Cloudkitty Specs

Release 0.0.1.dev59

OpenStack Cloudkitty Team

Mar 06, 2023

CONTENTS

CHAPTER

ONE

ANTELOPE

1.1 Set a time to live for rating rules on modules Hashmap and Pyscript.

This specification proposes an extension in the rating rules to add a life time to the rules and add audit.

1.1.1 Problem Description

In a dynamic cloud environment where computing resources are constantly replaced to improve the cloud platform, we see the computing resources prices being modified with the cloud environment upgrades; meaning, old resources become cheaper, and users can choose the newest available resources which usually are more expensive.

For that purpose, the price of computing resources is continually changing, which means that they can change every year, or even more often according to the cloud upgrades.

Nowadays in CloudKitty when we create rating rules (for both hash mapping and Pyscripts), the rules are applied immediately after they were created, and they keep working until they are deleted; the update of rating rules is also possible, and we can do it whenever we want.

To keep the prices of the resources updated, operators need to change the cost of each rating rule created, which causes some headaches when some are missed/forgotten to be updated when they should. This generated the need to execute data reprocessing. Besides that, as the rating rules updates take effect right when they are updated, operators have to work at a specific moment in time, which is not something very practical. For instance, if the price changes at the very beginning of the year, which would force somebody to update the rules at midnight of January of the new year, and so on.

1.1.2 Proposed Change

To facilitate operators work and make the Cloudkitty rating rules creation process more dynamic and flexible, we propose to add a start and end date in the hashmap and Pyscript rating rules modules.

Furthermore, as we are working with billing data, which will be used to charge the cloud environment users, we also propose to create an auditing mechanism for the rating rules; to achieve that, we will stop deleting the rating rules from the database; instead of it, we will mark them as deleted and at the same time, we will start denying the rule updating process in some specific situations. The goal of these changes is to make the system more secure, auditable, and predictable. Moreover, we will add three new columns, one to store the user that created the rule, one with the user that removed it, and the last one with the user that updated it.

The update is only going to be allowed if the rules have never been used to rate a metric by CloudKitty. If it has already been used, we will not allow updating it. The rationale is that we cannot change a rule while it is being used, so we maintain it consistent. Operators will always be able to delete a rating rule to be able to remove it from processing and reprocessing. If rating rules have None as the end date, we then allow an update (for rules that have not been used yet), which is the only update that can be set for rating rules that are in use.

For rating rules that have not been used yet, we allow operators to update anything they wish, including price, start, and end dates.

Another process that we will allow operators to do is to create rating rules that start in a past time. As long as they send an extra parameter saying that they understand that the data point that is in the past will not have this rule to be applied if a reprocessing is not executed.

1.1.3 Database changes

Here are presented the proposed changes in Cloudkitty database schema.

Add new columns in the tables *hashmap_mappings* and *pyscripts_scripts*:

- created_at: timestamp defining when the rule is created;
- start: timestamp defining when the rule starts to be applied by CloudKitty;
- end: timestamp defining when the rule ends/finishes/is not used anymore by CloudKitty. If NULL is used, it means endless; in other words, it is always applied;
- name: the rating rule name which identifies the rule;
- description: a rating rule description;
- deleted: deletion timestamp;
- created_by: keystone users id who created the rule;
- updated_by: last keystone users id who changed the rule;
- deleted_by: keystone users id who deleted the rule.

```
+---------------------+-------------+------+---------------------+
| Field               | Type        | Null | Default             |
+---------------------+-------------+------+---------------------+
| created_at          | timestamp   | NO   | CURRENT_TIMESTAMP   |
| start               | timestamp   | NO   | CURRENT_TIMESTAMP   |
| end                 | timestamp   | YES  |                     |
| name                | varchar(32) | NO   |                     |
| description         | varchar(256)| YES  |                     |
| deleted             | timestamp   | YES  |                     |
| created_by          | varchar(32) | NO   |                     |
| updated_by          | varchar(32) | YES  |                     |
| deleted_by          | varchar(32) | YES  |                     |
+---------------------+-------------+------+---------------------+
```

1.1.4 REST API

Here are presented the proposed changes in Cloudkitty REST API.

- path: /v1/rating/module_config/hashmap/mappings

  method: GET

  changes:

  - Filter by rating rule time to live window;
  - Filter by user who created/updated/deleted the rules;
  - Filter by description;
  - Show or not, via a flag, the deleted rating rules (marked as deleted with the deletion date);
  - Show or not, via a flag, the active rating rules (start and end dates period containing the current date);

- path: /v1/rating/module_config/hashmap/mappings

  method: POST

  changes:

  - Four new values were added when creating rules:

    start: the timestamp the rule will take effect; if none is provided, the current timestamp will be used; if a date without time is provided, the midnight time (00:00:00) will be used; if the timezone is not provided, we use the timezone of the system. Its value must be < than the provided end date and cannot be in the past;

    end: the timestamp the rule will have no more effect; if none is provided, it will be none, meaning that it is an endless rule (rule without an end date); if a date without time is provided, the next midnight day minus 1 minute (23:59:00) will be used; if the timezone is not provided, we use the timezone of the system. Its value must be > than the provided start and cannot be in the past;

    name: the rule name, it is a required value, must be a string. The name must be unique for all not deleted rules (not marked as deleted);

    description: the rule description, is a optional parameter, and must be a string;

    force: optional value to allow users to create rules with a start and end in the past, for reprocessing purposes;

  - A new value will be processed in background when creating rules:

    created_by: We will set this value with the Keystones authentication token used in the request, we will get the user id from the token and set it in this field;

- path: /v1/rating/module_config/hashmap/mappings

  method: PUT

  changes:

  - We will allow to update only the rules end date, nothing else, and only if the current end date is not defined (none). The new end value must be something in the future.

  - It will be also allowed to update rules where the start date is still in the future, for that case, the attributes allowed to be updated will be the:
    * start: if the provided one is in the future and < than end;
    * end: if the provided one is in the future and > than start;
    * cost;
    * description;
  - A new value will be processed in background when updating rules:

    updated_by: We will set this value with the Keystones authentication token used in the request, we will get the user id from the token and set it in this field;

- path: /v1/rating/module_config/hashmap/mappings

  method: DELETE

  changes:

  - We will not delete a rule anymore, instead, this endpoint will mark a rule as deleted, using the current date the request was called.
  - A new value will be processed in background when deleting rules:

    deleted_by: We will set this value with the Keystones authentication token used in the request, we will get the user id from the token and set it in this field;

- path: /v1/rating/module_config/pyscripts/scripts

  method: GET

  changes:

  - Filter by rating rule time to live window;
  - Filter by user who created/updated/deleted the rules;
  - Filter by description;
  - Show or not, via a flag, the deleted rating rules (marked as deleted with the deletion date);
  - Show or not, via a flag, the active rating rules (start and end dates period containing the current date);

- path: /v1/rating/module_config/pyscripts/scripts

  method: POST

  changes:

  - Four new values were added to request when creating rules:

    start: the timestamp the rule will take effect; if none is provided, the current timestamp will be used; if a date without time is provided, the midnight time (00:00:00) will be used; if the timezone is not provided, we use the timezone of the system. Its value must be < than the provided end date and cannot be in the past;

    end: the timestamp the rule will have no more effect; if none is provided, it will be none, meaning that it is an endless rule (rule without an end date); if a date without time is provided, the next midnight day minus 1 minute (23:59:00) will be used; if

the timezone is not provided, we use the timezone of the system. Its value must be > than the provided start and cannot be in the past;

name: the rule name, it is a required value, must be a string. The name must be unique for all not deleted rules (not marked as deleted);

description: the rule description, is a optional parameter, and must be a string;

force: optional value to allow users to create rules with a start and end in the past, for reprocessing purposes;

- A new value will be processed in background when creating rules:

  created_by: We will set this value with the Keystones authentication token used in the request, we will get the user id from the token and set it in this field;

- path: /v1/rating/module_config/pyscripts/scripts

  method: PUT

  changes:

  - We will allow to update only the rules end date, nothing else, and only if the current end date is not defined (none). The new end value must be something in the future.
  - It will be also allowed to update rules where the start date is still in the future, for that case, the attributes allowed to be updated will be the:
    - start: if the provided one is in the future and < than end;
    - end: if the provided one is in the future and > than start;
    - cost;
    - description;
  - A new value will be processed in background when updating rules:

    updated_by: We will set this value with the Keystones authentication token used in the request, we will get the user id from the token and set it in this field;

- path: /v1/rating/module_config/pyscripts/scripts

  method: DELETE

  changes:

  - We will not delete a rule anymore, instead, this endpoint will mark a rule as deleted, using the current date the request was called.
  - A new value will be processed in background when deleting rules:

    deleted_by: We will set this value with the Keystones authentication token used in the request, we will get the user id from the token and set it in this field;

1.1.5 Workflows

Here are presented the proposed changes in the processing and reprocessing workflows in Cloudkitty orchestrator.

1.1.6 Processing

- We will have three more filters (besides the tenant_id) when checking which rules must be used to process the data frame cost:
  - check if the rules start timestamp is <= than the current timestamp; and
  - check if the rules end timestamp is > than the current timestamp; and
  - check if there is no timestamp set in the deleted field of the rule;

Besides the changes introduced in this proposal, which we will filter the rules processed, the rest of the processing workflow will stay the same.

1.1.7 Reprocessing

- We will have three more filters (besides the tenant_id) when checking which rules must be used to process the data frames cost:
  - check if the rules start timestamp is <= than the processed data frame timestamp; and
  - check if the rules end timestamp is > than the processed data frame timestamp; and
  - check if there is no timestamp set in the deleted field of the rule;

Besides the changes introduced in this proposal, which we will filter the rules processed, the rest of the reprocessing workflow will stay the same.

1.1.8 Final thoughts

With this extension, if one needs to update any rule that has already been in use (the start value is in the past), we will need to delete it and create a new one with the updated values; alternatively, one can set the end date, and then create a new one. By using this approach we can track all changes a rule has, which facilitated auditing the system.

If all rules get expired, then we will start rating all the resources with 0 (zero) value. However, that would be expected, as there is no valid rating rules anymore. To avoid that, one can always use rules without and end date.

Alternatives

Create some cron triggers to update the rating rules cost in the server;

Data model impact

New columns will be added in the tables *hashmap_mappings* and *pyscripts_scripts*;

REST API impact

New fields will be added in the */v1/rating/module_config/hashmap/mappings* and */v1/rating/module_config/pyscripts/scripts* endpoints

Security impact

None

Notifications Impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

1.1.9 Implementation

Assignee(s)

Primary assignee:

- Pedro Henrique Pereira Martins <phpm13@gmail.com>

Work Items

1) Extend the database model;
2) Create the data migration process to fill the initial start dates;
3) **Adapt the REST APIs to accept the new fields;** 3.1) Adapt Hashmap mappings REST API; 3.2) Adapt Pyscript REST API;
4) Create the validation mechanism to the new fields;
5) Change the deletion REST API to not delete the rules anymore;
6) Adapt the processing workflow to the new rules/validations;
7) Adapt the reprocessing workflow to the new rules/validations;
8) Change Cloudkitty CLI to accept new fields.

1.1.10 Dependencies

None

1.1.11 Testing

Unit tested

1.1.12 Documentation Impact

None

1.1.13 References

None

CHAPTER

TWO

YOGA

2.1 Introduce reprocessing API

This specification proposes an API to enable reprocessing.

2.1.1 Problem Description

Sometimes due to a misconfiguration in rating rules, or due to backfill of data in the collector backend (Gnocchi, Prometheus, or others), operators need to reprocess CloudKitty data. The reprocessing process that is normally taken is the following.

- Stop all CloudKitty processors,
- reset the *state* (last processed/rated timestamp) in MySQL database for all scopes that need reprocessing,
- clean/delete all processed/rated data for the affected timeframe in the processed data backend (e.g. InfluxDB),
- restart CloudKitty processors.

CloudKitty has an API already to reset the processing/rating state of a scope[1]. However, this API only resets the state in the MySQL database. Therefore, it is important to have an API that implements the manual steps that operators are normally executing when reprocessing is required.

2.1.2 Proposed Change

To avoid human error during reprocessing, and also to make it easier for non-CloudKitty experts to execute it, we propose the introduction of a reprocessing API that schedules the reprocessing and also removes the processed/rated data for the affected timeframe from the backend (e.g. InfluxDB).

