aiohttp-tus Documentation

Release 1.0.0

Okumy

Jan 05, 2022

CONTENTS

1 Quickstart **3**

2 Chunk Size **5**

3 CORS Headers **7**

4 Reverse proxy and HTTPS **9**

5 Examples **11**

6 Installation **13**

7 License **15**

8 Contents **17**

8.1 Usage . . . 17
8.1.1 Default . . . 17
8.1.2 Understanding tus.io Chunk Size . . . 17
8.1.3 CORS Headers . . . 18
8.1.4 Reverse proxy and HTTPS . . . 19
8.1.5 User Uploads . . . 19
8.1.6 On Upload Done Callback . . . 20
8.1.7 Mutliple TUS upload URLs . . . 20
8.1.8 Upload resource name . . . 20
8.2 API Reference . . . 21
8.2.1 aiohttp_tus . . . 21
8.2.2 aiohttp_tus.data . . . 22
8.3 Authors & Contributors . . . 22
8.4 Changelog . . . 22
8.4.1 1.1.0 (2022-01-04) . . . 22
8.4.2 1.0.0 (2020-06-08) . . . 23
8.4.3 1.0.0rc1 (2020-04-02) . . . 23
8.4.4 1.0.0rc0 (2020-03-26) . . . 23
8.4.5 1.0.0b2 (2020-03-18) . . . 23
8.4.6 1.0.0b1 (2020-03-18) . . . 23
8.4.7 1.0.0b0 (2020-03-15) . . . 23
8.4.8 1.0.0a1 (2020-03-12) . . . 23
8.4.9 1.0.0a0 (2020-03-11) . . . 24

Index **25**

tus.io server implementation for aiohttp.web applications.

For uploading large files, please consider using aiotus (Python 3.7+) library instead.

- Works on Python 3.6+
- Works with aiohttp 3.5+
- BSD licensed
- Latest documentation on Read The Docs
- Source, issues, and pull requests on GitHub

CHAPTER

ONE

QUICKSTART

Code belows shows how to enable tus-compatible uploads on `/uploads` URL for `aiohttp.web` application. After upload, files will be available at `../uploads` directory.

```
from pathlib import Path

from aiohttp import web
from aiohttp_tus import setup_tus


app = setup_tus(
    web.Application(),
    upload_url="/uploads",
    upload_path=Path(__file__).parent.parent / "uploads",
)
```

CHAPTER

TWO

CHUNK SIZE

Please, make sure to configure `client_max_size` for `aiohttp.web` Application and supply proper `chunkSize` for Uppy.io or other tus.io client.

Documentation

CHAPTER

THREE

CORS HEADERS

To setup CORS headers you need to use cors_middleware from aiohttp-middlewares package. aiohttp-cors library not supported cause of aio-libs/aiohttp-cors#241 issue.

Documentation

CHAPTER

FOUR

REVERSE PROXY AND HTTPS

When `aiohttp` application deployed under the reverse proxy (such as nginx) with HTTPS support, it is needed to use https_middleware from aiohttp-middlewares package to ensure that `web.Request` instance has proper schema.

Documentation

CHAPTER

FIVE

EXAMPLES

examples/ directory contains several examples, which illustrate how to use `aiohttp-tus` with some tus.io clients, such as tus.py and Uppy.io.

CHAPTER

SIX

INSTALLATION

```
pip install aiohttp-tus
```

Or using poetry:

```
poetry add aiohttp-tus
```

CHAPTER

SEVEN

LICENSE

aiohttp-tus is licensed under the terms of BSD License.

CHAPTER

EIGHT

CONTENTS

8.1 Usage

8.1.1 Default

To allow upload files to `../uploads` directory for all clients via `/uploads` URL,

```
from pathlib import Path

from aiohttp import web
from aiohttp_tus import setup_tus


app = setup_tus(
    web.Application(),
    upload_path=Path(__file__).parent.parent / "uploads",
)
```

8.1.2 Understanding tus.io Chunk Size

By default, Uppy and some other tus.io clients do not setup chunk size and tries to upload as large chunk, as possible. However as `aiohttp.web.Application` setting up `client_max_size` on app initialization you might need to configure server to receive larger chunks as well as setup tus.io client to use respected chunk sizes.

Examples below shown on how to config different parts to upload files with chunk size of **4MB** (`4_000_000` bytes)

aiohttp.web configuration

```
from aiohttp import web
from aiohttp_tus import setup_tus

app = web.Application(client_max_size=4_000_000)
```

nginx configuration

```
location ~ ^/uploads.*$ {
    client_max_body_size 4M;
    proxy_pass http://localhost:8080;
}
```

tus.py configuration

```
tus-upload --chunk-size=4000000 \
    /path/to/large-file http://localhost:8080/uploads
```

uppy.io Configuration

```
uppy.use(Uppy.Tus, {
    endpoint: "http://localhost:8080/uploads",
    chunkSize: 3999999
})
```

Important: To make Uppy.Tus plugin work you need to specify chunk size **at least 1 byte smaller** than `client_max_size`. If you'll provide chunk size equals to client max size upload will not work properly.

