StackTach Documentation

Release 1.0

Sandy Walsh

Apr 19, 2017

Contents

Contents:

CHAPTER 1

An Introduction to StackTach

StackTach was initially created as a browser-based debugging tool for OpenStack Nova. Since that time, StackTach has evolved into a tool that can do debugging, performance monitoring and perform audit, validation and reconcilation of Nova and Glance usage in a manner suitable for billing.

How it works

Nearly all OpenStack components are capable of generating *notifications* when significant events occur. Notifications are messages placed on the OpenStack queue (generally RabbitMQ) for consumption by downstream systems.

The OpenStack wiki has info on the notification format.

StackTach has a *worker* that is configured to read these notifications and store them in a database (ideally a database separate from the OpenStack production database). From there, StackTach reviews the stream of notifications to glean usage information and assemble it in an easy-to-query fashion.

Users can inquire on instances, requests, servers, etc using the browser interface or command line tool (Stacky).

To get a general sense of notification adoption across OpenStack projects read this blog post

CHAPTER 2

Installing StackTach

The "Hurry Up" Install Guide

1. Create a database for StackTach to use. By default, StackTach assumes MySql, but you can modify the settings.py file to others.
2. Install django and the other required libraries listed in `./etc/pip-requires.txt` (please let us know if any are missing)
3. Clone this repo
4. Copy and configure the config files in `./etc` (see below for details)
5. Create the necessary database tables (python manage.py syncdb) You don't need an administrator account since there are no user profiles used.
6. Configure OpenStack to publish Notifications back into RabbitMQ (see below)
7. Restart the OpenStack services.
8. Run the Worker to start consuming messages. (see below)
9. Run the web server (`python manage.py runserver --insecure`)
10. Point your browser to `http://127.0.0.1:8000` (the default server location)
11. Click on stuff, see what happens. You can't hurt anything, it's all read-only.

Of course, this is only suitable for playing around. If you want to get serious about deploying StackTach you should set up a proper webserver and database on standalone servers. There is a lot of data that gets collected by StackTach (depending on your deployment size) ... be warned. Keep an eye on DB size.

The Config Files

There are two config files for StackTach. The first one tells us where the second one is. A sample of these two files is in `./etc/sample_*`. Create a local copy of these files and populate them with the appropriate config values as

described below.

The `sample_stacktach_config.sh` shell script defines the necessary environment variables StackTach needs. Most of these are just information about the database (assuming MySql) but some are a little different. Copy this file and modify it for your environment. `source` this `stacktach_config.sh` shell script to set up the necessary environment variables.

`STACKTACH_INSTALL_DIR` should point to where StackTach is running out of. In most cases this will be your repo directory, but it could be elsewhere if your going for a proper deployment. The StackTach worker needs to know which RabbitMQ servers to listen to. This information is stored in the deployment file. `STACKTACH_DEPLOYMENTS_FILE` should point to this json file. To learn more about the deployments file, see further down.

Finally, `DJANGO_SETTINGS_MODULE` tells Django where to get its configuration from. This should point to the `setting.py` file. You shouldn't have to do much with the `settings.py` file and most of what it needs is in these environment variables.

The `sample_stacktach_worker_config.json` file tells StackTach where each of the RabbitMQ servers are that it needs to get events from. In most cases you'll only have one entry in this file, but for large multi-cell deployments, this file can get pretty large. It's also handy for setting up one StackTach for each developer environment.

The file is in json format and the main configuration is under the `deployments` key, which should contain a list of deployment dictionaries.

A blank worker config file would look like this:

```
{"deployments": [] }
```

But that's not much fun. A deployment entry would look like this:

```
{"deployments": [
     {
         "name": "east_coast.prod.cell1",
         "durable_queue": false,
         "rabbit_host": "10.0.1.1",
         "rabbit_port": 5672,
         "rabbit_userid": "rabbit",
         "rabbit_password": "rabbit",
         "rabbit_virtual_host": "/",
         "topics": {
             "nova": [
                 {"queue": "notifications.info", "routing_key": "notifications.info"},
                 {"queue": "notifications.error", "routing_key": "notifications.error
↪"},
             ]
         }
     }
]}
```

where, *name* is whatever you want to call your deployment, and *rabbit_** are the connectivity details for your rabbit server. It should be the same information in your *nova.conf* file that OpenStack is using. Note, json has no concept of comments, so using `#`, `//` or `/* */` as a comment won't work.

By default, Nova uses ephemeral queues. If you are using durable queues, be sure to change the necessary flag here.

The topics section defines which queues to pull notifications from. You should pull notifications from all related queues (`.error`, `.info`, `.warn`, etc)

You can add as many deployments as you like.

Starting the Worker

Note: the worker now uses librabbitmq, be sure to install that first.

`./worker/start_workers.py` will spawn a worker.py process for each deployment defined. Each worker will consume from a single Rabbit queue.

Configuring Nova to Generate Notifications

In the OpenStack service you wish to have generate notifications, add the following to its `.conf` file:

```
--notification_driver=nova.openstack.common.notifier.rpc_notifier
--notification_topics=monitor
```

Note: *This will likely change once the various project switch to ''oslo.messaging'' which uses endpoints to define the notification drivers.*

This will tell OpenStack to publish notifications to a Rabbit exchange starting with `monitor.*` ... this may result in `monitor.info`, `monitor.error`, etc.

You'll need to restart Nova once these changes are made.

