EOSC Performance API

Release 0.0.1

Karlsruhe Institute of Technology

Oct 26, 2022

GETTING STARTED

Compare and analyze benchmark results from diverse cluster providers at the European Open Science Cloud (EOSC) using our Representational State Transfer (REST) Application Program Interface (API).

CHAPTER

ONE

GETTING STARTED WITH EOSC PERFORMANCE API

Learn more about how to use our endpoints to get valuable benchmark results from multiple providers or how to submit yours.

- **First steps**: *Getting started* | *First example*
- **Overview of core features**: *benchmarks* | *reports* | *results* | *sites* | *tags*

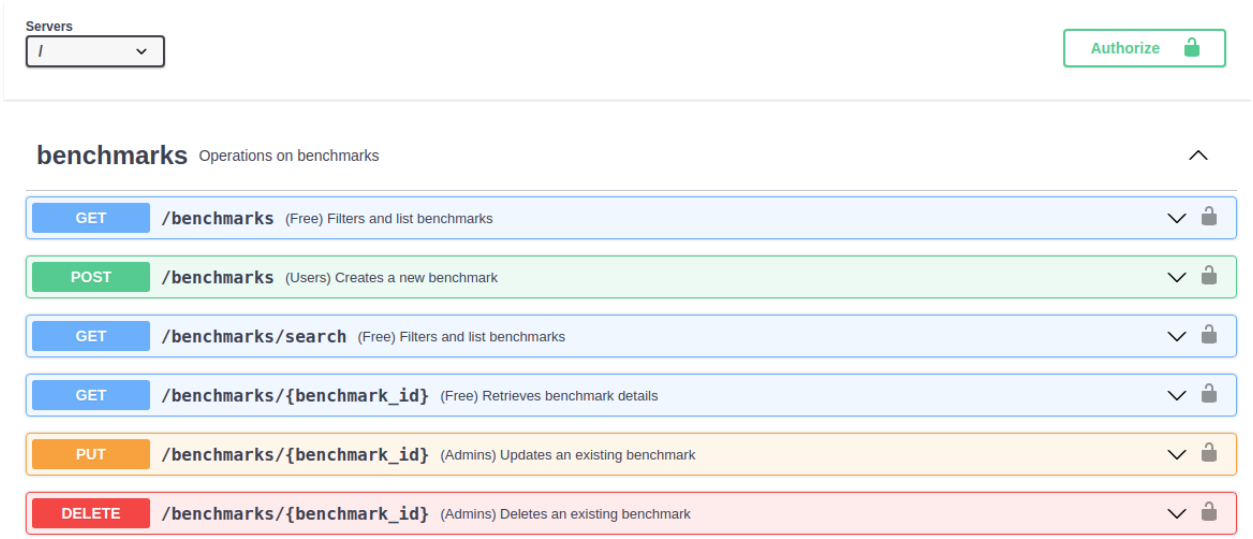
1.1 Introduction

EOSC Performance aims to bring a platform where users can compare multiple cloud providers to decide the best place and configuration to run their applications.

In order to achieve such objective, EOSC Performance API provides the backend with the logic to store and filter all the benchmark results provided by the community.

1.1.1 OpenAPI specification

In order to improve usability, the API provides an **OpenAPI v3** specification which can be accessed at the path `/api-spec.json`. Additionally the root path provides a Graphical User Interface (GUI) build with Swagger UI that users can use to test, interface and learn about the multiple methods offered by this API.



More information about the OpenAPI community can be found at their home page openapis.org.

1.1.2 Project source

EOSC Performance is an open source project. You can review, fork the code source, reproduce the application, open and review issues accessing our code repository. Any support and comments form the community on how to improve our application and code quality are welcome.

1.1.3 OIDC Tokens

Although most of the GET reports are open to any request from any user, some are restricted to identified users and administrators. In order to identify into the service, the technology used is **OpenID Connect**.

More information about OpenID Connect can be found at their home page openid.net.

In order to obtain an OIDC token you can use to identify yourself when using the API, we recommend the tool oidc-agent. Once configured, you can easily store your access token as environment variable, for example:

```
$ OIDC_TOKEN=oidc-token <your-account>
```

1.1.4 Registration process

Although most of the GET reports are open without registration required, those users that want to provide benchmark results from their platforms need to confirm that they have read and accepted our terms and conditions:

- Privacy policy

Once the terms are understood and you decide to continue with the registration process, you can register using the method `POST /users/self` with an OIDC token as authorization bearer. For example:

```
$ curl -X 'POST' \
    'http://localhost:5000/users/self' \
    -H "accept: application/json" \
    -H "Authorization: Bearer $OIDC_TOKEN"
```

The response will return your stored information:

```
$ curl -X 'POST' \
    'http://localhost:5000/users/self' \
    -H "accept: application/json" \
    -H "Authorization: Bearer $OIDC_TOKEN"
```

1.2 First example

In this first example, we are going to register ourselves into the application and collect some simple benchmarks and results.

1.2.1 Prepare your environment

Before starting, there are some pre requisites you need to full fill.

Read or terms and conditions

If you plan to upload data into EOSC Performance the first thing you would have to do is register. But before doing so, please read carefully our terms of service at performance.services.fedcloud.eu/terms-of-service.

Install and run oidc-agent

When you need to authenticate yourself at the API, for example to request POST methods, you need to attach an `access-token` to the http header. To get that header, we recommend the usage of oidc-agent.

Get your `access-token` in bash with the following command:

[1]:
```
access_token=$(oidc-token egi-prod)
```

Configure your environment

Not too complicate, for this example it is enough to select an API endpoint.

[2]:
```
eosc_perf_api="https://performance.services.fedcloud.eu/api/v1"
```

Register

Perform a POST request to `/users:register`. > You do not need to register for accessing benchmarks and results.

[ ]:
```
curl -X 'POST' \
  "$eosc_perf_api/users:register" \
  -H "Authorization: Bearer $access_token"
```

1.2.2 Search for a benchmark

Use the endpoint `GET /benchmarks` to download a list of all available benchmarks at EOSC Performance.

[4]:
```
benchmarks=$(curl -X 'GET' "$eosc_perf_api/benchmarks")
benchmarks_ids=$(echo $benchmarks | jq '.items[].id')
echo $benchmarks_ids | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2541  100  2541    0     0  15782      0 --:--:-- --:--:-- --:--:-- 15881
"5a0f5641-01ae-4f48-b5d1-84cd9c87dfac"
"8edf5bcf-d432-4f4f-930f-6245d7f49588"
```

A benchmark is composed by the following fields: - **docker_image**: Docker image referenced by the benchmark - **docker_tag**: Docker image version/tag referenced by the benchmark - **json_schema**: Schema used to validate benchmark results before upload

You should find the benchmark container image at Docker Hub using the `docker_image` and `docker_tag` fields.

[5]:
```
docker_image=$(echo $benchmarks | jq '.items[0].docker_image')
docker_tag=$(echo $benchmarks | jq '.items[0].docker_tag')
echo "$docker_image:$docker_tag"
```

```
"deephdc/deep-oc-benchmarks_cnn":"benchmark"
```

In addition, you can explore what can you expect from the schema accessing the benchmark schema.

[ ]:
```
echo $benchmarks | jq '.items[0].json_schema'
```

1.2.3 Collect all benchmark results

Now you have chosen a benchmark, use its `id` to perform a GET request to `/results` and retrieve the uploaded data from the community.

[7]:
```
benchmark_id='5a0f5641-01ae-4f48-b5d1-84cd9c87dfac'
results=$(curl -X 'GET' "$eosc_perf_api/results?benchmark_id=$benchmark_id")
results_ids=$(echo $results | jq '.items[].id')
echo $results_ids | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 43513  100 43513    0     0   148k      0 --:--:-- --:--:-- --:--:--  148k
"69516c84-c610-4656-ac7b-d41849fb6cbc"
"124195c0-e5c8-459e-aca2-19a9665a4845"
"5736c4f3-ea86-477a-8ae9-0e0ae05f62a3"
"3212d315-72eb-4e63-8a91-25ece0718b18"
"c99a91a1-2fb0-4d42-88a1-ddc12e7bdaf8"
"e79a99b4-46f1-4ace-8d27-bc5f254b7a42"
"8a7a15fd-b759-4c09-9117-3dc328b915c2"
"889286ad-9e8d-47d2-a3f0-c07de330c81f"
"553cc539-f32e-4db5-9a41-89d759dcf5ae"
"8fe67271-b839-4dd4-807d-1c0a2eb04382"
"51b861a3-d1fe-403e-b2db-fa8467a40c0b"
"6926ab40-9d1a-462c-979d-21f1dd5df061"
```

1.3 Using /benchmarks

In this example we are going to explore deeply the options available when collecting benchmarks from EOSC Performance. This example was created using Jupyter notebook, click here to the original notebook file.

1.3.1 Create the environment

To do so, we select an API endpoint and collect a token from our configuration. We also need an access token, in this example we use oidc-agent to get one.

[1]:
```
eosc_perf_api="https://performance.services.fedcloud.eu/api/v1/"
access_token=$(oidc-token egi-prod)
```

1.3.2 (Conditional) Register, if not done already

To use our service as user, first we need to accept the terms of usage and register. Make sure to read the terms and conditions.

[ ]:
```
curl -X 'POST' \
  "$eosc_perf_api/users:register" \
  -H "Authorization: Bearer $access_token"
```

1.3.3 Push a benchmark docker image in a public container repository

All benchmark must rely on a public and accessible container image in a container repository.

> You can find a tutorial on how to push a docker container image to Docker Hub here.
>
> From version **1.2.0** benchmarks accept images also outside from docker-hub.

After upload your docker image, you will need the docker_image and docker_tag identifications for the later POST method.

[3]:
```
image="deephdc/deep-oc-benchmarks_cnn"
tag="gpu"
```

1.3.4 Design a JSON Schema to accept or discard results from users

Results must be linked to a benchmark when submitted. You can control the required fields and their data types to ensure users do not upload invalid results. This functionality will simplify users to compare attributes between results as such fields will always be present and will share the same type.

If you do not want to use JSON Schemas, you can use `{}` for an always valid result.

---

[4]:
```
schema='{
        "$id": "https://example.com/benchmark.schema.json",
        "$schema": "https://json-schema.org/draft/2019-09/schema",
        "type": "object",
```

(continues on next page)

(continued from previous page)

```
        "properties": {
          "start_datetime": {
            "description": "The benchmark start datetime.",
            "type": "string",
            "format": "date-time"
          },
          "end_datetime": {
            "description": "The benchmark end datetime.",
            "type": "string",
            "format": "date-time"
          },
          "machine": {
            "description": "Execution machine details.",
            "type": "object",
            "properties": {
              "cpus": {
                "description": "Number of CPU.",
                "type": "integer"
              },
              "ram": {
                "description": "Available RAM in MB.",
                "type": "integer"
              }
            },
            "required": [
              "cpus",
              "ram"
            ]
          }
        },
        "required": [
          "start_datetime",
          "end_datetime",
          "machine"
        ]}'
```

You can learn more about JSON Schemas at json-schema.org.

1.3.5 Upload your benchmark

To upload the benchmark, you only need to use an authenticated POST request to `/benchmarks` and attach the following content to the body:

- **docker_image**: Name of the image in docker hub.
- **docker_tag**: Tag of the docker image you want this benchmark to reference.
- **json_schema**: Defined JSON Schema to accept community results.
- **description(Optional)**: Short description about the benchmark for the community users.

[5]:
```
curl -X 'POST' "$eosc_perf_api/benchmarks" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -H 'Content-Type: application/json' \
  -d "{\"docker_image\": \"$image\", \"docker_tag\": \"$tag\", \"json_schema\": $schema,␣
↪\
      \"description\": \"A free description for the community\"}" | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2128  100   853  100  1275    268    400  0:00:03  0:00:03 --:--:--   668
{
  "description": "A free description for the community",
  "docker_image": "deephdc/deep-oc-benchmarks_cnn",
  "docker_tag": "gpu",
  "id": "7722c571-234f-4db3-b4f6-7b8fb869cab0",
  "json_schema": {
    "$id": "https://example.com/benchmark.schema.json",
    "$schema": "https://json-schema.org/draft/2019-09/schema",
    "properties": {
      "end_datetime": {
        "description": "The benchmark end datetime.",
        "format": "date-time",
        "type": "string"
      },
      "machine": {
        "description": "Execution machine details.",
        "properties": {
          "cpus": {
            "description": "Number of CPU.",
            "type": "integer"
          },
          "ram": {
            "description": "Available RAM in MB.",
            "type": "integer"
          }
        },
        "required": [
          "cpus",
          "ram"
        ],
        "type": "object"
      },
      "start_datetime": {
        "description": "The benchmark start datetime.",
        "format": "date-time",
        "type": "string"
      }
    },
    "required": [
      "start_datetime",
      "end_datetime",
      "machine"
    ],
    "type": "object"
  },
  "upload_datetime": "2022-02-22T11:25:25.688828"
}
```

(continues on next page)

(continued from previous page)

1.3.6 Download your benchmark

You can check your benchmark is available by downloading the benchmark.

Note the upload of benchmarks needs to be validated by administrators. Therefore it will only be available to the community after the review is completed.

---

[6]:
```
benchmarks=$(curl -X 'GET' "$eosc_perf_api/benchmarks?docker_image=$image&docker_tag=$tag
↪")
echo $benchmarks | jq '.items[0].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   943  100   943    0     0   131k      0 --:--:-- --:--:-- --:--:--  131k
"7722c571-234f-4db3-b4f6-7b8fb869cab0"
```

1.4 Using /reports

Although EOSC Performance administrators review benchmarks and site uploads, it is not feasible nor recommended to review all uploaded results as it might impact the usage experience. Therefore, it relies in the hands of the community the responsibility to report any result that might create confusion or might seem not correct. In this example we are going to explore deeply the options available when reporting results at EOSC Performance. This example was created using Jupyter notebook, click here to the original notebook file.

