Santiago Documentation

Release 1.2.0

Top Free Games

November 07, 2016

Contents

Contents:

CHAPTER 1

Overview

What is Santiago? Santiago is a web hook dispatching application.

Why would you need such a thing? Because Santiago tries hard for your Web Hooks to be properly dispatched. It offers an easy-to-use API that allows any of your applications to integrate seamlessly with each other.

Santiago is also very cloud-friendly and comes bundled with docker containers pre-built for both Production use as well as dev usage.

1.1 Getting started

Let's get Santiago up and running in your machine in a few steps. For this quick start it is assumed you have docker available and running.

First let's start a server that will receive our webhook. You can use anything you want for this, but we'll run an echo server in node.js:

```
var http = require('http');

http.createServer(function(request,response){
  response.writeHead(200);
  request.on('data',function(message){
    console.log("RECEIVED " + message);
    response.write(message);
  });

  request.on('end',function(){
    response.end();
  });
}).listen(3000);
```

Now that we got our echo server, let's fire it up:

```
$ node echo.js
```

Echo server is now running at 3000. Then in another terminal, let's test it:

```
$ curl -dHello=World http://localhost:3000/
```

You should see `RECEIVED Hello=World` in the terminal running your echo server.

Now for the actual fun. Let's start our own Santiago server:

```
$ docker pull tfgco/santiago-dev
$ docker run -i -t --rm -p 8080:8080 tfgco/santiago-dev
```

Then let's enqueue a web hook to be dispatched in our Santiago server. For this part you'll need to know your network adapter IP address. You can find it out with this command:

```
$ ifconfig | egrep inet | egrep -v inet6 | egrep -v 127.0.0.1 | awk ' { print $2 } '
```

Now that you know your IP, just replace $IP with your actual IP address:

```
$ curl -dHello=World "http://localhost:8080/hooks?method=POST&url=http%3A//$IP%3A3000/"
```

Once more you should see `RECEIVED Hello=World` in the terminal running Santiago.

When you decide to run your Santiago app in production, please read our [Hosting] docs.

1.2 Features

- **Reliable** - Santiago is very simple and relies on Redis for its queueing system;
- **Delivery Retry** - Santiago will retry up to 10 times to deliver your web hook (configurable ammount);
- **Exponential Back-off** - After a failed web hook attempt, Santiago will wait some time before trying again. The time between attempts grows exponentially to allow you to react to it;
- **Log-Friendly** - We log almost any operation we do in Santiago, so you can easily debug it;
- **Easy to deploy** - Santiago comes with containers already exported to docker hub for every single of our successful builds. Just pick your choice and it should just work;
- **Easily support Sentry** - Set a configuration option to ensure errors get sent to Sentry.

1.3 Architecture

Whenever you add a new web hook to Santiago, it enqueues it with Redis. There are workers running that process this queue and try to send your web hooks.

If the web hook fail, it re-enqueues the message up to a max number of times.

That's pretty much all there's to know about Santiago's architecture. Running redis is out of the scope of this document.

1.4 Using Sentry

In your configuration file, just set `api.sentry.url` to your project's sentry URL. In the worker, pass –sentry-url `my-project-sentry-url` to ensure errors in the worker get sent to Sentry.

1.5 The Stack

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

- Web Framework - echo based on the insanely fast FastHTTP;
- Queueing - Redis.

1.6 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!

1.7 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

Hosting Santiago

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

2.1 Docker

Running santiago with docker is rather simple. Our docker container image already comes bundled with both the API and the Worker. All you need to do is load balance all the containers and you're good to go.

Santiago uses Redis to publish hooks to and to listen for incoming hooks. The container also takes parameters to specify this connection:

- `SNT_API_REDIS_HOST` - Redis host to publish hooks to;
- `SNT_API_REDIS_PORT` - Redis port to publish hooks to;
- `SNT_API_REDIS_PASSWORD` - Password of the Redis Server to listen for hooks;
- `SNT_API_REDIS_DB` - DB Number of the Redis Server to listen for hooks;
- `SNT_API_USE_FAST_HTTP` - Whether to use fasthttp for echo engine or not. This env should be either "–fast" or "".
- `SNT_NEWRELIC_KEY` - New Relic account key. If present will enable New Relic.