If you're using DevStack you may want to set up your `local.conf` to include the following:

```
[[post-config|$NOVA_CONF]]
[DEFAULT]
notification_driver=nova.openstack.common.notifier.rpc_notifier
notification_topics=notifications,monitor
notify_on_state_change=vm_and_task_state
notify_on_any_change=True
instance_usage_audit=True
instance_usage_audit_period=hour
```

Next Steps

Once you have this working well, you should download and install `Stacky` and play with the command line tool.

CHAPTER 3

The StackTach REST Interface

JSON Response Format

StackTach uses an tabular JSON response format to make it easier for Stacky to display generic results.

The JSON response format is as follows:

```
[
  ['column header', 'column header', 'column header', ...],
  ['row 1, col 1', 'row 1, col 2', 'row 1, col 3', ...],
  ['row 2, col 1', 'row 2, col 2', 'row 2, col 3', ...],
  ['row 3, col 1', 'row 3, col 2', 'row 3, col 3', ...],
  ...
]
```

stacky/deployments

GET http://example.com/stacky/deployments/
The list of all available deployments

Example request:

```
GET /stacky/deployments/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  ['#', 'Name'],
```

```
  [1, 'deployment name'],
  [2, 'deployment name'],
  ...
]
```

stacky/events

GET http://example.com/stacky/events/

The distinct list of all event names

Example request:

```
GET /stacky/events/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  ['Event Name'],
  ["add_fixed_ip_to_instance"],
  ["attach_volume"],
  ["change_instance_metadata"],
  ["compute.instance.create.end"],
  ["compute.instance.create.error"],
  ["compute.instance.create.start"],
  ["compute.instance.create_ip.end"],
  ...
]
```

Query Parameters

- **service** – nova or glance. default="nova"

stacky/hosts

GET http://example.com/stacky/hosts/

The distinct list of all hosts sending notifications.

Example request:

```
GET /stacky/hosts/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json
```

```
[
  ['Host Name'],
  ["compute-1"],
  ["compute-2"],
  ["scheduler-x"],
  ["api-88"],
  ...
```

Query Parameters

- **service** – nova or glance. default="nova"]

stacky/uuid

GET http://example.com/stacky/uuid/

Retrieve all notifications for instances with a given UUID.

Example request:

```
GET /stacky/uuid/?uuid=77e0f192-00a2-4f14-ad56-7467897828ea  HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  ["#", "?", "When", "Deployment", "Event", "Host", "State",
   "State'", "Task"],
  [
      40065869,
      " ",
      "2014-01-14 15:39:22.574829",
      "region-1",
      "compute.instance.snapshot.start",
      "compute-99",
      "active",
      "",
      ""
  ],
  [
      40065879,
      " ",
      "2014-01-14 15:39:23.599298",
      "region-1",
      "compute.instance.update",
      "compute-99",
      "active",
      "active",
      "image_snapshot"
  ],
```

```
  ...
]
```

Query Parameters

- **uuid** – UUID of desired instance.
- **service** – nova or glance. default=”nova”

stacky/timings/uuid/

GET http://example.com/stacky/timings/uuid/

Retrieve all timings for a given instance. Timings are the time deltas between related .start and .end notifications. For example, the time difference between compute.instance.run_instance.start and compute.instance.run_instance.end. This url works only for nova.

The first column of the response will be

- S if there is a .start event and no .end
- E if there is a .end event and no .start
- . if there was a .start and .end event

No time difference will be returned in the S or E cases.

Example request:

```
GET /stacky/timings/uuid/?uuid=77e0f192-00a2-4f14-ad56-7467897828ea HTTP/
↪1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  ["?", "Event", "Time (secs)"],
  [".", "compute.instance.create", "0d 00:00:55.50"],
  [".", "compute.instance.snapshot", "0d 00:14:11.71"],
  [".", "compute.instance.snapshot", "0d 00:17:31.33"],
  [".", "compute.instance.snapshot", "0d 00:16:48.88"]
  ...
]
```

Query Parameters

- **uuid** – UUID of desired instance.

stacky/summary

GET http://example.com/stacky/summary/

Returns timing summary information for each event type collected. Only notifications with `.start/.end` pairs are considered. This url works only for nova.

This includes:

```
* the number of events seen of each type (N)
* the Minimum time seen
* the Maximum time seen
* the Average time seen
```

Example request:

```
GET /stacky/summary/  HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  ["Event", "N", "Min", "Max", "Avg"],
  ["compute.instance.create", 50,
      "0d 00:00:52.88", "0d 01:41:14.27", "0d 00:08:26"],
  ["compute.instance.create_ip", 50,
      "0d 00:00:06.80", "5d 20:16:47.08", "0d 03:47:17"],
  ...
]
```

Query Parameters

- **uuid** – UUID of desired instance.
- **limit** – the number of timings to return.
- **offset** – offset into query result set to start from.

stacky/request

GET http://example.com/stacky/request/

Returns all notifications related to a particular Request ID.

The `?` column will be `E` if the event came from the `.error` queue. `State` and `State'` are the current state and the previous state, respectively. This url works only for nova.

