service.

4.2.1 Example Configuration

An example of a `gordon-janitor.toml` file for GCP-specific plugins:

```
# Example for Gordon Janitor GCP-related Config
[core]
plugins = ["gcp.gdns"]

[gcp]
cleanup_timeout = 60

[gcp.gdns]
keyfile = "/path/to/dns-service-account.json"
project = "gordon-dns-example"
```

```
scopes = ["ndev.clouddns.readonly"]
default_zone_prefix = "production"

[gcp.gpubsub]
keyfile = "/path/to/pubsub-service-account.json"
project = "gordon-pubsub-example"
topic = "dns-changes-topic"

[gcp.gce]
keyfile = "/path/to/crm-service-account.json"
scopes = ["cloud-platform"]
dns_zone = "example.com."
metadata_blacklist = [["key", "val"], ["other_key", "other_val"]]
tag_blacklist = []
project_blacklist = []
# This is passed directly to GCE's v1.instances.aggregatedList endpoint
instance_filter = ""
```

4.2.2 Plugin Configuration

The following sections are supported:

gcp

Any configuration key/value listed here may also be used in the specific plugin configuration. Values set in a plugin-specific config section will overwrite what's set in this general `[gcp]` section.

`project``="STR"`
: *Required*: Google Project ID which hosts the relevant GCP services (e.g. Cloud DNS, Pub/Sub, Compute Engine).

  To learn more about GCP projects, please see Google's docs on creating & managing projects.

`keyfile``="/path/to/keyfile.json"`
: *Optional*: Path to the Service Account JSON keyfile to use while authenticating against Google APIs. If not provided the default Service Account will be used instead.

  While one global key for all plugins is supported, it's advised to create a key per plugin with only the permissions it requires. To setup a service account, follow Google's docs on creating & managing service account keys.

  Attention: For the Pub/Sub plugin, `keyfile` is not required when running against the Pub/Sub Emulator that Google provides.

`scopes``=["STR","STR"]`
: *Optional*: A list of strings of the scope(s) needed when making calls to Google APIs. Defaults to `["cloud-platform"]`.

`cleanup_timeout``=INT`
: *Optional*: Timeout in seconds for how long each plugin should wait for outstanding tasks (e.g. processing remaining message from a channel) before cancelling. This is only used when a plugin has received all messages from a channel, but may have work outstanding. Defaults to `60`.

`default_zone_prefix``="STR"`
: *Optional*: Prefix associated with Google managed zone names, prepended with a '-' to the generated name.

For example prefix "production" will produced a managed zone name of "production-example-com" for the "example.com." DNS zone.

Note: This prefix must be the same as that used by the Gordon Service to work correctly.

gcp.gdns

All configuration options above in the general `[gcp]` may be used here. There are no specific DNS-related configuration options.

gcp.gpubsub

All configuration options above in the general `[gcp]` may be used here. Additional Google Cloud Pub/Sub-related configuration is needed:

`topic``="STR"`
: *Required*: Google Pub/Sub topic to receive the publish change messages.

> **Attention:** For the Pub/Sub plugin, `keyfile` is not required when running against the Pub/Sub Emulator that Google provides.

gcp.gce

All configuration options from the general `[gcp]` section may be used here.

Additional plugin-specific configuration is needed:

`dns_zone``="STR"`
: *Required*: DNS zone to pull records from. Must be a fully-qualified domain name (FQDN), ending in `.`, e.g. `example.com.`. If it's a reverse zone, it must be in the form 'A.B.in-addr.arpa.'.

Note: this is separate from Google's 'managed zone' names. Google uses custom string names with specific requirements for storing records. Gordon requires that managed zone names be based on DNS names. For all domains, remove the trailing dot and replace all other dots with dashes. For reverse records, then use only the two most significant octets, prepended with 'reverse-'. (E.g. `foo.bar.com.` -> `foo-bar-com` and `0.168.192.in-addr.arpa.` -> `reverse-168-192`.)

`metadata_blacklist``=[["STR","STR"],["STR","STR"]]`
: *Optional*: List of key-value pairs that will be used to filter out unwanted GCE instances by instance metadata. Note that both the key and the value must match for an instance to be filtered out.

`tag_blacklist``=["STR","STR"]`
: *Optional*: List of network tags that will be used to filter out unwanted GCE instances.

`project_blacklist``=["STR","STR"]`
: *Optional*: List of unique, user-assigned project IDs (`projectId`) that will be ignored when fetching projects.

`project_whitelist``=["STR","STR"]`
: *Optional*: List of unique, user-assigned project IDs (`projectId`) that will be used to fetch instances. If set, janitor will only look at these projects, it will not fetch active projects and the *project_blacklist* will be ignored.

`instance_filter``="STR"`
: *Optional*: String used to filter instances by instance attributes. It is passed directly to GCE's v1.instances.aggregatedList endpoint's *filter* parameter.

4.3 Schemas for Event Consuming

4.3.1 Definitions

This package defines two types of schemas for consuming events by the *Google Pub/Sub Event Consumer plugin*: *Google Audit Log Message* and a *General Event Message*.

Google Audit Log Message

The *Google Audit Log Message* schema is based off of Google's Audit Log datatype for Cloud Audit Logging. Documentation on setting up the exporting of Google Cloud audit logs to Google Pub/Sub (a.k.a. a "sink") can be found here. You may need to refine the export to include the following filters:

```
resource.type=gce_instance
protoPayload."@type"="type.googleapis.com/google.cloud.audit.AuditLog"
```

Schema Definition

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "id": "https://github.com/spotify/gordon-gcp/blob/master/gordon_gcp/schemas/audit-
↪log.schema.json",
    "title": "Google Audit Log Messages",
    "description": "Schema for parsing Google Audit Log Messages",
    "required": ["protoPayload", "logName"],
    "properties": {
        "protoPayload": {
            "type": "object",
            "properties": {
                "methodName": {
                    "type": "string",
                    "description": "Action performed on a GCP",
                    "enum": [
                        "v1.compute.instances.insert",
                        "v1.compute.instances.delete"
                    ]
                },
                "resourceName": {
                    "type": "string",
                    "description": "",
                    "examples": [
                        "projects/a-project-id/zones/a-zone-name/instances/an-
↪instance-name"
                    ],
                    "pattern": "^projects/([0-9]+|[a-z][a-z\\-0-9]{5,29})/zones/[a-
↪z\\-0-9]+/instances/[a-z\\-0-9]+"
                }
            },
            "required": ["methodName", "resourceName"]
        },
        "logName": {
            "type": "string",
            "description": "Name of audit log",
            "examples": [
                "projects/a-project-id/logs/cloudaudit.googleapis.com%2Factivity"
            ],
            "pattern": "^projects/[a-z][a-z\\-0-9]{5,29}/logs/cloudaudit.googleapis.
↪com%2Factivity"
        },
        "receiveTimestamp": {
            "type": "string",
            "description": "Time the log entry was received by Stackdriver Logging",
            "examples": [
                "2018-01-01T23:13:45.123456789Z"
            ]
        },
        "timestamp": {
            "type": "string",
            "description": "Time the operation status was reported",
            "examples": [
                "2018-01-01T23:13:45.123456789Z"
            ]
        }
    }
}
```

Examples

Audit Log Message with an `operation.first` key:

```
{
  "insertId": "123456789101112",
  "logName": "projects/some-project/logs/cloudaudit.googleapis.com%2Factivity",
  "operation": {
    "first": true,
    "id": "operation-1234-5678-9101-1121",
    "producer": "compute.googleapis.com"
  },
  "protoPayload": {
    "@type": "type.googleapis.com/google.cloud.audit.AuditLog",
    "authenticationInfo": {
      "principalEmail": "not-real@cloudservices.gserviceaccount.com"
    },
    "authorizationInfo": [
      {
        "granted": true,
        "permission": "compute.instances.create"
      }
    ],
    "methodName": "v1.compute.instances.insert",
    "request": {
      "@type": "compute.googleapis.com/compute.instances.insert"
    },
    "requestMetadata": {
      "callerSuppliedUserAgent": "Managed Infrastructure Mixer Client"
    },
    "resourceName": "projects/123456789101/zones/us-central1-c/instances/an-instance-
↪name-b34c",
    "response": {
      "@type": "compute.googleapis.com/operation",
      "id": "2345678910111121314",
      "insertTime": "2017-12-04T12:13:44.785-08:00",
      "name": "operation-1234-5678-9101-1121",
      "operationType": "insert",
      "progress": "0",
      "selfLink": "https://www.googleapis.com/compute/v1/projects/some-project/zones/
↪us-central1-c/operations/operation-1234-5678-9101-1121",
      "status": "PENDING",
      "targetId": "123456789101213141",
      "targetLink": "https://www.googleapis.com/compute/v1/projects/some-project/
↪zones/us-central1-c/instances/an-instance-name-b34c",
      "zone": "https://www.googleapis.com/compute/v1/projects/some-project/zones/us-
↪central1-c"
    },
    "serviceName": "compute.googleapis.com"
  },
  "receiveTimestamp": "2017-12-04T20:13:45.494149958Z",
  "resource": {
    "labels": {
      "instance_id": "123456789101213141",
      "project_id": "some-project",
      "zone": "us-central1-c"
    },
    "type": "gce_instance"
  },
  "severity": "NOTICE",
  "timestamp": "2017-12-04T20:13:44.181Z"
}
```

Audit Log Message with an `operation.last` key:

```
{
    "insertId": "123456789101112",
    "logName": "projects/some-project/logs/cloudaudit.googleapis.com%2Factivity",
    "operation": {
        "id": "operation-1234-5678-9101-1121",
        "last": true,
        "producer": "compute.googleapis.com"
    },
    "protoPayload": {
        "@type": "type.googleapis.com/google.cloud.audit.AuditLog",
        "authenticationInfo": {},
        "methodName": "v1.compute.instances.insert",
        "requestMetadata": {
            "callerSuppliedUserAgent": "Managed Infrastructure Mixer Client"
        },
        "resourceName": "projects/123456789101/zones/us-central1-c/instances/an-
↪instance-name-b45c",
        "serviceName": "compute.googleapis.com"
    },
    "receiveTimestamp": "2017-12-04T20:13:51.414016721Z",
    "resource": {
        "labels": {
            "instance_id": "123456789101213141",
            "project_id": "some-project",
            "zone": "us-central1-c"
        },
        "type": "gce_instance"
    },
    "severity": "NOTICE",
    "timestamp": "2017-12-04T20:13:50.717Z"
}
```

Audit Log Message with no `operation` object:

```
{
    "insertId": "123456789101112",
    "logName": "projects/a-project/logs/cloudaudit.googleapis.com%2Factivity",
    "protoPayload": {
        "@type": "type.googleapis.com/google.cloud.audit.AuditLog",
        "authenticationInfo": {},
        "methodName": "v1.compute.instances.insert",
        "requestMetadata": {
            "callerSuppliedUserAgent": "Managed Infrastructure Mixer Client"
        },
        "resourceName": "projects/123456789101/zones/us-central1-c/instances/an-
↪instance-name-a123b",
        "serviceName": "compute.googleapis.com"
    },
    "receiveTimestamp": "2017-12-04T20:13:51.414016721Z",
    "resource": {
        "labels": {
            "instance_id": "123456789101213141",
            "project_id": "a-project",
            "zone": "us-central1-c"
        },
        "type": "gce_instance"
    },
    "severity": "NOTICE",
    "timestamp": "2017-12-04T20:13:50.717Z"
}
```

General Event Message

The *General Event Message* schema is meant for any event that may not come from Google's audit log sink. For instance, one may need to add/update/delete a manual record (i.e. marketing-friendly `CNAME` s). Or there may be a reconciliation process between the current state of the world and what is reflected in DNS.

Schema Definition

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "type": "object",
    "id": "https://github.com/spotify/gordon-gcp/blob/master/gordon_gcp/schemas/event.
↪schema.json",
    "title": "Generic Event Messages",
    "description": "Schema for parsing generic event messages",
    "required": ["action", "timestamp", "resourceRecords"],
    "properties": {
        "action": {
            "description": "Add or delete resource record sets (rrdatas) listed",
            "type": "string",
            "enum": ["additions", "deletions"]
        },
        "timestamp": {
            "type": "string",
            "description": "Time the message was emitted",
            "examples": [
                "2018-01-01T23:13:45.123456789Z"
            ]
        },
        "resourceRecords": {
            "description": "DNS resource records upon which to act",
            "type": "object",
            "required": ["name", "rrdatas", "type"],
            "properties": {
                "name": {
                    "description": "Record name",
                    "type": "string",
                    "examples": [
                        "www.example.com.",
                        "some-fqdn.subdomain.example.com."
                    ]
                },
                "rrdatas": {
                    "description": "Resource record(s) that `name` point to",
                    "type": "array",
                    "minItems": 1,
                    "items": {
                        "type": "string"
                    },
                    "uniqueItems": true,
                    "examples": [
                        "198.1.1.50",
                        "example.com.",
                        "some text value"
                    ]
                },
                "ttl": {
                    "description": "TTL in seconds; default 300 seconds",
                    "type": "number",
                    "maximum": 86400,
                    "minimum": 60,
                    "default": 300
                },
                "type": {
                    "description": "Type of resource record(s) as supported by Google␣
↪Cloud DNS",
                    "type": "string",
                    "enum": ["A", "AAAA", "CNAME", "MX", "NS", "PTR", "SOA", "SRV",
↪"TXT"]
                }
            }
        }
    }
}
```