2.2 Binaries

Whenever we publish a new version of Santiago, we'll always supply binaries for both Linux and Darwing, 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 `snt` binary. It takes a configuration yaml file that specifies the connection to Redis and some additional parameters. You can learn more about it at default.yaml.

The workers are started by the `snt-worker` binary. This one takes all the parameters it needs via console options. To learn what options are available, use `snt-worker -h`. To start a new worker, use `snt-worker start`.

2.3 Source

Left as an exercise to the reader.

CHAPTER 3

Santiago API

3.1 Healthcheck Routes

3.1.1 Healthcheck

`GET /healthcheck`

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

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

```
"WORKING"
```

  - Headers:

    It will add an `KHAN-VERSION` header with the current khan module version.

- Error Response

  It will return an error if it failed to connect to redis.

  - Code: `500`

3.2 Status Routes

3.2.1 Status

`GET /status`

Returns statistics on the health of khan.

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

```
{
  "app": {
    "errorRate": [float]        // Exponentially Weighted Moving Average Error Rate
```

```
        },
        "dispatch": {
          "pendingJobs": [int]        // Pending hook jobs to be sent
        }
      }
```

3.3 WebHook Routes

3.3.1 Dispatch webhook

`POST /hooks?method=GET&url=http://some.server.com/my-webhook&expires=1478401023`

Creates a new webhook to be dispatched. This method takes Method and URL as querystring parameters and the payload to send to the webhook as the body.

An expiration timestamp (Unix Format) may be passed optionally to enforce an expiration for a message (in the case of retrying in a later point in time). This is very useful in the event of messages that only make sense to be sent in a very short period of time, but the system that receives them is down for more than that period.

- Querystring:
  - `method` - HTTP Method to use to call the webhook (GET, POST, etc);
  - `url` - Endpoint of the webhook to be called;
  - `expires` - Unix Timestamp that determines the expiration of this message. If Santiago's worker finds a message with an expiration date lesser than the current date it just ignores the message and it leaves the queue.
- Payload

The body of this request will be sent without modification to the webhook endpoint.

CHAPTER 4

Santiago's Benchmarks

You can see santiago's benchmarks in our CI server as they get run with every build.

4.1 Results

Runnning with Apache Benchmark on a Macbook Pro, with this command:

```
$ ab -n 10000 -c 30 -p ab.data "http://127.0.0.1:3333/hooks?method=POST&url=http%3A//127.0.0.1:3000/h
```

With the ab.data file containing:

```
hello=world
```

The results should be similar to these:

```
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        iris
Server Hostname:        127.0.0.1
Server Port:            3333

Document Path:          /hooks?method=POST&url=http%3A//10.0.23.64:3000/hooks/
Document Length:        2 bytes

Concurrency Level:      30
Time taken for tests:   2.034 seconds
```

```
Complete requests:      10000
Failed requests:        0
Total transferred:      1510000 bytes
Total body sent:        1940000
HTML transferred:       20000 bytes
Requests per second:    4916.58 [#/sec] (mean)
Time per request:       6.102 [ms] (mean)
Time per request:       0.203 [ms] (mean, across all concurrent requests)
Transfer rate:          725.00 [Kbytes/sec] received
                        931.46 kb/s sent
                        1656.47 kb/s total

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    2   0.7      2       9
Processing:     1    4   1.3      4      16
Waiting:        0    4   1.3      4      16
Total:          1    6   1.6      6      18

Percentage of the requests served within a certain time (ms)
  50%      6
  66%      6
  75%      6
  80%      6
  90%      8
  95%      9
  98%     13
  99%     13
 100%     18 (longest request)
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search