To achieve that, we will use a new table to store the reprocessing progress for scopes, the proposed table name is `storage_scope_reprocessing_schedule`.

```
+------------------------------+--------------+------+-----+------------------
↪-+----------------+
| Field                        | Type         | Null | Key | Default          ␣
↪ | Extra          |
+------------------------------+--------------+------+-----+------------------
↪-+----------------+
```

(continues on next page)

[1] https://docs.openstack.org/cloudkitty/latest/api-reference/v2/index.html?expanded=get-a-rating-summary-detail,reset-the-status-of-several-scopes-detail#reset-the-status-of-several-scopes

(continued from previous page)

```
| id                          | int(11)       | NO   | PRI | NULL                ␣
↪ | auto_increment |
| identifier                  | varchar(256)  | NO   | MUL | NULL                ␣
↪ | FK             |
| start_reprocess_time        | datetime      | NO   |     | -                   ␣
↪ |                |
| end_reprocess_time          | datetime      | NO   |     | -                   ␣
↪ |                |
| current_reprocess_time      | datetime      | yes  |     | NULL                ␣
↪ |                |
| reason                      | text          | NO   |     | NULL                ␣
↪ |                |
+-----------------------------+---------------+------+-----+---------------------
↪-+----------------+
```

The endpoint for the schedule reprocessing API will be:

- `POST /v2/task/reprocesses`
  - `scope_id`: a list of scope IDs to schedule reprocessing to.
  - `start_reprocess_time`: a timestamp to start reprocessing.
  - `end_reprocess_time`: a timestamp to end reprocessing.
  - `reason`: the reason for the reprocessing
- `GET /v2/task/reprocesses/<scope_id>` to retrieve the reprocessing schedules for a single scope.
- `GET /v2/task/reprocesses` - to retrieve the reprocessing schedules for multiple scopes

The `/v2/task/reprocesses` endpoint will receive a POST request similar to the following:

```
{
  "scope_id": ["scope_id_one", "scope_id_two", ...],
  "start_reprocess_time": "2021-05-01 00:00:00",
  "end_reprocess_time": "2021-05-30 23:00:00"
  "reason": "The reason why this reprocessing is scheduled."
}
```

The reason field is mandatory to force users to explain why the reprocessing is scheduled. We are forcing users to register why they are taking such drastic measures, such as reprocessing. Therefore, we hold a history of the scheduled reprocessing, and their respective reasons/explanations.

If one of the scope IDs informed via `scope_id` does not exist, we will raise an exception telling the operator that there are invalid scopes ID in the request.

One can only schedule reprocessing for timestamps of scopes that have already been processed. Therefore, if the *end_reprocess_time* is after the latest processing timestamp for a given resource, an error is thrown in the API.

One cannot schedule overlapping reprocessing. Therefore, it is only possible to reprocess a reprocessing of a scope, when the previously scheduled reprocessing is finished (if there is an overlapping of start/end timestamps).

After the schedule has been created, the CloudKitty processors will clean the affected time range for the given resource, and then move on with the reprocessing. Every time a timestamp is reprocessed, the `current_reprocess_time` is updated, similarly to the current processing workflow.

When the `current_reprocess_time` equals to `end_reprocess_time`, it means that the scheduled reprocessing has finished. Therefore, The reprocessing workers will do nothing for this entry.

We will also introduce an endpoint to consult the status of the scheduled reprocessing.

- To list all schedules: `GET /v2/task/reprocesses`
  - `scope_id`: optional field to filter scopes that have been scheduled for reprocessing. If no scope is provided, we list all.

Alternatives

Manually execute the reprocessing steps previously described.

Data model impact

A new table (`storage_scope_reprocessing_schedule`) will be added to the database.

REST API impact

A new API will be introduced: `/v2/task/reprocesses`.

Security impact

Only admins must have access to this new API

Notifications Impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

2.1.3 Implementation

Assignee(s)

Primary assignee:

- Rafael <rafael@apache.org>

Work Items

1) propose, discuss, and merge the spec
2) execute the implementations as described.
3) implement changes in the CloudKitty client to support the new API

2.1.4 Dependencies

None

2.1.5 Testing

Unit tested

2.1.6 Documentation Impact

None

2.1.7 References

CHAPTER

THREE

WALLABY

3.1 Add active column to storage scope, and API to manage it

This spec proposes an extension for the scope state endpoint. The proposal goal is to add a new option called active in the storage scope table (cloudkitty_storage_states), and an API that enables operators to manage it.

3.1.1 Problem Description

When using Gnocchi as a fetcher (so, we can process resources that are not just OpenStack ones), we noticed that CloudKitty keeps processing resources that have already been deleted in the systems they were created. Therefore, they do not have measures in Gnocchi anymore. This causes a slowdown in the processing of CloudKitty.

To present some figures, in a production environment, there are only ~330 resources that we want to process, but in CloudKitty storage states table, we can see ~990 (this number just grows with time), and CloudKitty is processing all of them.

During our meeting on January 11 (http://eavesdrop.openstack.org/meetings/cloudkitty/2021/cloudkitty.2021-01-11-14.00.html), we discussed that maybe we could do that automatically. However, when I reviewed the code changes that I needed, this would not be that simple. A project in OpenStack could, for instance, have a single VM, and then this VM could be deleted, and the project would not have measures anymore. Therefore, we would mark it as inactive. Then, if the project receives VMs in the future they would not be billed. Therefore, we would need to introduce a method to change resources from inactive to active states, which is not that simple, and therefore, this other mechanism would need to somehow process inactive resources at some point in time to check if they still do not have measures in Gnocchi.

Therefore, we came up with a simpler solution.

3.1.2 Proposed Change

The proposal is to add two new fields in the storage state table (cloudkitty_storage_states). A boolean column called active, which indicates if the CloudKitty scope is active for billing, and another one called scope_activation_toggle_date (timestamp field) would store the latest timestamp when the scope moved between the active/deactivated states.

Then, during CloudKitty processing, we would check the active column. If the resource is not active, we could ignore it during the processing.

Moreover, we propose an API to allow operators to set the active field. The scope_activation_toggle_date will not be exposed for operators to change it. It would be updated automatically according to the changes in the active field.

The proposal will add a new HTTP method to /v2/scope endpoint. We then will use patch HTTP method to allow operators to patch a storage scope. The API will require the scope_id, and then, it takes into account some of the fields we allow operators to change, and active field is one of them.

Assignee(s)

Primary assignee: Rafael Weingärtner <rafael@apache.org>

Work Items

1) Implement proposed changes
2) Update documentation and samples
3) implement changes in CloudKitty and OpenStack python clients

3.1.3 Dependencies

None

3.2 Deprecate *state* field and propose *last_processed_at* field

This spec proposes a name change for the *state* field in *cloudkitty_storage_states* table. The goal is to use a more descriptive and meaningful name. The proposed new column name is *last_processed_at*.

3.2.1 Problem Description

The *state* column in *cloudkitty_storage_states* table holds the timestamp of the last processing cycle for a scope. The problem with that name is that it does not represent the data it stores. A column name like *state* gives the expectation that we would have some info about the resource such as *active/disabled/enabled* or something similar. However, that is not what we get in the *state* column.

3.2.2 Proposed Change

Therefore, to avoid confusions, and to adopt a more consistent naming for this variable, we propose to rename the column to *last_processed_at*. Moreover, we propose to deprecate the wording *state* in the scope API, therefore, instead of using *state* to represent the last processing timestamp, we would use *last_processed_at*.

Alternatives

Other names can be considered. It is up for other people to propose, and then we can discuss and try to reach a consensus.

Data model impact

Rename the column *state* in the *cloudkitty_storage_states* table to *last_processed_at*.

REST API impact

Introduce the new field *last_processed_at* alongside the *state* field. Then, we announce its deprecation and remove it completely in after Wallaby.

Security impact

None

Notifications Impact

Renaming of the *state* field in the *cloudkitty_storage_states*, and as a consequence in the scope API, we will deprecate it and introduce a new field along side the *state* attribute.

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None Implementation ==========

Assignee(s)

Primary assignee:

- Rafael <rafael@apache.org>

Work Items

1) propose, discuss, and merge the spec
2) execute the implementations to rename the column, introduce the new attribute in the API, and then execute a warning against the use of the *state* variable.
3) implement changes in the CloudKitty client
4) After Wallaby, we remove the *state* field from the API

3.2.3 Dependencies

None

3.2.4 Testing

Unit tested

3.2.5 Documentation Impact

None

3.2.6 References

None so far

CHAPTER

FOUR

USSURI

4.1 Adding some Prometheus query functions to the Prometheus collector

https://storyboard.openstack.org/#!/story/2006427

4.1.1 Problem Description

The Prometheus collector cannot handle `Counter` and `Gauge` data types.

A `Counter` is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart. For example, you can use a counter to represent the number of requests served, tasks completed, or errors.

A `Gauge` is a metric that represents a single numerical value that can arbitrarily go up and down.

4.1.2 Proposed Change

If the delta function was used, it would be possible to get the difference between two collect cycles, and therefore to charge metrics of the `Counter` and `Gauge` type.

Adding two fields (`range_function` and `query_function`) to the `extra_args` section of the `metrics.yml` file would solve this problem.

Prometheus functions can be grouped in two categories, each matching one of the new fields:

Some functions take a `Range Vector` as argument and some take an `Instant vector`. Both categories return an `Instant vector`.

The new fields would only have a limited set of allowed functions because of the expected result format.

The following Prometheus query functions will be allowed:

Functions taking a `Range Vector`:

- `changes()`
- `delta()`
- `deriv()`
- `idelta()`
- `irange()`

- irate()
- rate()

Functions taking an Instant Vector:

- abs()
- ceil()
- exp()
- floor()
- ln()
- log2()
- log10()
- round()
- sqrt()

Functions accepting a Range vector will be will be set through the new range_function field. They will replace the current implicit {aggregation_method}_over_time function that is applied by the collector to obtain an Instant vector. The field will be optional and will default to {aggregation_method}_over_time.

Functions accepting an Instant vector will be set through the new query_function field. This will allow to apply an extra transformation to data before it is rated.

Both fields can be combined.

Example

The following config

```
metrics:
  gateway_function_invocation_total:
    unit: total
    groupby:
      - function_name
      - code
    extra_args:
      aggregation_method: max
      range_function: delta
```

would result in this query: max(delta(gateway_function_invocation_total{}[3600s])) by (function_name,code)

Another example:

```
metrics:
  gateway_function_invocation_total:
    unit: total
    groupby:
      - function_name
```

(continues on next page)

(continued from previous page)

```
      - code
    extra_args:
      aggregation_method: max
      range_function: delta
      query_function: abs
```

Would result in: `max(abs(delta(gateway_function_invocation_total{}[3600s]))) by (function_name,code)`

And this:

```
metrics:
  gateway_function_invocation_total:
    unit: total
    groupby:
      - function_name
      - code
    extra_args:
      aggregation_method: max
      query_function: abs
```

Would result in: `max(abs(max_over_time(gateway_function_invocation_total{}[3600s]))) by (function_name,code)`

Alternatives

The PyScript module could be used in some cases but its a bit complex just for some simple operations. It would also require to save the latest state of the metric (in case of the delta function).

Data model impact

None

REST API impact

None

Security impact

None

Notifications Impact

None

Other end user impact

End users will be able to perform more operations on metrics retrieved by the Prometheus collector especially on `Gauge` and `Counter` metrics.

Performance Impact

None

Other deployer impact

None

Developer impact

None

4.1.3 Implementation

Assignee(s)

Primary assignee: <aimbot31>

Work Items

- Add support for the `query_function` field to the Prometheus collector
- Add support for the `range_function` field to the Prometheus collector

4.1.4 Dependencies

None

4.1.5 Testing

The proposed changes will be tested with Unit Tests.

4.1.6 Documentation Impact

An entry detailing the configuration of the new field will be added to : `Admin/Configuration/Collector`.

4.1.7 References

- Prometheus functions documentation: https://prometheus.io/docs/prometheus/latest/querying/functions/
- Prometheus type documentation: https://prometheus.io/docs/prometheus/latest/querying/basics/

4.2 V1 API rating modules endpoints migration to V2 API

https://storyboard.openstack.org/#!/story/2006572

With the v1 API being frozen, migrating endpoints that are not present in the v2 API is required to achieve a complete v2 API with both existing v1 features and new v2 ones. This spec is about the `/v1/rating/modules` endpoints.

4.2.1 Problem Description

As the v1 API is planned to be deprecated, `/v1/rating/modules` endpoints must be ported to v2. The motivation for the v2 API is documented in the relevant spec.