Example request:

```
GET /stacky/request/?request_id=req-a7517402-6192-4d0a-85a1-e14051790d5a
↪HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  ["#", "?", "When", "Deployment", "Event", "Host", "State",
   "State'", "Task'"
  ],
  [
      40368306,
      " ",
      "2014-01-15 15:39:34.130286",
      "region-1",
      "compute.instance.update",
      "api-1",
      "active",
      "active",
      null
  ],
  [
      40368308,
      " ",
      "2014-01-15 15:39:34.552434",
      "region-1",
      "compute.instance.update",
      "api-1",
      "active",
      null,
      null
  ],

  ...
]
```

Query Parameters

- **request_id** – desired request ID
- **when_min** – unixtime to start search
- **when_max** – unixtime to end search
- **limit** – the number of timings to return.
- **offset** – offset into query result set to start from.

stacky/reports

GET http://example.com/stacky/reports/

Returns a list of all available reports.

The `Start` and `End` columns refer to the time span the report covers (in unixtime).

Example request:

```
GET /stacky/reports/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  ["Id", "Start", "End", "Created", "Name", "Version"],
  [
      5971,
      1389726000.0,
      1389729599.0,
      1389730212.9474499,
      "summary for region: all",
      4
  ],
  [
      5972,
      1389729600.0,
      1389733199.0,
      1389733809.979934,
      "summary for region: all",
      4
  ],

  ...
]
```

Query Parameters

- **created_from** – unixtime to start search
- **created_to** – unixtime to end search
- **limit** – the number of timings to return.
- **offset** – offset into query result set to start from.

stacky/report/<report_id>

GET http://example.com/stacky/report/<report_id>

Returns a specific report.

The contents of the report varies by the specific report, but all are in row/column format with Row 0 being a special *metadata* row.

Row 0 of each report is a dictionary of metadata about the report. The actual row/columns of the report start at Row 1 onwards (where Row 1 is the Column headers and Rows 2+ are the details, as with other result sets)

Example request:

```
GET /stacky/report/1/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
```

```
Content-Type: text/json

[
  {
      "4xx failure count": 0,
      "4xx failure percentage": 0.0,
      "5xx failure count": 1,
      "5xx failure percentage": 0.018284904,
      "> 30 failure count": 13,
      "> 30 failure percentage": 1.13479794,
      "cells": [
          "c0001",
          "global",
          "c0003",
          "c0004",
          "c0011",
          "c0010",
          "a0001",
          "c0012",
          "b0002",
          "a0002"
      ],
      "end": 1389729599.0,
      "failure_grand_rate": 0.2445074415308293,
      "failure_grand_total": 14,
      "hours": 1,
      "pct": 0.014999999999999999,
      "percentile": 97,
      "region": null,
      "start": 1389726000.0,
      "state failure count": 0,
      "state failure percentage": 0.0,
      "total": 411
  },
  ["Operation", "Image", "OS Type", "Min", "Max", "Med", "97%", "Requests",
   "4xx", "% 4xx", "5xx", "% 5xx", "> 30", "% > 30", "state", "% state"],
  [
      "aux",
      "snap",
      "windows",
      "0s",
      "5s",
      "0s",
      "5s",
      6,
      0,
      0.0,
      0,
      0.0,
      0,
      0.0,
      0,
      0.0
  ],
  [
      "resize",
      "base",
      "linux",
```

```
      "1s",
      "5:44s",
      "1:05s",
      "3:44s",
      9,
      0,
      0.0,
      0,
      0.0,
      0,
      0.0,
      0,
      0.0
  ],

  ...
]
```

stacky/reports/search/

GET http://example.com/stacky/reports/search

Returns reports that match the search criteria in descending order of id.

The contents of the report varies by the specific report, but all are in row/column format with Row 0 being a special *metadata* row. The actual row/columns of the report start at Row 1 onwards.

Example request:

```
GET /stacky/reports/search/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  [
      "Id",
      "Start",
      "End",
      "Created",
      "Name",
      "Version"
  ],

  [
      4253,
      "2013-11-21 00:00:00",
      "2013-11-22 00:00:00",
      "2013-11-22 01:44:55",
      "public outbound bandwidth",
      1
  ],
```

```
  [
      4252,
      "2014-01-18 00:00:00",
      "2013-11-22 00:00:00",
      "2013-11-22 01:44:55",
      "image events audit",
      1
  ],
  [
      4248,
      "2013-11-21 00:00:00",
      "2013-11-22 00:00:00",
      "2013-11-22 01:44:55",
      "Error detail report",
      1
  ],

  ...
]
```

Query Parameters

- **id** – integer report id
- **name** – string report name(can include spaces)
- **period_start** – start of period, which the report pertains to, in the following format: YYYY-MM-DD HH:MM[:ss[.uuuuuu]][TZ]
- **period_end** – end of period, which the report pertains to, in the following format: YYYY-MM-DD HH:MM[:ss[.uuuuuu]][TZ]
- **created** – the day, when the report was created, in the following format: YYYY-MM-DD

stacky/show/<event_id>

GET http://example.com/stacky/show/<event_id>/

Show the details on a specific notification.

The response of this operation is non-standard. It returns 3 rows:

•The first row is the traditional row-column result set used by most commands.

•The second row is a prettied, stringified version of the full JSON payload of the raw notification.

•The third row is the UUID of the related instance, if any.