1.4.1 Create the environment

To do so, we select an API endpoint and collect a token from our configuration. We also need an access token, in this example we use oidc-agent to get one.

[1]:
```
eosc_perf_api="https://performance.services.fedcloud.eu/api/v1/"
access_token=$(oidc-token egi-prod)
```

1.4.2 (Conditional) Register, if not done already

To use our service as user, first we need to accept the terms of usage and register. Make sure to read the terms and conditions.

[ ]:
```
curl -X 'POST' \
  "$eosc_perf_api/users:register" \
  -H "Authorization: Bearer $access_token"
```

1.4.3 Evaluate result details

You can access to the result details using the `json` field on a `/results` response. For more information about how to filter and search for results, see the example on *how to use /results*.

[3]:
```
results=$(curl -X 'GET' "$eosc_perf_api/results")
echo $results | jq '.items[0].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 77177  100 77177    0     0   746k      0 --:--:-- --:--:-- --:--:--  746k
"7a3fa35c-2ab8-40fe-837c-e8c81fbcc657"
```

1.4.4 Report the result

You can create a result `claim` with a POST request to the `/results/{id}:claim` endpoint. To do so, you need the `id` of the result to report and attach as body a json with **message** field where explaining the reasons for the claim.

[4]:
```
result_id='7a3fa35c-2ab8-40fe-837c-e8c81fbcc657'
curl -X 'POST' "$eosc_perf_api/results/$result_id:claim" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -H 'Content-Type: application/json' \
  -d '{"message": "This is a claim example"}' | jq  '.uploader = "..."'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   456  100   418  100    38    533     48 --:--:-- --:--:-- --:--:--   582
{
  "id": "d5ca59a4-d7e6-45ad-9887-9a60fbf094ea",
  "message": "This is a claim example",
  "resource_id": "7a3fa35c-2ab8-40fe-837c-e8c81fbcc657",
  "resource_type": "result",
  "upload_datetime": "2022-02-23T11:14:05.722832",
  "uploader": "..."
}
```

Once the claim is submitted the result will remain hidden for the community when using the default searches and list methods. The result will remain hidden until the administrators review the result and the claim. If the result is legitimate, it will be restored otherwise it will be continue hidden from the community.

Note that results cannot be edited. If claimed results is hidden due to simple mistakes the best way to fix them is to create a new result.

1.5 Using /results

In this example we are going to explore deeply the options available when collecting and uploading benchmark results at EOSC Performance. This example was created using Jupyter notebook, click here to the original notebook file.

1.5.1 Create the environment

To do so, we select an API endpoint and collect a token from our configuration. We also need an access token, in this example we use oidc-agent to get one.

[1]:
```
eosc_perf_api="https://performance.services.fedcloud.eu/api/v1"
access_token=$(oidc-token egi-prod)
```

1.5.2 Search for results

Configure and limit your search using the multiple arguments and terms available.

Search for the benchmark id that produced our result

You can get a list of all available benchmarks using GET `/benchmarks`.

[2]:
```
benchmarks=$(curl -X 'GET' "$eosc_perf_api/benchmarks?docker_image=thechristophe/
↪openbench-c-ray")
benchmark=$(echo $benchmarks | jq '.items[0]')
echo $benchmark | jq '.json_schema = "..."'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1441  100  1441    0     0  11620      0 --:--:-- --:--:-- --:--:-- 11528
{
  "description": "Compare cpu perf with multithreaded raytracing",
  "docker_image": "thechristophe/openbench-c-ray",
  "docker_tag": "latest",
  "id": "1cc7814e-131f-4002-803a-434a287cf135",
  "json_schema": "...",
  "upload_datetime": "2022-02-01T07:58:17.555822"
}
```

[3]:
```
benchmark_id=$(echo $benchmark | jq -r '.id')
curl -X 'GET' "$eosc_perf_api/results?benchmark_id=$benchmark_id" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 74245  100 74245    0     0   895k      0 --:--:-- --:--:-- --:--:--  895k
"07decfe3-8f03-4dcf-b8ee-b47090322e0c"
"c2517c02-e195-4bad-8d8d-707ffa8cff9b"
"1196b021-7923-44bf-a9f1-d7abd683fe5a"
"c45d2d84-2bd7-4fb7-83a7-b9a25d2d034a"
"2a04564b-adad-40f0-9c86-062546bf15dd"
"f5cdf98c-28db-4c8e-97f8-459ae1e94ee6"
```

(continues on next page)

(continued from previous page)

```
"7ddecf46-75c6-46b1-abc7-f566b38de10c"
"dd74701b-052d-4302-9d76-96ddc0a11f0c"
"f97528bd-f350-427a-8cf0-44dbb2eaf487"
"138f4780-1cd5-4d4d-bb36-2b4c86f61147"
"a6dde02e-fbf4-4342-ba29-729802aa7b36"
"ddad3d39-eae3-43d4-a4b5-36fd9e427e3c"
"f8f9beee-705d-4096-870e-6b037ebab86b"
"eb2ba92d-479b-420d-bdda-4ac0fda780b4"
"244d5872-cfc4-4359-850e-a74314edbe65"
"615539ce-c123-4b2c-9de9-aa68dde88055"
"768b83de-58a4-426b-ac6a-3d7b1ee1db51"
"5c6a7dc1-14be-4b18-af26-cb7089f0607c"
"96e95029-c24b-468b-b659-4a0ef6886be2"
"0a1d1595-f4d4-4364-99a3-7e1cb2e642e3"
"105357bb-8054-4174-9bda-b0020a824ef9"
"7f66e8d3-2218-47ad-b697-726343bffb27"
"fb0ee217-5f19-4d24-bb86-a1baaf6262d0"
```

Search for the site id used to run our benchmark

You can get a list of all available benchmarks using GET `/sites`.

[4]:
```
sites=$(curl -X 'GET' "$eosc_perf_api/sites?name=CESNET-MCC")
site=$(echo $sites | jq '.items[0]')
echo $site | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   223  100   223    0     0  22300      0 --:--:-- --:--:-- --:--:-- 22300
{
  "address": "unknown",
  "id": "17fb17c9-107c-4571-ab1e-120299878342",
  "name": "CESNET-MCC",
  "upload_datetime": "2022-02-01T07:57:42.821704"
}
```

[5]:
```
site_id=$(echo $site | jq -r '.id')
curl -X 'GET' "$eosc_perf_api/results?site_id=$site_id" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 13839  100 13839    0     0   409k      0 --:--:-- --:--:-- --:--:--  409k
"2a04564b-adad-40f0-9c86-062546bf15dd"
"f5cdf98c-28db-4c8e-97f8-459ae1e94ee6"
"7ddecf46-75c6-46b1-abc7-f566b38de10c"
"dd74701b-052d-4302-9d76-96ddc0a11f0c"
```

Search for the flavor id used to run our benchmark

If needed you can also get a list of all available flavors in that site using GET `/sites{id}/flavors`.

[6]:
```
site_id=$(echo $site | jq -r '.id')
flavors=$(curl -X 'GET' "$eosc_perf_api/sites/$site_id/flavors:search?terms=hpc.8core-
↪16ram")
flavor=$(echo $flavors | jq '.items[0]')
echo $flavor | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   225  100   225    0     0  15000      0 --:--:-- --:--:-- --:--:-- 15000
{
  "description": "",
  "id": "01210775-cc77-472d-a061-c98760dbb883",
  "name": "hpc.8core-16ram",
  "upload_datetime": "2022-02-01T07:57:48.652778"
}
```

[7]:
```
flavor_id=$(echo $flavor | jq -r '.id')
curl -X 'GET' "$eosc_perf_api/results?flavor_id=$flavor_id" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3684  100  3684    0     0   179k      0 --:--:-- --:--:-- --:--:--  179k
"dd74701b-052d-4302-9d76-96ddc0a11f0c"
```

Search for the tags to relate your result

Collect the tags you want to link to your result so users can find it easily.

[8]:
```
tag_gpu=$(curl -X 'GET' "$eosc_perf_api/tags?name=gpu" | jq '.items[0]')
echo $tag_gpu | jq
tag_hpc=$(curl -X 'GET' "$eosc_perf_api/tags?name=hpc" | jq '.items[0]')
echo $tag_hpc | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   191  100   191    0     0  27285      0 --:--:-- --:--:-- --:--:-- 27285
{
  "description": "Result executed using gpu",
  "id": "08156712-4607-469d-903a-d3033b44ab9d",
  "name": "gpu"
}
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   195  100   195    0     0  16250      0 --:--:-- --:--:-- --:--:-- 16250
{
  "description": "Result executed in hpc system",
  "id": "92f7443d-1991-4bee-8b58-a4189c586a08",
  "name": "hpc"
}
```

If you do not know the name of the tag you can use `:search` as generic filter.

[9]:
```
tag_1=$(echo $tag_hpc | jq -r '.id')
curl -X 'GET' "$eosc_perf_api/results?tags_ids=$tag_1" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3684  100  3684    0     0   133k      0 --:--:-- --:--:-- --:--:--  133k
"dd74701b-052d-4302-9d76-96ddc0a11f0c"
```

Search for results between dates

If is also possible to filter results by upload and execution date.

[10]:
```
upload_before='2023-01-01'
execution_after='2000-01-01'

curl -X 'GET' "$eosc_perf_api/results?upload_before=$upload_before&execution_after=
↪$execution_after" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 74245  100 74245    0     0  1050k      0 --:--:-- --:--:-- --:--:-- 1050k
"07decfe3-8f03-4dcf-b8ee-b47090322e0c"
"c2517c02-e195-4bad-8d8d-707ffa8cff9b"
"1196b021-7923-44bf-a9f1-d7abd683fe5a"
"c45d2d84-2bd7-4fb7-83a7-b9a25d2d034a"
"2a04564b-adad-40f0-9c86-062546bf15dd"
"f5cdf98c-28db-4c8e-97f8-459ae1e94ee6"
"7ddecf46-75c6-46b1-abc7-f566b38de10c"
"dd74701b-052d-4302-9d76-96ddc0a11f0c"
"f97528bd-f350-427a-8cf0-44dbb2eaf487"
"138f4780-1cd5-4d4d-bb36-2b4c86f61147"
"a6dde02e-fbf4-4342-ba29-729802aa7b36"
"ddad3d39-eae3-43d4-a4b5-36fd9e427e3c"
"f8f9beee-705d-4096-870e-6b037ebab86b"
"eb2ba92d-479b-420d-bdda-4ac0fda780b4"
"244d5872-cfc4-4359-850e-a74314edbe65"
"615539ce-c123-4b2c-9de9-aa68dde88055"
"768b83de-58a4-426b-ac6a-3d7b1ee1db51"
"5c6a7dc1-14be-4b18-af26-cb7089f0607c"
"96e95029-c24b-468b-b659-4a0ef6886be2"
"0a1d1595-f4d4-4364-99a3-7e1cb2e642e3"
"105357bb-8054-4174-9bda-b0020a824ef9"
"7f66e8d3-2218-47ad-b697-726343bffb27"
"fb0ee217-5f19-4d24-bb86-a1baaf6262d0"
```

Search using custom filters

If is also possible to filter results specific values inside the result (if you know the result structure).

[11]:
```
benchmark_id='1cc7814e-131f-4002-803a-434a287cf135'
filter_1='machine.cpu.count%20%3E%204'
filter_2='machine.cpu.count%20%3C%2020'

curl -X 'GET' "$eosc_perf_api/results?benchmark_id=$benchmark_id&filters=$filter_1&
↪filters=$filter_2" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 25917  100 25917    0     0   377k      0 --:--:-- --:--:-- --:--:--  383k
"c2517c02-e195-4bad-8d8d-707ffa8cff9b"
"1196b021-7923-44bf-a9f1-d7abd683fe5a"
"dd74701b-052d-4302-9d76-96ddc0a11f0c"
"ddad3d39-eae3-43d4-a4b5-36fd9e427e3c"
"f8f9beee-705d-4096-870e-6b037ebab86b"
"768b83de-58a4-426b-ac6a-3d7b1ee1db51"
"105357bb-8054-4174-9bda-b0020a824ef9"
"7f66e8d3-2218-47ad-b697-726343bffb27"
```

If you do not want results that does not include the field in the response, we recommend that you use a benchmark id.

Configure sorting and use pagination

For all the previous options, it is possible to sort the results using the following fields:

- **id**: Result id.
- **upload_datetime**: EOSC Performance upload datetime of the result.
- **json**: Result json values.
- **execution_datetime**: Execution date of the benchmark.
- **benchmark_id**: Benchmark id used to obtain the result.
- **flavor_id**: Favor id used to run the benchmark.
- **site_id**: Site id where the benchmark was executed.

In addition, sometimes you might get more results that expected. In such case you will have to use pagination to collect all the items. To do so you can use the following parameters:

- **per_page**: The number of items to be displayed on a page (maximum 100)
- **page**: The return page number (1 indexed)

[12]:
```
curl -X 'GET' "$eosc_perf_api/results?per_page=4&page=2&sort_by=%2Bid" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 13086  100 13086    0     0   399k      0 --:--:-- --:--:-- --:--:--  399k
```

(continues on next page)

(continued from previous page)

```
"138f4780-1cd5-4d4d-bb36-2b4c86f61147"
"244d5872-cfc4-4359-850e-a74314edbe65"
"2a04564b-adad-40f0-9c86-062546bf15dd"
"5c6a7dc1-14be-4b18-af26-cb7089f0607c"
```

1.5.3 Upload results

(Conditional) Register, if not done already

To use our service as user, first we need to accept the terms of usage and register. Make sure to read the terms and conditions.