8.1.3 CORS Headers

At a moment (*May 17 2020*), `aiohttp-tus` supports setting up CORS Headers for `aiohttp.web` application only via cors_middleware from aiohttp-middlewares package.

As `aiohttp-tus` registers *OPTIONS* handlers it doesn't work with aiohttp-cors library cause of known issue aio-libs/aiohttp-cors#241. (Full discussion)

To enable CORS Headers for your `aiohttp.web` application, which is using `aiohttp-tus`, you need to,

1. Install aiohttp-middlewares
2. In your *app.py*,

```
from pathlib import Path

from aiohttp import web
from aiohttp_middlewares import cors_middleware
from aiohttp_tus import setup_tus


# Allow CORS Headers for requests from http://localhost:3000
app = web.Application(
    middlewares=(
        cors_middleware(origins=("http://localhost:3000",)),
    )
)
setup_tus(
    app, upload_path=Path(__file__).parent.parent / "uploads",
)
```

8.1.4 Reverse proxy and HTTPS

When aiohttp web application with `aiohttp-tus` deployed under the reverse proxy (such as nginx), with HTTPS support **you need to** setup https_middleware from aiohttp-middlewares package to ensure that `aiohttp.web.Request` instance has proper schema.

To use HTTPS middleware you need to,

1. Install aiohttp-middlewares
2. In *app.py*,

```
from pathlib import Path

from aiohttp import web
from aiohttp_middlewares import https_middleware
from aiohttp_tus import setup_tus

app = web.Application(middlewares=(https_middleware(),))
setup_tus(
    app, upload_path=Path(__file__).parent.parent / "uploads"
)
```

8.1.5 User Uploads

To allow upload files to `/files/{username}` directory only for authenticated users via `/users/{username}/uploads` URL,

```
from aiohttp_tus.annotations import Handler


def upload_user_required(handler: Handler) -> Handler:
    async def decorator(request: web.Request) -> web.Response:
        # Change ``is_user_authenticated`` call to actual call,
        # checking whether user authetnicated for given request
        # or not
        if not is_user_authenticated(request):
            raise web.HTTPForbidden()
        return await handler(request)

    return decorator


app = setup_tus(
    web.Application(),
    upload_path=Path("/files") / r"{username}",
    upload_url=r"/users/{username}/uploads",
    decorator=upload_user_required,
)
```

8.1.6 On Upload Done Callback

There is a possibility to run any coroutine after upload is done. Example below, illustrates how to achieve that,

```
from aiohttp_tus.data import Resource


async def notify_on_upload(
    request: web.Request, resource: Resource, file_path: Path,
) -> None:
    redis = request.config_dict["redis"]
    await redis.rpush("uploaded_files", resource.file_name)


app = setup_tus(
    web.Application(),
    upload_path=Path(__file__).parent.parent / "uploads",
    on_upload_done=notify_on_upload,
)
```

8.1.7 Mutliple TUS upload URLs

It is possible to setup multiple TUS upload URLs. Example below illustrates, how to achieve anonymous & authenticated uploads in same time for one `aiohttp.web.Application` instance.

```
app = web.Application()
base_upload_path = Path(__file__).parent.parent / "uploads"

# Anonymous users uploads
setup_tus(app, upload_path=base_upload_path / "anonymous")

# Authenticated users uploads
setup_tus(
    app,
    upload_path=base_upload_path / r"{username}",
    upload_url=r"/users/{username}/uploads",
    decorator=upload_user_required,
)
```

8.1.8 Upload resource name

In most cases there is no need to specify `aiohttp.web.Resource` name for upload resource, but when it is necessary, it is possible to specify custom `upload_resource_name` and lately use it for URL reversing.

Example below illustrates how to achieve it,

In `app.py`,

```
setup_tus(
    web.Application(),
    upload_path=(
        Path(__file__).parent.parent / "uploads" / r"{username}"
    ),
    upload_url="/user/{username}/uploads",
```

(continues on next page)

(continued from previous page)

```
    upload_resource_name="user_upload",
)
```

In `views.py`,

```
async def user_profile(request: web.Request) -> web.Response:
    upload_url = request.app.router["uploads"].url_for(
        username=request.match_info["username"]
    )
    return aiohttp_jinja2.render(
        "users/profile.html",
        request,
        {"upload_url": upload_url},
    )
```

8.2 API Reference

8.2.1 aiohttp_tus

aiohttp_tus.**setup_tus**(*app*, ***, *upload_path*, *upload_url='/uploads'*, *upload_resource_name=None*, *allow_overwrite_files=False*, *decorator=None*, *on_upload_done=None*, *json_dumps=<function dumps>*, *json_loads=<function loads>*)

Setup tus protocol server implementation for aiohttp.web application.