4.2.2 Proposed Change

Implementing the existing `/v1/rating/modules` into the v2 API. This is part of the broader effort to migrate v1 endpoints to the v2 API.

Alternatives

None.

Data model impact

None.

REST API impact

This will add `/v2/rating/modules` endpoints similarly to the v1 API.

`GET /v2/rating/modules` returns the list of loaded modules. It does not require any parameter. The body of the response will contain the module list:

```
{
  "modules": [
    {
        "module_id": "f6f4f726-583z-4a09-a972-c908dea4c291"
        "description": "Sample extension.",
        "enabled": true,
        "hot-config": false,
        "priority": 2
    },
    [...]
  ]
}
```

`GET /v2/rating/modules/<module_id>` returns the module specified by the id in the url. It does not need any parameter. The body of the response contains a single module:

```
{
   "module_id": "f6f4f726-583z-4a09-a972-c908dea4c291",
   "description": "Sample extension.",
   "enabled": true,
   "hot-config": false,
   "priority": 2
}
```

`PUT /v2/rating/modules/<module_id>` changes the state and priority of a module. It requires fields to be updated to be present in the query body:

```
{
  "enabled": false,
  "priority": 42
}
```

For both GET endpoints, the expected response code is a 200 OK. For the PUT endpoint, the expected response is a 204 NO CONTENT as the body is empty. Expected HTTP error response codes for all 3 endpoints are:

- `400 Bad Request`: Malformed request.
- `401 Unauthorized`: The user is not authenticated.
- `403 Forbidden`: The user is not authorized.

Security impact

All added endpoints will keep the same policy as the v1 API: admin credentials are needed to query the API endpoints. Added endpoints only port existing functionality from v1 to v2 API and will not have any other security impact.

Notifications Impact

A notification reload triggered by queries on added endpoints is planned in the future.

Other end user impact

None.

Performance Impact

None.

Other deployer impact

None.

Developer impact

None.

4.2.3 Implementation

Assignee(s)

Primary assignee: qanglade/qanglade

Other contributors: lukapeschke/peschk_l

Work Items

- Implement the `/v1/rating/modules` endpoints in the v2 API, including unit tests and documentation
- Add functional tests to the tempest plugin
- Add support to CloudKittys client

4.2.4 Dependencies

None.

4.2.5 Testing

Besides regular unit tests, the tempest plugin will be updated to add new tests for added endpoints.

4.2.6 Documentation Impact

v2 API reference will be updated to reflect changes.

4.2.7 References

None.

4.3 Cloudkitty monasca fetcher implementation

https://storyboard.openstack.org/#!/story/2006675

Monasca is already supported in cloudkitty through its collector. However, cloudkitty lacks a dedicated fetcher for monasca, relying instead on other fetchers.

4.3.1 Problem Description

No scope discovery is available when using the monasca collector. This forces users to use the keystone fetcher and assign the rating role to every project where cloudkitty is needed. Another less than ideal solution is to use the source fetcher, but that requires to specify every scope in the configuration file.

4.3.2 Proposed Change

Implementing a new fetcher using monasca to discover scopes:

- The fetcher will connect to monasca and retrieve a list of scopes (dimensions values in monasca) given a specified dimension name. This is achieved through monascas python client and its `metrics.list_dimension_values()` method.
- The monasca endpoint will be retrieved from keystone in similar fashion to the monasca collector, as it is needed for the monasca client initialization. This is a good opportunity to mutualize the monascas client bootstraping code in a common file.

Configuration options include:

- `dimension_name`: the monasca dimension from which the scope_ids should be retrieved, defaults to `project_id`.
- `monasca_tenant_id`: The monasca tenant id, has no default value.
- `monasca_service_name`: Name of the monasca service, defaults to `monasca`
- `interface`: Endpoint type, defaults to `internal`

Alternatives

None

Data model impact

None

REST API impact

None

Security impact

The fetcher will use the same authentication mechanism as the current monasca collector (keystone for authentication and python-monascaclient for monasca interactions), and wont introduce any new dependency. Thus, the new fetcher shouldnt have any security impact.

Notifications Impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

4.3.3 Implementation

Assignee(s)

Primary assignee: qanglade/qanglade

Other contributors: lukapeschke/peschk_l

Work Items

- Implement a cloudkitty monasca scope fetcher using python-monascaclient.

4.3.4 Dependencies

No new dependency is needed.

4.3.5 Testing

Regular unit tests will be included.

4.3.6 Documentation Impact

Documentation of the new fetcher will be added, mostly covering configuration of the fetcher.

4.3.7 References

None.

4.4 Optimize driver loading in v2 API

https://storyboard.openstack.org/#!/story/2006654

4.4.1 Problem Description

Currently, required drivers are loaded in the `__init__` method of v2 API resources. However, `flask_restful` instanciates resources on each request, meaning that drivers are also loaded on each request. This leads to sub-optimal performance.

4.4.2 Proposed Change

Implement two new classmethods in `cloudkitty.api.v2.base.BaseResource`:

- `reload`: Reloads all required drivers.
- `load`: Loads all required drivers by calling `reload`. In case the drivers are already loaded, does nothing

In order to be able to persist drivers between class instantiations, they will be set as class attributes.

Alternatives

None.

Data model impact

None.

REST API impact

None.

Security impact

None.

Notifications Impact

None.

Other end user impact

The average response time for v2 API endpoints will be reduced.

Performance Impact

Loading the drivers only once will result in a performance gain.

Other deployer impact

Log messages that appear on driver instantiation will appear only once rather than on each request.

Developer impact

Driver loading will have to be done in the `reload` method rather than in `__init__`.

4.4.3 Implementation

Assignee(s)

Primary assignee: <lukapeschke/peschk_l>

Work Items

- Implement the `reload` and `load` methods.

4.4.4 Dependencies

None.

4.4.5 Testing

Existing unit tests for the v2 API along with planned tempest tests will test this.

4.4.6 Documentation Impact

The developer documentation for the v2 API will be updated to include information about the `load` and `reload` methods

4.4.7 References

None.

CHAPTER

FIVE

TRAIN

5.1 Add DataFrame/DataPoint objects

CloudKitty has an inner data format called DataFrame. It is used almost everywhere: The API returns dataframes, the storage driver expects to store dataframes, collected data is retrieved as a dataframe But dataframes are always passed around as dicts, making their manipulation tedious. This is a proposal to add a DataFrame and DataPoint class definition, which would allow easier conversion/manipulation of dataframes.

https://storyboard.openstack.org/#!/story/2005890

5.1.1 Problem Description

The dataframe format is specified in multiple places, but there is no true implementation of it: dicts respecting the format specifications are passed around instead. This can be error-prone: the integrity of these objects is not guaranteed (a function might modify them, even without intending to), and some specific details may vary from one part of the codebase to another (for example a `float` may be used instead of a `decimal.Decimal`).

Furthermore, the dataframe format is not exactly the same in the v1 and v2 storage interfaces. v1 has a single `desc` key containing every metadata attribute of a data point, whereas v2 provides two keys, `metadata` and `groupby`, depending on the type of the attribute. This leads to conversions between v1 and v2 format in several places in the code. Example taken from the `CloudKittyFormatTransformer`:

```
def format_item(self, groupby, metadata, unit, qty=1.0):
    data = {}
    data['groupby'] = groupby
    data['metadata'] = metadata
    # For backward compatibility.
    data['desc'] = data['groupby'].copy()
    data['desc'].update(data['metadata'])
    data['vol'] = {'unit': unit, 'qty': qty}

    return data
```

5.1.2 Proposed Change

The proposed solution is to introduce two new classes: `DataPoint` and `DataFrame`.

`DataPoint`

`DataPoint` replaces a single data point represented by a dict with the following format:

```
{
    "vol": {
        "unit": "GiB",
        "qty": 1.2,
    },
    "rating": {
        "price": 0.04,
    },
    "groupby": {
        "group_one": "one",
        "group_two": "two",
    },
    "metadata": {
        "attr_one": "one",
        "attr_two": "two",
    },
}
```

The following attributes will be accessible in a `DataPoint` object:

- `qty`: `decimal.Decimal`
- `price`: `decimal.Decimal`
- `groupby`: `werkzeug.datastructures.ImmutableMultiDict`
- `metadata`: `werkzeug.datastructures.ImmutableMultiDict`
- `desc`: `werkzeug.datastructures.ImmutableMultiDict`

Note: `desc` will be a combination of `metadata` and `groupby`

In order to ensure data consistency, the `DataPoint` object will inherit `collections.namedtuple`. The `groupby` and `metadata` attributes will be stored as `werkzeug.datastructures.ImmutableDict`.

In addition to its base attributes, the `DataPoint` class will have a `desc` attribute (implemented as a property), which will return an `ImmutableDict` (a merge of `metadata` and `groupby`).

`DataPoint` instances will expose the following methods:

- `set_price`: Set the price of the `DataPoint`. Returns a new instance.
- `as_dict`: Returns an (optionally mutable) dict representation of the object. For convenience with API backward compatibility, it will be possible to obtain the result in legacy format (`desc` will replace `metadata` and `groupby`).

- `json`: Returns a json representation of the object. For convenience with API backward compatibility, it will be possible to obtain the result in legacy format (`desc` will replace `metadata` and `groupby`).
- `from_dict`: Creates a `DataPoint` from its dict representation.

DataFrame

`DataFrame` replaces a dataframe represented by a dict with the following format:

```
{
    "period": {
        "begin": datetime.datetime,
        "end": datetime.datetime,
    },
    "usage": {
        "metric_one": [], # list of datapoints
        [...]
    }
}
```

A `DataFrame` is a wrapper around a collection of `DataPoint` objects. `DataFrame` instances will have two read-only attributes: `start` and `end` (stored as `datetime.datetime` objects).

`DataFrame` instances will expose the following methods:

- `as_dict`: Returns an (optionally mutable) dict representation of the object. For convenience with API backward compatibility, it will be possible to obtain the result in legacy format.
- `json`: Returns a json representation of the object. For convenience with API backward compatibility, it will be possible to obtain the result in legacy format.
- `from_dict`: Creates a `DataFrame` from its dict representation.
- `add_points`: Adds a list of `DataPoint` objects to a dataframe for a given metric.
- `iterpoints`: Generator function iterating over all points in the `DataFrame`. Yields (metric_name, `DataPoint`) tuples.

Note: Given that the `from_dict` method of both classes will mainly be used at the API level, voluptuous schemas matching the classes will be added and a schema validation will be executed on the argument `from_dict` is called with.

Alternatives

The code-base could be left as is, letting developers deal with the tedious dataframe manipulations.

Data model impact

Data structures manipulated internally get hardened.

REST API impact

None. However, this would ease a future endpoint allowing to push dataframes to cloudkitty.

Security impact

None.

Notifications Impact

None.

Other end user impact

None.

Performance Impact

Instantiating `DataPoints` might be slightly slower than instantiating dicts. However, `namedtuple` is a high-performance container, and several dict formatting steps that are currently executed will be skipped if we use a `namedtuple` subclass, so there may be no overhead at all.

Other deployer impact

None.

Developer impact

Manipulating objects with a clear and strict interface should make developing with dataframes easier and way less error-prone.

No extra dependencies are required.

5.1.3 Implementation

Assignee(s)

Primary assignee:

peschk_l

Work Items

- Create validation utils that will allow to check the datapoint/dataframe format.
- Submit the new classes along with tests.

5.1.4 Dependencies

None.

5.1.5 Testing

This will be tested with unit tests. A 100% test coverage is expected.

5.1.6 Documentation Impact

None.

5.1.7 References

None.

5.2 Add a delete method to the v2 storage interface

https://storyboard.openstack.org/#!/story/2005395

5.2.1 Problem Description

Note: This spec is the detailed implementation of one part of a larger spec. If you havent read it yet, please see https://specs.openstack.org/openstack/cloudkitty-specs/specs/train/reset_scope_state.html

---

Currently, there is no way to delete data from a v2 storage backend *via* cloudkitty. However, this is required in order to reset the state of a scope (see note above).

5.2.2 Proposed Change

A `delete` method will be added to the v2 storage interface and implemented in the v2 storage drivers. It will take three parameters:

- `begin` (datetime object, optional): The start of the period for which data should be deleted.
- `end` (datetime object, optional): The end of the period for which data should be deleted.
- `filters` (dict, optional): A dict of optional filters allowing to select data marked for deletion.

Alternatives

None.

Data model impact

No changes will be made to the data model. However, it will be possible to delete data from the storage backend through cloudkitty now.

REST API impact

None. Described in another spec.