Examples

Event for an `A` record:

```
{
    "resourceRecords": {
        "name": "lynntest-a-1234.foo.spotify.net.",
        "rrdatas": [
            "10.1.2.3"
        ],
        "ttl": 300,
        "type": "A"
    },
    "timestamp": "2017-12-04T20:13:51.414016721Z",
    "action": "additions"
}
```

Event for an `CNAME` record:

```
{
    "action": "additions",
    "timestamp": "2017-12-04T20:13:51.414016721Z",
    "resourceRecords": {
        "name": "lynntest-a-1234.spotify.net.",
        "rrdatas": [
            "lynntest-a-1234.foo.spotify.net."
        ],
        "ttl": 300,
        "type": "CNAME"
    }
}
```

Event for an `NS` record:

```
{
    "action": "additions",
    "timestamp": "2017-12-04T20:13:51.414016721Z",
    "resourceRecords": {
        "name": "gordontest.spotify.net.",
        "rrdatas": [
            "ns-cloud-c1.googledomains.com.",
            "ns-cloud-c2.googledomains.com.",
            "ns-cloud-c3.googledomains.com.",
            "ns-cloud-c4.googledomains.com."
        ],
        "ttl": 21600,
        "type": "SOA"
    }
}
```

4.3.2 Validating

Module to load and validate GCP-related JSON schemas.

Schema file discovery is based on any JSON file in the gordon_gcp/schema/schemas/ directory.

For more information on this plugin's schemas, see *Schemas for Event Consuming*.

Warning: The following documentation may change since the calling/using of the this module is not yet incorporated into the plugin logic.

Schemas are loaded once at service startup. A JSON message/object is validated against a given loaded schema upon receipt of message/object. The schema to validate against is determined by the topic from which the PubSub message is pulled.

To use:

```
>>> from gordon_gcp.schema import validate
>>> validator = validate.MessageValidator()
>>> validator.schemas
{
    'event': {
        'title': 'Generic Event Message',
        'type': 'object',
        'required': ...},
    'audit-log': {
        'title': 'Google Audit Log Message',
        'type': 'object',
        'required': ...}
}
>>> example_message = {'foo': 'bar'}
>>> validator.validate(example_message, 'event')
```

class gordon_gcp.**MessageValidator**

Load packaged JSON schemas and validate a given JSON message.

schemas

dict – schema name based on filename mapped to its loaded JSON schema.

Raises *GCPGordonError* – if unable to find or load schemas.

validate (*message*, *schema_name*)

Validate a message given a schema.

Parameters

- **message** (*dict*) – Loaded JSON of pulled message from Google PubSub.
- **schema_name** (*str*) – Name of schema to validate `message` against. `schema_name` will be used to look up schema from *MessageValidator.schemas* dict

Raises

- *InvalidMessageError* – if message is invalid against the given schema.

- *InvalidMessageError* – if given schema name can not be found.

4.3.3 Parsing

Module to parse loaded JSON messages according to GCP-related schemas and return only the actionable data.

To use:

```
>>> import json
>>> from gordon_gcp.schema import parse
>>> parser = parse.MessageParser()
>>> # using an example file
>>> exp = 'gordon_gcp/schema/examples/audit-log.last-operation.json'
>>> with open(exp, 'r') as f:
...   data = json.load(f)
>>> message = parser.parse(data, 'audit-log')
>>> message
{
    'action': 'additions',
    'resourceName': 'projects/.../instances/an-instance-name-b45c',
    'resourceRecords': []
}
```

class gordon_gcp.**MessageParser**

Parse a message provided a given GCP schema.

parse (*message*, *schema*)

Parse message according to schema.

message should already be validated against the given schema. See *Schema Definition* for more information.

Parameters

- **message** (*dict*) – message data to parse.
- **schema** (*str*) – valid message schema.

Returns parsed message

Return type (dict)

4.4 Load and Validate Schemas

Module to load and validate GCP-related JSON schemas.

Schema file discovery is based on any JSON file in the gordon_gcp/schema/schemas/ directory.

For more information on this plugin's schemas, see *Schemas for Event Consuming*.

Warning: The following documentation may change since the calling/using of the this module is not yet incorporated into the plugin logic.

Schemas are loaded once at service startup. A JSON message/object is validated against a given loaded schema upon receipt of message/object. The schema to validate against is determined by the topic from which the PubSub message is pulled.

To use:

```
>>> from gordon_gcp.schema import validate
>>> validator = validate.MessageValidator()
>>> validator.schemas
{
    'event': {
        'title': 'Generic Event Message',
        'type': 'object',
        'required': ...},
    'audit-log': {
        'title': 'Google Audit Log Message',
        'type': 'object',
        'required': ...}
}
>>> example_message = {'foo': 'bar'}
>>> validator.validate(example_message, 'event')
```

class `gordon_gcp.schema.validate.`**`MessageValidator`**

Load packaged JSON schemas and validate a given JSON message.

`schemas`

dict – schema name based on filename mapped to its loaded JSON schema.

Raises `GCPGordonError` – if unable to find or load schemas.

`validate`(*message*, *schema_name*)

Validate a message given a schema.

Parameters

- **`message`** (*`dict`*) – Loaded JSON of pulled message from Google PubSub.
- **`schema_name`** (*`str`*) – Name of schema to validate `message` against. `schema_name` will be used to look up schema from *`MessageValidator.schemas`* dict

Raises

- `InvalidMessageError` – if message is invalid against the given schema.
- `InvalidMessageError` – if given schema name can not be found.

4.5 Plugins

Currently available Google Cloud Platform plugins for the gordon and gordon-janitor services.

Attention: These plugins are internal modules for the core gordon and gordon-janitor logic. No other use cases are expected.

Todo: Add prose documentation for how to implement a plugin.

4.6 Gordon Service

4.6.1 Enricher

Client module to enrich an event message with any missing information (such as IP addresses to a new hostname) and to generate the desired record(s) (e.g. `A` or `CNAME` records). Once an event message is done (either successfully enriched or met with errors along the way), it will be placed into the appropriate channel, either the `success_channel` or `error_channel` to be further handled by the `gordon` core system.

Attention: The enricher client is an internal module for the core gordon logic. No other use cases are expected.

class `gordon_gcp.`**`GCEEnricher`**(*config*, *metrics*, *http_client*, *dns_client*, ***kwargs*)

Get needed instance information from Google Compute Engine.