[ ]:
```
curl -X 'POST' \
  "$eosc_perf_api/users:register" \
  -H "Authorization: Bearer $access_token"
```

Search for the benchmark id that produced our result

You can get a list of all available benchmarks using GET `/benchmarks`.

[14]:
```
benchmarks=$(curl -X 'GET' "$eosc_perf_api/benchmarks?docker_image=thechristophe/
↪openbench-c-ray")
benchmark=$(echo $benchmarks | jq '.items[0]')
echo $benchmark | jq '.json_schema = "..."'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1441  100  1441    0     0   234k      0 --:--:-- --:--:-- --:--:--  234k
{
  "description": "Compare cpu perf with multithreaded raytracing",
  "docker_image": "thechristophe/openbench-c-ray",
  "docker_tag": "latest",
  "id": "1cc7814e-131f-4002-803a-434a287cf135",
  "json_schema": "...",
  "upload_datetime": "2022-02-01T07:58:17.555822"
}
```

Benchmarks are public to the Internet, access token is not needed.

Search for the flavor id used to run our benchmark

First you need to find the site where the benchmark was run. Once the site id is collected, it is possible to access and select the site flavors.

[15]:
```
sites=$(curl -X 'GET' "$eosc_perf_api/sites?name=CESNET-MCC")
site=$(echo $sites | jq '.items[0]')
echo $site | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   223  100   223    0     0  31857      0 --:--:-- --:--:-- --:--:-- 31857
{
  "address": "unknown",
  "id": "17fb17c9-107c-4571-ab1e-120299878342",
  "name": "CESNET-MCC",
  "upload_datetime": "2022-02-01T07:57:42.821704"
}
```

In this example we will use `:search` endpoint to find the flavor.

[16]:
```
site_id=$(echo $site | jq -r '.id')
flavors=$(curl -X 'GET' "$eosc_perf_api/sites/$site_id/flavors:search?terms=hpc.8core-
↪16ram")
flavor=$(echo $flavors | jq '.items[0]')
echo $flavor | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   225  100   225    0     0  25000      0 --:--:-- --:--:-- --:--:-- 25000
{
  "description": "",
  "id": "01210775-cc77-472d-a061-c98760dbb883",
  "name": "hpc.8core-16ram",
  "upload_datetime": "2022-02-01T07:57:48.652778"
}
```

Search for the tags to relate your result

Collect the tags you want to link to your result so users can find it easily.

If you do not know the name of the tag you can use `:search` as generic filter.

[17]:
```
tag_hpc=$(curl -X 'GET' "$eosc_perf_api/tags?name=hpc" | jq '.items[0]')
echo $tag_hpc | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   195  100   195    0     0   8863      0 --:--:-- --:--:-- --:--:--  8863
{
  "description": "Result executed in hpc system",
  "id": "92f7443d-1991-4bee-8b58-a4189c586a08",
  "name": "hpc"
}
```

(continues on next page)

(continued from previous page)

Upload your result

Use an execution datetime and the collected *site_id* and *flavor_id* to upload a result.

[18]:
```
execution_datetime="2022-01-01T10:00:00.00000Z"
benchmark_id=$(echo $benchmark | jq -r '.id')
flavor_id=$(echo $flavor | jq -r '.id')
tag1_id=$(echo $tag_hpc | jq -r '.id')
result_json='{"arguments": "Total Time - 4K, 16 Rays Per Pixel", "machine": {"cpu": {
↪"count": 4}}, "result": {"all_results": "155.995:157.626:156.195", "score": 156.605},
↪"test": "pts/c-ray-1.2.0", "units": "Seconds"}'
```

[19]:
```
query="execution_datetime=$execution_datetime&benchmark_id=$benchmark_id&flavor_id=
↪$flavor_id&tags_ids=$tag1_id"
curl -X 'POST' "$eosc_perf_api/results?$query" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -d 'Content-Type: application/json' \
  -d "$result_json" | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2299  100  2095  100   204   1847    179  0:00:01  0:00:01 --:--:--  2027
{
  "benchmark": {
    "description": "Compare cpu perf with multithreaded raytracing",
    "docker_image": "thechristophe/openbench-c-ray",
    "docker_tag": "latest",
    "id": "1cc7814e-131f-4002-803a-434a287cf135",
    "json_schema": {
      "$schema": "https://json-schema.org/draft/2020-12/schema",
      "properties": {
        "arguments": {
          "const": "Total Time - 4K, 16 Rays Per Pixel"
        },
        "machine": {
          "properties": {
            "cpu": {
              "properties": {
                "arch": {
                  "description": "Processor architecture",
                  "suggestToUser": true,
                  "type": "string"
                },
                "bits": {
                  "description": "Processor address size",
                  "type": "number"
                },
                "brand_raw": {
                  "description": "Human-readable processor branding",
                  "type": "string"
                },
                "count": {
                  "description": "Processor core count",
                  "suggestToUser": true,
                  "type": "number"
                },
```

(continues on next page)

(continued from previous page)

1.6 Using /sites

In this example we are going to explore deeply the options available when collecting and uploading sites from EOSC Performance. This example was created using Jupyter notebook, click here to the original notebook file.

1.6.1 Create the environment

To do so, we select an API endpoint and collect a token from our configuration. We also need an access token, in this example we use oidc-agent to get one.

[1]:
```
eosc_perf_api="https://performance.services.fedcloud.eu/api/v1/"
access_token=$(oidc-token egi-prod)
```

1.6.2 (Conditional) Register, if not done already

To use our service as user, first we need to accept the terms of usage and register. Make sure to read the terms and conditions.

[ ]:
```
curl -X 'POST' \
  "$eosc_perf_api/users:register" \
  -H "Authorization: Bearer $access_token"
```

1.6.3 Upload your site

To upload the site, you only need to use an authenticated POST request to `/sites` and attach the following content to the body:

- **name**: Name of the site to display to the community.
- **address**: Address of the site to display to the community (it might be virtual or physical).
- **description(Optional)**: Short description about the site for the community.

[3]:
```
name="my_site"
address="my_address.com"
curl -X 'POST' "$eosc_perf_api/sites" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -H 'Content-Type: application/json' \
  -d "{\"name\": \"$name\", \"address\": \"$address\", \
       \"description\": \"A free description for the community\"}" | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   300  100   190  100   110    230    133 --:--:-- --:--:-- --:--:--   364
{
  "address": "my_address.com",
  "description": "A free description for the community",
  "id": "d9a42645-5cd4-403b-866c-793cc00e1665",
  "name": "my_site",
  "upload_datetime": "2022-02-24T13:32:25.830999"
}
```

(continues on next page)

(continued from previous page)

1.6.4 Download your site

You can check your site is available by downloading the site.

Note the upload of sites needs to be validated by administrators. Therefore it will only be available to the community after the review is completed.

---

[4]:
```
sites=$(curl -X 'GET' "$eosc_perf_api/sites?name=$name")
echo $sites | jq '.items[0].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   280  100   280    0     0  12727      0 --:--:-- --:--:-- --:--:-- 12727
"d9a42645-5cd4-403b-866c-793cc00e1665"
```

1.6.5 Upload your site flavors

Additionally to the site main attributes, you can also upload deployment flavors to a specific site. To upload a flavor, you only need to use an authenticated POST request to `/sites/{id}/flavors` and attach the following content to the body:

- **name**: Name of the flavor to display to the community.
- **description(Optional)**: Short description about the flavor for the community.

[5]:
```
site_id=$(echo $sites | jq -r '.items[0].id')
name="flavor_1"
curl -X 'POST' "$eosc_perf_api/sites/$site_id/flavors" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -H 'Content-Type: application/json' \
  -d "{\"name\": \"$name\", \
       \"description\": \"A free description for the community\"}" | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   246  100   164  100    82    206    103 --:--:-- --:--:-- --:--:--   309
{
  "description": "A free description for the community",
  "id": "efbd2510-8b45-4113-b691-9a6dae7dd5cf",
  "name": "flavor_1",
  "upload_datetime": "2022-02-24T13:34:11.627946"
}
```

1.6.6 Download your site flavor

You can check your site flavor is available by downloading the site flavors.

Note the upload of flavors needs to be validated by administrators. Therefore it will only be available to the community after the review is completed.

---

[6]:
```
flavors=$(curl -X 'GET' "$eosc_perf_api/sites/$site_id/flavors?name=$name")
echo $flavors | jq '.items[0].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   254  100   254    0     0   4618      0 --:--:-- --:--:-- --:--:--  4618
"efbd2510-8b45-4113-b691-9a6dae7dd5cf"
```

1.7 Using /tags

In this example we are going to explore deeply the options available when collecting and uploading tags to EOSC Performance. This example was created using Jupyter notebook, click here to the original notebook file.

1.7.1 Create the environment

To do so, we select an API endpoint and collect a token from our configuration. We also need an access token, in this example we use oidc-agent to get one.

[1]:
```
eosc_perf_api="https://performance.services.fedcloud.eu/api/v1/"
access_token=$(oidc-token egi-prod)
```

1.7.2 (Conditional) Register, if not done already

To use our service as user, first we need to accept the terms of usage and register. Make sure to read the terms and conditions.

[ ]:
```
curl -X 'POST' \
  "$eosc_perf_api/users:register" \
  -H "Authorization: Bearer $access_token"
```

1.7.3 Upload your tag

To upload the tag, you only need to use an authenticated POST request to `/tags` and attach the following content to the body:

- **name**: Name of the tag to display to the community.
- **description(Optional)**: Short description about the tag for the community.

[3]:
```
name="my_tag"
description="A free description for the community"
curl -X 'POST' "$eosc_perf_api/tags" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -H 'Content-Type: application/json' \
  -d "{\"name\": \"$name\", \"description\": \"$description\"}" | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   188  100   115  100    73    148     94 --:--:-- --:--:-- --:--:--   243
{
  "description": "A free description for the community",
  "id": "6186498a-929b-4745-9ac7-fc1d8b75b398",
  "name": "my_tag"
}
```

1.7.4 Download your tag

You can check your tag is available by downloading the tag.

Note the upload of tags needs to be validated by administrators. Therefore it will only be available to the community after the review is completed.

---

[4]:
```
tags=$(curl -X 'GET' "$eosc_perf_api/tags?name=$name")
echo $tags | jq '.items[0].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   205  100   205    0     0  14642      0 --:--:-- --:--:-- --:--:-- 14642
"6186498a-929b-4745-9ac7-fc1d8b75b398"
```

1.7.5 Update results with your tags

You can use a PUT request at `/results/{id}/tags` to update your result tags. For example when a user has created an interesting tag that you think it might relate some of your results. You can use a GET request at `/users/self/results` to collect the list of results uploaded by your user. Note that you need to include your access token to use this request.

[5]:
```
results=$(curl -X 'GET' "$eosc_perf_api/users/self/results" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -H 'Content-Type: application/json')
echo $results | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2081  100  2081    0     0   2627      0 --:--:-- --:--:-- --:--:--  2624
"76c5773a-52d7-408b-961b-e6d3ad06c640"
```

[6]:
```
result_id=$(echo $results | jq -r '.items[0].id')
tag_id=$(echo $tags | jq -r '.items[0].id')
curl -X 'PUT' "$eosc_perf_api/results/$result_id/tags" \
  -H 'accept: application/json' \
  -H "Authorization: Bearer $access_token" \
  -H 'Content-Type: application/json' \
  -d "{\"tags_ids\": [\"$tag_id\"]}" | jq
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    54    0     0  100    54      0     32  0:00:01  0:00:01 --:--:--    32
```

You can only update tags on your uploaded results. However you can use all the community.

1.7.6 Download results with your tags

With this functionality, users can easily find relevant results by just attaching the tags they are interested in into the `/results` query.

[7]:
```
curl -X 'GET' "$eosc_perf_api/results?tags_ids=$tag_id" \
  -H 'accept: application/json' | jq '.items[].id'
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  2195  100  2195    0     0   178k      0 --:--:-- --:--:-- --:--:--  178k
"76c5773a-52d7-408b-961b-e6d3ad06c640"
```

Note: From version **1.2.0** benchmarks accept images out of docker-hub.

CHAPTER

TWO

ADVANCED FEATURES OF EOSC PERFORMANCE API

EOSC Performance API offers many advanced features and options. Learn more about these integrations and how you can get the most out of your application when using our endpoints.

- **Advanced search and result filtering**: *Using advanced search* | *Filter results by json values* |
- **Pagination and sorting of responses**: *How to use sorting and pagination*

2.1 Generic search

All main request endpoints have the standard API methods List which should include enough argument options to configure most of your required searches. However, in those cases where the search is not required to be so strict, some endpoints include the *custom method* `:search`.

Use this method to get a list of items based on a general search of terms. For example, calling this custom method with `terms=A` and `terms=B` would return all those endpoint items that contain a 'A' **and** 'B' on the main json fields (except *id*). For example in the case of `\sites:search`, it will return all the sites that contain an 'A' and a 'B' on the *name*, *address* or *description* fields.

As usual, the response returns a pagination object with the filtered items or the corresponding error code.

2.2 JSON Filters

Although the usage of all the standard and custom methods should be enough to collect most of the items for analysis, in the case of results, some of the required variables to compare or limit in the search might be inside the result data.

However, the flexibility provided to the `json` field to store results information impedes the design of specific requests, the amount of options available are just infinite.

As the usage of conventional queries is not suitable to limit the results of the search inside json fields, EOSC Performance proposes a customizable filter argument which can be configured to filter the results directly into the database following some rules.

