Seed Scheduler Documentation

Release 0.9.12

Praekelt.org

Jun 25, 2018

Contents

CHAPTER 1

Seed Scheduler documentation

The Seed Scheduler is one of the microservices in the Seed Stack.

The Scheduler has the following key responsibilities:

- Allow other services to create scheduled callbacks via the API.
- At the scheduled time, call the configured URLs with the configured payload.

Getting started

The following resources are provided to help you get started running or developing the Seed Scheduler:

- Learn about the *Requirements* for running the service and basic *Setup* instructions.
- Read about the *Data Models* used by the service.
- Read about the *Authorization* requirements.
- Browse the *API Documentation* for the available endpoints and parameters.
- Learn about what is required when running the service in *Production*

How the task scheduling works

The Seed Scheduler uses Celery and the django-celery integration package to handle the scheduling of the periodic tasks.

The schedules are kept in tables in the PostgreSQL database that a Celery beat worker processes and then hands off actual task execution to another Celery worker process.

When a new Schedule object is created via the *Seed Scheduler API*, a corresponding django-celery PeriodicTask object is created for that schedule.

CHAPTER 2

Requirements

Overview

The Seed Scheduler requires the following dependencies to run:

- Python 2.7
- PostgreSQL >= 9.3
- Redis >= 2.10 or RabbitMQ >= 3.4 as the Celery Broker

Python requirements

The full list of Python packages required are detailed in the project's setup.py file, but the major ones are:

- Django 1.9
- Django REST Framework 3.3
- Celery 3.1

Note: A celery worker needs to be running to process post-save tasks and scheduled metric firing tasks.

Seed Requirements

The Seed Scheduler only depends on one other seed service, the Go Metrics API.

CHAPTER 3

Setup

Installing

The steps required to install the Seed Scheduler Service are:

1. Get the code from the Github Project with git:

   ```
   $ git clone https://github.com/praekelt/seed-scheduler.git
   ```

   This will create a directory `seed-scheduler` in your current directory.

1. Install the Python requirements with pip:

   ```
   $ pip install -r requirements.txt
   ```

   This will download and install all the Python packages required to run the project.

2. Setup the database:

   ```
   $ python manage migrate
   ```

   This will create all the database tables required.

   Note: The PostgreSQL database for the Seed Scheduler needs to exist before running this command. See *SCHEDULER_DATABASE* for details.

3. Run the development server:

   ```
   $ python manage.py runserver
   ```

   Note: This will run a development HTTP server. This is only suitable for testing and development, for production usage please see *Running in Production*

Configuration Options

The main configuration file is `seed_scheduler/settings.py`.

The following environmental variables can be used to override some default settings:

SECRET_KEY
This overrides the Django SECRET_KEY setting.

DEBUG
This overrides the Django DEBUG setting.

USE_SSL
Whether to use SSL when build absolute URLs. Defaults to False.

SCHEDULER_DATABASE
The database parameters to use as a URL in the format specified by the DJ-Database-URL format.

SCHEDULER_SENTRY_DSN
The DSN to the Sentry instance you would like to log errors to.

HOOK_AUTH_TOKEN
The token to use when posting to webhooks.

BROKER_URL
The Broker URL to use with Celery.

METRICS_URL
The URL to the Go Metrics API instance to push metrics to.

METRICS_AUTH_TOKEN
The *auth token* to use to connect to the Go Metrics API above.

CHAPTER 4

Data Models

Schedule

Fields

id A UUID 4 unique identifier for the record.

frequency (Deprecated) An optional integer number of times a task should be run in total.

triggered (Deprecated) An integer representing the number of times a task has been run.

cron_definition A character based representation of the schedule in cron format.

celery_cron_definition A reference to the Celery crontab schedule.

interval_definition An character based representation of the schedule in interval format. Given as an integer and period (from: days, hours, minutes, seconds, microseconds) e.g. *1 minute*.

celery_interval_definition A reference to the Celery interval schedule.

endpoint The URL to POST to when the task is run.

auth_token The auth token to use when POSTing to the *endpoint*.

payload The payload to use as the POST body to the *endpoint*.

next_send_at An rough estimate of when the next run is scheduled.

enabled A boolean enabled flag.

created_at A date and time field of when the record was created.

updated_at A date and time field of when the record was last updated.

created_by A reference to the User account that created this record.

updated_by A reference to the User account that last updated this record.