Parameters

- **`config`** (*`dict`*) – configuration relevant to Compute Engine.
- **`metrics`** (*`obj`*) – `IMetricRelay` implementation.
- **`http_client`** (`AIOConnection`) – client for interacting with the GCE API.

`handle_message`(*event_message*)

Enrich message with extra context and send it to the publisher.

When a message is successfully processed, it is passed to the `self.success_channel`. However, if there is a problem during processing, the message is passed to the `self.error_channel`.

Parameters **`event_message`** (*`GEventMessage`*) – message requiring additional information.

4.6.2 Event Consumer

Client module to consume Google Cloud Pub/Sub messages and create an event message to be passed around the `gordon` core system and its plugins. Once an event message is created, it will be placed into the `success_channel` to be further handled by the `gordon` core system. While being processed by the other plugin(s), the consumer will continuously extend the message's `ack` deadline. When an event message is done (either successfully published or met with errors) the consumer will then `ack` the message in Pub/Sub to signify that the work is complete.

Attention: The event consumer client is an internal module for the core gordon logic. No other use cases are expected.

class `gordon_gcp.`**`GPSEventConsumer`**(*config*, *success_channel*, *error_channel*, *metrics*, *subscriber*, *flow_control*, *validator*, *parser*, *loop*, ***kwargs*)

Consume messages from Google Cloud Pub/Sub.

Pub/Sub messages are continually consumed via google-cloud-python's *pubsub* module using gRPC. Every consumed message will create an asyncio task that handles the message schema validation, creation of a `GEventMessage` instance (*event_msg*), and the forwarding on for further processing via the `self.success_channel`. The *pubsub* module handles the message ack deadline extension behind the scenes. Once the *event_msg* is done processing, gordon's core routing system will submit it back to this consumer to be *ack*'ed via the *handle_method* method used for pubsub cleanup.

Parameters

- **config** (*dict*) – configuration relevant to Cloud Pub/Sub.
- **success_channel** (*asyncio.Queue*) – a sink for successfully processed `interfaces.IEventMessages`.
- **error_channel** (*asyncio.Queue*) – a sink for `interfaces.IEventMessages` that were not processed due to problems.

handle_message(*event_msg*)

Ack Pub/Sub message and update event message history.

Parameters **event_msg** (*GEventMessage*) – message to clean up

run()

Start consuming messages from Google Pub/Sub.

Once a Pub/Sub message is validated, a `GEventMessage` is created for the message, then passed to the `self.success_channel` to be handled by an IEnricher plugin.

4.6.3 GDNS Publisher

Client module to publish DNS records from an event message. Once an event message is done (either successfully published, or met with errors along the way), it will placed into the appropriate channel, either the *success_channel‘* or `error_channel` to be further handled by the `gordon` core system.

Attention: The publisher client is an internal module for the core gordon logic. No other use cases are expected.

class gordon_gcp.**GDNSPublisher**(*config*, *metrics*, *dns_client*, ***kwargs*)

Publish records to Google Cloud DNS.

Parameters

- **config** (*dict*) – Configuration relevant to Cloud DNS.
- **success_channel** (*asyncio.Queue*) – A sink for successfully processed `interfaces.IEventMessages`.
- **error_channel** (*asyncio.Queue*) – A sink for `interfaces.IEventMessages` that were not processed due to problems.
- **dns_client** (*gdns.GDNSClient*) – A Google DNS HTTP connection class.

handle_message(*event_msg*)

Publish changes extracted from the event message.

Parameters **event_msg** (*event_consumer.GEventMessage*) – Contains the changes to publish.

Raises *InvalidDNSZoneInMessageError if the DNS zone of a resource record* – does not match our configured zone.

4.7 Gordon Janitor

4.7.1 Reconciler

Module to compare desired record sets produced from a Resource Authority (i.e. `GCEInstanceAuthority` for Google Compute Engine) and actual record sets from Google Cloud DNS, then publish corrective messages to the internal `changes_channel` if there are differences.

This client makes use of the asynchronous DNS client as defined in *`GDNSClient`*, and therefore must use service account/JWT authentication (for now).

See *Janitor Configuration* for the required Google DNS configuration.

Attention: This reconciler client is an internal module for the core janitor logic. No other use cases are expected.

To use:

```
import asyncio
import gordon_gcp

config = {
    'keyfile': '/path/to/keyfile.json',
    'project': 'a-dns-project'
}
rrset_chnl = asyncio.Queue()
changes_chnl = asyncio.Queue()

reconciler = gordon_gcp.GDNSReconciler(
    config, rrset_chnl, changes_chnl)

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(reconciler.start())
finally:
    loop.close()
```

The keyfile is optional. If not provided the default service account will be used.

class `gordon_gcp.`**`GDNSReconciler`**(*config*, *metrics*, *dns_client*, *rrset_channel=None*, *changes_channel=None*, ***kw*)

Validate current records in DNS against desired source of truth.

`GDNSReconciler` will create a change message for the configured publisher client plugin to consume if there is a discrepancy between records in Google Cloud DNS and the desired state.

Once validation is done, the Reconciler will emit a `None` message to the `changes_channel` queue, signalling a Publisher client (e.g. *`GPubsubPublisher`*) to publish the message to a pub/sub to which Gordon subscribes.

Parameters

- **`config`** (*`dict`*) – Google Cloud DNS-related configuration.
- **`metrics`** (*obj*) – `IMetricRelay` implementation.
- **`dns_client`** (`GDNSClient`) – Client to interact with Google Cloud DNS API.
- **`rrset_channel`** (*`asyncio.Queue`*) – Queue from which to consume record set messages to validate.

- **changes_channel** (*asyncio.Queue*) – Queue to publish message to make corrections to Cloud DNS.

cleanup()

Clean up & notify `changes_channel` of no more messages.

This method collects all tasks that this particular class initiated, and will cancel them if they don't complete within the configured timeout period.

Once all tasks are done, `None` is added to the `changes_channel` to signify that it has no more work to process. Then the HTTP session attached to the `dns_client` is properly closed.

static create_rrset_set(*zone*, *rrsets*, *source=None*)

Create a set of ResourceRecordSets excluding SOA and zone's NS.

Parameters

- **zone** (*str*) – zone of the rrsets, for NS record exclusion.
- **rrsets** (*list (dict)*) – collection of dict representation of RRSets.
- **source** (*str*) – (optional) source to add to the rrset

Returns set of `ResourceRecordSet`

publish_change_messages(*desired_rrsets*, *action='additions'*)

Publish change messages to the `changes_channel`.

NOTE: Only *'additions'* are currently supported. *'deletions'* may be supported in the future.