Filters are composed by 3 arguments separated by spaces ('%20' on URL-encoding): `<path.separated.by.dots> <operator> <value>` and there are five filter operators:

- **Equals (==)**: Return results where path value is exact to the query value. For example *filters=cpu.count == 5*
- **Greater than (>)**: Return results where path value strictly greater than the query value. For example *filters=cpu.count > 5*
- **Less than (<)**: Return results where path value strictly lower than the query value. For example *filters=cpu.count < 5*

- **Greater or equal (>=)**: Return results where path value is equal or greater than the query value. For example *filters=cpu.count >= 5*
- **Less or equal (<=)**: Return results where path value is equal or lower than the query value. For example *filters=cpu.count <= 5*

Note: Note that in the provided examples the filter is not URL-encoded as most libraries do it automatically.

When designing your query using filters, remember that in general it is a good idea to apply in addition a `benchmark_id` argument to limit the search on results that share the same json structure. Take a look on the required fields inside the benchmark schema to ensure that your filter applies. Filtering fields that are not available on the result are ignored.

2.3 Sorting and Pagination

EOSC Performance database expects to include hundreds of results from multiple providers. Therefore, every query into the database has to be optimized to what it is really required by the user.

In addition, multiple list methods are public and available to the internet. Therefore, it is required for security reasons and performance that most responses from the API are paginated and limited.

2.3.1 Pagination response

API responses for list methods (called by *GET*) are limited by default to a maximum of **100** items per page. In order to provide the user with enough information about the omitted items (if any) on the search, a paginated body has always the following fields:

- **has_next**: True if a next page exists.
- **has_prev**: True if a previous page exists
- **items**: The items for the current page
- **next_num**: Number of the next page
- **prev_num**: Number of the previous page.
- **page**: The current page number (1 indexed)
- **pages**: The total number of pages
- **per_page**: The number of items to displayed on the page.
- **total**: The total number of items matching the query

To control the returned page, you can configure the following arguments on the request query:

- **per_page**: The number of items to be displayed on a page.
- **page**: The page index to return.

Default query argument values (when not included) are `per_page=100` and `page=1`. The maximum value for `per_page` is 100.

Note: Note that modifying the argument `per_page` might shift the items returned on previous pages therefore not matching with the expected `page` items. The number of global pages depends on the `per_page` assigned value.

2.3.2 Sorting response items

It is possible to sort the response items including sorting fields into the the `sort_by` argument. This argument is available on every list method for the main endpoints (`/benchmark`, `/results`, `/sites` and `/flavors`).

All sorting fields are composed by a **sorting operator** which defines the order of the sorting and a **field id** which indicates the field to use. In addition, it is possible to include more than one sorting requirement into the argument by separating them using commas `,` without spaces. Note the following available sorting operators and equivalent order:

- **+: Sorts the results in ascendant order. For example *sort=+name***

```
["item_1", "item_2", "item_3"]
```

- **-: Sorts the results in descendant order. For example *sort=-name***

```
["item_3", "item_2", "item_1"]
```

In the case of results, you might be interested into sorting by an specific result field inside the json attribute. This behavior is similar to the already explained into *JSON Filters*.

When sorting by a json field use a *path separated by dots* to indicate where to find the value you desire to use to sort. In addition, remember that in general it is a good idea to apply take a look into the benchmark schema related to the search you want to perform in order to ensure your filter applies. Sorting fields that are not available on the result are ignored.

CHAPTER

THREE

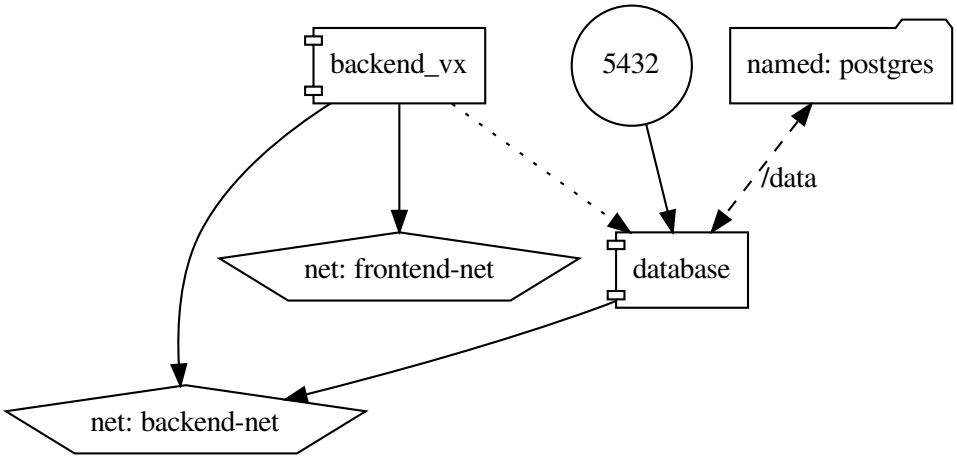
EXTENDING EOSC PERFORMANCE

EOSC Performance is an open source project which hopes to evolve and provide with the time better functionalities to the users. Our source code was build in order to be understandable and easy to maintain by the community. Any help is welcome, therefore we have prepare a section where developers can understand better the bases of the application.

- **Design principles**: *containers, networks and databases*
- **Application package documentation**: *Modules, Classes and Functions*
- **Database interface with Python**: *Instantiate multiple models with Factories*

3.1 Design principles

EOSC Performance API is designed to run as a container service so it can be deployed with container technologies such as Docker and Kubernetes.



You should be able to access the backend from the frontend network. in production it is recommended to run a reverse proxy container which provides HTTPS layer although for development it can be enough just to export the port 5000

where Flask normally runs.

3.1.1 Data storage

The main container includes all the dependencies to run the software, however, it is designed to store the data into an external postgresql database, normally deployed as a container. See the *configuration settings* to know more details about how to configure the application to connect to the external database.

When deploying the database as a container using a container orchestrator, it recommended to connect it to the "backend-net" network so all the backend related containers such as the backend itself or a backup service can access the required ports.

In case the database has to be managed outside the container network, it is generally a good idea to export the port where postgresql listens the incoming connections. By default it is the 5432, although for security reasons is always recommended to use a different port and always keep the container running with the last security updates.

3.1.2 Production vs development

You can control the environment as production or development configuring the environmental variable *FLASK_ENV*. This variable can take the following values and behaviors:

- *production*: Backend service runs enforcing security requirements and performance processing. You should always run the application as production except for specific short executions for development. Testing should run as well in production to ensure the correct behavior of components.
- *development*: Backend service runs with minimum security requirements and with additional loads for debugging. When running in development the number open queries might be limited to the specific used extension. Therefore it is not suitable for long running operations. In addition, some variables required for production might be optional, see *configuration settings* for more details about defaults when running on development mode.

3.1.3 Settings

You can get a full list of configurable environment variables from the *configuration settings* page.

3.2 EOSC Performance API

This is the main web application package for the EOSC Performance Application Program Interface (API).

The application is developed using python and flask as main engines. By default, Flask does not provide database or specific web abstraction layers. API and any other functionality such database or authentication are handled by independent libraries and extension.

`create_app`(*config_base='backend.settings'*, ***settings_override*)

Create application factory, as explained here: http://flask.pocoo.org/docs/patterns/appfactories

Parameters `config_base` (*str, optional*) – Configuration object, defaults to "backend.settings"

Returns EOSC Performance API instance

Return type `flask.app.Flask`

The EOSC Performance API is composed by the following main components:

3.2.1 Settings module

3.2.2 Extensions module

This module implements the loading of multiple flask extensions to extend the basic provided functionality to the API requirements. For more information about flask and extensions see:

https://flask.palletsprojects.com/en/2.0.x/extensions

Each extension requires of a specific class initialization which is lately initialized in the application factory using the settings and configurations from the environment.

flaat = <flaat.flask.Flaat object>
Flask extension that provides support for handling oidc Access Tokens

api = <flask_smorest.Api object>
Flask framework library for creating REST APIs (i.e. OpenAPI)

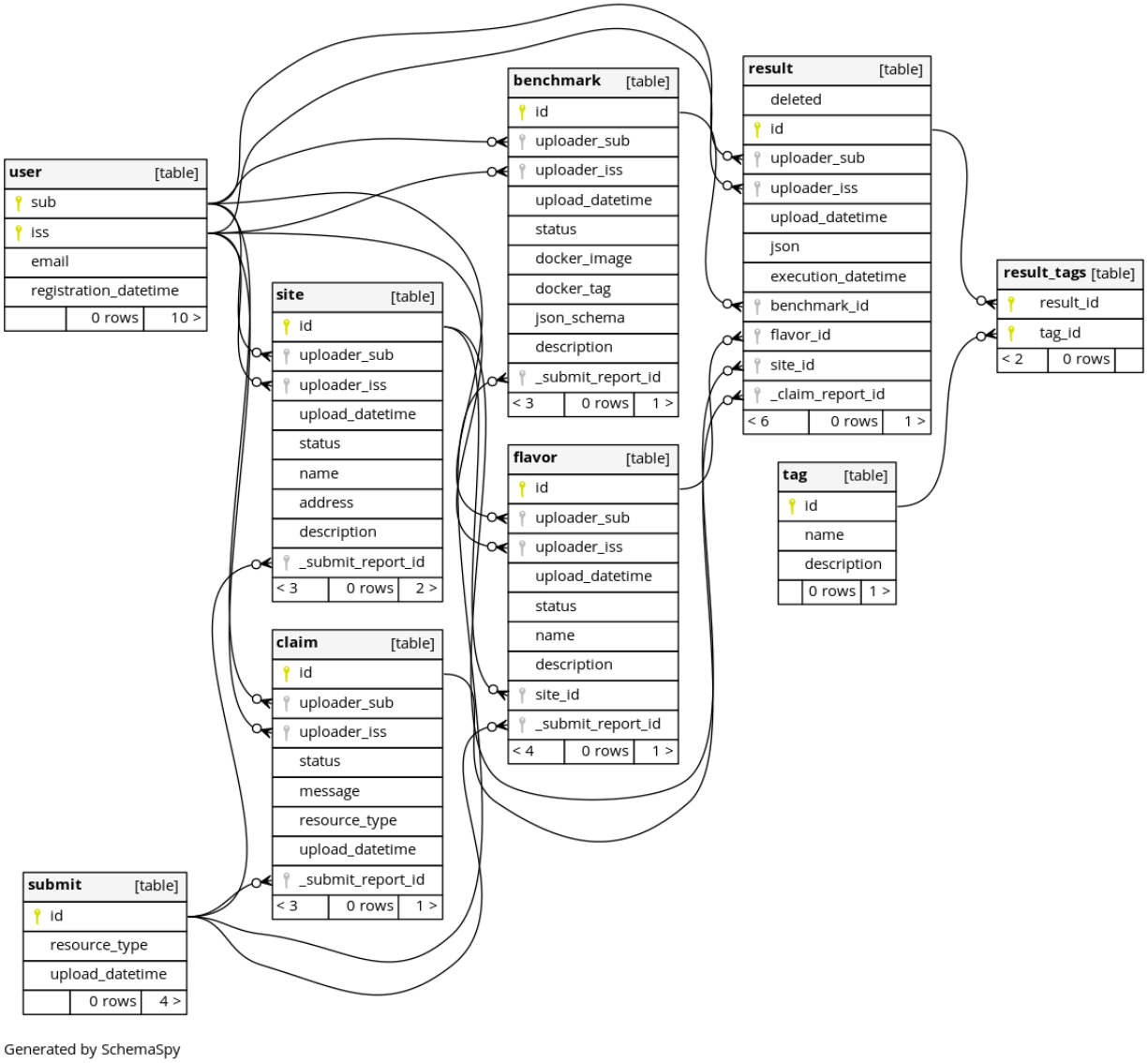
db = <SQLAlchemy engine=None>
Flask extension hat adds support for SQLAlchemy

migrate = <flask_migrate.Migrate object>
Flask extension that handles SQLAlchemy database migrations using Alembic

mail = <flask_mailman.Mail object>
Flask extension providing simple email sending capabilities

3.2.3 Models package

Backend package for models definition. Includes all necessary models to store benchmark results with all the necessary contextual information. The following diagram represents a relationships between the SQL models.



Note the classes representing the models might contain additional properties. For example association_proxies or column_properties which are simple mirroring or calculations between self and relationships properties.

All exported models are based on *backend.models.core.BaseCRUD*:

class BaseCRUD(***kwargs*)

Base model that adds CRUD methods to the model.

classmethod create(*properties*)

Creates and saves a new record into the database.

Parameters **properties** (*dict*) – Values to set on the model properties

Returns The record instance

Return type MixinCRUD

classmethod read(*primary_key*)

Returns the record from the database matching the id.