Example request:

```
GET /stacky/show/1234/  HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

```

```
[
  [
    ["Key", "Value"],
    ["#", 1234 ],
    ["When", "2014-01-15 20:39:44.277745"],
    ["Deployment", "region-1"],
    ["Category", "monitor.info"],
    ["Publisher", "compute-1"],
    ["State", "active"],
    ["Event", "compute.instance.update"],
    ["Service", "compute"],
    ["Host", "compute-1"],
    ["UUID", "8eba1a6d-43eb-1343-8d1a-5e596f5233b5"],
    ["Req ID", "req-1368539d-f645-4d96-842e-03b5c5c9dc8c"],
    ...
  ],
  "[\n  \"monitor.info\", \n  {\n    \"_context_request_id\": \"req-
↪13685e9d-f645-4d96-842e-03b5c5c9dc8c\", \n    \"_context_quota_class\":␣
↪null, \n    \"event_type\": \"compute.instance.update\", \n    \"_
↪context_service_catalog\": [], \n    \"_context_auth_token\": \
↪"d81a25d03bb340bb82b4b67d105cc42d\", \n    \"_context_user_id\": \
↪"b83e2fac644c4215bc449fb4b5c9bbfa\", \n    \"payload\": {\n      \
↪"state_description\": \"\", \n      \"availability_zone\": null, \n␣
↪ \"terminated_at\": \"\", \n      \"ephemeral_gb\": 300, \n ...",
  "8eba1a6d-43eb-1343-8d1a-5e596f5233b5"
]
```

Query Parameters

- **service** – nova or glance. default="nova"
- **event_id** – desired Event ID

stacky/watch/<deployment_id>

GET http://example.com/stacky/watch/<deployment_id>/

Get a real-time feed of notifications.

Once again, this is a non-standard response (not the typical row-column format). This call returns a tuple of information:

- A list of column widths, to be used as a hint for formatting.
- A list of events that meet the query criteria. * the db id of the event * the type of event (E for errors, . otherwise) * stringified date of the event * stringified time of the event * deployment name * the event name * the instance UUID, if available
- The ending unixtime timestamp. The last time covered by this query (utcnow, essentially)

Example request:

```
GET /stacky/watch/14/  HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  [10, 1, 15, 20, 50, 36],
  [
  ... events ...
  ]
  "1389892207"
]
```

Query Parameters

- **service** – nova or glance. default="nova"
- **since** – get all events since unixtime. Defaults to 2 seconds ago.
- **event_name** – only watch for event_name notifications. Defaults to all events.

stacky/search

GET http://example.com/stacky/search/

Search for notifications.

Returns:

•Event ID

•E for errors, . otherwise

•unixtime for when the event was generated

•the deployment name

•the event name

•the host name

•the instance UUID

•the request ID

Example request:

```
GET /stacky/search/  HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

[
  [...event info as listed above...]
]
```

Query Parameters

- **service** – nova or glance. default=”nova”
- **field** – notification field to search on.
- **value** – notification values to find.
- **when_min** – unixtime to start search
- **when_max** – unixtime to end search

CHAPTER 4

StackTach Usage Verification

Usage Basics

In OpenStack, usage is tracked through notifications. The notifications are emitted by each service as users request changes and each service performs those changes. Services like Nova can also be configured to emitted periodic audit notifications exposing the state of the database at the time of the audit. The periodic audit notifications are useful for billing as it is not necessary to store past states.

But, we want to be sure what we're billing for is correct and that we've received audit notifications for every instance that should be billable. Thus, it is a good idea to track instance state so that periodic audit notifications can be validated against that state. The notifications each service sends as changes are requested and performed are extremely useful for tracking instance state through different billable states.

The idea behind StackTach's Usage Verification is to track changes through instantaneous notifications, then compare them to the periodic audit notifications for correctness. After being validated, StackTach itself will emit a copy of the notification with a new event_type indicating that is has been verified. StackTach also provides a set of scripts which can be used to confirm that exists were sent for all instances in a billable state.

Configuring Usage Verification

Usage Verification in StackTach is done by a separate verifier process. A sample configuration file can be found at `./etc/sample_stacktach_verifier_config.sjon`

The default config provides most all settings that are required for the verifier.

```
{
    "tick_time": 30,
    "settle_time": 5,
    "settle_units": "minutes",
    "pool_size": 2,
    "enable_notifications": true,
    "validation_level": "all",
    "flavor_field_name": "instance_type_id",
```

```
    "rabbit": {
        "durable_queue": false,
        "host": "10.0.0.1",
        "port": 5672,
        "userid": "rabbit",
        "password": "rabbit",
        "virtual_host": "/",
        "topics": {
            "nova": ["notifications.info"],
            "glance": ["notifications.info"]
        }
    }
}
```

- tick_time: Time in seconds to sleep before attempting to retrieve pending usage entries for verifications
- settle_time: Amount of time between when a usage notification was emitted and when it should be picked up for verification.
- settle_units: Units for the settle_time value
- pool_size: Amount of verifier processes to create for the verifier pool.
- enable_notifications: Whether or not to emit verified notifications.
- validation_level: Determines how strict datatype validation will be on usage notifications. Values are `none`, `basic`, and `all`.
- flavor_field_name: Field to use for flavor verification. Values are `instance_type_id` and `instance_flavor_id`.
- rabbit: Rabbit config, please see *StackTach install guide* for rabbit config details.
- The topics here are how the verifier determines which services to verify. For example, Nova and Glance services will be verified and verified notifications will be emitted with a routing_key of notifications.info with our sample config.
- An alternate config that would only verify Nova and emit verified notifications on notifications.info and monitor.info:

```
"topics": {
    "nova": ["notifications.info", "monitor.info"]
}
```

- Other Config Options:
  - nova_event_type: Event type to emit for Nova events
  - Default: compute.instance.exists.verified.old
  - glance_event_type: Event type to emit for Glance events
  - Default: image.exists.verified.old

Starting the Verifier

`./verifier/start_verifier.py` will spawn a verifier.py process for each service being verified along with a pool of processes to verify each usage entry.