Parameters

- **desired_rrsets** (*list (ResourceRecordSet)*) – Desired record sets that are not in Google Cloud DNS.
- **action** (*str*) – (optional) action for these corrective messages. Defaults to `'additions'`.

run()

Publish necessary DNS changes to the `changes_channel`.

Consumes zone/rrset-list messages from `rrset_channel`, compares them to the current records, and publishes the changes. Once `None` is received from the channel, emits a final `None` message to the `changes_channel`.

validate_rrsets_by_zone(*zone*, *rrsets*)

Given a zone, validate current versus desired rrsets.

Returns lists of missing rrsets (in desired but not in current) and extra rrsets (in current but not in desired). Extra rrsets that are the result of updates in the desired list will not be returned, and root SOA/NS comparisons are skipped.

Parameters

- **zone** (*str*) – zone to query Google Cloud DNS API.
- **rrsets** (*list*) – desired record sets to which to compare the Cloud DNS API's response.

Returns The missing and extra rrset sets.

Return type tuple[set(rrset), set(rrset)]

4.7.2 GPubSub Publisher

Client module to publish any required DNS changes initiated from *GDNSReconciler* to Google Cloud Pub/Sub. The consumer of these messages is the Gordon service.

This client wraps around google-cloud-pubsub using grpc rather than inheriting from *AIOConnection*.

Attention: This publisher client is an internal module for the core janitor logic. No other use cases are expected.

To use:

```
import asyncio
import gordon_gcp

config = {
    'keyfile': '/path/to/keyfile.json',
    'project': 'a-dns-project',
    'topic': 'a-topic',
}
changes_channel = asyncio.Queue()

publisher = gordon_gcp.get_gpubsub_publisher(
    config, changes_channel)

loop = asyncio.get_event_loop()
try:
    loop.run_until_complete(publisher.start())
finally:
    loop.close()
```

The keyfile is optional. If not provided the default service account will be used.

class gordon_gcp.**GPubsubPublisher**(*config*, *metrics*, *publisher*, *changes_channel=None*, ***kw*)
Client to publish change messages to Google Pub/Sub.

Parameters

- **config** (*dict*) – Google Cloud Pub/Sub-related configuration, ex. 'projects/test-example/topics/a-topic'.
- **publisher** (*google.cloud.pubsub_v1.publisher.client.Client*) – client to interface with Google Pub/Sub API.
- **metrics** (*obj*) – IMetricRelay implementation.
- **changes_channel** (*asyncio.Queue*) – queue to publish message to make corrections to Cloud DNS.

cleanup()
Clean up outstanding tasks and emit final logs + metrics.

This method collects all tasks that this particular class initiated, and will cancel them if they don't complete within the configured timeout period.

publish(*message*)
Publish received change message to Google Pub/Sub.

Parameters message (*dict*) – change message received from the changes_channel to emit.

run()

Start consuming from `changes_channel`.

Once `None` is received from the channel, finish processing records and clean up any outstanding tasks.

4.7.3 Authority

A GCEAuthority retrieves a list of all instances in all projects that it has access to, and which belong to the configured zone. For every project, it will create a message containing domain record information and put it into the rrset channel. Projects can be filtered by 'project name'. Instances can be filtered by tags and metadata.

To use:

```
import asyncio

from gordon_gcp.plugins import janitor

async def run():
    rrset_channel = asyncio.queue()
    authority = janitor.get_authority(config, rrset_channel)
    await authority.start()
    msg = await rrset_channel.get()
    print(msg)

loop = asyncio.get_event_loop()
loop.run_until_complete(run())
# prints: {'zone': 'example.com', 'resourceRecords': [...]}
```

class gordon_gcp.**GCEAuthority**(*config*, *metrics*, *crm_client*, *gce_client*, *rrset_channel=None*, ***kwargs*)

Gather instance data from GCE.

Parameters

- **config** (*dict*) – plugin-specific configuration.
- **metrics** (*obj*) – `IMetricRelay` implementation.
- **crm_client** (`GCRMClient`) – client used to fetch GCE projects.
- **gce_client** (`GCEClient`) – client used to fetch instances for a project.
- **rrset_channel** (*asyncio.Queue*) – channel to send resource record messages to.

cleanup()

Clean up after a run.

run()

Batch instance data and send it to the `self.rrset_channel`.

4.8 API Clients

4.8.1 HTTP Sessions

Todo: Update prose of "...you can pass in your own session into an API client" with specific API client class names once they're defined. Also update the example code using one API client.

By default, the HTTP session used for getting credentials is reused for API calls (recommended if there are many). If this is not desired, you can pass in your own `aiohttp.ClientSession` instance into an API client or *`AIOConnection`*. The auth client *`GAuthClient`* may also take an explicit session object, but is not required to assert a different HTTP session is used for the API calls.

```
import aiohttp
import gordon_gcp

keyfile = '/path/to/service_account_keyfile.json'
session = aiohttp.ClientSession()  # optional
auth_client = gordon_gcp.GAuthClient(
    keyfile=keyfile, session=session
)

new_session = aiohttp.ClientSession()

# basic HTTP client
client = gordon_gcp.AIOConnection(
    auth_client=auth_client, session=new_session
)
```

4.8.2 Asynchronous GCP HTTP Client

Module to interact with Google APIs via asynchronous HTTP calls. *`AIOConnection`* is meant to be used/inherited by other product-specific API clients as it handles Google authentication and automatic refresh of tokens.

Todo: Include that it also handles retries once implemented.

To use:

```
import gordon_gcp

keyfile = '/path/to/service_account_keyfile.json'
auth_client = gordon_gcp.GAuthClient(keyfile=keyfile)

client = AIOConnection(auth_client=auth_client)
resp = await client.request('get', 'http://api.example.com/foo')
```

The keyfile is optional. If not provided the default service account will be used.

class `gordon_gcp.`**`AIOConnection`**(*auth_client=None*, *session=None*)

Async HTTP client to Google APIs with service-account-based auth.

Parameters

- **auth_client** (`GAuthClient`) – client to manage authentication for HTTP API requests.
- **session** (*`aiohttp.ClientSession`*) – (optional) `aiohttp` HTTP session to use for sending requests. Defaults to the session object attached to `auth_client` if not provided.

get_all(*url*, *params=None*)

Aggregate data from all pages of an API query.

Parameters

- **url** (*`str`*) – Google API endpoint URL.

- **`params`** (*`dict`*) – (optional) URL query parameters.

Returns Parsed JSON query response results.

Return type list

`get_json`(*url*, *json_callback=None*, ***kwargs*)
Get a URL and return its JSON response.

Parameters

- **`url`** (*`str`*) – URL to be requested.
- **`json_callback`** (*`func`*) – Custom JSON loader function. Defaults to `json.loads()`.
- **`kwargs`** (*`dict`*) – Additional arguments to pass through to the request.