Parameters **primary_key** (*any*) – Record with primary_key to retrieve

Returns Record to retrieve or None

Return type `BaseModel` or None

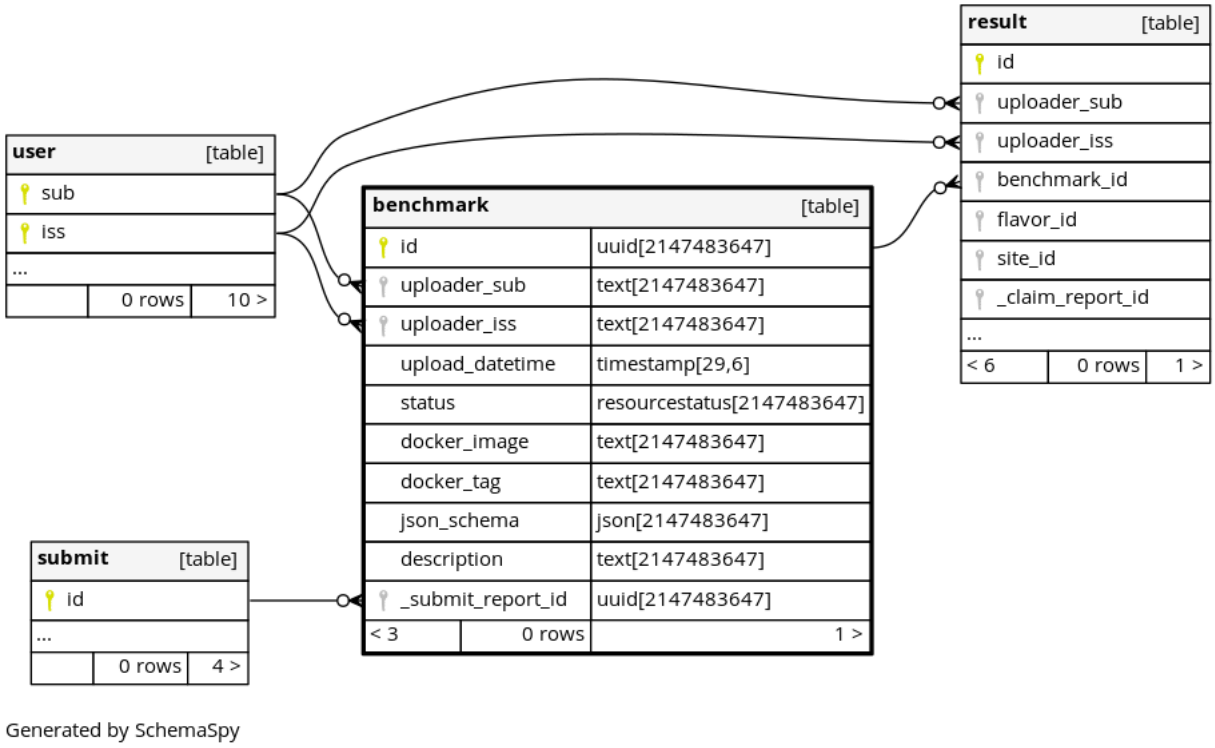
`update`(*properties*)
Updates specific fields from a record in the database.

Parameters **`properties`** (*`dict`*) – Values to set on the model properties

`delete`()
Deletes a specific record from the database.

The following chapters represent the expected usable models and their main accessible properties. Some tables such as results_tags or report_association are built-in tables automatically generated by the corresponding mixin (i.e. report.NeedsApprove)

Benchmark model



`class Benchmark`(***properties*)
Bases: `backend.models.models.reports.submit.NeedsApprove`, `backend.models.models.user.HasUploader`, `backend.models.core.PkModel`

The benchmark model represents a single type of docker container designed to run and produce benchmark results from virtual machines.

Benchmarks are tied down to a specific docker image and version to avoid confusion and misleading comparisons in case a benchmark container changes its metrics or scoring scale between versions.

It also includes a valid "JSON Schema" which is used to validate the results linked to the benchmark and uploaded into the system.

Description is optional but offers a valuable text that can help possible users to understand the benchmark main features.

Properties:

`docker_image`
(Text, required) Docker image referenced by the benchmark

`docker_tag`
(Text, required) Docker image version/tag referenced by the benchmark

`name`
(Text, read_only) Benchmark name: image:tag

`url`
(JSON, required) Schema used to validate benchmark results before upload

`json_schema`
(JSON, required) Schema used to validate benchmark results before upload

`description`
(Text) Short text describing the main benchmark features

`id`
(UUID) Primary key with an Unique Identifier for the model instance

`status`
(ItemStatus) Status of the resource

`submit_report`
(Report) Submit report related to the model instance

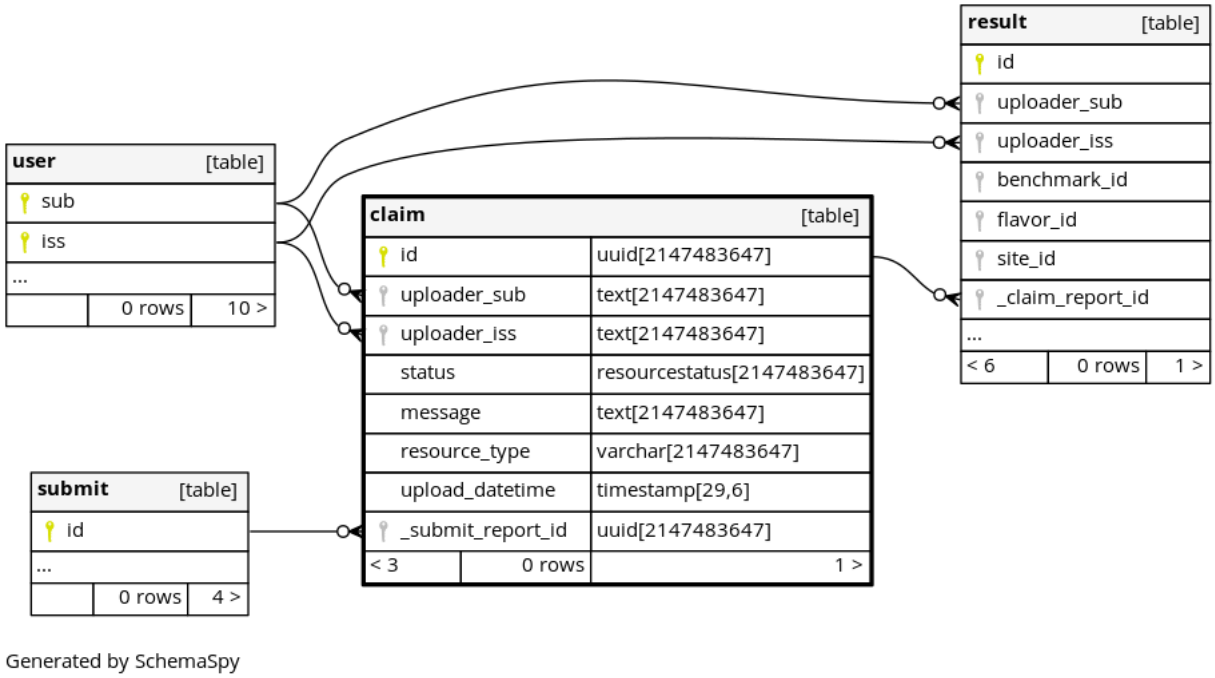
`upload_datetime`
(ISO8601) Upload datetime of the model instance

`uploader`
(User class) User that uploaded the model instance

`uploader_iss`
(Text) OIDC issuer of the user that created the model instance, *conflicts with uploader*

`uploader_sub`
(Text) OIDC subject of the user that created the model instance, *conflicts with uploader*

Claim model



class Claim(***properties*)

Bases: backend.models.models.reports.submit.NeedsApprove, backend.models.models.user.HasUploader, backend.models.core.PkModel

The Claim model represents an user's claim regarding a resource which should be processed by administrators.

Claims can be manually generated by the users comunity if they suspect a resource may be falsified or incorrect.

Properties:

message
: (Text) Information created by user to describe the issue

resource = NotImplementedError()
: (Resource) Resource the claim is linked to

resource_type
: (String) Refers to the type report

resource_id
: (Read_only) Resource unique identification

upload_datetime
: (ISO8601) Upload datetime of the model instance

delete()
: Deletes the claim report and restores the resource.

id
: (UUID) Primary key with an Unique Identifier for the model instance

status
: (ItemStatus) Status of the resource

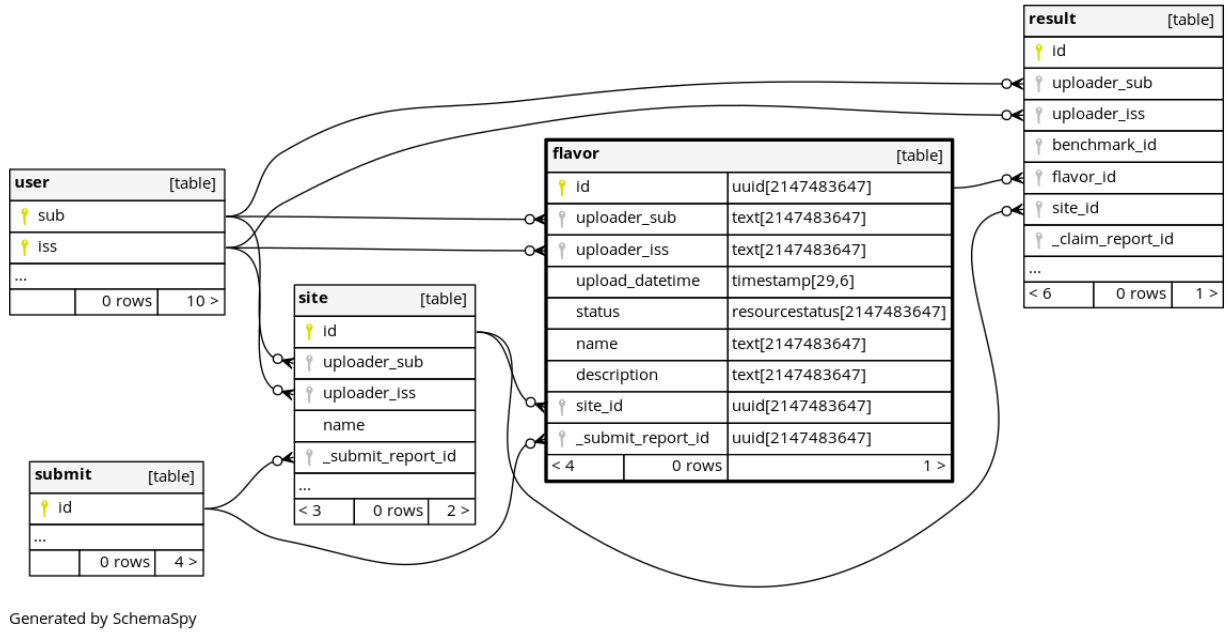
submit_report
(Report) Submit report related to the model instance

uploader
(User class) User that uploaded the model instance

uploader_iss
(Text) OIDC issuer of the user that created the model instance, *conflicts with uploader*

uploader_sub
(Text) OIDC subject of the user that created the model instance, *conflicts with uploader*

Flavor model



class Flavor(***properties*)
Bases: `backend.models.models.reports.submit.NeedsApprove`, `backend.models.models.user.HasUploader`, `backend.models.core.PkModel`

The Flavor model represents a flavor of virtual machines available for usage on a Site.

Flavours can be pre-existing options filled in by administrators or a custom configuration by the user.

Properties:

name
(Text, required) Text with virtual hardware template identification

description
(Text) Text with useful information for users

site_id
(Site.id, required) Id of the Site the flavor belongs to

site
(Site, required) Id of the Site the flavor belongs to

id
(UUID) Primary key with an Unique Identifier for the model instance

status
(ItemStatus) Status of the resource

submit_report
(Report) Submit report related to the model instance

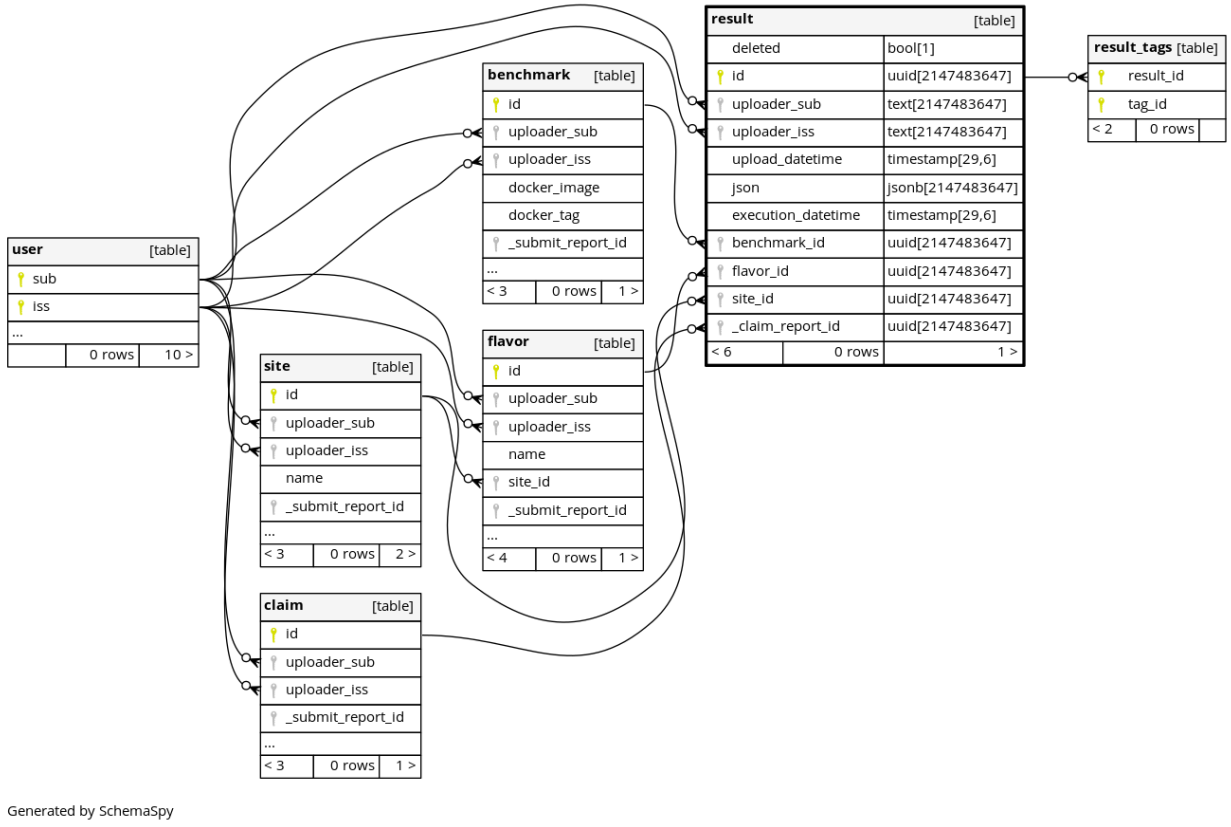
upload_datetime
(ISO8601) Upload datetime of the model instance

uploader
(User class) User that uploaded the model instance

uploader_iss
(Text) OIDC issuer of the user that created the model instance, *conflicts with uploader*

uploader_sub
(Text) OIDC subject of the user that created the model instance, *conflicts with uploader*

Result model



class Result(*site=None*, *site_id=None*, ***properties*)
Bases: `backend.models.models.reports.claim.HasClaims`, `backend.models.models.tag.HasTags`, `backend.models.models.user.HasUploader`, `backend.models.core.PkModel`

The Result model represents the results of the execution of a specific Benchmark on a specific Site and Flavor.