Security impact

None. API security impact is described in other specs.

Notifications Impact

None. Described in another spec.

Other end user impact

None.

Performance Impact

Data deletion may slow down and put some load on the storage backend.

Other deployer impact

None.

Developer impact

Changes related to scope state reset will need to be rebased on this change.

5.2.3 Implementation

Assignee(s)

Primary assignee: peschk_l

Work Items

Add the `delete` method to the v2 storage interface and implement it in the InfluxDB storage driver.

5.2.4 Dependencies

Spec: allowing to get/reset the state of a scope. https://specs.openstack.org/openstack/cloudkitty-specs/specs/train/reset_scope_state.html

5.2.5 Testing

In addition to unit tests, Tempest scenarios testing the whole scope state reset action will be added.

5.2.6 Documentation Impact

The `delete` method will be added to the autogenerated documentation.

5.2.7 References

Spec: allowing to get/reset the state of a scope. https://specs.openstack.org/openstack/cloudkitty-specs/specs/train/reset_scope_state.html

5.3 Add a v2 API endpoint to retrieve DataFrame objects

https://storyboard.openstack.org/#!/story/2005890

CloudKitty needs an endpoint in its v2 API to retrieve DataFrame objects. This spec aims at defining what is to be done and why. Everything in this document is open to discussion, equally on the associated storyboard and gerrit.

5.3.1 Problem Description

Now that there is an endpoint available bound on `POST /v2/dataframes` that allows us to push DataFrames objects into the CloudKitty storage backend, we also need a v2 API endpoint to retrieve these objects from it.

Also this feature is available in the v1 API on `GET /v1/storage/dataframes`. Therefore it is necessary to have it in the v2 API.

5.3.2 Proposed Change

The proposed endpoint will support pagination, a dict of filters, a `begin` and an `end` parameter as ISO 8601 strings that support timezone offsetting e.g. 2019-08-29T07:18:40+00:00.

It will be available on `GET /v2/dataframes`.

Example:

- Getting all dataframes for `image.size` metric type for project X and user Y for a specific week:

  This would result in the following HTTP request:

```
GET /v2/dataframes?filter=type:image.size&filter=project_id:X&filter=user_
↪id:Y&begin=2019-05-01T00:00:00&end=2019-05-07T00:00:00
```

The feature will also be available to the CloudKitty client library and CLI.

Alternatives

None.

Data model impact

None.

REST API impact

This will add an endpoint on `/v2/dataframes` with support for the `GET` HTTP method.

The endpoint will support the following query parameters:

- `offset`: (optional, defaults to 0) The offset of the first element that should be returned.
- `limit`: (optional, defaults to 100) The maximal number of results to return.
- `filter`: (optional) A dict of metadata filters to apply.
- `begin`: (optional, defaults to the first day of the month at midnight) start of the period the request should apply to.
- `end`: (optional, defaults to the first day of the next month at midnight) end of the period the request should apply to.

A `GET` request on this endpoint will return a JSON object of the following format:

```
{
    "total": 2,
    "dataframes": [
        {
            "usage": {
                "volume.size": [
                    {
                        "vol": {
                            "unit": "GiB",
                            "qty": 1.9
                        },
                        "rating": {
                            "price": 3.8
                        },
                        "groupby": {
                            "project_id": "8ace6f139a1742548e09f1e446bc9737",
                            "user_id": "b28fd3f448c34c17bf70e32886900eed",
                            "id": "be966c6d-78a0-42cf-bab9-e833ed996dee"
                        },
                        "metadata": {
                            "volume_type": ""
                        }
                    }
                ]
            },
            "period": {
                "begin": "2019-08-01T01:00:00+00:00",
                "end": "2019-08-01T02:00:00+00:00"
            }
        },
        {
            "usage": {
                "image.size": [
                    {
                        "vol": {
                            "unit": "MiB",
                            "qty": 3.55339050293
                        },
                        "rating": {
                            "price": 1.77669525146
                        },
                        "groupby": {
                            "project_id": "5994682e63af4aa8873d247aa28b876e",
                            "user_id": "b28fd3f448c34c17bf70e32886900eed",
                            "id": "f7541950-96d5-4e89-ac76-9d4eacf59b83"
                        },
                        "metadata": {
                            "container_format": "foo",
                            "disk_format": "bar"
                        }
```


(continues on next page)

(continued from previous page)

```
                    }
                ]
            },
            "period": {
                "begin": "2019-08-01T02:00:00+00:00",
                "end": "2019-08-01T03:00:00+00:00"
            }
        }
    ]
}
```

The expected HTTP success response code for a `GET` request on this endpoint is `200 OK`.

Expected HTTP error response codes for a `GET` request on this endpoint are:

- `400 Bad Request`: Malformed request.
- `403 Forbidden`: The user does not have the necessary rights to retrieve dataframes.
- `404 Not Found`: No dataframes were found for the provided parameters.

This endpoint will be authorized for admins and for the tenant/project owners of the requested dataframes (regarding the specified `project_id` filter).

Security impact

Any user with access to this endpoint will be able to retrieve information about data rated by CloudKitty. Thus, access to this endpoint should be granted to non-admin users with parsimony.

Notifications Impact

None.

Other end user impact

The client will also be updated in order to include a function and a CLI command allowing to retrieve DataFrame objects.

Performance Impact

None.

Other deployer impact

Dataframes will be easier to retrieve for admins and deployers.

Developer impact

None.

5.3.3 Implementation

Assignee(s)

Primary assignee: <jferrieu>

Work Items

- Implement the API endpoint with unit tests
- Add tempest tests
- Support this endpoint in the client with unit and functional tests.

5.3.4 Dependencies

None.

5.3.5 Testing

Tempest tests for this endpoint will be added.

5.3.6 Documentation Impact

The endpoint will be added to the API reference.

5.3.7 References

- Spec: Add DataFrame/DataPoint objects: https://specs.openstack.org/openstack/cloudkitty-specs/specs/train/add_dataframe_datapoint_object.html
- Spec: Add a v2 API endpoint to push DataFrame objects: https://specs.openstack.org/openstack/cloudkitty-specs/specs/train/add_push_dataframes_api_endpoint.html

5.4 Add a v2 API endpoint to retrieve the state of different scopes

https://storyboard.openstack.org/#!/story/2005395

Note: This spec is the detailed implementation of one part of a larger spec. If you havent read it yet, please see https://review.opendev.org/#/c/657393/

5.4.1 Problem Description

For now, there is no way for admins to know until which date a scopes data has been processed, except by looking at the `cloudkitty_storage_states` SQL table. This is, however, critical information to have before an invoice/report is generated, as you have to make sure that the whole period youre billing for has been processed.

5.4.2 Proposed Change

This information should be made available through the API. This describes a V2 API endpoint. It will only be available to admins.

Alternatives

None.

Data model impact

None.

REST API impact

This will add an endpoint on `/v2/scope` with support for the `GET` HTTP method.

A `GET` request on this endpoint returns a paginated list of all scopes, with support for optional filters on the scope, the fetcher, the collector and/or the scope_key.

The method will support the following parameters:

- `offset`: (optional, defaults to 0) The offset of the first scope that should be returned.
- `limit`: (optional, defaults to 100) The maximal number or results to return.
- `scope_id`: (optional) Can be specified several times. One or several scope_ids to filter on.
- `collector`: (optional) Can be specified several times. One or several collectors to filter on.
- `fetcher`: (optional) Can be specified several times. One or several fetchers to filter on.
- `scope_key`: (optional) Can be specified several times. One or several scope_keys to filter on.

The response will have the following format:

```
{
    "results": [
        {
            "collector": "gnocchi",
            "fetcher": "keystone",
            "scope_id": "7a7e5183264644a7a79530eb56e59941",
            "scope_key": "project_id",
            "state": "2019-05-09 10:00:00"
        },
        {
            "collector": "gnocchi",
            "fetcher": "keystone",
            "scope_id": "9084fadcbd46481788e0ad7405dcbf12",
            "scope_key": "project_id",
            "state": "2019-05-08 03:00:00"
        },
        {
            "collector": "gnocchi",
            "fetcher": "keystone",
            "scope_id": "1f41d183fca5490ebda5c63fbaca026a",
            "scope_key": "project_id",
            "state": "2019-05-06 22:00:00"
        }
    ]
}
```


The expected HTTP success response code for a `GET` request on this endpoint is `200 OK`.

Expected HTTP error response codes for a `GET` request on this endpoint are:

- `400 Bad Request`: Malformed request.
- `403 Forbidden`: The user hasnt the necessary rights to retrieve the state of the scopes.
- `404 Not Found`: No scope was found for the provided filters.

This endpoint will only be authorized for admins.

Security impact

Any user with access to this endpoint will be able to retrieve information about the state of all scopes rated by cloudkitty. Thus, access to this endpoint should be granted to non-admin users with parsimony.

Notifications Impact

None.

Other end user impact

The client will also be updated in order to include a function and a CLI command allowing to retrieve information about the state of the different scopes.

Performance Impact

None.

Other deployer impact

Scope state information will be easier to retrieve for admins and deployers.

Developer impact

None.

5.4.3 Implementation

Assignee(s)

Primary assignee: <peschk_l>

Work Items

- Implement the API endpoint with unit tests
- Add tempest tests
- Support this endpoint in the client.

5.4.4 Dependencies

None.

5.4.5 Testing

Tempest tests for this endpoint will be added.

5.4.6 Documentation Impact

The endpoint will be added to the API reference.

5.4.7 References

Spec to get/reset the state of a scope: https://review.opendev.org/#/c/657393/

5.5 Add a v2 API endpoint to push DataFrame objects

https://storyboard.openstack.org/#!/story/2005890

CloudKitty needs an endpoint in its v2 API to push DataFrame objects. This spec aims at defining what is to be done and why. Everything in this document is open to discussion, equally on the associated storyboard and gerrit.

5.5.1 Problem Description

With the current state of CloudKitty, the only way to import DataFrames into the CloudKitty storage is to load a storage driver and to call the `push` method but it is impossible through the CloudKitty API.

This inconvenience prevents CloudKitty from having external processors being able to easily push data into its storage. e.g. CloudKitty cannot have DataFrame fixtures provisioned so it can be functionally tested with OpenStack Tempest.

A new admin API endpoint is needed to allow this.

The Tempest tests would also enable a more efficient workflow for validating forthcoming features and therefore improve the overall quality and stability of them in a long term goal and, more generally speaking, having this feature implemented would also represent an opportunity to anyone to extend their usage of CloudKitty in a more handy and flexible way.

5.5.2 Proposed Change

A new endpoint will be made available to admin users on `POST /v2/dataframes`. This will allow end users to push DataFrames in the form of JSON objects.

Alternatives

We could keep the old way, but external processors would need to be written in Python to be able to load the storage driver modules plus they would need the storage backend credentials exposed to them.

Otherwise CloudKitty could be provisioned by using more hackish ways such as manipulating directly the storage backend and the database metadata but this is a laborious and error-prone process that should definitely be avoided.

Data model impact

None.

REST API impact

This will add an endpoint on `/v2/dataframes` with support for the `POST` HTTP method.

The endpoint will support the following body parameters:

- `dataframes`: (required) A json array describing DataFrame objects (see below for details).

Inside the body of the request, a collection of DataFrame json objects can be specified as follows:

```
{
  "dataframes": [
    # first DataFrame
    {
      "period": {
        "begin": "20190723T122810Z",  # valid ISO 8601 datetime format
        "end": "20190723T132810Z"     # valid ISO 8601 datetime format
      },
      "usage": {
        "metric_one": [  # list of DataPoint json objects
          {
            "vol": {
              "unit": "GiB",
              "qty": 1.2
            },
            "rating": {
              "price": 0.04
            },
            "groupby": {
              "group_one": "one",
              "group_two": "two"
            },
            "metadata": {
```

(continues on next page)

(continued from previous page)

```
              "attr_one": "one",
              "attr_two": "two"
            }
          }
          # { second DataPoint for metric_one here }
        ],
        "metric_two": []
      }
    }
    # { second DataFrame here }
  ]
}
```

The expected HTTP success response code for a `POST` on this endpoint is `204 No Content`. This decision have been made in order to reduce the network bandwidth consumption for this kind of operation. The user will be able to consult DataFrames through a `GET /v2/dataframes` endpoint that will be available in the future.

Expected HTTP error response codes for a `POST` request on this endpoint are:

- `400 Bad Request`: Malformed request.
- `401 Unauthorized`: The user is not authenticated.
- `403 Forbidden`: The user is not authorized to push DataFrame objects.