It is a cornerstone of `aiohttp-tus` library and in most cases only thing developers need to know for setting up tus.io server for aiohttp.web application.

Parameters

- **app** (`Application`) – `aiohttp.web.Application` instance
- **upload_path** (`Path`) – `pathlib.Path` instance to point the directory where to store uploaded files. Please, esnure that given directory is exists before application start and is writeable for current user.

  It is possible to prepend any `match_info` param from named URL.
- **upload_url** (`str`) – tus.io upload URL. Can be plain as `/uploads` or named as `/users/{username}/uploads`. By default: `"/uploads"`
- **upload_resource_name** (`Optional[str]`) – By default `aiohttp-tus` will provide auto name for the upload resource, as well as for the chunk resource. But sometimes it might be useful to provide exact name, which can lately be used for URL reversing.
- **allow_overwrite_files** (`bool`) – When enabled allow to overwrite already uploaded files. This may harm consistency of stored data, cause please use this param with caution. By default: `False`
- **decorator** (`Optional[Callable[[Callable[[Request], Awaitable[StreamResponse]]], Callable[[Request], Awaitable[StreamResponse]]]]`) – In case of guarding upload views it might be useful to decorate them with given decorator function. By default: `None` (which means **ANY** client will able to upload files)
- **on_upload_done** (`Optional[Callable[[Request, Resource, Path], Awaitable[None]]]`) – Coroutine to call after upload is done. Coroutine will receive three arguments: `request`, `resource` & `file_path`. Request is current

aiohttp.web.Request instance. Resource will contain all data about uploaded resource such as file name, file size (*aiohttp_tus.data.Resource* instance). While file path will contain pathlib.Path instance of uploaded file.

- **json_dumps** (Callable[[Any], str]) – To store resource metadata between chunk uploads aiohttp-tus using JSON files, stored into upload_path / ".metadata" directory.

  To dump the data builtin Python function used: json.dumps(), but you might customize things if interested in using ujson, orjson, rapidjson or other implementation.

- **json_loads** (Callable[[str], Any]) – Similarly to json_dumps, but for loading data from JSON metadata files. By default: json.loads()

Return type Application

8.2.2 aiohttp_tus.data

class aiohttp_tus.data.**Resource**(*file_name*, *file_size*, *offset*, *metadata_header*, *uid=NOTHING*)

Dataclass to store resource metadata.

Given dataclass used internally in between resource chunk uploads and is passed to on_upload_done callback if one is defined at *aiohttp_tus.setup_tus()* call.

Parameters

- **uid** (str) – Resource UUID. By default: str(uuid.uuid4())
- **file_name** (str) – Resource file name.
- **file_size** (int) – Resource file size.
- **offset** (int) – Current resource offset.
- **metadata_header** (str) – Metadata header sent on initiating resource upload.

8.3 Authors & Contributors

aiohttp-tus is a @okumy project, which never happened without its authors & contributors, who listed below.

- Mikhail Kashkin
- Igor Davydenko
- Alwin Wang

8.4 Changelog

8.4.1 1.1.0 (2022-01-04)

- Fix that support aiohttp >3.8

8.4.2 1.0.0 (2020-06-08)

- Final **1.0.0** release, which marks `aiohttp-tus` as library ready to be used in production

8.4.3 1.0.0rc1 (2020-04-02)

- Fix upload large files via multiple chunks

8.4.4 1.0.0rc0 (2020-03-26)

- Add example to ensure that upload via Uppy JavaScript library works as expected
- Fix resuming uploads by passing missed `Upload-Length` header: #5
- Add documentation about CORS Headers
- Allow to provide upload resource name, which can be lately used for URL reversing

8.4.5 1.0.0b2 (2020-03-18)

- Ensure trailing slash upload URLs working as well

8.4.6 1.0.0b1 (2020-03-18)

- Add brief documentation
- Use canonical upload URL for tus config mapping

8.4.7 1.0.0b0 (2020-03-15)

- Allow to setup tus upload URLs multiple times for one `aiohttp.web` application
- Allow to call callback after upload is done
- Provide many unit tests for tus views

8.4.8 1.0.0a1 (2020-03-12)

- Allow to decorate upload views for authentication or other (for example *to check whether entity for upload exists or not*) needs
- Allow to upload on named upload paths, when using named upload URLs
- Ensure named upload URLs (e.g. `/user/{username}/uploads`) works as well
- Ensure package is typed by adding `py.typed`

8.4.9 1.0.0a0 (2020-03-11)

- First public release with minimal valuable coverage of `tus.io` protocol for `aiohttp.web` applications

INDEX

R

Resource (*class in aiohttp_tus.data*), 22

S

setup_tus() (*in module aiohttp_tus*), 21