# Stashboard Documentation

***Release 1.5.0***

**Twilio**

November 23, 2016

Contents

CHAPTER 1

# About

Stashboard was written by Twilio to provide status information on our phone, SMS, and Communication APIs. We open sourced the code and to provide a generic status page designed to be customized by any hosted services company to provide customers up-to-date status information. The code can be downloaded, customized with your logo, and run on any Google App Engine account.

CHAPTER 2

# User's Guide

A complete guide to installing, running, and developing Stashboard

## 2.1 Getting Started

### 2.1.1 Running Locally

Download and install the App Engine SDK for Python.

You'll then need to checkout the Stashboard repository.

```
git clone git://github.com/twilio/stashboard.git
```

Open the SDK, choose `File > Add Existing Application...` and select the `stashboard` folder inside the cloned repository. Then click the green "Run" button to start the application.

Visit http://localhost:8080/admin/setup to complete the installation.

### 2.1.2 Customizing

Open the `settings.py` file and change the `SITE_NAME`, `SITE_URL`, and `REPORT_URL` options to the desired values.

If you're planning on authenicating using the REST API, you need to register your application with Google to get your `CONSUMER_KEY` and `CONSUMER_SECRET` values.

### 2.1.3 Configuring Twitter Updates

To configure posting to Twitter, you must create an application and follow the instructions for generating access tokens for a single user.

1. Create a Twitter account for posting status messages.
2. Log into dev.twitter.com with the account you want to use for posting statuses.
3. Create an application with Read and Write permissions.
4. Visit your application's page and create an access token.
5. Update `settings.py` with your Twitter username, consumer key, consumer secret, access token, and access token secret.

### 2.1.4 Deploying

Before you can deploy to Appspot, you'll need to create an application on App Engine. Once you've done that, update the `app.yaml` with your application id.

Hit the "Deploy" button, wait a couple of seconds, and then naviagate to `http://{application}.appspot.com` to enjoy your new status dashboard.

## 2.2 Community

Can't seem to get Stashboard up and running? We're always here to help

### 2.2.1 Google Group

All Stashboard development and discussion happens in the Stashboard google group

### 2.2.2 IRC

Join the #stashboard channel on freenode

### 2.2.3 Twitter

To keep up to date, you can follow @stashboard on Twitter

### 2.2.4 Wiki and Issue Tracker

Hosted on the stashboard Github repository.

## 2.3 REST API Documentation

The Stashboard REST API is split in two portions. The public facing REST API only responds to GET and lives at the `/api/v1/` endpoint. This API requires no authentication.

The admin-only REST API lives at the `/admin/api/v1/` endpoint and responsds to GET, POST, PUT, and DELETE. You'll need to authenticate via OAuth. You can obtain your OAuth keys on the OAuth Credentials page at `https://{application-id}.appspot.com/admin/credentials`

### 2.3.1 Services

The Services resource represents all web services currently tracked via Stashboard.

| Property | Description |
|---|---|
| id | The unique identifier by which to identify the service |
| name | The name of the service, defined by the user |
| description | The description of the web service |
| current-event | The current event for the service |
| url | The URL of the specific service resource |

### List Resource

```
/admin/api/v1/services
```

#### GET

Returns a list of all current services tracked by Stashboard

```
GET /admin/api/v1/services HTTP/1.1
```

```
{
    "services": [
        {
            "name": "Example Foo",
            "id": "example-foo",
            "description": "An explanation of this service"
            "url": "/api/v1/services/example-foo",
            "current-event": {
                'message': 'What an event!',
                'sid': 'ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M',
                'status': {
                    'description': 'Hey, dude',
                    'id': 'up',
                    'image': '/images/status/tick-circle.png',
                    'level': 'NORMAL',
                    'name': 'Up',
                    'url': '/statuses/up'
        },
                'timestamp': 'Mon, 28 Jun 2010 22:17:06 GMT',
                'url': '/services/twilio/events/ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M'},

        },
        {
            "name": "Example Bar",
            "id": "example-bar",
            "description": "An explanation of this service"
            "url": "/api/v1/services/example-bar",
            "current-event": null,
        }
    ]
}
```