This endpoint will only be authorized for administrator end users.

Security impact

Any user with access to this endpoint will be able to alterate data on the target platform which is an action carrying heavy side effects. Thus, access to this endpoint should be granted to non admin users with uttermost concern or not at all if possible.

Notifications Impact

None.

Other end user impact

The client will also be updated to include a function and a CLI command allowing to push DataFrame objects.

Performance Impact

None.

Other deployer impact

Importing DataFrame objects of any nature to CloudKitty will now be an easy process. This will be handy to provision CloudKitty if felt necessary.

Developer impact

Importing DataFrame objects will allow to push fixtures into the CloudKitty storage and therefore to add scenarios to the Tempest plugin. This will be handy to write integration tests afterwards.

5.5.3 Implementation

Assignee(s)

Primary assignee: <jferrieu>

Other contributors: <peschk_l>

Work Items

- Implement the API endpoint with unit tests.
- Add tempest tests.
- Support this endpoint in the client.

5.5.4 Dependencies

This endpoint depends on the DataFrame and DataPoint objects specification:

- Spec: https://review.opendev.org/#/c/668669/

5.5.5 Testing

Unit tests and Tempest tests for this endpoint will be added.

5.5.6 Documentation Impact

The endpoint will be added to the API reference.

5.5.7 References

Spec: Add a v2 API endpoint to push dataframe objects:

https://review.opendev.org/#/c/668669/

5.6 Add a v2 API endpoint to reset the state of different scopes

https://storyboard.openstack.org/#!/story/2005395

CloudKitty needs an endpoint int its v2 API to reset the state of one or several scopes. This spec aims at defining what is to be done and why. Everything in this document is open to discussion, equally on the associated storyboard and gerrit.

5.6.1 Problem Description

With the current state of CloudKitty, if an administrator is willing to reset calculations for a given scope for any reason e.g. changes in CloudKittys collection configuration, update for some rating rules, etc. He has to stop all the processors, update the state of said scope in the database and restart all the processors. This is a laborious process and we should make an easier way available to the end user in the form of a new v2 API endpoint.

5.6.2 Proposed Change

A new endpoint will be made available to admin users on `PUT /v2/scope` with relatively similar parameters that are to be found on the `GET /v2/scope` endpoint regarding filtering. This will allow end users to reset the scope state of several scopes at once if they are willing to. There is an additional parameter detailed later for preventing the end user from performing any irreversible change too quickly.

Alternatives

We could only allow resetting scope state on `PUT /v2/scope/<scope_id>` but that would prevent the end user to reset several scope states at once and would enforce one request per scope state reset attempt.

Also, a list of the effected scopes by the request could be returned to the end user but we have already a specified endpoint `GET /v2/scope` for this use case. That wouldnt be therefore necessary and would add dispensable complexity to the implementation for no expected usage.

Data model impact

None.

REST API impact

This will add an endpoint on `/v2/scope` with support for the `PUT` HTTP method.

The endpoint will support the following parameters:

- `all_scopes`: (mutually exclusive with `scope_id`, defaults to False) Whether we target all scopes at once or not. This parameter intends to prevent the end user from doing any mistake too quickly. If all_scopes and scope_id parameters are both absent or present no data would be affected by the request and a `400 Bad Request` response will be returned.
- `scope_id`: (mutually exclusive with `all_scopes`) Can be specified several times. One or several scope_ids to filter on to reset associated scope states. If all_scopes and scope_id parameters are both absent or present no data would be affected by the request and a `400 Bad Request` response will be returned.
- `collector`: (optional) Can be specified several times. One or several collectors to filter on to reset associated scope states.
- `fetcher`: (optional) Can be specified several times. One or several fetchers to filter on to reset associated scope states.
- `scope_key`: (optional) Can be specified several times. One or several scope_keys to filter on to reset associated scope states.

As the scope state reset is a time consuming operation and relies on the underlying AMQP system, it will be an asynchronous call to the API and will respond to the caller with a `202 Accepted`, once the notifications have been sent across the message queue. The response body of this request is empty.

The expected HTTP success response code for a `PUT` request on this endpoint is therefore `202 Accepted`.

Expected HTTP error response codes for a `PUT` request on this endpoint are:

- `400 Bad Request`: Malformed request. This will also happen in the case where `all_scopes` and `scope_id` parameters are both present or missing.
- `403 Forbidden`: The user is not authorized to reset the state of any scope.
- `404 Not Found`: No scope was found for the provided parameters.

This endpoint will be only authorized to admins.

Security impact

Any user with access to this endpoint will be able to perform a scope state reset on one or several scopes which is an action carrying heavy side effects. Thus, access to this endpoint should be granted to non-admin users with uttermost concern or not at all if possible.

Notifications Impact

Notifications will be sent to an AMQP listener in order to trigger the scope state reset. The structure of the notification to be sent will be the following:

```
{
   "scopes": [
       {
           "collector": "gnocchi",
           "fetcher": "keystone",
           "scope_id": "7a7e5183264644a7a79530eb56e59941",
           "scope_key": "project_id"
       },
       {
           "collector": "gnocchi",
           "fetcher": "keystone",
           "scope_id": "9084fadcbd46481788e0ad7405dcbf12",
           "scope_key": "project_id"
       },
       {
           "collector": "gnocchi",
           "fetcher": "keystone",
           "scope_id": "1f41d183fca5490ebda5c63fbaca026a",
           "scope_key": "project_id"
       }
    ]
}
```

Other end user impact

The client will also be updated in order to include a function and a CLI command allowing to reset state for one or several given scopes.

Performance Impact

The deletion operation carried by the scope state reset call will depend on the amount of data associated with the targeted scopes and may be time consuming, impairing the database performance in the same course.

Other deployer impact

Scope state reset will cease to be a tricky and laborious process and will profit admins and deployers for operating scopes.

Developer impact

None.

5.6.3 Implementation

Assignee(s)

Primary assignee: <jferrieu>

Other contributors: <peschk_l>

Work Items

- Implement the API endpoint with unit tests
- Add tempest tests
- Support this endpoint in the client.

5.6.4 Dependencies

None.

5.6.5 Testing

Tempest tests for this endpoint will be added.

5.6.6 Documentation Impact

The endpoint will be added to the API reference.

5.6.7 References

Spec to get/reset the state of a scope: https://specs.openstack.org/openstack/cloudkitty-specs/specs/train/reset_scope_state.html

5.7 Add a v2 reporting endpoint

https://storyboard.openstack.org/#!/story/2005664

5.7.1 Problem Description

Now that the v2 storage has been merged, new features can be implemented on top of it. This is a proposal for a v2 endpoint replacing v1s `/report/summary`. The endpoint should support filters and grouping on any `groupby` attribute.

5.7.2 Proposed Change

The proposed endpoint will support pagination, a dict of filters, and a list of attributes to group data by.

Example usecases:

- A per-user total for the current month, for the whole cloud:

  Example client usage:

  ```
  cloudkitty summary get --groupby user_id
  ```

  This would result in the following HTTP request:

  ```
  GET /v2/summary?groupby=user_id
  ```

- A per-resource total for scope X and user Y for a specific week:

  Example client usage:

  ```
  cloudkitty summary get --filter project_id:X --filter user_id:Y --groupby␣
  ↪id --begin 2019-05-01T00:00:00 --end 2019-05-07T00:00:00
  ```

  This would result in the following HTTP request:

  ```
  GET /v2/summary?filter=project_id:X&filter=user_id:Y&groupby=id&
  ↪begin=2019-05-01T00%3A00%3A00&end=2019-05-07T00%3A00%3A00
  ```

Alternatives

None

Data model impact

There is no direct change to the data model. However, the v2 storage interface will have to be updated: no distinction should be made between `groupby` and `metadata` filters. Thus, the `group_filters` parameter should be removed from the prototypes of the `retrieve` and `total` function.

REST API impact

This will add an endpoint on `/v2/summary` with support for the `GET` HTTP method.

The method will support the following parameters:

- `offset`: (optional, defaults to 0) The offset of the first element that should be returned.
- `limit`: (optional, defaults to 100) The maximal number of results to return.
- `filter`: (optional) A dict of metadata filters to apply.
- `groupby`: (optional) A list of attributes to group by.
- `begin`: (optional, defaults to the first day of the month at midnight) start of the period the request should apply to.
- `end`: (optional, defaults to the first day of the next month at midnight) end of the period the request should apply to.

A `GET` request on this endpoint will return a JSON object of the following format:

```
type: "object"
properties:
  total:
    type: "integer"
  columns:
    type: "array"
    items:
      type: "string"
  results:
    type: "array"
    items:
      oneOf:
        - type: string
        - type: string
          format: datetime
        - type: integer
        - type: float
```

`total` contains the total number of matching elements (for pagination).

`columns` contains the list of columns available for each element of `results`.

`results` is a list of same-length arrays sorted in the same way as `columns`.

Example response:

```
GET /v2/summary?groupby=user_id

{
  "total": 20,
  "columns": [
     "begin",
     "end",
     "qty",
```

(continues on next page)

(continued from previous page)

```
    "rate",
    "user_id"
  ],
  "results": [
    [
      "2019-05-01T00:00:00Z",
      "2019-06-01T00:00:00Z",
      42.21,
      13.37,
      "f6b331ad-af19-45b9-a4a3-2d27e8ab76e0"
    ],
    [...]
  ]
}
```

Security impact

There is no security impact introduced by this patch.

Note: In order to limit access, requests from non-admin users will automatically have a filter on their project added.

Notifications Impact

None

Other end user impact

The clients `summary get` method will be updated when using the v2 API in order to provide support for pagination, filters and grouping. User experience will be improved, as users will have a more precise view of their resource usage.

Performance Impact

None

Other deployer impact

None

Developer impact

None

5.7.3 Implementation

Assignee(s)

Primary assignee: peschk_l

Other contributors: jferrieu

Work Items

- Update the prototype of the `retrieve` and `total` methods of the v2 storage interface in order to remove the `group_filters` parameter.
- Implement the endpoint.
- Add tempest tests for this endpoint.
- Add support for the endpoint to the client.

5.7.4 Dependencies

None

5.7.5 Testing

Tempest tests for this endpoint will be added.

5.7.6 Documentation Impact

The endpoint will be added to the API reference.

5.7.7 References

None

5.8 Add an Elasticsearch v2 storage driver

https://storyboard.openstack.org/#!/story/2006332

5.8.1 Problem Description

For now, there is only one v2 storage driver: InfluxDB. However there should always be several proposed choices for each of CloudKittys modules.

The following strengths make Elasticsearch a great candidate:

- **Its a widespread solution.** On most deployments, it is likely that an Elasticsearch cluster is available (for example for log centralization).
- **HA and clustering**. Elasticsearch features HA and clustering by default, whereas it is only available in the paid version of InfluxDB.
- **Its performant.** And Elasticsearch allows some tuning by admins.
- **Data visualization** Data from the InfluxDB storage driver can be visualized with Grafana, the Elastic stack provides Kibana.

5.8.2 Proposed Change

A v2 storage driver for Elasticsearch, available through the `cloudkitty.storage.v2.backends.elasticsearch` entrypoint.

Heres a summary of the routes and aggregation methods that will be used for each of the v2 storage driver interfaces methods:

- `init`: `PUT /<index>` See Data model impact for mapping details.
- `push`: `POST /<index>/<mapping>/_bulk`
- `retrieve`: `GET /<index>/_search` A standard search query with filters.
- `total`: `GET /<index>/_search` The `composite` aggregation will be used: Several `terms` aggregations in the `must` clause of a `bool` query will allow to group data on specific attributes. A `sum` aggregation will then be applied to the buckets to obtain the qty and price for each of them.
- `delete`: `POST /<index>/_delete_by_query` Same principle as the `retrieve` method, but for deletion.

> **Warning:** The composite query is stable since Elasticsearch version 6.5. In order to be compatible with 6.x and 7.x, cloudkitty will use the `include_type_name` parameter for mapping creation. This parameter was added in Elasticsearch 6.8. This parameter will be removed in Elasticsearch 8. Thus, CloudKitty will require Elasticsearch >= 6.5 and < to 8.

Note: **About pagination:** Given that `offset + size` cant exceed 15000 in the `search` API, the `retrieve` function will use scrolling. The `search_after` feature will not be used, as it is stateless, which means that consecutive requests may return unexpected results depending on the index updates happening at the same time. The duration for which scroll contexts should be kept open will be configurable through a config file option marked as **advanced**.

