Donations Documentation

Release 0.1.1

Top Free Games

February 07, 2017

Contents

Contents:

CHAPTER 1

Overview

What is Donations? Donations is an HTTP "resty" API for managing donations of items in games.

Donations allows your app to focus on the interaction required to donating items and keeping players engaged, instead of the backend required for actually doing it.

1.1 Features

- **Multi-tenant** - Donations already works for as many games as you need, just keep adding new games;
- **Item Donation Management** - Donate items to other players and request items from them in return;
- **New Relic Support** - Natively support new relic with segments in each API route for easy detection of bottlenecks;
- **Easy to deploy** - Donations comes with containers already exported to docker hub for every single of our successful builds. Just pick your choice!

1.2 Architecture

Donations is based on the premise that you have a backend server for your game. That means we do not employ any means of authentication.

There's no validation if the actions you are performing are valid as well. We have TONS of validation around the operations themselves being valid.

What we don't have are validations that test whether the source of the request can perform the request (remember the authentication bit?).

1.3 The Stack

For the devs out there, our code is in Go, but more specifically:

- Web Framework - Echo based on the insanely fast FastHTTP;
- Database - MongoDB;
- Locks - Redis.

1.4 Who's Using it

Well, right now, only us at TFG Co, are using it, but it would be great to get a community around the project. Hope to hear from you guys soon!

1.5 How To Contribute?

Just the usual: Fork, Hack, Pull Request. Rinse and Repeat. Also don't forget to include tests and docs (we are very fond of both).

CHAPTER 2

Installing Donations

TBW.

CHAPTER 3

Hosting Donations

There are three ways to host Donations: docker, binaries or from source.

3.1 Docker

Running Donations with docker is rather simple. Our docker container image comes bundled with the API binary. All you need to do is load balance all the containers and you're good to go. The API runs at port `8080` in the docker image.

Donations uses MongoDB to store clans information. The container takes environment variables to specify this connection:

- `DONATIONS_MONGO_HOST` - MongoDB host to connect to;
- `DONATIONS_MONGO_PORT` - MongoDB port to connect to;
- `DONATIONS_MONGO_DB` - Database name of the MongoDB Server to connect to.

Donations uses Redis for global locks. The container takes environment variables to specify this connection:

- `DONATIONS_REDIS_URL` - Redis URL to connect to;
- `DONATIONS_REDIS_MAXIDLE` - Max Idle connection to Redis;
- `DONATIONS_REDIS_IDLETIMEOUTSECONDS` - Number of seconds to consider a connection idle.

Other than that, there are a couple more configurations you can pass using environment variables:

- `DONATIONS_NEWRELIC_KEY` - If you have a New Relic account, you can use this variable to specify your API Key to populate data with New Relic API;
- `DONATIONS_NEWRELIC_APPNAME` - If you have a New Relic account, you can use this variable to specify the name of the application to use in your New Relic dashboard;
- `DONATIONS_SENTRY_URL` - If you have a sentry server you can use this variable to specify your project's URL to send errors to.

If you want to expose Donations outside your internal network it's advised to use Basic Authentication. You can specify basic authentication parameters with the following environment variables:

- `DONATIONS_BASICAUTH_USERNAME` - If you specify this key, Donations will be configured to use basic auth with this user;
- `DONATIONS_BASICAUTH_PASSWORD` - If you specify `BASICAUTH_USERNAME`, Donations will be configured to use basic auth with this password.

3.1.1 Example command for running with Docker

```
$ docker pull tfgco/donations
$ docker run -t --rm -e "DONATIONS_MONGO_HOST=<mongoDB host>" -e "DONATIONS_MONGO_PORT=<mongoDB p
```

3.2 Binaries

Whenever we publish a new version of Donations, we'll always supply binaries for both Linux and Darwin, on i386 and x86_64 architectures. If you'd rather run your own servers instead of containers, just use the binaries that match your platform and architecture.

The API server is the `donations` binary. It takes a configuration yaml file that specifies the connection to MongoDB and some additional parameters. You can learn more about it at default.yaml.

3.3 Source

Left as an exercise to the reader.

CHAPTER 4

Game Configuration

Being a multi-tenant server, Donations allows for many different configurations per tenant. Each tenant is a different game and is identified by it's game ID.

Before any operation can be performed, you must create a game in Donations. The good news here is that updating games are idempotent operations. You can keep executing it any time your game changes. That's ideal to be executed in a deploy script, for instance.

4.1 Creating/Updating a Game

The `Update` operation of the Game is idempotent (you can run it as many times as you want with the same result). If your game does not exist yet, it will create it, otherwise just updated it with the new configurations.

To Create/Update your game, just do a `PUT` request to `http://my-donations-server/games/my-game-public-id`, where `my-game-public-id` is the ID you'll be using for all your game's operations in the future. The payload for the request is a JSON object in the body and should be as follows:

```
{
  "name":                          [string],
  "donationCooldownHours":          [int],
  "donationRequestCooldownHours":   [int]
}
```

If the operation is successful, you'll receive a JSON object with the game details.

4.2 Game Configuration Settings

As can be seen from the previous section, there are some configurations you can do per game. These will be thoroughly explained in this section.

4.2.1 name

The name of your game. This is used mainly for easier reasoning of what this game is when debugging.