#### POST

Creates a new service (or updates an existing service) and returns the new service object.

| Param | Description |
|---|---|
| name | Name of the service |
| description | Description of service |

```
POST /admin/api/v1/services HTTP/1.1 name=New%20Service&description=A%20great%20service
```

```
{
    "name": "New Service",
    "id": "new-service",
    "description": "A great service"
```

```
    "url": "/api/v1/services/new-service",
    "current-event": null,
}
```

### Instance Resource

```
/admin/api/v1/services/{service}
```

The Service Instance resources represents an individual web service tracked by StashBoard

#### GET

```
GET /admin/api/v1/services/{service} HTTP/1.1
```

```
{
    "name": "Example Service",
    "id": "example-service",
    "description": "An explanation of what this service represents"
    "url": "/api/v1/services/example-service",
    "current-event": null,
}
```

#### POST

Updates a service's description and returns the updated service object. All the listed parameters are optional.

| Param | Description |
|---|---|
| name | Name of the service |
| description | Description of service |

```
POST /admin/api/v1/services/{service} description=System%20is%20now%20operational
```

```
{
    "name": "Example Service",
    "id": "example-service",
    "description": "System is now operational",
    "url": "/api/v1/services/example-service",
    "current-event": null,
}
```

#### DELETE

Deletes a service and returns the deleted service object

```
DELETE /admin/api/v1/services/{service} HTTP/1.1
```

```
{
    "name": "Example Service",
    "id": "example-service",
    "description": "System is now operational",
    "url": "/api/v1/services/example-service",
    "current-event": null,
}
```

### 2.3.2 Service List

The Service List resource represents all a collection of related services

| Property | Description |
|---|---|
| id | The unique identifier by which to identify the service list |
| name | The name of the service list, defined by the user |
| description | The description of the service list |
| url | The URL of the specific service list resource |

#### List Resource

```
/admin/api/v1/service-lists
```

##### GET

Returns a list of all current service lists tracked by Stashboard

```
GET /admin/api/v1/service-lists HTTP/1.1
```

```
{
    "lists": [
        {
            "name": "Example Foo",
            "id": "example-foo",
            "description": "An explanation of this service"
            "url": "/api/v1/service-lists/example-foo",
        },
        {
            "name": "Example Bar",
            "id": "example-bar",
            "description": "An explanation of this service"
            "url": "/api/v1/service-lists/example-bar",
        }
    ]
}
```

##### POST

Creates a new service list and returns the new service list object.

| Param | Description |
|---|---|
| name | Name of the service list |
| description | Description of service list |

```
POST /admin/api/v1/service-lists HTTP/1.1 name=New%20Service&description=A%20great%20service
```

```
{
    "name": "New List",
    "id": "new-list",
    "description": "A great service"
    "url": "/api/v1/service-list/new-list",
}
```

#### Instance Resource

```
/admin/api/v1/service-lists/{service-list}
```

The Service Instance resources represents an individual service list

#### GET

```
GET /admin/api/v1/service-lists/{service} HTTP/1.1
```

```
{
    "name": "Example List",
    "id": "example-list",
    "description": "An explanation of what this list represents"
    "url": "/api/v1/service-lists/example-list",
}
```

#### POST

Updates a service list's description and returns the updated service list. All the listed parameters are optional.

| Param | Description |
|---|---|
| name | Name of the service list |
| description | Description of service list |

```
POST /admin/api/v1/service-lists/{service-list} description=System%20is%20now%20operational
```

```
{
    "name": "Example List",
    "id": "example-list",
    "description": "System is now operational",
    "url": "/api/v1/service-lists/example-list",
}
```