The `total` function will use the `after` parameter of the `composite` aggregation

Note: The CloudKitty storage driver will only require the OSS version of Elasticsearch to work. However, some X-Pack features of the Basic version, like authentication, will be supported (but not mandatory) in the future.

Alternatives

None.

Data model impact

The data model used in Elasticsearch will be as follows:

Each DataPoint will be a single document. An existing empty index is required (this will allow tuning from admins). In order to improve overall performance, a mapping with the following attributes will be created.

- `start`: (date) The start of the period the datapoint applies to.
- `end`: (date) The end of the period the datapoint applies to.
- `type`: (keyword) The type of the datapoint.
- `unit`: (keyword) The unit of the datapoint.
- `qty`: (double) The qty of the datapoint.
- `price`: (double) The price of the datapoint.
- `groupby`: (object) Dict of the datapoints groupby attributes.
- `metadata`: (object) Dict of the datapoints metadata attributes.

Note: In order to allow flexible groupby/metadata, the associated objects will be flexible.

Note: Given that we will only do exact value searches, every `string` attribute will be converted to a `keyword`. This will be achieved using dynamic templates.

Warning: By default, the `_source` field will be enabled. An option to disable it in order to improve storage size may be added, but this should be done with care. See the link to the Elasticsearch documentation in the references for details.

In the end, the mapping will be defined as follows:

```
{
  "mappings": {
    "_doc": {
      // cast all strings to keywords
      "dynamic_templates": [
        {
          "strings_as_keywords": {
            "match_mapping_type": "string",
            "mapping": {
              "type": "keyword"
            }
          }
        }
      ],
      // we won't add any attribute to the base object, so dynamic must be␣
↪false
      "dynamic": false,
      "properties": {
        "start": {"type": "date"},
        "end": {"type": "date"},
        "type": {"type": "keyword"},
        "unit": {"type": "keyword"},
        "qty": {"type": "double"},
        "price": {"type": "double"},
        // groupby and metadata will accept new attributes
        "groupby": {"dynamic": true, "type": "object"},
        // even though metadata should not be indexed, disabling it can't be
        // undone, and disabled objects are only available through the "_
↪source"
        // field, which may also be disabled
        "metadata": {"dynamic": true, "type": "object"}
      }
    }
  }
}
```

Note: Given that a term to filter on may be part of `groupby` or `metadata`, each filter will add two `term` queries to the `should` part of the `bool` query (one for the `groupby` section and one for the `metadata` section). Thus, the `minimum_should_match` parameter of the `bool` query will be set to half of the number of terms in the `should` query.

REST API impact

None.

Security impact

In the first iteration, there will be no support for x-pack authentication. It will be up to the admins to secure the connections between the Elasticsearch cluster and CloudKitty. Authentication will be introduced in future releases.

Notifications Impact

None.

Other end user impact

None.

Performance Impact

On most benchmarks (and from what could be determined from POCs), data insertion into Elasticsearch is slower than insertion into InfluxDB. However, Elasticsearch is faster for aggregations. However, once CloudKitty has caught up with the current timestamp, not many insertions are required. Moreover, Elasticsearchs support for clustering and for tuning should allow for a better overall performance in the end.

Other deployer impact

The new backend will require more configuration from the admins:

- Index aliases and lifecycles
- Shards and replicas

Developer impact

None

5.8.3 Implementation

Assignee(s)

Primary assignee: peschk_l

Work Items

- Implement an Elasticsearch storage driver
- Add support for the driver to the Devstack plugin.
- Add a Tempest job where the Elasticsearch storage driver is used.

5.8.4 Dependencies

Elasticsearch >= 6.5.

5.8.5 Testing

In addition to unit tests, this will be tested with Tempest.

5.8.6 Documentation Impact

The configuration options provided of this driver will be detailed in the documentation. There will also be a section dedicated to the configuration of the Elasticsearch index.

5.8.7 References

- Dynamic templates: https://www.elastic.co/guide/en/elasticsearch/reference/current/dynamic-templates.html
- `_source` field: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping-source-field.html
- `search_after` parameter: https://www.elastic.co/guide/en/elasticsearch/reference/7.x/search-request-body.html#request-body-search-search-after
- Elasticsearch and InfluxDB benchmarks: https://jolicode.com/blog/influxdb-vs-elasticsearch-for-time-series-and-metrics-data

5.9 Adding Prometheus fetcher

https://storyboard.openstack.org/#!/story/2005427

CloudKitty needs a fetcher describing how to get scopes (formerly tenants) from a Prometheus service. This spec aims at defining what is to be done and why. Everything in this document is open to discussion, equally on the associated storyboard and gerrit.

5.9.1 Problem Description

In the present state, CloudKitty does not implement any method for discovering and fetching scope IDs from a Prometheus service and it needs one.

5.9.2 Proposed Change

The fetcher will be configurable under a `[fetcher_prometheus]` section in `cloudkitty.conf`.

The configuration options will be the folllowing :

- A `metric` option to provide a metric name from which we can infer the scope ID (default: none).
- A `scope_attribute` option to provide the key referring the scope IDs (default: `project_id`).
- A `filters` option as a key-value dictionary to filter out the discovery query response with some metadata (optional).
- A `prometheus_url` option to define the HTTP API endpoint URL of a Prometheus service.
- A `prometheus_user` and a `prometheus_password` option to provide credentials for authentication (both defaults: none).
- A `cafile` option to allow custom certificate authority file (default: none).
- An `insecure` option to explicitly allow insecure HTTPS requests (default: false).

The fetcher will request an aggregation on the specified `metric` using PromQL query and send it over HTTP to the specified Prometheus service API instant query endpoint. It will also group the result by the specified `scope_attribute` and filter out the response if some metadata `filters` have been specified in order to gather the scope_ids.

Example

```
# cloudkitty.conf

# [...]

[fetcher_prometheus]
metric=container_memory_usage_bytes
scope_attribute=namespace
prometheus_url=https://prometheus-dn.tld/api/v1
prometheus_user=foobar
prometheus_password=foobar
insecure=true
filters=label1:foo,label2:bar
```

The PromQL request will look the following:

```
max(container_memory_usage_bytes{label1="foo", label2="bar"}) by (namespace)
```

The HTTP URL with the url-encoded PromQL will then look the following :

```
http://prometheus-dn.tld/api/v1/query?query=max%28container_memory_usage_bytes%7Blabel1%3D%
```

Data model impact

None

REST API impact

None

Security impact

None

Notifications Impact

None

Other end user impact

End users will benefit from new Prometheus fetcher to dynamically discover scope IDs.

It will be particularly handy in conjunction with the Prometheus collector.

Performance Impact

None

Other deployer impact

None

Developer impact

None

5.9.3 Implementation

Assignee(s)

Justin Ferrieu is assigned to work on the spec as well as to develop the evoked points.

Primary assignee: jferrieu

Other contributors: None

Work Items

- Implement Prometheus fetcher (with unit tests)
- Add fetcher documentation

5.9.4 Dependencies

None

5.9.5 Testing

The proposed changes will be tested with unit tests.

5.9.6 Documentation Impact

We will add an entry detailing the Prometheus fetcher configuration in `Administration Guide/ Configuration Guide/Fetcher/Prometheus`.

5.9.7 References

- Prometheus documentation: https://prometheus.io/docs/

5.10 Adding some concurrency/parallelism to the processor

The processor is currently single-threaded (except for AMQP listeners) and running on a single process. This is a proposal to add some concurrency to the processor, which will lead to a huge performance improvement.

https://storyboard.openstack.org/#!/story/2005423

5.10.1 Problem Description

Given that the processor spends most of its time waiting once it has caught up with the current timestamp, this is not a critical feature. However, this does not imply much changes to the code, and would lead to a huge performance improvement.

Having faster processors allows to catch up with the current timestamp quicker, which is useful on new deployments, or when a long period needs to be re-processed.

5.10.2 Proposed Change

This change can be split into two parts. The first (and simplest) part would be to retrieve all metrics for a given scope and period (pseudo-)simultaneously, by making use of eventlet greenthreads. CloudKitty already depends on eventlet, so this requires no new dependency. Once support for python2 will have been dropped, this part can be updated in order to use the STL, and remove the dependency on eventlet (which could be replaced by concurrent.futures or asyncio).

The change is rather straightforward: rather than making consecutive calls to `_collect` (one per metric type) in the workers, these should be made simultaneously.

The second part would be to spawn several workers for each cloudkitty-processor instance. For this, the proposal would be to use the **cotyledon** library, which is already used by other OpenStack services, like ceilometer. Again, the change is quite small: Instead of inheriting `object`, the `Orchestrator` would inherit from `cotyledon.Service`. A `cotyledon.ServiceManager` spawning several orchestrator services will also be added.

In order to limit the load on the network and memory, the number of workers will be configurable.

Alternatives

- Instead of using cotyledon, the STLs multiprocessing could be used. However, cotyledon has some useful features, like forwarding signals to the workers. Moreover, some new kind of services may be added in the future (rating agents etc); cotyledon would allow to easily intergrate those without having to make much changes to the code.

Data model impact

None

REST API impact

None

Security impact

None

Notifications Impact

None

Other end user impact

None, apart from a notable performance improvement.

Performance Impact

A significant improvement to the processors performance will be made.

Other deployer impact

None

Developer impact

This adds a new dependency to cloudkitty: **cotyledon**. Introducing concurrency and parallelism will encourage developpers to adopt a functional coding style, in order to avoid race issues.

5.10.3 Implementation

Assignee(s)

Gerrit topic: `adding-concurrency`.

Primary assignee: peschk_l

Work Items

- Retrieve metrics in eventlet greenthreads.
- Make cloudkitty-processor run several workers with cotyledon.
- Add an **orchestration** section to the documentation. It will contain details about how to configure the number of workers and how to configure the coordination URL.

5.10.4 Dependencies

- `cotyledon`

5.10.5 Testing

This will be tested by tempest scenarios, once these are implemented.

5.10.6 Documentation Impact

An **orchestration** section will be added to the documentation. It will contain details about how to configure the number of workers and how to configure the coordination URL.

5.10.7 References

- Cotyledon documentation: https://cotyledon.readthedocs.io/en/latest/index.html
- Eventlet documentation: https://eventlet.net/

5.11 Allowing to get/reset the state of a scope

https://storyboard.openstack.org/#!/story/2005395

5.11.1 Problem Description

With the current state of CloudKitty, if an admin wants to reset calculations for a given scope for any reason (changes in cloudkittys collection configuration, update of some rating rules), he has to stop all the processors, update the state of said scope in the database, and restart all the processors. Furthermore, there is no way for an admin to get the current state of a scope, unless he directly queries the SQL database.

5.11.2 Proposed Change

> **Warning:** This spec only covers the global schema. If it is accepted, three related specs detailing each part of the solution will be proposed.

In order to solve this, the following three changes are proposed:

- Add a `delete` method to the v2 storage interface allowing to delete all dataframes after a specific timestamp for one or several scopes. This will be the subject of another spec.
- Add a v2 API endpoint allowing to reset the state of one or several scopes. This will be the subject of another spec.
- Add a v2 API endpoint allowing to retrieve the state of one or several scopes. This endpoint will be a replacement for v1s `/report/tenants`. This will be the subject of another spec.

The proposed logic is the following:

- The admin sets the status of a scope through the API. He can regularly check if the status of the previously updated scope has been updated through the scope state endpoint.
- The API endpoint sends an AMQP notification indicating that one or several scopes should be reset to a given state. Once an AMQP reply has been received by the API, either a 202 Accepted or 400 Bad Request status code will be returned (it is implied that authentication-related issues leading to 401 or 403 errors will have been handled before reaching this point).
- Upon receiving a notification, the associated AMQP listener goes through the following steps:

  - The notification is parsed and validated. An AMQP reply indicating if it is valid or not is returned to the API. If the notification was invalid, the listener stops here.
  - For each scope, the following steps are applied:
    * To prevent a scope from being processed while its state is being reset and its data being deleted, the listener acquires a tooz lock. This operation is blocking.
    * Once the lock is acquired, all the data after the given state is deleted from the storage backend.
    * After the data has been deleted, the state of the scope is set to the given state.
    * Once this is done, the lock is released.

Alternatives

This could be done without the AMQP notification. However, this would imply to execute the locking and data update/deletion from the API, which would lead to request timeouts.

Data model impact

None.

REST API impact

The REST API impact will be detailed in the specs related to the API endpoints described above.

Security impact

This will allow admins to delete user data through the API. This must be used with care.