CHAPTER 5

Authentication and Authorization

Basics

Authentication to the Seed Scheduler API is provided the Token Authentication feature of the Django REST Framework.

In short, each user of this API needs have been supplied a unique secret token that must be provided in the `Authorization` HTTP header of every request made to this API.

An example request with the `Authorization` header might look like this:

```
POST /endpoint/ HTTP/1.1
Host: <scheduler-domain>
Content-Type: application/json
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Users and Groups

User and *Group* objects are provided by the Django Auth framework and can be added and created through the normal maintenance methods (Django Admin, Dgango Shell, ...).

There is also a rudimentary API endpoint: *`POST /user/token/`* that will create a user and token for a given email address (or just a token if a user with that email address already exists).

Authorization and permissions

All of the current API endpoints do not require any specific permissions other than a valid authenticated user.

The only exception to this is *`POST /user/token/`* which requires an admin level user.

CHAPTER 6

API Details

The Scheduler provides REST like API with JSON payloads.

The root URL for all of the endpoints is:

```
https://<scheduler-domain>/api/
```

Authenticating to the API

Please see the *Authentication and Authorization* document.

Pagination

When the results set is larger than a configured amount, the data is broken up into pages using the limit and offset parameters.

Paginated endpoints will provide information about the total amount of items available along with links to the previous and next pages (where available) in the returned JSON data.

GET / (*any*) **/**

Query Parameters

- **limit** – the amount of record to limit a page of results to.
- **offset** – the starting position of the query in relation to the complete set of unpaginated items

Response JSON Object

- **count** (*int*) – the total number of results available
- **previous** (*string*) – the URL to the previous page of results (if available)
- **next** (*string*) – the URL to the next page of results (if available)

Example request:

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

{
    "count": 50,
```

```
    "next": "http://sbm.example.org/api/v1/enpoint/?limit=10&offset=30",
    "previous": "http://smb.example.org/api/v1/endpoint/?limit=10&offset=10",
    "results": []
}
```

Endpoints

The endpoints provided by the Seed Scheduler are split into two categories, core endpoints and helper endpoints

Core

The root URL for all of the core endpoints includes the version prefix (`https://<scheduler-domain>/api/v1/`)

`POST /user/token/`

Creates a user and token for the given email address.

If a user already exists for the given email address, the existing user account is used to generate a new token.

Request JSON Object

- **`email`** (*`string`*) – the email address of the user to create or use.

Response JSON Object

- **`token`** (*`string`*) – the auth token generated for the given user.

Status Codes

- 201 Created – token successfully created.
- 400 Bad Request – an email address was not provided or was invalid.
- 401 Unauthorized – the token is invalid/missing.

Example request:

```
POST /user/token/ HTTP/1.1
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b

{
    "email": "bob@example.org"
}
```

Example response:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "token": "c05fbab6d5f912429052830c77eeb022249324cb"
}
```

Users and Groups

`GET /user/`

Returns a list of users for the Seed Scheduler service.

Status Codes

- 200 OK – no error.
- 401 Unauthorized – the token is invalid/missing.

Example request:

```
GET /user/ HTTP/1.1
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Example response:

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

{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "email": "john@example.org",
            "groups": [],
            "url": "http://scheduler.example.org/api/v1/user/1/",
            "username": "john"
        }
    ]
}
```

GET /user/ (**int:** *user_id*) **/**
Returns the details of the specified user ID.

Parameters

- **user_id** (*int*) – a user's unique ID.

Status Codes

- 200 OK – no error.
- 401 Unauthorized – the token is invalid/missing.

Example request:

```
GET /user/1/ HTTP/1.1
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Example response:

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

{
    "email": "john@example.org",
    "groups": [],
    "url": "http://scheduler.example.org/api/v1/user/1/",
    "username": "john"
}
```

GET /group/
Returns a list of groups for the Seed Scheduler service.

Status Codes

- 200 OK – no error
- 401 Unauthorized – the token is invalid/missing.

Example request:

```
GET /group/ HTTP/1.1
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Example response:

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

{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "name": "Admins",
            "url": "http://scheduler.example.org/api/v1/group/1/"
        }
    ]
}
```

GET /group/ (**int:** *group_id*) **/**
Returns the details of the specified group ID.

Parameters

- **group_id** (*int*) – a group's unique ID.

Status Codes

- 200 OK – no error.
- 401 Unauthorized – the token is invalid/missing.

Example request:

```
GET /group/1/ HTTP/1.1
Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
```

Example response:

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

{
    "name": "Admins",
    "url": "http://scheduler.example.org/api/v1/group/1/"
}
```

Schedules

GET /schedule/
Returns a list of Schedules.

POST /schedule/
Creates a new Schedule.

Request JSON Object

- **frequency** (*int*) – (Deprecated) an optional integer number of times a task should be run in total.
- **endpoint** (*string*) – a URL to POST to when this schedule is run.
- **cron_definition** (*string*) – A crontab definition of when to run this schedule.
- **inteval_definition** (*string*) – An interval definition of when to run this schedule.
- **auth_token** (*string*) – An auth token to use when POSTing to the endpoint.
- **payload** (*json*) – The JSON payload to include when POSTing to the endpoint.
- **enabled** (*boolean*) – A boolean flag of whether this schedule is enabled.

Response Headers

- Location – the URL to the newly created resource.

Status Codes

- 201 Created – created.
- 400 Bad Request – invalid data.
- 401 Unauthorized – the token is invalid/missing.

GET /schedule/(**uuid:** *schedule_id*)**/**
Retuns the Schedule record for a given schedule_id.

PUT /schedule/(**uuid:** *schedule_id*)**/**
Updates the Schedule record for a given schedule_id.

Request JSON Object

- **frequency** (*int*) – an optional integer number of times a task should be run in total.
- **endpoint** (*string*) – a URL to POST to when this schedule is run.
- **cron_definition** (*string*) – A crontab definition of when to run this schedule.
- **inteval_definition** (*string*) – An interval definition of when to run this schedule.
- **auth_token** (*string*) – An auth token to use when POSTing to the endpoint.
- **payload** (*json*) – The JSON payload to include when POSTing to the endpoint.
- **enabled** (*boolean*) – A boolean flag of whether this schedule is enabled.

Status Codes

- 200 OK – updated.
- 400 Bad Request – invalid data.
- 401 Unauthorized – the token is invalid/missing.

DELETE /schedule/(**uuid:** *schedule_id*)**/**
Deletes the Schedule record for a given schedule_id.

Helpers

The root URL for the helper endpoints does not include a version prefix (`https://<scheduler-domain>/api/`)

`GET /metrics/`
Returns a list of all the available metric keys provided by this service.

Status Codes

- 200 OK – no error
- 401 Unauthorized – the token is invalid/missing.

`POST /metrics/`
Starts a task that fires all scheduled metrics.

Status Codes

- 200 OK – no error
- 401 Unauthorized – the token is invalid/missing.

`GET /health/`
Returns a basic health check status.

Status Codes

- 200 OK – no error
- 401 Unauthorized – the token is invalid/missing.

CHAPTER 7

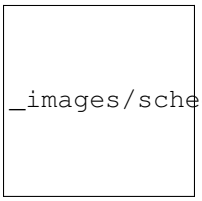
Production requirements and setup

Running in Production

The Seed Scheduler is expected to be run in a Docker container and as such a Docker file is provided in the source code repository.

The web service portion and celery work portion of the Scheduler are expected to be run in different instances of the same Docker container.

An example production setup might look like this:



CHAPTER 8

Indices, glossary and tables

- genindex
- modindex
- Glossary

/group

/schedule

/user

E

S