They carry the JSON data output by the executed benchmarks.

Properties:

`json`
(JSON, required) Benchmark execution results

`execution_datetime`
(ISO8601, required) Benchmark execution **START**

`benchmark_id`
(Conflicts Benchmark) Id of the benchmar used

`benchmark`
(Benchmark, required) Benchmark used to provide the results

`benchmark_name`

`flavor_id`
(Conflicts Flavor) Id of the flavor used to executed the benchmark

`flavor`
(Flavor, required) Flavor used to executed the benchmark

`flavor_name`

`site_id`
(Collected from flavor) Id of the site where the benchmar was executed

`site`
(Collected from flavor) Site where the benchmark was executed

`site_name`

`site_address`

`claims`
([Report]) Claim report related to the model instance

`deleted`
(Bool) Flag to hide the item from normal queries

`id`
(UUID) Primary key with an Unique Identifier for the model instance

`tags`
([Tag]) List of associated tags to the model

`tags_ids = ColumnAssociationProxyInstance(AssociationProxy('tags', 'id'))`

`tags_names = ColumnAssociationProxyInstance(AssociationProxy('tags', 'name'))`

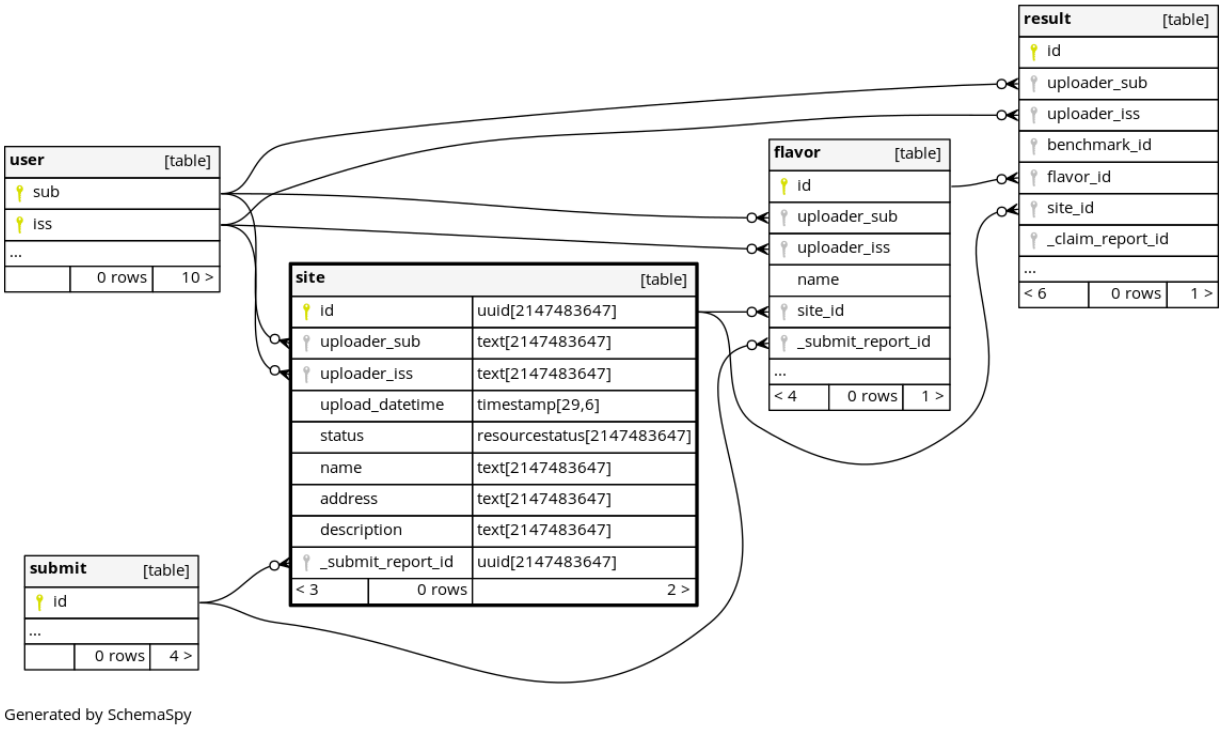
`upload_datetime`
(ISO8601) Upload datetime of the model instance

`uploader`
(User class) User that uploaded the model instance

`uploader_iss`
(Text) OIDC issuer of the user that created the model instance, *conflicts with uploader*

`uploader_sub`
(Text) OIDC subject of the user that created the model instance, *conflicts with uploader*

Site model



`class Site`(***properties*)

Bases: `backend.models.models.reports.submit.NeedsApprove`, `backend.models.models.user.HasUploader`, `backend.models.core.PkModel`

The Site model represents a location where a benchmark can be executed.

This generally refers to the different virtual machine providers and should include a human readable name and physical location.

`name`
(Text, required) Human readable institution identification

`address`
(Text, required) Place where a site is physically located

`description`
(Text) Useful site information to help users

`flavors`
([Flavor], read_only) List of flavors available at the site

`id`
(UUID) Primary key with an Unique Identifier for the model instance

`status`
(ItemStatus) Status of the resource

`submit_report`
(Report) Submit report related to the model instance

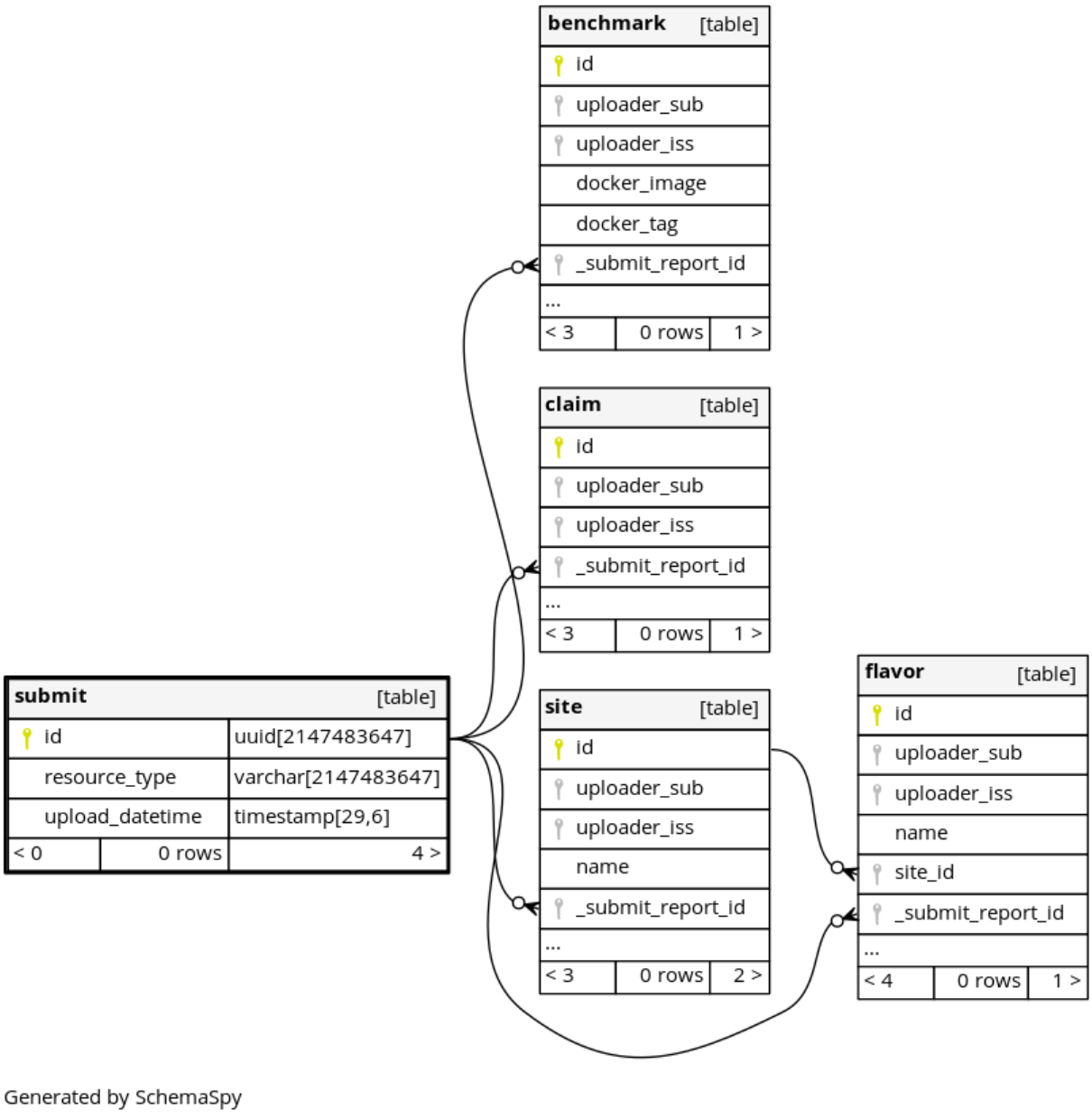
`upload_datetime`
(ISO8601) Upload datetime of the model instance

uploader
(User class) User that uploaded the model instance

uploader_iss
(Text) OIDC issuer of the user that created the model instance, *conflicts with uploader*

uploader_sub
(Text) OIDC subject of the user that created the model instance, *conflicts with uploader*

Submit model



class Submit(***properties*)
Bases: `backend.models.core.PkModel`

The Submit model represents an automated request for review. For example the creation of items which need to be approved before activated in the database or the notification that a new claim is created.

Properties:

resource = NotImplementedError()
(Resource) Resource the submit is linked to

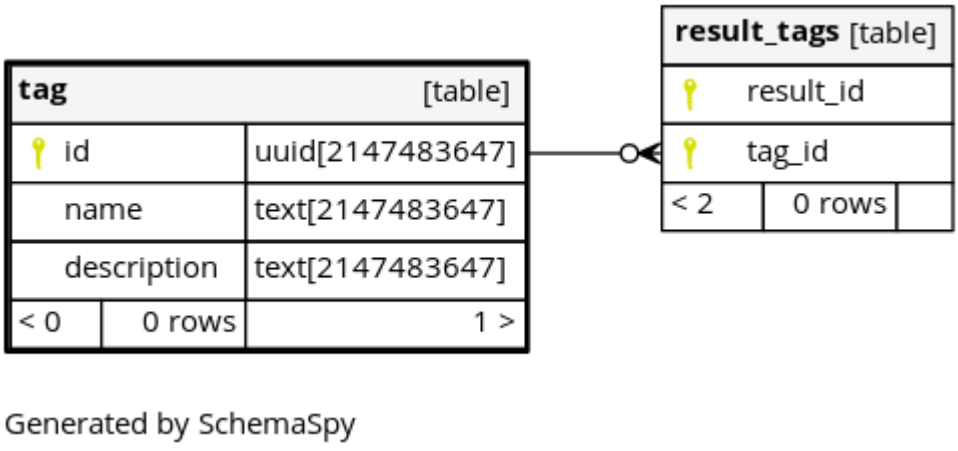
resource_type
(String) Refers to the type report

resource_id
(Read_only) Resource unique identification

upload_datetime

id
(UUID) Primary key with an Unique Identifier for the model instance

Tag model



class Tag(***properties*)
Bases: backend.models.core.PkModel

The Tag model represents a user-created label that can be used for filtering a list of results.