Audit Reports

StackTach also provides a few reports for auditing the audit notifications, which can be useful for confirming all usage was sent for a deployment.

- `./reports/nova_usage_audit.py`
- Suggested Arguments:
- –period_length: `day` or `year`, default: `day`
- –utcdatetime: Overrides datetime used to audit, default: current utc datetime
- –store: `True` or `False`, whether or not to store report in StackTach database
- `./reports/glance_usage_audit`
- Suggested Arguments:
- –period_length: `day` or `year`, default: `day`
- –utcdatetime: Overrides datetime used to audit, default: current utc datetime
- –store: `True` or `False`, whether or not to store report in StackTach database

CHAPTER 5

The StackTach Database REST Interface

JSON Response Format

The StackTach Database API uses a more standard data model for access to database objects. The Database API is read only, with the exception of usage confirmation, which is used to indicate that usage has been sent downstream.

The JSON response format uses an envelope with a single key to indicate the type of object returned. This object can be either a dictionary in the case of queries that return single objects, or a list when multiple objects are turned.

Sample JSON response, single object:

```
{
  "enitity":
  {
    "id": 1
    "key1": "value1",
    "key2": "value2"
  }
}
```

Sample JSON response, multiple objects:

```
{
  "enitities":
  [
    {
      "id": 1,
      "key1": "value1",
      "key2": "value2"
    },
    {
      "id": 2,
      "key1": "value1",
      "key2": "value2"
    }
```

```
  ]
}
```

Write APIs

db/confirm/usage/exists/batch/

PUT http://example.com/db/confirm/usage/exists/batch/

Uses the provided message_id's and http status codes to update image and instance exists send_status values.

Example V0 request:

```
PUT db/confirm/usage/exists/batch/ HTTP/1.1
Host: example.com
Accept: application/json

{
  "messages":
  [
    {"nova_message_id": 200},
    {"nova_message_id": 400}
  ]
}
```

Example V1 request:

```
PUT db/confirm/usage/exists/batch/ HTTP/1.1
Host: example.com
Accept: application/json

{
  "messages":
  [
    {
      "nova":
      [
        {"nova_message_id1": 200},
        {"nova_message_id2": 400}
      ],
      "glance":
      [
        {"glance_message_id1": 200},
        {"glance_message_id2": 400}
      ]
    }
  ]
  "version": 1
}
```

Example V2 request:

```
PUT db/confirm/usage/exists/batch/ HTTP/1.1
Host: example.com
Accept: application/json
```

```
{
  "messages":
  [
    {
      "nova":
      [
        {"nova_message_id1":
            {
              "event_id": "AH_event_id_1",
              "status": 201
            }
        },
        {"nova_message_id2":
            {
              "event_id": "AH_event_id_2",
              "status": 201
            }
        }
      ],
      "glance":
      [
        {"glance_message_id1":
             {
              "event_id": "AH_event_id_3",
              "status": 201},
              }
        },
        {"glance_message_id2":
             {
              "event_id": "AH_event_id_3",
              "status": 201
              }
        }
      ]
    }
  ]
  "version": 2
}
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json
```

Read APIs

db/stats/events

GET http://example.com/db/stats/events/

Returns a count of events stored in Stacktach's Rawdata tables from `when_min` to `when_max`

Query Parameters

- `event`: event type to filter by

- `when_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `when_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `service`: `nova` or `glance`. default="nova"

Example request:

```
GET db/stats/events/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  count: 10
}
```

db/stats/nova/exists/

`GET http://example.com/db/stats/nova/exists`

Returns a list of status combinations and count of events with those status combinations.

Note: Only status combinations with >0 count will show up.

Query Parameters

- `audit_period_beginning_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_beginning_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_ending_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_ending_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `launched_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `launched_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `received_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `received_max`: datetime (yyyy-mm-dd hh:mm:ss)

Example request:

```
GET /db/stats/nova/exists/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "stats":
  [
    {"status": "pending", "send_status": 0, "event_count": 1},
    {"status": "verified", "send_status": 200, "event_count": 100},
    {"status": "reconciled", "send_status": 200, "event_count": 2},
    {"status": "failed", "send_status": 0, "event_count": 1},
  ]
}
```

db/stats/glance/exists/

GET http://example.com/db/status/usage/glance/exists

Returns a list of status combinations and count of events with those status combinations.

Note: Only status combinations with >0 count will show up.

Query Parameters

- audit_period_beginning_min: datetime (yyyy-mm-dd hh:mm:ss)
- audit_period_beginning_max: datetime (yyyy-mm-dd hh:mm:ss)
- audit_period_ending_min: datetime (yyyy-mm-dd hh:mm:ss)
- audit_period_ending_max: datetime (yyyy-mm-dd hh:mm:ss)
- created_at_min: datetime (yyyy-mm-dd hh:mm:ss)
- created_at_max: datetime (yyyy-mm-dd hh:mm:ss)
- deleted_at_min: datetime (yyyy-mm-dd hh:mm:ss)
- deleted_at_max: datetime (yyyy-mm-dd hh:mm:ss)
- received_min: datetime (yyyy-mm-dd hh:mm:ss)
- received_max: datetime (yyyy-mm-dd hh:mm:ss)

Example request:

```
GET /db/stats/nova/exists/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "stats":
  [
    {"status": "verified", "send_status": 200, "event_count": 200},
    {"status": "failed", "send_status": 0, "event_count": 2},
```

```
  ]
}
```

db/usage/launches/

GET http://example.com/db/usage/launches/

Deprecated, see: *db/usage/nova/launches/*

db/usage/nova/launches/

GET http://example.com/db/usage/nova/launches/

Returns a list of instance launches matching provided query criteria.