Returns response body returned by `json_callback()` function.

`request`(*method*, *url*, *params=None*, *headers=None*, *data=None*, *json=None*, *token_refresh_attempts=2*, ***kwargs*)
Make an asynchronous HTTP request.

Parameters

- **`method`** (*`str`*) – HTTP method to use for the request.
- **`url`** (*`str`*) – URL to be requested.
- **`params`** (*`dict`*) – (optional) Query parameters for the request. Defaults to `None`.
- **`headers`** (*`dict`*) – (optional) HTTP headers to send with the request. Headers pass through to the request will include `DEFAULT_REQUEST_HEADERS`.
- **`data`** (*`obj`*) – (optional) A dictionary, bytes, or file-like object to send in the body of the request.
- **`json`** (*`obj`*) – (optional) Any json compatible python object. NOTE: json and body parameters cannot be used at the same time.
- **`token_refresh_attempts`** (*`int`*) – (optional) Number of attempts a token refresh should be performed.

Returns (str) HTTP response body.

Raises *`GCPHTTPError`* – if any exception occurred, specifically a *`GCPHTTPResponseError`*, if the exception is associated with a response status code.

`valid_token_set`()
Check for validity of token, and refresh if none or expired.

4.8.3 GCP Auth Client

Module to create a client interacting with Google Cloud authentication.

An instantiated client is needed for interacting with any of the Google APIs via the *`AIOConnection`*.

The GAuthClient supports both service account (JSON Web Tokens/JWT) authentication with keyfiles, and default credentials. To setup a service account, follow Google's docs. More information on default credentials can be found *here*. To setup default credentials, follow Application Default Credentials.

If a keyfile is not provided, the Application Default Credentials will be used.

To use:

```
>>> import asyncio
>>> import google_gcp
>>> loop = asyncio.get_event_loop()
>>> keyfile = '/path/to/service_account_keyfile.json'
# with keyfile
>>> auth_client = google_gcp.GAuthClient(keyfile=keyfile)
# with Application Default Credentials
>>> auth_client = google_gcp.GAuthClient()
>>> auth_client.token is None
True
>>> loop.run_until_complete(auth_client.refresh_token())
>>> auth_client.token
'c0ffe3'
```

class gordon_gcp.**GAuthClient**(*keyfile=None*, *scopes=None*, *session=None*, *loop=None*)

Async client to authenticate against Google Cloud APIs.

SCOPE_TMPL_URL

str – template URL for Google auth scopes.

DEFAULT_SCOPE

str – default scope if not provided.

JWT_GRANT_TYPE

str – grant type header value when requesting/refreshing an access token.

Parameters

- **keyfile** (*str*) – path to service account (SA) keyfile.
- **scopes** (*list*) – (optional) scopes with which to authorize the SA. Default is `'cloud-platform'`.
- **session** (*aiohttp.ClientSession*) – (optional) `aiohttp` HTTP session to use for sending requests.
- **loop** – (optional) asyncio event loop to use for HTTP requests. NOTE: if `session` is given, then `loop` will be ignored. Otherwise, `loop` will be used to create a session, if provided.

refresh_token()

Refresh oauth access token attached to this HTTP session.

Raises

- *GCPAuthError* – if no token was found in the response.
- *GCPHTTPError* – if any exception occurred, specifically a *GCPHTTPResponseError*, if the exception is associated with a response status code.

google.auth._default.**default**(*scopes=None*, *request=None*, *quota_project_id=None*, *default_scopes=None*)

Gets the default credentials for the current environment.

Application Default Credentials provides an easy way to obtain credentials to call Google APIs for server-to-server or local applications. This function acquires credentials from the environment in the following order:

1. If the environment variable `GOOGLE_APPLICATION_CREDENTIALS` is set to the path of a valid service account JSON private key file, then it is loaded and returned. The project ID returned is the project ID defined in the service account file if available (some older files do not contain project ID information).

   If the environment variable is set to the path of a valid external account JSON configuration file (workload identity federation), then the configuration file is used to determine and retrieve the external credentials from the current environment (AWS, Azure, etc). These will then be exchanged for Google access tokens via the Google STS endpoint. The project ID returned in this case is the one corresponding to the underlying workload identity pool resource if determinable.

   If the environment variable is set to the path of a valid GDCH service account JSON file (Google Distributed Cloud Hosted), then a GDCH credential will be returned. The project ID returned is the project specified in the JSON file.

2. If the Google Cloud SDK is installed and has application default credentials set they are loaded and returned.

   To enable application default credentials with the Cloud SDK run:

   ```
   gcloud auth application-default login
   ```

   If the Cloud SDK has an active project, the project ID is returned. The active project can be set using:

   ```
   gcloud config set project
   ```

3. If the application is running in the App Engine standard environment (first generation) then the credentials and project ID from the App Identity Service are used.

4. If the application is running in Compute Engine or Cloud Run or the App Engine flexible environment or the App Engine standard environment (second generation) then the credentials and project ID are obtained from the Metadata Service.

5. If no credentials are found, `DefaultCredentialsError` will be raised.

Example:

```
import google.auth

credentials, project_id = google.auth.default()
```

Parameters

- **`scopes`** (*`Sequence[str]`*) – The list of scopes for the credentials. If specified, the credentials will automatically be scoped if necessary.
- **`request`** (*`Optional[google.auth.transport.Request]`*) – An object used to make HTTP requests. This is used to either detect whether the application is running on Compute Engine or to determine the associated project ID for a workload identity pool resource (external account credentials). If not specified, then it will either use the standard library http client to make requests for Compute Engine credentials or a google.auth.transport.requests.Request client for external account credentials.
- **`quota_project_id`** (*`Optional[str]`*) – The project ID used for quota and billing.
- **`default_scopes`** (*`Optional[Sequence[str]]`*) – Default scopes passed by a Google client library. Use 'scopes' for user-defined scopes.

Returns the current environment's credentials and project ID. Project ID may be None, which indicates that the Project ID could not be ascertained from the environment.

Return type Tuple[Credentials, Optional[str]]

Raises `DefaultCredentialsError` – If no credentials were found, or if the credentials found were invalid.

4.8.4 GCP Cloud DNS HTTP Client

Client module to interact with the Google Cloud DNS API.

This client makes use of the asynchronus HTTP client as defined in *`AIOConnection`*, and therefore must use service account/JWT authentication (for now).