Type: `string` **Sample Value**: `My Sample Game`

4.2.2 donationCooldownHours

Number of hours that must ellapse before a player can donate again. This is used in conjunction with the max donation weight passed in with the donation.

Type: `Integer` **Sample Value**: `24`

4.2.3 donationRequestCooldownHours

Number of hours a player must wait before requesting donations again.

Type: `Int` **Sample Value**: `8`

4.3 Game Items

In order to use donations, the items that can be donated must be previously created for that specific game.

As with the game, creating/updating items can be done idempotently, thus allowing for easier maintenance of a game's donatable items.

4.3.1 Creating/Updating an Item

The `Update` operation of the Item is idempotent (you can run it as many times as you want with the same result). If your item does not exist yet, it will create it, otherwise just updated it with the new configurations.

To Create/Update your item, just do a `PUT` request to `http://my-donations-server/games/my-game-public-id/items` where `my-game-public-id` is the ID of the game you registered in the previous step and `my-item-key` is the item key you'll be using in your game's donation requests and donations in the future. The payload for the request is a JSON object in the body and should be as follows:

```
{
  "metadata":                           [JSON],
  "weightPerDonation":                  [int],
  "limitOfItemsPerPlayerDonation":      [int],
  "limitOfItemsInEachDonationRequest":  [int]
}
```

If the operation is successful, you'll receive a JSON object with the item details.

4.3.2 Item Configuration Settings

As can be seen from the previous section, there are some configurations you can do per item. These will be thoroughly explained in this section.

4.3.3 metadata

This sections allow you to store metadata for this item.

Donations treats this as a black box and won't use it for any operations. It will be returned whenever the item is returned, though.

Type: `JSON` **Sample Value**: `{ "cost": 100, "currency": "gold" }`

4.3.4 weightPerDonation

Donations keep track of the weight that has been donated by player in case you want to limit the max weight donated per player per time.

This configuration allows you to set different weights per donation per item (epic donations weigh more than rare and so on).

Type: `int` **Sample Value**: `3`

4.3.5 limitOfItemsPerPlayerDonation

The limit of items that can be donated by a single player for this item in a single donation request.

Type: `int` **Sample Value**: `3`

4.3.6 limitOfItemsInEachDonationRequest

The amount of this item that can be donated in a single donation request.

Type: `int` **Sample Value**: `8`

CHAPTER 5

Donations API

5.1 Healthcheck Routes

5.1.1 Healthcheck

`GET /healthcheck`

Validates that the app is still up, including the database connection.

- Success Response
  - Code: `200`
  - Content:

```
"WORKING"
```

- Error Response

  It will return an error if it failed to connect to the database.
  - Code: `500`
  - Content:

```
"Error connecting to database: <error-details>"
```

5.2 Game Routes

5.2.1 Update Game

`PUT /games/:gameID`

Updates the game with that has publicID `gameID`.

- Payload

```
{
  "name":                           [string],  // 2000 characters max
  "donationCooldownHours":          [int],
  "donationRequestCooldownHours":   [int]
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "name":                              [string],  // 2000 characters max
  "donationCooldownHours":             [int],
  "donationRequestCooldownHours":      [int]
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

5.3 Item Routes

5.3.1 Update Item

`PUT /games/:gameID/items/:itemKey`

Updates the item with key `itemKey` in the game with public ID `gameID`.

- Payload

```
{
  "metadata":                             [JSON],
  "weightPerDonation":                    [int],
  "limitOfItemsPerPlayerDonation":        [int],
  "limitOfItemsInEachDonationRequest":    [int]
}
```

- Success Response
  - Code: `200`
  - Content:

```
{
  "metadata":                             [JSON],
  "weightPerDonation":                    [int],
  "limitOfItemsPerPlayerDonation":        [int],
```

```
    "limitOfItemsInEachDonationRequest":  [int]
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

5.4 Donation Request Routes

5.4.1 Create Donation Request

`POST /games/:gameID/donation-requests`

Creates a new donation request for the item specified in the payload.

- Payload

```
{
  "item":      [string],
  "player":    [string],
  "clan":      [string],
}
```

  - `item` is the key for the item to create the donation request for;
  - `player` is the player id that will receive the donations;
  - `clan` is the team/clan/group the player belongs to. This is useful for grouping donations. Leave this empty if player does not belong to a team/clan/group.

- Success Response
  - Code: `200`
  - Content: Serialized donation request.
- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.

  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

- Code: `500`
- Content:

```
{
  "success": false,
  "reason": [string]
}
```

5.5 Donation Routes

5.5.1 Donate to a Donation Request

`POST /games/:gameID/donation-requests/:donationRequestID`

Donates items to a donation request with public ID `donationRequestID` in the game `gameID`.

- Payload

```
{
  "player":                [string],
  "amount":                [string],
  "maxWeightPerPlayer":    [string]
}
```

  - `player` is the player id that will receive the donations;
  - `amount` is the quantity of the item being donated by this player;
  - `maxWeightPerPlayer` is the maximum weight this player can donate per time period.

- Success Response
  - Code: `200`
  - Content:

```
{
  "success": true
}
```

- Error Response

  It will return an error if an invalid payload is sent or if there are missing parameters.
  - Code: `400`
  - Content:

```
{
  "success": false,
  "reason": [string]
}
```

  - Code: `500`

- Content:

```
{
  "success": false,
  "reason": [string]
}
```

CHAPTER 6

Indices and tables

- genindex
- modindex
- search