Django Twined

Release x.y.unknown

Octue Ltd

Jul 14, 2023

CONTENTS

Attention: This library is in very early stages. Like the idea of it? Please star us on GitHub and contribute via the issues board and the twined roadmap.

django-twined helps run data services based on the twined framework from your own django server.

If you're a scientist or engineer getting started with creating online data services, here is definitely NOT the right place to start! Check out the documentation for twined and the example app templates in the SDK.

> *"Twined" [t-why-nd] ~ encircled, twisted together, interwoven*

CHAPTER

ONE

AIMS

This is an installable app for django, that allows management of octue-based services from django.

This is great for advanced use cases where:

- you have specific security/firewalling requirements, or
- you want to manage your own auth, or
- you have specific/unusual data integration needs, or
- you have a pre-existing django-based web app and want to connect it into the twined ecosystem
- you want to run your apps on a cluster, and provide a single entrypoint for external services to connect

Health warning: to use this plugin to deploy your twined apps, you'll need to handle all your own data storage/orchestration, devops, server management, security and auth. Contact Octue if this doesn't sound like your bag - we can help!

CHAPTER

TWO

RAISON D'ETRE

To help scientists and engineers solve the crisis. More here.

CHAPTER

THREE

CONTENTS

3.1 Installation

3.1.1 Install the library

django-twined is available on pypi, so installation into your python virtual environment is dead simple:

```
poetry add django-twined
```

Not using poetry yet? You definitely should, there's a small learning curve then it removes a world of pip agony :)

3.1.2 Install the django app

You'll need to install `django_twined`, `django_gcp` and `jsoneditor` as apps in your django settings:

```
INSTALLED_APPS = [
    # ...
    'django_gcp',  # For event handlers and flexible storages
    'django_twined',
    'jsoneditor',  # For editing JSON in modeladmin views
    # ...
]
```

Tip: You can use django-gcp for your media/static storage, event handlers and task queues too!

3.1.3 Add the services endpoint

Include the django-twined URLs in your `your_app/urls.py`:

```
from django.urls import include, re_path

urlpatterns = [
   # ...other routes
   # Use whatever regex you want:
   re_path(r"^integrations/octue/", include("django_twined.urls")),
]
```

Using `python manage.py show_urls` you can now see the endpoint for registering and getting service revisions appear in your app.

> **Warning:** The registry URLs are CSRF-exempt by default to allow automatic service revision registration from, for example, GitHub Actions. If you don't want this behaviour, you can import the view at `django_twined.views.service_revision` and use it in your URL patterns as you like.

3.1.4 Run migrations

Then run `python manage.py migrate django_twined` to add the models used for managing services, events and questions to your database.

3.1.5 Add the base URL

Finally, make sure the `BASE_URL` setting is present in `settings.py` - it's used to create absolute URLs for webhooks.

```
BASE_URL = "https://your-server.com"
```

3.2 Quick Start

> **Attention:** LIBRARY IS UNSTABLE! WATCH THIS SPACE!
>
> We suggest at this point you don't try to use this yourself; contact Octue for support and we'll help you out.

3.3 Running a service registry

Once the *services endpoint* has been added, your app can be used as a service registry - ie service revisions can be registered and requested from it.

3.3.1 Registering a service revision

To register a service revision:

```
import requests

response = requests.post(
    "<base_url>/<chosen_path_for_django_twined_urls>/services/<namespace>/<name>",
    json={"revision_tag": "<revision_tag>"},
)
```

For example, if your base URL is `myapp.org/api`, you've registered the `django-twined` URLs under `integrations/octue`, and the service revision you want to register is `my-org/my-service:1.2.9`, the request would be:

```
import requests

response = requests.post(
    "https://myapp.org/api/integrations/octue/services/my-org/my-service",
    json={"revision_tag": "1.2.9"},
)
```

Tip: To override the registry deciding if the service revision being registered should be set as the default (see below), add the `"is_default"` key to the request body and set it to either `True` or `False`.