To use:

```
import asyncio
import gordon_gcp

keyfile = '/path/to/service_account_keyfile.json'
auth_client = gordon_gcp.GAuthClient(keyfile=keyfile)
client = gordon_gcp.GDNSClient(
    project='my-dns-project', auth_client=auth_client)

async def print_first_record(client)
    records = await client.get_records_for_zone('testzone')
    print(records[0])

loop = asyncio.get_event_loop()
loop.run_until_complete(print_first_record(client))
```

class `gordon_gcp.`**`GDNSClient`**(*project=None*, *auth_client=None*, *api_version='v1'*, *session=None*, *default_zone_prefix=None*)

Async HTTP client to interact with Google Cloud DNS API.

`BASE_URL`

str – base call url for the DNS API

Parameters

- **project** (*`str`*) – Google project ID that hosts the managed DNS.
- **auth_client** (`GAuthClient`) – client to manage authentication for HTTP API requests.
- **api_version** (*`str`*) – DNS API endpoint version. Defaults to `v1`.
- **session** (*`aiohttp.ClientSession`*) – (optional) `aiohttp` HTTP session to use for sending requests. Defaults to the session object attached to `auth_client` if not provided.

`get_managed_zone`(*zone*)

Get the GDNS managed zone name for a DNS zone.

Google uses custom string names with specific requirements for storing records. The scheme implemented here chooses a managed zone name which removes the trailing dot and replaces other dots with dashes, and in the case of reverse records, uses only the two most significant octets, prepended with 'reverse'. At least two octets are required for reverse DNS zones.

Example

get_managed_zone('example.com.') = 'example-com' get_managed_zone('20.10.in-addr.arpa.) = 'reverse-20-10' get_managed_zone('30.20.10.in-addr.arpa.) = 'reverse-20-10' get_managed_zone('40.30.20.10.in-addr.arpa.) = 'reverse-20-10'

Parameters **zone** (*str*) – DNS zone.

Returns str of managed zone name.

get_records_for_zone (*dns_zone*, *params=None*)
Get all resource record sets for a managed zone, using the DNS zone.

Parameters

- **dns_zone** (*str*) – Desired DNS zone to query.
- **params** (*dict*) – (optional) Additional query parameters for HTTP requests to the GDNS API.

Returns list of dicts representing rrsets.

is_change_done (*zone*, *change_id*)
Check if a DNS change has completed.

Parameters

- **zone** (*str*) – DNS zone of the change.
- **change_id** (*str*) – Identifier of the change.

Returns Boolean

publish_changes (*zone*, *changes*)
Post changes to a zone.

Parameters

- **zone** (*str*) – DNS zone of the change.
- **changes** (*dict*) – JSON compatible dict of a Change.

Returns string identifier of the change.

4.8.5 GCRM Client

Client classes to retrieve project and instance data from GCE.

These clients use the asynchronous HTTP client defined in *AIOConnection* and require service account or JWT-token credentials for authentication.

To use:

```
import asyncio

import aiohttp
import gordon_gcp

loop = asyncio.get_event_loop()

async def main():
    session = aiohttp.ClientSession()
    auth_client = gordon_gcp.GAuthClient(
        keyfile='/path/to/keyfile', session=session)
    client = gordon_gcp.GCEClient(auth_client, session)
    instances = await client.list_instances('project-id')
    print(instances)
```

```
loop.run_until_complete(main())
# example output
# [{'hostname': 'instance-1', 'internal_ip': '10.10.10.10',
#   'external_ip': '192.168.1.10'}]
```

class gordon_gcp.**GCRMClient**(*auth_client=None*, *session=None*, *api_version='v1'*)
Async client to interact with Google Cloud Resource Manager API.

You can find the endpoint documentation here.

BASE_URL
str – Base endpoint URL.

Parameters

- **auth_client** (GAuthClient) – client to manage authentication for HTTP API requests.
- **session** (*aiohttp.ClientSession*) – (optional) `aiohttp` HTTP session to use for sending requests. Defaults to the session object attached to `auth_client` if not provided.
- **api_version** (*str*) – version of API endpoint to send requests to.

list_all_active_projects(*page_size=1000*)
Get all active projects.

You can find the endpoint documentation here.

Parameters **page_size** (*int*) – hint for the client to only retrieve up to this number of results per API call.

Returns all active projects

Return type list(dicts)

4.8.6 GCE Client

Client classes to retrieve project and instance data from GCE.

These clients use the asynchronous HTTP client defined in *AIOConnection* and require service account or JWT-token credentials for authentication.

To use:

```
import asyncio

import aiohttp
import gordon_gcp

loop = asyncio.get_event_loop()

async def main():
    session = aiohttp.ClientSession()
    auth_client = gordon_gcp.GAuthClient(
        keyfile='/path/to/keyfile', session=session)
    client = gordon_gcp.GCEClient(auth_client, session)
    instances = await client.list_instances('project-id')
    print(instances)

loop.run_until_complete(main())
```

```
# example output
# [{'hostname': 'instance-1', 'internal_ip': '10.10.10.10',
#   'external_ip': '192.168.1.10'}]
```

class gordon_gcp.**GCEClient**(*auth_client=None*, *session=None*, *api_version='v1'*, *blacklisted_tags=None*, *blacklisted_metadata=None*)

Async client to interact with Google Cloud Compute API.

BASE_URL

str – base compute endpoint URL.

Parameters

- **auth_client** (GAuthClient) – client to manage authentication for HTTP API requests.
- **session** (*aiohttp.ClientSession*) – (optional) aiohttp HTTP session to use for sending requests. Defaults to the session object attached to auth_client if not provided.
- **api_version** (*str*) – version of API endpoint to send requests to.
- **blacklisted_tags** (*list*) – Do not collect an instance if it has been tagged with any of these.
- **blacklisted_metadata** (*list*) – Do not collect an instance if its metadata key:val matches a {key:val} dict in this list.

list_instances(*project*, *page_size=100*, *instance_filter=None*)

Fetch all instances in a GCE project.

You can find the endpoint documentation here.

Parameters

- **project** (*str*) – unique, user-provided project ID.
- **page_size** (*int*) – hint for the client to only retrieve up to this number of results per API call.
- **instance_filter** (*str*) – endpoint-specific filter string used to retrieve a subset of instances. This is passed directly to the endpoint's "filter" URL query parameter.

Returns

data of all instances in the given project

Return type list(dicts)

4.9 Exceptions

Any exception that this module could throw.

class gordon_gcp.**GCPGordonError**

General Gordon GCP Plugin Error.

class gordon_gcp.**GCPGordonJanitorError**

General Gordon GCP Janitor Plugin Error.

class gordon_gcp.**InvalidMessageError**

Consumed an invalid message from Google Pub/Sub.

class gordon_gcp.**InvalidDNSZoneInMessageError**
Raised when a message with an invalid DNS zone is consumed.

class gordon_gcp.**GCPHTTPError**
An error occurred while processing an HTTP request.

class gordon_gcp.**GCPHTTPResponseError**(*message*, *status*)
An HTTP response had an error associated with a status code.

class gordon_gcp.**GCPAuthError**
Authentication error with Google Cloud.

class gordon_gcp.**GCPConfigError**
Improper or incomplete configuration for plugin.

class gordon_gcp.**GCPPublishRecordTimeoutError**
Time out error when attempting to publish records.