Query Parameters

- `launched_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `launched_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `instance`: uuid
- `limit`: int, default: 50, max: 1000
- `offset`: int, default: 0

Example request:

```
GET /db/usage/nova/launches/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "launches":
  [
    {
      "os_distro": "org.centos",
      "os_version": "5.8",
      "instance_flavor_id": "2",
      "instance_type_id": "2",
      "launched_at": "2014-01-17 15:35:44",
      "instance": "72e4d8e8-9f63-47cb-a904-0193e5edac6e",
      "os_architecture": "x64",
      "request_id": "req-7a86ed49-e1f4-4403-b3ef-22636f7acb7d",
      "rax_options": "0",
      "id": 91899,
      "tenant": "5853600"
    },
    {
      "os_distro": "org.centos",
      "os_version": "5.8",
```

```
      "instance_flavor_id": "performance1-4",
      "instance_type_id": "11",
      "launched_at": "2014-01-17 15:35:20",
      "instance": "932bcfd9-af68-4261-805e-6e43156c3b40",
      "os_architecture": "x64",
      "request_id": "req-6bfe911f-40f2-4fd8-946a-070c10bed014",
      "rax_options": "0",
      "id": 91898,
      "tenant": "5853595"
    }
  ]
}
```

db/usage/glance/images/

GET http://example.com/db/usage/glance/images/

Returns a list of images matching provided query criteria.

Query Parameters

- `created_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `created_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `limit`: int, default: 50, max: 1000
- `offset`: int, default: 0

Example request:

```
GET /db/usage/glance/images/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "images":
  [
    {
      "uuid": "2048efd8-fdce-4123-bdbc-add3bfe64b83",
      "created_at": "2014-01-17 02:28:08",
      "owner": null,
      "last_raw": 299977,
      "id": 4837,
      "size": 9192352
    },
    {
      "uuid": "aa2c07dd-fd1c-4ad3-9f73-6a6d7d8a0dbd",
      "created_at": "2014-01-17 02:24:18",
      "owner": "5937488",
      "last_raw": 299967,
      "id": 4836,
      "size": 9
```

```
    }
  ]
}
```

db/usage/launches/<launch_id>/

GET http://example.com/db/usage/launches/<launch_id>/

Deprecated, see: *db/usage/nova/launches/<launch_id>/*

db/usage/nova/launches/<launch_id>/

GET http://example.com/db/usage/nova/launches/<launch_id>/

Returns the single launch with id matching the provided id.

Example request:

```
GET /db/usage/nova/launches/91898/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "launch":
  {
    "os_distro": "org.centos",
    "os_version": "5.8",
    "instance_flavor_id": "performance1-4",
    "instance_type_id": "11",
    "launched_at": "2014-01-17 15:35:20",
    "instance": "932bcfd9-af68-4261-805e-6e43156c3b40",
    "os_architecture": "x64",
    "request_id": "req-6bfe911f-40f2-4fd8-946a-070c10bed014",
    "rax_options": "0",
    "id": 91898,
    "tenant": "5853595"
  }
}
```

db/usage/glance/images/<image_id>/

GET http://example.com/db/usage/glance/images/<image_id>/

Returns the single image with id matching the provided id.

Example request:

```
GET /db/usage/glance/images/4836/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "launch":
  {
    "uuid": "aa2c07dd-fd1c-4ad3-9f73-6a6d7d8a0dbd",
    "created_at": "2014-01-17 02:24:18",
    "owner": "5937488",
    "last_raw": 299967,
    "id": 4836,
    "size": 9
  }
}
```

db/usage/deletes/

GET http://example.com/db/usage/deletes/

Deprecated, see: *db/usage/nova/deletes/*

db/usage/nova/deletes/

GET http://example.com/db/usage/nova/deletes/

Returns a list of instance deletes matching provided query criteria.

Query Parameters

- `launched_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `launched_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `instance`: uuid
- `limit`: int, default: 50, max: 1000
- `offset`: int, default: 0

Example request:

```
GET /db/usage/nova/deletes/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "deletes":
  [
    {
      "raw": 14615347,
      "instance": "b36a8c2d-af88-4371-b14c-14dadf7073e5",
      "deleted_at": "2014-01-17 16:07:30",
      "id": 65110,
      "launched_at": "2014-01-17 16:06:54"
    },
    {
      "raw": 14615248,
      "instance": "3fd6797d-bc35-42d9-ad85-157a2ea93023",
      "deleted_at": "2014-01-17 16:05:23",
      "id": 65108,
      "launched_at": "2014-01-17 16:05:00"
    }
  ]
}
```

db/usage/glance/deletes/

GET http://example.com/db/usage/glance/deletes/

Returns a list of image deletes matching provided query criteria.