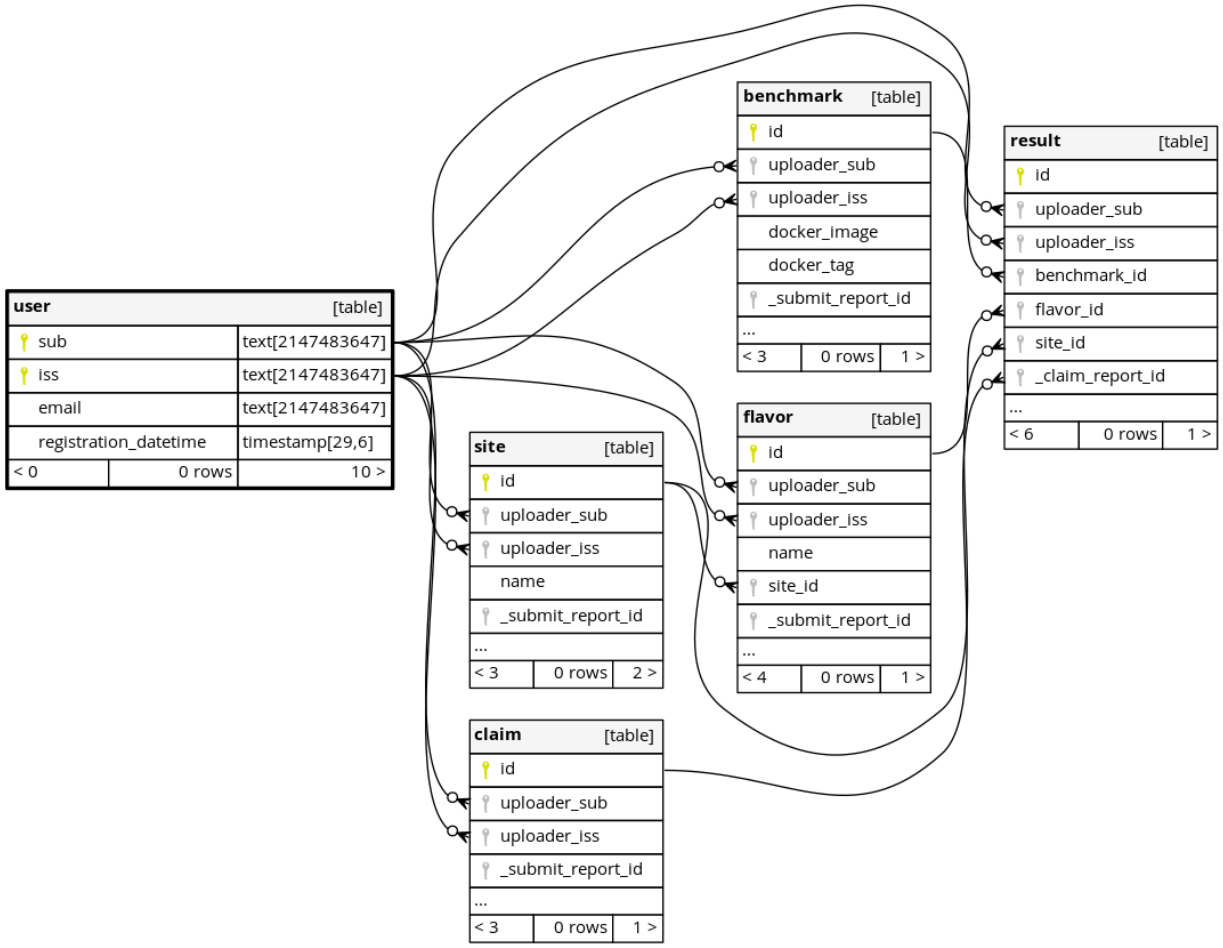
These are entirely created by users and may not necessarily be related to any benchmark output data. These may be used to indicate if, for example, a benchmark is used to measure CPU or GPU performance, since some benchmarks may be used to test both.

name
(Text, required) Human readable feature identification

description
(Text) Useful information to help users to understand the label context

id
(UUID) Primary key with an Unique Identifier for the model instance

User model



`class User`(***properties*)

Bases: `backend.models.core.TokenModel`

The User model represents the users of the application. Users are build over a OIDC token model, therefore are identified based on the 'Subject' and 'issuer' identifications provided by the OIDC provider.

Also an email is collected which is expected to match the one provided by the ODIC introspection endpoint.

Note an user might have multiple accounts with different ODIC providers and on each of them use the same email. Therefore the email cannot be a unique entity.

Properties:

`email`

`registration_datetime`

(DateTime) Time when the user was registered

`iss`

(Text) Primary key containing the OIDC issuer of the model instance

sub
(Text) Primary key containing the OIDC subject the model instance

3.2.4 Routes package

Backend package for the collection of the API URLs routes definitions.

The API uses routes to determine which controller and action method to execute. Routes should use meaningful URL users can remember and use directly when interfacing the API.

All routes defined in this package are build based on flask blueprints and the extension flask_smorest. This allows to split the routes between multiple sections collecting the related methods per module.

The flask extension flask_smorest allows to produce a OpenAPI JSON specification which can be used by automation tools. For example swagger can use such specification to produce an user friendly GUI for the API.

Benchmarks routes

Benchmark URL routes. Collection of controller methods to create and operate existing benchmarks on the database.

approve(**args*, ***kwargs*)
(Admins) Approves a benchmark to include it on default list methods

Use this method to approve an specific benchmark submitted by an user. It is a custom method, as side effect, it removes the submit report associated as it is no longer needed.

create(**args*, ***kwargs*)
(Users) Uploads a new benchmark

Use this method to create a new benchmarks in the database so it can be accessed by the application users. The method returns the complete created benchmark (if succeeds).

Note: Benchmark use JSON Schemas to implement results validation.

delete(**args*, ***kwargs*)
(Admins) Deletes an existing benchmark

Use this method to delete a specific benchmark from the database.

get(**args*, ***kwargs*)
(Public) Retrieves benchmark details

Use this method to retrieve a specific benchmark from the database.

list(**args*, ***kwargs*)
(Public) Filters and list benchmarks

Use this method to get a list of benchmarks filtered according to your requirements. The response returns a pagination object with the filtered benchmarks (if succeeds).

reject(**args*, ***kwargs*)
(Admins) Rejects a benchmark to safe delete it.

Use this method instead of DELETE as it raises 422 in case the resource was already approved. Use this method to reject an specific benchmark submitted by an user. It is a custom method, as side effect, it removes the submit report associated as it is no longer needed.

search(**args*, ***kwargs*)
(Public) Filters and list benchmarks

Use this method to get a list of benchmarks based on a general search of terms. For example, calling this method with terms=v1&terms=0 returns all benchmarks with 'v1' and '0' on the 'docker_image', 'docker_tag' or 'description' fields. The response returns a pagination object with the filtered benchmarks (if succeeds).

update(**args*, ***kwargs*)
(Admins) Implements JSON Put for benchmarks

Use this method to update a specific benchmark from the database.

Reports routes

Report URL routes. Collection of controller methods to create and operate existing reports on the database.

approve_claim(**args*, ***kwargs*)
(Admin) Accepts an existing claim

Use this method to approve an specific resource submitted by an user. It is a custom method, as side effect, it removes the submit report associated as it is no longer needed.

get(**args*, ***kwargs*)
(Public) Retrieves claim details

Use this method to retrieve a specific claim from the database.

list_claims(**args*, ***kwargs*)
(Admins) Filters and lists claims

Use this method to get a list of claims filtered according to your requirements. The response returns a pagination object with the filtered claims (if succeeds).

list_submits(**args*, ***kwargs*)
(Admins) Filters and list submits

Use this method to get a list of submits filtered according to your requirements. The response returns a pagination object with the filtered submits (if succeeds).

reject_claim(**args*, ***kwargs*)
(Admin) Refuses an existing claim

Use this method to reject an specific resource submitted by an user. It is a custom method, as side effect, it removes the resource and the submit report associated as it is no longer needed.

Results routes

Results URL routes. Collection of controller methods to create and operate existing benchmark results on the database.

claim(**args*, ***kwargs*)
(Users) Reports a result

Use this method to create a report for a specific result so the administrators are aware of issues. The reported result is hidden from generic responses until the issue is corrected and approved by the administrators.

create(**args*, ***kwargs*)
(Users) Uploads a new result

Use this method to create a new result in the database so it can be accessed by the application users. The method returns the complete created result (if succeeds).

The uploaded result must pass the benchmark JSON Schema to be accepted, otherwise 422 UnprocessableEntity is produced. In addition, an execution_datetime must be provided in order to indicate the time when the benchmark was executed. It should be in ISO8601 format and include the timezone.

delete(**args*, ***kwargs*)
(Admin) Deletes an existing result

Use this method to delete a specific result from the database.

get(**args*, ***kwargs*)
(Public) Retrieves result details

Use this method to retrieve a specific result from the database.

get_uploader(**args*, ***kwargs*)
(Admins) Retrieves result uploader

Use this method to retrieve the uploader of a specific result from the database.

list(**args*, ***kwargs*)
(Public) Filters and list results

Use this method to get a list of results filtered according to your requirements. The response returns a pagination object with the filtered results (if succeeds).

This method allows to return results filtered by values inside the result. The filter is composed by 3 arguments separated by spaces ('%20' on URL-encoding): <path.separated.by.dots> <operator> <value>

There are five filter operators:

- **Equals (==)**: Return results where path value is exact to the query value. For example *filters=cpu.count == 5*
- **Greater than (>)**: Return results where path value strictly greater than the query value. For example *filters=cpu.count > 5*
- **Less than (<)**: Return results where path value strictly lower than the query value. For example *filters=cpu.count < 5*
- **Greater or equal (>=)**: Return results where path value is equal or greater than the query value. For example *filters=cpu.count >= 5*
- **Less or equal (<=)**: Return results where path value is equal or lower than the query value. For example *filters=cpu.count <= 5*

Note that in the provided examples the filter is not URL-encoded as most libraries do it automatically, however there might be exception. In such cases, use the url encoding guide at: https://datatracker.ietf.org/doc/html/rfc3986#section-2.1

list_claims(**args*, ***kwargs*)
(Owner or Admins) Returns the result claims.

Use this method to retrieve all the result claims.

search(**args*, ***kwargs*)
(Public) Filters and list results

Use this method to get a list of results based on a general search of terms. For example, calling this method with terms=v1&terms=0 returns all results with 'v1' and '0' on the 'docker_image', 'docker_tag', 'site_name', 'flavor_name' fields or 'tags'. The response returns a pagination object with the filtered results (if succeeds).

update_tags(**args*, ***kwargs*)
(Owner or Admin) Updates an existing result tags

Use this method to update tags on a specific result from the database.

Sites routes

Site URL routes. Collection of controller methods to create and operate existing sites on the database.

approve(**args*, ***kwargs*)
(Admins) Approves a site to include it on default list methods

Use this method to approve an specific site submitted by an user. It is a custom method, as side effect, it removes the submit report associated as it is no longer needed.

create(**args*, ***kwargs*)
(Users) Uploads a new site

Use this method to create a new site in the database so it can be accessed by the application users. The method returns the complete created site (if succeeds).

create_flavor(**args*, ***kwargs*)
(Users) Uploads a new flavor

Use this method to create a new flavors in the database so it can be accessed by the application users. The method returns the complete created flavor (if succeeds).

delete(**args*, ***kwargs*)
(Admins) Deletes an existing site

Use this method to delete a specific site from the database.

get(**args*, ***kwargs*)
(Public) Retrieves site details

Use this method to retrieve a specific site from the database.

list(**args*, ***kwargs*)
(Public) Filters and list sites

Use this method to get a list of sites filtered according to your requirements. The response returns a pagination object with the filtered sites (if succeeds).

list_flavors(**args*, ***kwargs*)
(Public) Filters and list flavors

Use this method to get a list of flavors filtered according to your requirements. The response returns a pagination object with the filtered flavors (if succeeds).

reject(**args*, ***kwargs*)
(Admins) Rejects a site to safe delete it.

Use this method instead of DELETE as it raises 422 in case the resource was already approved.

Use this method to reject an specific site submitted by an user. It is a custom method, as side effect, it removes the submit report associated as it is no longer needed.

search(**args*, ***kwargs*)
(Public) Filters and list sites

Use this method to get a list of sites based on a general search of terms. For example, calling this method with terms=K&terms=T returns all sites with 'K' and 'T' on the 'name', 'address', or 'description' fields. The response returns a pagination object with the filtered sites (if succeeds).

search_flavors(**args*, ***kwargs*)
(Public) Filters and list flavors

Use this method to get a list of flavors based on a general search of terms. For example, calling this method with terms=K&terms=T returns all flavors with 'K' and 'T' on the 'name', or 'description' fields. The response returns a pagination object with the filtered flavors (if succeeds).

update(**args*, ***kwargs*)
(Admins) Updates an existing site

Use this method to update a specific site from the database.

Flavor routes

Flavor URL routes. Collection of controller methods to operate existing flavors on the database.

approve(**args*, ***kwargs*)
(Admins) Approves a flavor to include it on default list methods

Use this method to approve an specific flavor submitted by an user. It is a custom method, as side effect, it removes the submit report associated as it is no longer needed.

delete(**args*, ***kwargs*)
(Admins) Deletes an existing flavor

Use this method to delete a specific flavor from the database.

get(**args*, ***kwargs*)
(Public) Retrieves flavor details

Use this method to retrieve a specific flavor from the database.

reject(**args*, ***kwargs*)
(Admins) Rejects a flavor to safe delete it.

Use this method instead of DELETE as it raises 422 in case the resource was already approved.

Use this method to reject an specific flavor submitted by an user. It is a custom method, as side effect, it removes the submit report associated as it is no longer needed.

site(**args*, ***kwargs*)
(Public) Retrieves flavor site details

Use this method to retrieve the site information from a specific flavor in the database.

update(**args*, ***kwargs*)
(Admins) Updates an existing flavor

Use this method to update a specific flavor from the database.

Tags routes

Tag URL routes. Collection of controller methods to create and operate existing user tags on the database.

create(**args*, ***kwargs*)
(Users) Uploads a new tag

Use this method to create a new tags in the database so it can be accessed by the application users. The method returns the complete created tag (if succeeds).

delete(**args*, ***kwargs*)
(Admins) Deletes an existing tag

Use this method to delete a specific tag from the database.

get(**args*, ***kwargs*)
(Public) Retrieves tag details

Use this method to retrieve a specific tag from the database.

list(**args*, ***kwargs*)
(Public) Filters and list tags

Use this method to get a list of tags filtered according to your requirements. The response returns a pagination object with the filtered tags (if succeeds).

search(**args*, ***kwargs*)
(Public) Filters and list tags

Use this method to get a list of tags based on a general search of terms. For example, calling this method with terms=v1&terms=0 returns all tags with 'v1' and '0' on the 'name' or 'description' fields. The response returns a pagination object with the filtered tags (if succeeds).

update(**args*, ***kwargs*)
(Admins) Updates an existing tag

Use this method to update a specific tag from the database.