CHAPTER 5

Project Information

5.1 License and Credits

`gordon-gcp` 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`, `gordon-janitor`, and their plugins are 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 `develop`.
- 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.dev38 (2021-04-09)

Fixed

- Update aiohttp due to a security bug

5.3.2 0.0.1.dev37 (2020-06-01)

Changed

- Make keyfile optional

5.3.3 0.0.1.dev36 (2020-05-08)

Changed

- Ensure to use correct email when calling metadata token url

5.3.4 0.0.1.dev35 (2020-04-30)

Changed

- Support compute engine credentials when refreshing token.

5.3.5 0.0.1.dev34 (2020-04-20)

Changed

- Fallback to default token URI if not set.

5.3.6 0.0.1.dev33 (2020-04-17)

Changed

- Widen regex in auditlog schema to work with more use cases.

5.3.7 0.0.1.dev32 (2019-02-26)

Changed

- Exit (code=1) on failure to create pubsub subscription.

Fixed

- Fix reconciler bug introduced in v0.0.1.dev31.

5.3.8 0.0.1.dev31 (2019-02-18)

Updated

- Adjust metrics for graph optimization and to include source in janitor.

5.3.9 0.0.1.dev30 (2018-12-12)

Added

- Add metrics for record dispatching.

5.3.10 0.0.1.dev29 (2018-12-05)

Added

- Add project_whitelist config to janitor authority.

5.3.11 0.0.1.dev28 (2018-10-19)

Added

- Allow supplying custom managed zone name prefix.

5.3.12 0.0.1.dev27 (2018-10-16)

Removed

- Revert filtering out instances with status = TERMINATED in the authority.

5.3.13 0.0.1.dev26 (2018-10-15)

Added

- Filter out instances with status = TERMINATED in the authority.

Changed

- Lower minimum TTL for the event schema to 60.

Fixed

- Correct an external URL used in doc generation.

5.3.14 0.0.1.dev25 (2018-09-20)

Removed

- Removed superfluous jsonschema validation logging.

5.3.15 0.0.1.dev23 & 0.0.1.dev24 (2018-09-12/13)

Added

- Add logging on aiohttp request calls.

5.3.16 0.0.1.dev22 (2018-09-10)

Fixed

- Fix incorrect function call in GDNS publisher.

5.3.17 0.0.1.dev21 (2018-09-07)

Changed

- Bump upstream Gordon package requirements.

5.3.18 0.0.1.dev20 (2018-09-07)

Fixed

- Fix bug in date comparison in drop-old-message functionality.

Added

- Add metrics to the janitor.

Changed

- Speed up the reconciler.

5.3.19 0.0.1.dev19 (2018-08-14)

Fixed

- Fix paging of GDNS responses.
- Make setup.py work with direct github commit links.

Added

- Drop messages older than a limit.

Changed

- Remove managed zone from plugin configuration in favor of automated conversion from DNS zone.
- Do DNS zone to managed zone conversion only in GDNSClient.

5.3.20 0.0.1.dev18 (2018-08-01)

Added

- Add HTTP 403 response code to janitor PROJECT_SKIP_RESP_CODES constant.

Changed

- Default max results to 100 when listing instances in GCE client.

5.3.21 0.0.1.dev17 (2018-07-27)

Added

- Add callback in pubsub publisher.

5.3.22 0.0.1.dev16 (2018-07-26)

Fixed

- Fix incorrect and superfluous authority logging.

Added

- Add deletions to the janitor reconciler.

Changed

- Simplify HTTP error response handling.
- Fail authority if it cannot get a full view of of all instances.

Removed

- Removed GCPHTTPNotFoundError and GCPHTTPConflictError.

5.3.23 0.0.1.dev15 (2018-07-17)

Changed

- Increase level of detail in HTTP request/response logging.

Fixed

- Properly support ‘deletions’ action.

5.3.24 0.0.1.dev14 (2018-07-10)

Added

- Add `kind` attribute to `GCPResourceRecordSet` object.
- Add request concurrency to GCE listing of instances.

5.3.25 0.0.1.dev13 (2018-07-03)

Changed

- Update gordon-cloud-pubsub version to `0.35.4`.

Removed

- Remove the use of `_GPSThreads`.

5.3.26 0.0.1.dev12 (2018-06-28)

Fixed

- Clean up GPThread instances once done.

5.3.27 0.0.1.dev11 (2018-06-25)

Changed

- Janitor: Skip project if listing instances fails.
- Extract response rrsets properly.
- Make params optional when calling http.get_all.

5.3.28 0.0.1.dev10 (2018-06-20)

Changed

- Updated the Google API compute v1 endpoint URL.

5.3.29 0.0.1.dev9 (2018-06-20)

Added

- Add threadsafety when adding a message to the success channel from `GPSEventConsumer`.
- Add flow control when consuming from Pub/Sub.

Changed

- Update interface implementation of `GEventMessage`.

Removed

- Remove date validation in schemas.

5.3.30 0.0.1.dev8 (2018-06-18)

Changed

- Reorder args for GCEEnricher.

5.3.31 0.0.1.dev7 (2018-06-15)

Changed

- Update gordon-dns to 0.0.1.dev3.

Removed

- Remove routing logic from plugins.

5.3.32 0.0.1.dev6 (2018-06-07)

Changed

- Internal API improvements.

5.3.33 0.0.1.dev5 (2018-06-07)

Changed

- Fix failure for core to instantiate GDNSPublisher plugin.
- Internal API improvements.

5.3.34 0.0.1.dev4 (2018-06-05)

Added

- Merged gordon-janitor-gcp repo into gordon-gcp.
- Added janitor plugin summaries.
- Added missing exception docs.

Changed

- Updated and fixed OWNERS.
- Cleaned up some capitalizations and wordings.
- Suppressed a test warning.
- Fixed namespace collapses (`__all__` / `import *`).

Added

- Add implementation of IEventConsumer.
- Add implementation of IPublisher.
- Add implementation of IEnricher.
- Add support on loading credentials with application default credentials.
- Add support for `POST` JSON requests to HTTP client.

5.3.35 0.0.1.dev2 (2018-03-29)

Changed

Fixed packaging.

5.3.36 0.0.1.dev1 (2018-03-28)

Changed

Initial development release.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index