Notifications Impact

This will add a notification and a new AMQP listener to the processor. The exact change will be described in the associated spec.

Other end user impact

None.

Performance Impact

Depending on the amount of data associated to a scope, the deletion may take a consequent amount of time and affect the performance of the database.

Moreover, the scope to reset will be locked during data deletion, and processors will thus not be able to process it.

Other deployer impact

This will help admins, as resetting a scope wont require to manipulate the database directly anymore.

Developer impact

None

5.11.3 Implementation

Assignee(s)

Primary assignee: peschk_l

Other contributors: jferrieu

Work Items

- Write a spec to add a `delete` method to the v2 storage interface allowing to delete all dataframes after a specific timestamp for one or several scopes.
- Write a spec to add a v2 API endpoint allowing to reset the state of one or several scopes.
- Write a spec to add a v2 API endpoint allowing to retrieve the state of one or several scopes. This endpoint will be a replacement for v1s `/report/tenants`.

5.11.4 Dependencies

None.

5.11.5 Testing

Testing will be detailed in each of the associated specs.

5.11.6 Documentation Impact

- The API reference will be updated
- Some documentation and examples on how to reset the state of a scope will be added.

5.11.7 References

None

5.12 Making CloudKitty timezone-aware

Associated story: https://storyboard.openstack.org/#!/story/2005319

Currently, CloudKitty is not timezone-aware and every timestamp is considered to be UTC. In order to improve user experience, avoid confusion and potential bugs (when doing time object conversions), CloudKitty should be made timezone-aware.

5.12.1 Problem Description

Internally, CloudKitty manipulates timezone-unaware `datetime` objects and timestamps. Every object representing some point in time is considered to be UTC. This can be confusing for users (which expect the timestamps of their data to be in their current timezone) as well as a source of bugs: A `datetime` object (converted to an iso8601 format timestamp) has no timezone information. There is no guarantee about how this kind of string will be interpreted by another API: as UTC, or in the APIs local time ?

5.12.2 Proposed Change

In order to secure CloudKittys behaviour about timezones, the following points should be applied:

- All `datetime` objects must be made timezone-aware.
- CloudKitty must only manipulate `datetime` objects. Functions or interfaces accepting several types of objects for parameters containing time information must be changed in order to only accept timezone-aware `datetime` objects.
- The conversion to/from iso8601 timestamps or unix timestamps must be done at the interface level. This means that conversions to/from timezone-aware `datetime` objects must only be done by CloudKittys HTTP API and the different drivers (storage, fetcher, collector, messaging), and only when this type is not supported.
- Any timezone-unaware object retrieved by the API must be considered as UTC, made timezone-aware and raise a warning.
- The handling of timezone-aware objects is not consistent between different SQL databases. In consequence, the storage state will still be stored with no timezone information and will be stored as UTC. The storage state driver will be in charge of making objects aware/unaware and will only provide/accept timezone-aware `datetime` objects. To avoid complex migrations and inconsistencies, the same rule will apply to the timestamps of dataframes: They will be stored as timezone-unaware timestamps and considered to be UTC. The storage driver will be responsible for the conversions.

- Client functions querying the v2 API will send iso8601 timestamps with timezone information. If no timezone information is provided in the CLI arguments, they will be considered as local time and adequate timezone information will be added.

For convenience, microseconds will not be supported for now, and will be set to 0 in all `datetime` objects.

Alternatives

- Leaving the code as is and making clear in the documentation that everything is UTC. This does not prevent unexpected behaviour when sending timezone-unaware info to other APIs.

Data model impact

None, the storage state model is not modified.

REST API impact

None.

Security impact

None.

Notifications Impact

None.

Other end user impact

For v2 API endpoints, timestamps with no timezone information passed to the client will be considered as localtime by the client, and timezone information will be updated accordingly.

Performance Impact

None.

Other deployer impact

None. The upgrade will impact neither the state storage nor the dataframes storage as no storage migration will be required. In consequence, the existing data will not be impacted.

Developer impact

Clear and stricter rules will reduce potential time-related bugs in new features implementation.

5.12.3 Implementation

Assignee(s)

Gerrit topic: timezone-aware

Primary assignee: peschk_l

Other contributors: None

Work Items

Some of the following points are highly interdependent and may need to be implemented in the same commit:

- Removing usage of Unix timestamps from the codebase in order to only manipulate `datetime` objects internally.
- Adding timezone information to all `datetime` objects manipulated internally.
- Make the functions of the client querying the v2 API timezone-aware.

5.12.4 Dependencies

- `python-dateutil`: Library providing extensions to the standard librarys `datetime` module.

5.12.5 Testing

Unit tests are sufficient until the v2 API has more endpoints. Tempest tests testing v2 API endpoints should be ran with and without timezone information in timestamps to ensure that the API has the expected behaviour.

5.12.6 Documentation Impact

The v2 API documentation and client documentation will include a description of their behaviour with timezone-aware/unaware timestamps.

5.12.7 References

None.

CHAPTER

SIX

STEIN

6.1 Add some details to state management

CloudKitty is becoming more and more generic, at all levels. With the current state of the project, scopes can be anything, and can even be different when several collectors are running with different configurations. In consequence, some more details should be added to the storage state management.

Associated storyboard story: https://storyboard.openstack.org/#!/story/2004957

6.1.1 Problem Description

Currently, CloudKitty only stores the ID of a scope in its SQL database when checking/updating a scopes state. Given that scope IDs are UUIDs in most of the cases (project IDs for example), this is (in theory) collision-safe. However, since CloudKitty does now support multiple sources for data collection, collisions are more likely to occur, especially if data about the same resources is collected from different sources. In addition to that, a scope ID is **not** required to be an UUID, even if it is considered best practice.

Example of a problematic configuration:

- Collector A collects data from monasca with the keystone fetcher and scope being `project_id`.
- Collector B collects data from gnocchi with the keystone fetcher and scope being `project_id`.

This does currently have unpredictable behavior, as both collectors would update the same scope.

6.1.2 Proposed Change

In order to improve support for multiple sources and avoid collisions or unexpected behavior, the fetcher by which a scope ID was retrieved along with the name of the collector that was used to retrieve the data should also be stored in the state table. In order to avoid all hypothetic collisions, the scope_key that was used for this specific scope ID should also be added.

Thus, each collection unit would be defined by the following tuple: `(scope_id, scope_key, collector_name, fetcher_name)`.

This would allow collectors collecting data from different sources or with different scopes to synchronize in a consistent way.

In order to be backward-compatible, the state manager will have the following behavior: When checking a scope `(A, B, C, D)`, it will try to retrieve the state of the scope `(A, B, C, D)`. If no such scope

is found, it will try to retrieve the scope `(A, None, None, None)`. If that scope is found, its empty columns will be updated with values `B`, `C`, and `D`.

Alternatives

One alternative would be to document discouraged configurations. However, this would add some constraints and edge cases which could be supported with the current proposal.

Data model impact

Three nullable columns will be added to the `cloudkitty_storage_states` database, along to the `identifier` column:

- `scope_key`
- `collector`
- `fetcher`

REST API impact

None

Security impact

None

Notifications Impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None, the migration will be handled by the `cloudkitty-dbsync upgrade` command.

Developer impact

None

6.1.3 Implementation

Assignee(s)

Primary assignee: lukapeschke/peschk_l

Work Items

- Add additional information to the state management table, and update the state checking logic.

6.1.4 Dependencies

None

6.1.5 Testing

Given that this is just an algorithmic change (no new feature / API change / behavior impact), unit tests will be enough. No tempest integration tests are required.

6.1.6 Documentation Impact

Some details about how the state of a scope is stored will be added to the documentation.

6.1.7 References

This has been discussed during the CloudKitty IRC meeting of February 1st 2019. The logs of this meeting are available at http://eavesdrop.openstack.org/meetings/cloudkitty/2019/cloudkitty.2019-02-01-15.00.log.html (Adding collector + fetcher to the state database and Q & A topics, starting at 15:42:26).

6.2 Adding a v2 API to CloudKitty

https://storyboard.openstack.org/#!/story/2004208

CloudKitty has gone through a lot of changes recently (v2 storage, support for Prometheus, standalone mode). This requires big changes on the API: given that the internal data format is evolving, we need to expose the data differently, in order to support the new features of the v2 storage: pagination, grouping, filtering

6.2.1 Problem Description

Several distinct problems can be identified:

1. CloudKittys data needs to be exposed with more granularity. A non-exhaustive list of currently missing features:
   - Pagination
   - Filtering
   - Grouping
   -
2. For its current API, CloudKitty uses WSME + pecan. Code written using these tools can be quite difficult to understand for new contributors. In addition to that, pecans development seems to have slowed down: https://github.com/pecan/pecan/commits/master .
3. Semantics. A lot of endpoints are still using OpenStack-related terms (tenant, service).

6.2.2 Proposed Change

In consequence to the previously evoked reasons, the CloudKitty development team proposes to implement a new API, which will be based on Flask/Werkzeug and Flask-RESTful. These frameworks have been chosen for the following reasons:

- Flask is a well-known and easy-to-use framework. This will make contributions easier for new contributors. It is solid and used by other OpenStack projects, like Keystone.
- Flask-RESTful makes code easier to read and routing is also eased. The idea would be to have each important endpoint (for example `/rating/hashmap`) mapped to a blueprint, and each sub-endpoint (`rating/hashmap/services`) mapped to a Flask-RESTful resource. This would make adding new endpoints easy, and leaves space for the v2 API to evolve. Below a working code sample, with voluptuous input and output validation:

```
@api_utils.add_output_schema(GET_OUTPUT_EXAMPLE_SCHEMA)
def get(self):
    policy.authorize(flask.request.context, 'example:get_example', {})
    return {}

@api_utils.add_input_schema(POST_INPUT_EXAMPLE_SCHEMA)
def post(self, fruit='banana'):
    policy.authorize(flask.request.context, 'example:submit_fruit', {})
    if fruit not in ['banana', 'strawberry']:
```

(continues on next page)

(continued from previous page)

```
        raise http_exceptions.Forbidden(
            'You submitted a forbidden fruit',
        )
    return {
        'msg': 'Your fruit is a ' + fruit
    }
```

- Flask-RESTful forces (or at least encourages) contributors to have a restful approach to the API, whereas Flask leaves them too much freedom.

This new API will make it easy to add a new endpoint, and will be a container to new features. Any new endpoint will be added to this API.

This v2 API will only be compatible with the v2 storage, allowing new endpoints to completely exploit the capacities of the v2 storage, without having to handle backward compatibility. The v1 API is completely compatible with the v2 storage, so data will still be accessible through the v1 API, compatibilty with existing scripts etc, will be kept. The v2 API will be disabled when a v1 storage backend is used.

In order to limit the workload, not all v1 endpoints will be migrated at once. The proposition will be to have one WSGI app per API version, and to distinguish between them through routing. This could be done with Werkzeugs `DispatcherMiddleware`: http://werkzeug.pocoo.org/docs/0.14/middlewares/#werkzeug.wsgi.DispatcherMiddleware. In case a v1 storage backend is used, the v2 API will simply not be loaded.

V1 endpoints will be migrated one after another. Once the entire v1 API has been migrated, the v2 API will be marked as `CURRENT`, and the v1 API will be marked as `DEPRECATED`. Until that point is reached, v1 will be `CURRENT` and v2 will be `EXPERIMENTAL`.

The proposed workflow for adding an endpoint is the following:

- Write a spec detailing what the endpoint does, its impact on the client, the dashboard and the tempest plugin.
- **Once the spec is reviewed and merged**, create a storyboard story linking to your spec and containing one task for each of the following points:
  - Adding the endpoint to the API
  - Adding support for the endpoint to the client
  - Adding tests for this endpoint to the tempest plugin
  - (Optional) Adding support for the endpoint to the dashboard

  Each of these tasks must be the subject of a new patch. Each task which is not marked as optional is mandatory.
- The patches adding client/dashboard support as well as the patch adding tests to the tempest plugin must depend on the patch adding the endpoint to the API.

Once the v2 API is stable and marked as the current API, it will be **microversioned** in order to have an indicator of its capabilities. A microversion increase will be indicating a new feature (new endpoint). However, API endpoints in OpenStacks service catalog will **not** be microversioned. The exact version of the API will be available at the API root (`/`).

Versioning will be done using **semantic versioning** (https://semver.org/). Once all v1 endpoint have been migrated, and the v2 API is considered stable, its version will be `2.0`. Before that version, each

version will be suffixed with `-beta.version`.