Users routes

Users URL routes. Collection of controller methods to create and operate existing users on the database.

claims(**args*, ***kwargs*)
(Users) Returns your uploaded pending claims

Use this method to retrieve all the claims uploaded by your user.

get(**args*, ***kwargs*)
(Users) Retrieves the logged in user info

Use this method to retrieve your user data stored in the database.

list(**args*, ***kwargs*)
(Admins) Filters and list users

Use this method to get a list of users filtered according to your requirements. The response returns a pagination object with the filtered users (if succeeds).

register(**args*, ***kwargs*)
(OIDC Token) Registers the logged in user

Use this method to register yourself into the application. By using this method, you recognize that you have read and understood our terms, conditions and privacy policy at: *https://performance.services.fedcloud.eu/privacy_policy*

The method will return your stored information.

remove(**args*, ***kwargs*)
(Admins) Removes one or multiple users

Use this method to delete the users filtered according to your requirements. To prevent unintentionally delete all users, the method requires of query arguments, otherwise UnprocessableEntity exception is raised.

results(**args*, ***kwargs*)
(Users) Returns your uploaded results

Use this method to retrieve all the results uploaded by your user. You can use the query parameter to retrieve also those with pending claims.

search(**args*, ***kwargs*)
(Admins) Filters and list users

Use this method to get a list of users based on a general search of terms. For example, calling this method with terms=@hotmail&terms=de returns all users with 'hotmail' and 'de' on the 'email'. The response returns a pagination object with the filtered users (if succeeds).

try_admin()
(Admins) Returns 204 if you are admin

Use this method to check that you have the administration rights. If so, the access returns 204, otherwise 401 or 403 are expected. —

Raises

- **Unauthorized** – The server could not verify your identity
- **Forbidden** – You don't have the administrator rights

update(**args*, ***kwargs*)
(Users) Updates the logged in user info

Use this method to update your user data in the database. The method returns by default 204, use a GET method to retrieve the new status of your data.

3.2.5 Schemas package

Backend package for schemas definition. This module is based on the group of python framework **marshmallow**, an ORM/ODM/framework-agnostic library for converting complex datatypes, such as objects, to and from native Python datatypes.

Combined with the flask extension flask_smorest, it allows to generate the OpenAPI specification schemas based on python class objects.

This objects come into 2 types:

- Schemas: JSON structures used to operate model instances
- Arguments: Query arguments to control route method parameters

class BaseSchema(**, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None*)
Bases: marshmallow.schema.Schema

Base schema to control common schema features.

class Meta
Bases: object

marshmallow options object for BaseSchema.

ordered = True
Enforce Order in OpenAPI Specification File

class Pagination(**, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None*)
Bases: marshmallow.schema.Schema

Pagination schema to limit the amount of results provided by a method

has_next
(Bool, required, dump_only): True if a next page exists.

has_prev
(Bool, required, dump_only): True if a previous page exists.

next_num
(Int, required, dump_only): Number of the next page.

prev_num
(Int, required, dump_only): Number of the previous page.

pages
(Int, required, dump_only): The total number of pages

per_page
(Int, required, dump_only): The number of items to be displayed on a page.

page
(Int, required, dump_only): The return page number (1 indexed).

total
(Int, required, dump_only): The total number of items matching the query.

class Id(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: marshmallow.schema.Schema

id
(UUID, required, dump_only): Primary key with an Unique Identifier for the model instance

class Status(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: marshmallow.schema.Schema

status
(Str): Resource current state (approved, on_review, etc.)

class Search(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: marshmallow.schema.Schema

terms
([Text]): Group of strings to use as general search on model instances

sort_by
(Str): Order to return the results separated by coma

class UploadDatetime(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: marshmallow.schema.Schema

upload_datetime
(ISO8601): Upload datetime of the model instance

class **UploadFilter**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: marshmallow.schema.Schema

upload_before
(ISO8601, attribute="upload_datetime"): Upload datetime of the instance before a specific date

upload_after
(ISO8601, attribute="upload_datetime"): Upload datetime of the instance after a specific date

Benchmark schemas

class **Benchmark**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: *backend.schemas.Id*, *backend.schemas.UploadDatetime*, *backend.schemas.schemas.CreateBenchmark*

class **Benchmarks**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: *backend.schemas.Pagination*, *backend.schemas.BaseSchema*

items
([Benchmark], required): List of benchmark items for the pagination object

class **CreateBenchmark**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)
Bases: *backend.schemas.BaseSchema*

docker_image
(Text, required): Docker image referenced by the benchmark

docker_tag
(Text, required): Docker image version/tag referenced by the benchmark

url
(Text, required): URL to the benchmark container documentation

json_schema
(JSON, required): Schema used to validate benchmark results before upload

description
(Text): Short text describing the main benchmark features

Benchmark arguments

class **BenchmarkFilter**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.Status, backend.schemas.BaseSchema

docker_image

(Text, required): Docker image referenced by the benchmark

docker_tag

(Text, required): Docker image version/tag referenced by the benchmark

sort_by

(Str): Order to return the results separated by coma

class **BenchmarkSearch**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.Status, backend.schemas.Search, backend.schemas.BaseSchema

Flavor schemas

class **Flavor**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.Id, backend.schemas.UploadDatetime, backend.schemas.schemas.CreateFlavor

class **Flavors**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.Pagination, backend.schemas.BaseSchema

items

([Flavor], required): List of flavor items for the pagination object

class **CreateFlavor**(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.BaseSchema

name

(Text, required): Text with virtual hardware template identification

description

(Text, required): Text with useful information for users

Flavor arguments

class FlavorFilter(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: `backend.schemas.args.Pagination`, *backend.schemas.UploadFilter*, *backend.schemas.Status*, *backend.schemas.BaseSchema*

name
(Text): Text with virtual hardware template identification

sort_by
(Str): Order to return the results separated by coma

class FlavorSearch(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: `backend.schemas.args.Pagination`, *backend.schemas.UploadFilter*, *backend.schemas.Status*, *backend.schemas.Search*, *backend.schemas.BaseSchema*

Report schemas

class Submit(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: *backend.schemas.UploadDatetime*, *backend.schemas.BaseSchema*

resource_type
(String, required): Resource discriminator

resource_id
(UUID, required): Resource unique identification

uploader
(User, required): Resource uploader/creator

class Submits(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: *backend.schemas.Pagination*, *backend.schemas.BaseSchema*

items
([Submit], required): List of submit items for the pagination object

class CreateClaim(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: *backend.schemas.BaseSchema*

message
(String, required): Claim text describing the resource issue

class Claim(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.Id, backend.schemas.UploadDatetime, backend.schemas.schemas.CreateClaim

resource_type

(UUID, required): Resource unique identification

resource_id

(UUID, required): Resource unique identification

uploader

(User, required): Claim uploader/creator

class Claims(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.Pagination, backend.schemas.BaseSchema

items

([Claim], required): List of claim items for the pagination object

Report arguments

class SubmitFilter(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.BaseSchema

resource_type

(String): Resource discriminator

sort_by

(Str): Order to return the results separated by coma

class ClaimFilter(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.Status, backend.schemas.BaseSchema

sort_by

(Str): Order to return the results separated by coma

Result schemas

class Result(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.Id, backend.schemas.UploadDatetime, backend.schemas.BaseSchema

execution_datetime
(ISO8601, required) : Benchmark execution **START**

benchmark
(Benchmark, required): Benchmark used to provide the results

site
(Site, required): Site where the benchmark was executed

flavor
(Flavor, required): Flavor used to executed the benchmark

tags
([Tag], required): List of associated tags to the model

json
(JSON, required): Benchmark execution results

class Results(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.Pagination, backend.schemas.BaseSchema

items
([Result], required): List of results items for the pagination object

class Json(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.BaseSchema

Special schema to allow free JSON property

class Meta
Bases: object

marshmallow options object for JSON properties

unknown = 'include'
Accept and include the unknown fields

Result arguments

class ResultFilter(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.BaseSchema

execution_before
(ISO8601): Execution datetime of the instance before a specific date

execution_after
(ISO8601): Execution datetime of the instance after a specific date

benchmark_id
(Benchmark.id): Unique Identifier for result associated benchmark

site_id
(Site.id): Unique Identifier for result associated site

flavor_id
(Flavor.id): Unique Identifier for result associated flavor

tags_ids
([Tag.id], required): Unique Identifiers for result associated tags

filters
(String; <json.path> <operation> <value>) Expression to condition the returned results on JSON field

sort_by
(Str): Order to return the results separated by coma

class ResultContext(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.BaseSchema

execution_datetime
(ISO8601, required) : Benchmark execution **START**

benchmark_id
(Benchmark.id, required): Unique Identifier for result associated benchmark

flavor_id
(Flavor.id, required): Unique Identifier for result associated flavor

tags_ids
([Tag.id], default=[]): Unique Identifiers for result associated tags

class ResultSearch(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.Search, backend.schemas.BaseSchema

Site schemas

class Site(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.Id, backend.schemas.UploadDatetime, backend.schemas.schemas.CreateSite

class Sites(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.Pagination, backend.schemas.BaseSchema

items

([Site], required): List of site items for the pagination object

class CreateSite(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.BaseSchema

name

(Text, required): Human readable institution identification

address

(Text, required): Place where a site is physically located

description

(Text, required): Useful site information to help users

Site arguments

class SiteFilter(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.Status, backend.schemas.BaseSchema

name

(Text): Human readable institution identification

address

(Text): Place where a site is physically located

sort_by

(Str): Order to return the results separated by coma

class SiteSearch(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.UploadFilter, backend.schemas.Status, backend.schemas.Search, backend.schemas.BaseSchema

Tag schemas

class Tag(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.Id, backend.schemas.schemas.CreateTag

class Tags(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.Pagination, backend.schemas.BaseSchema

items

([Tag], required): List of tag items for the pagination object

class TagsIds(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.BaseSchema

tags_ids

([UUID]): List of tag ids

class CreateTag(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.BaseSchema

name

(Text, required): Human readable feature identification

description

(Text): Useful information to help users to understand the label context

Tag arguments

class TagFilter(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.BaseSchema

name

(Text): Human readable feature identification

sort_by

(Str): Order to return the results separated by coma

class TagSearch(*, only: Optional[Union[Sequence[str], Set[str]]] = None, exclude: Union[Sequence[str], Set[str]] = (), many: bool = False, context: Optional[Dict] = None, load_only: Union[Sequence[str], Set[str]] = (), dump_only: Union[Sequence[str], Set[str]] = (), partial: Union[bool, Sequence[str], Set[str]] = False, unknown: Optional[str] = None)

Bases: backend.schemas.args.Pagination, backend.schemas.Search, backend.schemas.BaseSchema

User schemas

class User(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: *backend.schemas.BaseSchema*

sub
(Text, required, dump_only): Primary key containing the OIDC subject the model instance

iss
(Text, required, dump_only): Primary key containing the OIDC issuer of the model instance

email
(Email, required): Electronic mail collected from OIDC access token

registration_datetime
(Email) Electronic mail collected from OIDC access token

class Users(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: *backend.schemas.Pagination*, *backend.schemas.BaseSchema*

items
([User], required): List of site items for the pagination object

User arguments

class UserFilter(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.args.Pagination, *backend.schemas.BaseSchema*

sub
(Text, required, dump_only): Primary key containing the OIDC subject the model instance

iss
(Text, required, dump_only): Primary key containing the OIDC issuer of the model instance

email
(Email, required): Electronic mail collected from OIDC access token

class UserDelete(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: *backend.schemas.args.UserFilter*

class UserSearch(*, *only: Optional[Union[Sequence[str], Set[str]]] = None*, *exclude: Union[Sequence[str], Set[str]] = ()*, *many: bool = False*, *context: Optional[Dict] = None*, *load_only: Union[Sequence[str], Set[str]] = ()*, *dump_only: Union[Sequence[str], Set[str]] = ()*, *partial: Union[bool, Sequence[str], Set[str]] = False*, *unknown: Optional[str] = None*)

Bases: backend.schemas.args.Pagination, *backend.schemas.Search*, *backend.schemas.BaseSchema*

3.2.6 Authorization module

is_registered(*user_infos*)
Assert user is registered in the database.

is_admin(*user_infos*)
Assert registration and entitlements.

3.2.7 Notifications module

Module with notification definitions for users and admins.

warning_if_fail(*notification*)

user_welcome(*user*)

email_updated(*user*)

resource_submitted(*resource*)

resource_approved(*resource*)

resource_rejected(*uploader*, *resource*)

result_claimed(*result*, *claim*)

result_restored(*result*)

3.2.8 Utils package

Utils subpackage with tools to extend controller functions.

Imagerepo module

manifest(*imagerepo*, *tag*)

- *Settings module*: Configuration variables from environment
- *Extensions module*: Flask additional components
- *Models package*: Database models and tables used by the API
- *Routes package*: URL routes and controller methods
- *Schemas package*: Defined OpenAPI schemas to interface the API
- *Authorization module*: Authorization methods to access the API
- *Notifications module*: Notification functions for email messages
- *Utils package*: Group of tools to simplify internal components

3.3 Database factories

All exported models are based on `factories.core.BaseMeta`:

The following chapters represent the expected usable factories with their main accessible properties.

3.3.1 Benchmark factory

3.3.2 Flavor factory

3.3.3 Result factory

3.3.4 Site factory

3.3.5 Tag factory

3.3.6 User factory

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W