Query Parameters

- `deleted_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `limit`: int, default: 50, max: 1000
- `offset`: int, default: 0

Example request:

```
GET /db/usage/glance/deletes/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "deletes":
  [
    {
      "raw": 300523,
      "deleted_at": "2014-01-17 15:28:18.154927",
      "id": 3169,
```

```
      "uuid": "f8b02f0e-b392-40f5-9d39-0458ae6ebfb3"
    },
    {
      "raw": 300512,
      "deleted_at": "2014-01-17 14:28:20.544617",
      "id": 3168,
      "uuid": "4c9dc0be-856b-4e98-81a5-1b63df108e7d"
    }
  ]
}
```

db/usage/deletes/<delete_id>/

GET http://example.com/db/usage/deletes/

Deprecated, see: *db/usage/nova/deletes/<delete_id>/*

db/usage/nova/deletes/<delete_id>/

GET http://example.com/db/usage/nova/deletes/<deleted_id>

Returns the single instance delete with id matching the provided id.

Example request:

```
GET /db/usage/nova/deletes/65110/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "delete":
  {
    "raw": 14615347,
    "instance": "b36a8c2d-af88-4371-b14c-14dadf7073e5",
    "deleted_at": "2014-01-17 16:07:30",
    "id": 65110,
    "launched_at": "2014-01-17 16:06:54"
  }
}
```

db/usage/glance/deletes/<delete_id>/

GET http://example.com/db/usage/glance/deletes/<deleted_id>

Returns the single image delete with id matching the provided id.

Example request:

```
GET /db/usage/glance/deletes/3168/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "delete":
  {
    "raw": 300512,
    "deleted_at": "2014-01-17 14:28:20.544617",
    "id": 3168,
    "uuid": "4c9dc0be-856b-4e98-81a5-1b63df108e7d"
  }
}
```

db/usage/exists/

GET http://example.com/db/usage/exists/

Deprecated, see: *db/usage/nova/exists/*

db/usage/nova/exists/

GET http://example.com/db/usage/nova/exists/

Returns a list of instance exists matching provided query criteria.

Query Parameters

- `audit_period_beginning_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_beginning_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_ending_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_ending_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `launched_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `launched_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `received_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `received_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `instance`: uuid
- `limit`: int, default: 50, max: 1000
- `offset`: int, default: 0

Example request:

```
GET /db/usage/nova/exists/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "exists":
  [
    {
      "status": "verified",
      "os_distro": "org.centos",
      "bandwidth_public_out": 0,
      "received": "2014-01-17 16:16:43.695474",
      "instance_type_id": "2",
      "raw": 14615544,
      "os_architecture": "x64",
      "rax_options": "0",
      "audit_period_ending": "2014-01-17 16:16:43",
      "deleted_at": null,
      "id": 135106,
      "tenant": "5889124",
      "audit_period_beginning": "2014-01-17 00:00:00",
      "fail_reason": null,
      "instance": "978b32ea-374b-48c6-814b-bb6151e2fb5c",
      "instance_flavor_id": "2",
      "launched_at": "2014-01-17 16:16:09",
      "os_version": "6.0",
      "usage": 91932,
      "send_status": 201,
      "message_id": "9d28fa15-d163-40c7-8195-2853ad13179b",
      "delete": null
    },
    {
      "status": "verified",
      "os_distro": "org.centos",
      "bandwidth_public_out": 0,
      "received": "2014-01-17 16:10:42.112505",
      "instance_type_id": "2",
      "raw": 14615459,
      "os_architecture": "x64",
      "rax_options": "0",
      "audit_period_ending": "2014-01-17 16:10:42",
      "deleted_at": null,
      "id": 135105,
      "tenant": "5824940",
      "audit_period_beginning": "2014-01-17 00:00:00",
      "fail_reason": null,
      "instance": "860b5df0-d58b-498d-8838-7156d701732c",
      "instance_flavor_id": "2",
      "launched_at": "2014-01-17 16:10:08",
      "os_version": "5.9",
      "usage": 91937,
      "send_status": 201,
```

```
      "message_id": "0a6b1c58-8443-4788-ac08-05cd03e6be53",
      "delete": null
    }
  ]
}
```

db/usage/glance/exists/

GET http://example.com/db/usage/glance/exists/

Returns a list of instance exists matching provided query criteria.

Query Parameters

- `audit_period_beginning_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_beginning_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_ending_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `audit_period_ending_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `created_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `created_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `deleted_at_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `received_min`: datetime (yyyy-mm-dd hh:mm:ss)
- `received_max`: datetime (yyyy-mm-dd hh:mm:ss)
- `limit`: int, default: 50, max: 1000
- `offset`: int, default: 0

Example request:

```
GET /db/usage/glance/exists/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "exists":
  [
    {
      "status": "verified",
      "audit_period_beginning": "2014-01-13 00:00:00",
      "fail_reason": null,
      "uuid": "d39a04bd-6ba0-4d20-8591-937ab43897dc",
      "usage": 2553,
      "created_at": "2013-05-11 15:37:34",
      "size": 11213393920,
      "owner": "389886",
```

```
      "message_id": "9c5fd5af-60b4-45ad-b524-c4a9964f31e4",
      "raw": 283303,
      "audit_period_ending": "2014-01-13 23:59:59",
      "received": "2014-01-13 09:20:02.777965",
      "deleted_at": null,
      "send_status": 0,
      "id": 5301,
      "delete": null
    },
    {
      "status": "verified",
      "audit_period_beginning": "2014-01-13 00:00:00",
      "fail_reason": null,
      "uuid": "6713c136-0555-4a93-b726-edb181d4b69e",
      "usage": 1254,
      "created_at": "2013-05-11 15:37:56",
      "size": 11254732800,
      "owner": "389886",
      "message_id": "9c5fd5af-60b4-45ad-b524-c4a9964f31e4",
      "raw": 283303,
      "audit_period_ending": "2014-01-13 23:59:59",
      "received": "2014-01-13 09:20:02.777965",
      "deleted_at": null,
      "send_status": 0,
      "id": 5300,
      "delete": null
    }
  ]
}
```


db/usage/exists/<exist_id>/

GET http://example.com/db/usage/exists/<exist_id>

Deprecated, see: *db/usage/nova/exists/<exist_id>/*

db/usage/nova/exists/<exist_id>/

GET http://example.com/db/usage/nova/exists/<exist_id>

Returns a single instance exists matching provided id

Example request:

```
GET /db/usage/nova/exists/135105/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:


```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "exist":
```

```
  {
    "status": "verified",
    "os_distro": "org.centos",
    "bandwidth_public_out": 0,
    "received": "2014-01-17 16:10:42.112505",
    "instance_type_id": "2",
    "raw": 14615459,
    "os_architecture": "x64",
    "rax_options": "0",
    "audit_period_ending": "2014-01-17 16:10:42",
    "deleted_at": null,
    "id": 135105,
    "tenant": "5824940",
    "audit_period_beginning": "2014-01-17 00:00:00",
    "fail_reason": null,
    "instance": "860b5df0-d58b-498d-8838-7156d701732c",
    "instance_flavor_id": "2",
    "launched_at": "2014-01-17 16:10:08",
    "os_version": "5.9",
    "usage": 91937,
    "send_status": 201,
    "message_id": "0a6b1c58-8443-4788-ac08-05cd03e6be53",
    "delete": null
  }
}
```


db/usage/glance/exists/<exist_id>/

GET http://example.com/db/usage/glance/exists/<exist_id>/

Returns a single instance exists matching provided id

Example request:

```
GET /db/usage/glance/exists/5300/ HTTP/1.1
Host: example.com
Accept: application/json
```

Example response:


```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: application/json

{
  "exist":
  {
    "status": "verified",
    "audit_period_beginning": "2014-01-13 00:00:00",
    "fail_reason": null,
    "uuid": "6713c136-0555-4a93-b726-edb181d4b69e",
    "usage": 1254,
    "created_at": "2013-05-11 15:37:56",
    "size": 11254732800,
    "owner": "389886",
    "message_id": "9c5fd5af-60b4-45ad-b524-c4a9964f31e4",
    "raw": 283303,
```

```
    "audit_period_ending": "2014-01-13 23:59:59",
    "received": "2014-01-13 09:20:02.777965",
    "deleted_at": null,
    "send_status": 0,
    "id": 5300,
    "delete": null
  }
}
```

/db/repair

POST http://example.com/db/repair/

Changes the status of all the exists of message-ids sent with the request from 'pending' to 'sent_unverified' so that the verifier does not end up sending .verified for all those exists(since the .exists have already been modified as .verified and sent to AH by Yagi). It sends back the message-ids of exists which could not be updated in the json response.

Example request:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/json

{
u'exists_not_pending': [u'494ebfce-0219-4b62-b810-79039a279620'],
u'absent_exists': [u'7609f3b2-3694-4b6f-869e-2f13ae504cb2',
                   u'0c64032e-4a60-44c0-a99d-5a4f2e46afb0']
}
```

Query Parameters

- **message_ids** – list of message_ids of exists messages
- **service** – `nova` or `glance`. default="nova"

CHAPTER 6

Indices and tables

- genindex
- search

HTTP Routing Table

/http:

GET http://example.com/db/stats/events/, 29
GET http://example.com/db/stats/nova/exists, 30
GET http://example.com/db/status/usage/glance/exists, 31
GET http://example.com/db/usage/deletes/, 37
GET http://example.com/db/usage/exists/, 38
GET http://example.com/db/usage/exists/<exist_id>, 41
GET http://example.com/db/usage/glance/deletes/, 36
GET http://example.com/db/usage/glance/deletes/<deleted_id>, 37
GET http://example.com/db/usage/glance/exists/, 40
GET http://example.com/db/usage/glance/exists/<exist_id>/, 42
GET http://example.com/db/usage/glance/images/, 33
GET http://example.com/db/usage/glance/images/<image_id>/, 34
GET http://example.com/db/usage/launches/, 32
GET http://example.com/db/usage/launches/<launch_id>/, 34
GET http://example.com/db/usage/nova/deletes/, 35
GET http://example.com/db/usage/nova/deletes/<deleted_id>, 37
GET http://example.com/db/usage/nova/exists/, 38
GET http://example.com/db/usage/nova/exists/<exist_id>, 41
GET http://example.com/db/usage/nova/launches/, 32
GET http://example.com/db/usage/nova/launches/<laun 34
GET http://example.com/stacky/deployments/, 9
GET http://example.com/stacky/events/, 10
GET http://example.com/stacky/hosts/, 10
GET http://example.com/stacky/report/<report_id>, 15
GET http://example.com/stacky/reports/, 14
GET http://example.com/stacky/reports/search, 17
GET http://example.com/stacky/request/, 13
GET http://example.com/stacky/search/, 20
GET http://example.com/stacky/show/<event_id>/, 18
GET http://example.com/stacky/summary/, 12
GET http://example.com/stacky/timings/uuid/, 12
GET http://example.com/stacky/uuid/, 11
GET http://example.com/stacky/watch/<deployment_id> 19
POST http://example.com/db/repair/, 43
PUT http://example.com/db/confirm/usage/exists/bat 28