Example: `v2.0-beta.1 -> -> v2.0-beta.x -> v2.0`.

Alternatives

Migrate the whole v1 API to Flask and extend existing endpoints in order to support the features listed above: This would imply to migrate the whole API codebase at once, which cant be done. Also, this wouldnt address the semantics problem.

Data model impact

This particular change (creating a v2 WSGI app and changing the way requests are routed) has no impact on the datamodel. Each v2 API endpoint will be the subject of a new spec. Potential data model impact will be detailed in these upcoming specs.

REST API impact

- v1 API will go from `EXPERIMENTAL` to `CURRENT` current state.
- A v2 API, marked as `EXPERIMENTAL`, will be available.
- A new route (`/v2`) will be available. It will contain a placeholder indicating that this will be the prefix for all upcoming v2 endpoints.

Security impact

None

Notifications Impact

None

Other end user impact

None. However, `python-cloudkittyclient` will need to be compatible with the new v2 endpoints.

Performance Impact

Pagination will allow to lower the load on the network.

Other deployer impact

None

Developer impact

For developers, adding a new enpoint should be way easier.

A dependency on Flask and Flask-RESTful will be added.

6.2.3 Implementation

Assignee(s)

Primary assignee: lukapeschke/peschk_l

Gerrit topic: cloudkitty-v2-api

Work Items

- Mark the v1 API as `CURRENT`
- Route the API with Werkzeug instead of Pecan
- Add an Example endpoint showing how to implement an endpoint, and how to document it. This example endpoint should be removed as soon as the first real endpoint is implemented.
- Update the documentation: Update the developer documentation with an explanation about how to add an enpoint and generate the documentation of the v2 API.

6.2.4 Dependencies

- Flask
- Flask-RESTful
- os-api-ref (for API reference generation)

6.2.5 Testing

This particular feature will be tested with unit tests only (gabbi and unittest). Endpoints to come will also add some tests to the tempest plugin.

6.2.6 Documentation Impact

The developer documentation will be updated to detail how requests are routed and how to add a new endpoint. The user documentation will also be updated in order to include the v2 API reference. The API reference will be generated with `os-api-ref`.

6.2.7 References

- os-api-ref documentation: https://docs.openstack.org/os-api-ref/latest/
- API WG wiki: https://wiki.openstack.org/wiki/API_Special_Interest_Group

6.3 Refactoring CloudKittys documentation

https://storyboard.openstack.org/#!/story/2004179

CloudKittys documentation needs to be refactored. This spec aims at proposing a new content layout. It is of course subject to discussion, as well on storyboard as on gerrit.

6.3.1 Problem Description

Since Cloudkitty has been through major changes in the last releases, the current documentation layout and content causes several problems:

- The layout is not intuitive: many people on IRC ask for information that is available in the documentation, but very hard to find.
- Cloudkittys architecture as well as the role of each component (fetcher, collector, rating module, storage backend) is not explained clearly enough. This is a problem for operators, as they dont know how CloudKitty works internally, which is an issue when troubleshooting.
- The configuration of metric collection is very poorly documented, leading most operators to stick to the default `metrics.yml` file, without knowing exactly what each entry means.
- The developer documentation is almost non-existent: It only contains information about the v2 storage interface. It should be extended to help contributors.
- The documentation also lacks an FAQ/troubleshooting part: Typical question is I deployed Cloudkitty with Devstack, why is summary 0 ?.

6.3.2 Proposed Change

The first thing in the documentation should be a very brief introduction about what CloudKitty is and does. Concerning people reading CloudKittys documentation: they can be split into three groups; users, operators/admins, and developers. In consequence, proposed top-level sections after the short introduction would be the following:

- Common introduction: Whats the matter/need Cloudkitty aims to solve ? Whats possible ? Whats not (yet) ? What will never be supported ?

  In addition to these questions, there should be a general introduction on the different components, their roles and how they interact.

- User documentation: How do I retrieve information through cloudkittys API ? What kind of information can I get ?
- HTTP API reference. However, reference of the v2 API will be published to `developer.openstack.org/api-ref` once it is considered stable.
- Operator documentation: This section should detail the role of each component, and provide information about how to configure it. This is also where information on rating rule creation should live. Operators are also the target audience for the troubleshooting section.

  In opposition to the current approach (with keystone/openstack vs. without keystone/openstack), I believe that this part of the documentation should explain each component one by one: its purpose, how it can be used, existing implementations (list different collectors/fetchers/rating modules, and ideally provide a support matrix), and how it can be configured.

  Example of a usecase where configuration is hard to deduce from the documentation: CloudKitty is used without keystone authentication and collects prometheus metrics. However, in order to collect metrics from gnocchi, the collector needs to authenticate to keystone. How to configure this ?
- Developer documentation: This section should provide implementation details about the different components cloudkitty is made of. It should explain contributors what they need to implement, which interfaces they should use, and how their contribution should be tested. Ideally, this should complete the docstrings of an interface. A (small and probably not complete enough) example of such a documentation can be found here: https://docs.openstack.org/cloudkitty/latest/developer/storage.html

  Here again, the documentation should be split by component.

Alternatives

I dont have any alternative layouts or contents in mind, but proposals are of course welcome!

Data model impact

None

REST API impact

None, except maybe a better documentation of the API.

Security impact

None

Notifications Impact

None

Other end user impact

End users and contributors will have a better understanding of the project.

Performance Impact

None

Other deployer impact

Better understanding of the projet, troubleshooting made easier.

Developer impact

Better understanding of the projet, contributing made easier.

6.3.3 Implementation

Assignee(s)

Luka Peschke is assigned for the work on the spec, and the integration of the new layout to the doc. Work on the content of the documentation will be dispatched once the spec has been reviewed, corrected and merged.

Primary assignee: peschk_l

Other contributors: None

Work Items

- Apply the new layout to cloudkittys documentation
- Write an introduction with a few schemas
- Improve the documentation (sub-)section by (sub-)section, with one commit/task couple per subject.

6.3.4 Dependencies

None

6.3.5 Testing

None

6.3.6 Documentation Impact

The documentation will be completely modified, and a lot of content will be added.

6.3.7 References

- Glance documentation, which is a good example of a documentation organised per user profile: https://docs.openstack.org/glance/latest/
- Some support matrixes:
  - Elastic product / OS: https://www.elastic.co/support/matrix
  - Kubernetes persistent volumes access modes: https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes

6.4 Reworking Prometheus Collector

CloudKittys Prometheus collector needs some rework. This spec aims at defining what is to be done and why. Everything in this document is open to discussion, equally on the associated storyboard and gerrit.

6.4.1 Problem Description

In the present state, Prometheus collector is not consistent with the other collectors present in CloudKitty in terms of configuration and operations.

- The end user fully defines, on its own, the PromQL query to be executed in the `metrics.yml` file in a `query` option under `extra_args` subkey. Permitting the end user such complexity for building its request at the expense of genericity is not relevant regarding the use cases covered by CloudKitty.
- The collector does not use native PromQL `groupby` capabilities. The process is done post Prometheus service response causing an overhead operational cost.
- The collector ignores `scope_key` configuration option, disallowing scoping requests however it is the standard behaviour to be found in other collectors of the project. Besides, it is a major concern for orchestration. Lets say you deploy several CK processors. Without scoping, you will end up with data duplications likely to result in false reports.

The collector also needs the following new features:

- It needs authentication configuration options to dial into some secured Prometheus service.

- It needs a configuration option to take in account a custom certificate authority file in order to authorize HTTPS requests signed with custom certificate.
- It needs an insecure configuration option to explicitly trust HTTPS requests signed with an untrusted certificate.

These features will be useful to end users for configuring CloudKitty to effectively connect to a Prometheus service.

6.4.2 Proposed Change

In `cloudkitty.conf`, under `[collector_prometheus]` section:

- Add a `prometheus_user` and a `prometheus_password` option to provide credentials for authentication (both defaults: none).
- Add a `cafile` option to allow custom certificate authority file (default: none).
- Add an `insecure` option to explicitly allow insecure HTTPS requests (default: false).

Possible deprecation of `extra_args/query` in `metrics.yml` for a defined metric (see *Alternatives* section).

In `metrics.yml`, for a defined metric, add an `aggregation_method` option under `extra_args` subkey to define which aggregation method to use collecting metric data.

For instance:

```
# metrics.yml

metrics:
  # [...]

  volume_size:
    unit: GiB
    groupby:
      - id
      - project_id
    metadata:
      - volume_type
    extra_args:
      aggregation_method: max
```

Valid aggregation methods will be the following :

- `avg`: the average value of all points in the time window.
- `min`: the minimum value of all points in the time window.
- `max`: the maximum value of all points in the time window.
- `sum`: the sum of all values in the time window.
- `count`: the count of all values in the time window.
- `stddev`: the population standard deviation of the values in the time window.
- `stdvar`: the population standard variance of the values in the time window.

Note: Time window is computed from the `period` configuration option under `[collect]` section in `cloudkitty.conf`.

Other changes at this stage are mostly focused on the query building process happening inside the Prometheus collector:

- `[collect]/scope_key` option value will be added to filter every query with the corresponding `scope_id`.
- The collector will now use the `groupby` capabilities offered by PromQL using the syntax `by (scope_key, groupby, metadata)` instead of deferring it to the collector implementation.

Example

For the following `cloudkitty.conf` :

```
# Some options have been omitted for the sake of clarity

[collect]
collector = prometheus
scope_key = namespace
period = 3600

[collector_prometheus]
prometheus_url = http://prometheus-dn.tld/api/v1
```

And the following `metrics.yml` :

```
metrics:
  # [...]

  # Let's say we want to rate the following metric
  container_memory_usage_bytes:
    unit: GiB
    groupby:
      - container_id
    metadata:
      - volume_type
    extra_args:
      aggregation_method: max
```

The PromQL request will look the following :

```
max(max_over_time(container_memory_usage_bytes{namespace="foobar"}[3600s])) by
(namespace, container_id, volume_type)
```

Start and end timestamps will be computed from the `period` option. Lets say that we are starting from `January 30th, 2019 at 1pm (timestamp: 1548853200)`. Then the end timestamp will be `1548853200 + period (= 1548856800)`.

The HTTP URL with the url-encoded PromQL will then look the following :

```
http://prometheus-dn.tld/api/v1/query?query=max%28max_over_time%28container_memory_usage_byt
```

Alternatives

Instead of deprecation for the `extra_args/query` option defined for a metric in `metrics.yml` file, we could remove it.

Data model impact

None

REST API impact

None

Security impact

None

Notifications Impact

None

Other end user impact

End users will probably have to reconfigure their metrics definition if they are using the Prometheus collector in the present state.

However, after these changes are made, they will have less overhead configuration to do in order to switch from a collector to another, due to the consistency improvement in the way metrics are defined in `metrics.yml` for Prometheus collector.

They will also benefit from the added configuration options for the `[collector_prometheus]` section in `cloudkitty.conf`.

Performance Impact

None

Other deployer impact

None

Developer impact

None

6.4.3 Implementation

Assignee(s)

Justin Ferrieu is assigned to work on the spec as well as to develop the evoked points.

Primary assignee: jferrieu

Other contributors: None

Work Items

- Change configuration schema, query building process and query response formatting process accordingly inside Prometheus collector.
- Add HTTPS and authentication support.

6.4.4 Dependencies

None

6.4.5 Testing

The proposed changes will be tested with unit tests.

6.4.6 Documentation Impact

We will add an entry for detailing the Prometheus collector configuration in `Administration Guide/Configuration Guide/Collector/Prometheus`.

6.4.7 References

- Prometheus documentation: https://prometheus.io/docs/

CHAPTER

SEVEN

PIKE

7.1 (Placeholder Spec)

This file is just a placeholder for Pike specs directory. It will be removed soon after some spec for Pike is merged.

The latest spec template is found at `specs/template.rst` in the `cloudkitty-specs` repository.

7.1.1 Problem Description

Sphinx toctree complains if no file exists in a directory specified in toctree glob.

7.1.2 Proposed Change

Add this file.

7.1.3 References

None.

CHAPTER

EIGHT

OCATA

8.1 (Placeholder Spec)

This file is just a placeholder for Ocata specs directory. It will be removed soon after some spec for Ocata is merged.

The latest spec template is found at `specs/template.rst` in the `cloudkitty-specs` repository.

8.1.1 Problem Description

Sphinx toctree complains if no file exists in a directory specified in toctree glob.

8.1.2 Proposed Change

Add this file.

8.1.3 References

None.

CHAPTER

NINE

INDICES AND TABLES

- search