3.3.2 Getting the default service revision

You can request the default service revision by not specifying a revision tag. By default, the service revision with the latest semantic version revision tag will be returned.

```
import requests

response = requests.get(
    "https://myapp.org/api/integrations/octue/services/my-org/my-service",
)

response.json()
>>> {
    "namespace": "my-org",
    "name": "my-service",
    "revision_tag": "1.2.9",
    "is_default": True,
}
```

Tip: If you know the exact revision you want to use, you can still fetch further information for it.

```
import requests

response = requests.get(
    "https://myapp.org/api/integrations/octue/services/my-org/my-service"
    "?revision_tag=1.2.9",
)

response.json()
>>> {
    "namespace": "my-org",
    "name": "my-service",
    "revision_tag": "1.2.9",
    "is_default": True,
}
```

Currently, the only useful information this provides is whether the requested service revision is the default or not. Later, more useful information will be returned (eg how to send a question to that specific service revision and access tokens to do so).

3.3.3 Controlling whether a service revision is set as the default at registration

The `TWINED_SERVICE_REVISION_IS_DEFAULT_CALLBACK` setting can be set to a user-defined callable to control whether a service revision is set as the default for its service during registration. The callable must take one argument, `service_revision` (an instance of the `ServiceRevision` model), and return a boolean indicating whether the revision should be set as the default. The default callable sets the service revision as the default if its revision tag is the latest semantic version for the service.

Examples of how this feature can be used include:

- A/B testing
- Controlling the availability of beta versions of services
- Other custom selection of service revisions

Click here to see the default callable as an example.

3.4 Settings

Name	Type	Description
`TWINED_BASE_URL`	str	The server address for generating absolute webhook URLs, eg `"https://api.you.com"`
`TWINED_DEFAULT_NAMESPACE`	str	The namespace used by default (if none specified) when creating Service Revisions. Typically your organisation, and should be in kebab case, eg `"mega-corp"`.
`TWINED_DEFAULT_PROJECT_NAME`	str	The GCP project name used by default (if none specified) when creating Service Revisions. This is the project where the default-namespace services reside. Often (but not necessarily), this is the same as the namespace eg `"mega-corp"`.
`TWINED_DEFAULT_TAG`	str	The tag used by default (if none specified) when creating new Service Revisions. `"latest"` is used if not specified.
`TWINED_SERVICES`	dict	DEPRECATED - DO NOT USE. The `ServiceRevision` model replaces the outgoing `RegisteredService` model, allows update of the parameters specified here, without rebooting django.
`TWINED_DATA_STORES`	dict	A dictionary defining one or more Data Stores, which map a database table (django Model) to a bucket on GCP, syncing metadata between the files in the bucket and filterable / searchable columns in teh DB table.
`TWINED_SERVICE_REVISION_IS_DEFAULT_CALLBACK`	callable	A function that takes one argument, `service_revision`, which is an instance of the `ServiceRevision` model, and returns a boolean indicating whether the revision should be set as the default during service revision registration. The default callable sets a service revision as the default if its revision tag is the latest semantic version for the service.

3.5 Examples

Here, we look at example use cases for the library, and show how to use it in python.

It's also well worth looking at the unit test cases.

3.5.1 Tabs with different examples

Scenario

You need to provide stuff

Tab2

We need tabs!

```
{
    "thats": "right",
    "code": "goes in tabs too",
}
```

3.6 License

3.6.1 The Boring Bit

See the django-twined license.

3.6.2 Third Party Libraries

django-twined includes or is linked against code from third party libraries, see our attributions page.

3.7 Version History

3.7.1 Origins

django-twined is the result of a refactor - Octue is progressively open-sourcing our stack for managing and connecting digital twins and data services.

As you do when you're a bootstrapped startup, we build our MVP as a massive monolithic django app, but wanted to reuse parts of it in applications for clients. Copying/pasting code never makes sense, so here we are!

A huge thank you to Wind Pioneers whose business made the initial refactor possible, and who now get to use twined to build the most kick-ass Wind Energy Resource Assessment applications in the world!

..ATTENTION:

```
**django-twined** will be unstable in 0.x versions. Consider every semiver increment to␣
↪be breaking!
```

3.7.2 Releases

We create release notes automatically using our conventional commits tools for completely automating code versions, release numbering and release history.

So for a full version history, check our releases page.