#### DELETE

Deletes a service list and returns the deleted service object

```
DELETE /admin/api/v1/service-lists/{service-list} HTTP/1.1
```

```
{
    "name": "Example List",
    "id": "example-list",
    "description": "System is now operational",
    "url": "/api/v1/service-lists/example-list",
}
```

### 2.3.3 Events

The Events List resource represents all event associated with a given service

| Property | Description |
|---|---|
| sid | The unique identifier by which to identify the event |
| message | The message associated with this event |
| timestamp | The time at which this event occurred, given in RFC 1132 format. |
| url | The URL of the specific event resource |
| status | The status of this event, as described by the Statuses resource |

### List Resource

```
/admin/api/v1/services/{service}/events
```

#### GET

Returns all events associated with a given service in reverse chronological order.

```
GET /admin/api/v1/services/{service}/events HTTP/1.1
```

```
{
    "events": [
        {
            "timestamp": "Mon, 28 Jun 2010 22:17:06 GMT",
            "message": "Problem fixed",
            "sid": "ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GBAM",
            "url": "/api/v1/services/example-service/events/ahJpc215d2Vic2VydmljZWRvd2",
            "status": {
                "id": "down",
                "name": "Down",
                "description": "An explanation of what this status represents",
                "level": "ERROR",
                "image": "/images/status/cross-circle.png",
                "url": "/api/v1/statuses/down",
            },
        },
        {
            "timestamp": "Mon, 28 Jun 2010 22:18:06 GMT",
            "message": "Might be up",
            "sid": "ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
            "url": "/api/v1/services/example-service/events/ahJpc215d2Vic..."
            "status": {
                "id": "down",
                "name": "Down",
                "description": "An explanation of what this status represents",
                "level": "ERROR",
                "image": "/images/status/cross-circle.png",
                "url": "/api/v1/statuses/down",
            },
        }
    ]
}
```

The Events List resource also supports filtering events via dates. To filter events, place on of the following options into the query string for a GET request

While the format of these parameters is very flexible, we suggested either the RFC 2822 or RFC 1123 format due to their support for encoding timezone information.

Events List URL Filtering Options

| Option | Description |
|---|---|
| start | Only show events which started after this date, inclusive. |
| end | Only show events which started before date, inclusive. |

```
GET /admin/api/v1/services/{service}/events?start=2010-06-10 HTTP/1.1
```

would return all events starting after June 6, 2010.

Similarly, both "start" and "end" can be used to create date ranges

```
GET /admin/api/v1/services/{service}/events?end=2010-06-17&start=2010-06-01 HTTP/1.1
```

would return all events between June 6, 2010 and June 17, 2010

#### POST

Creates a new event for the given service and returns the newly created event object. All arguments are required.

| Param | Description |
|---|---|
| status | The system status for the event. This must be a valid system status identifier found in the Statuses List resource |
| message | The message for the event |

```
POST /admin/api/v1/services/{service}/events HTTP/1.1 status=AVAILABLE&message=System%20is%20now%20op
```

```
{
    "timestamp": "Mon, 28 Jun 2010 22:18:06 GMT"
    "message": "Might be up",
    "sid": "ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "url": "/api/v1/services/example-service/events/ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "status": {
        "id": "down",
        "name": "Down",
        "description": "An explanation of what this status represents",
        "level": "ERROR",
        "image": "/images/status/cross-circle.png",
        "url": "/api/v1/statuses/down",
    },
}
```

### Current Event

The Current Service Event resource simply returns the current event for a given service.

```
/admin/api/v1/services/{service}/events/current
```

#### GET

Returns the current event for a given service.

```
GET /admin/api/v1/services/{service}/events/current HTTP/1.1
```

```
{
    "timestamp": "Mon, 28 Jun 2010 22:17:06 GMT",
    "message": "Might be up",
    "sid": "ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "url": "/api/v1/services/example-service/events/ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "status": {
        "id": "down",
        "name": "Down",
        "description": "An explanation of what this status represents",
        "level": "ERROR",
        "image": "/images/status/cross-circle.png",
        "url": "/api/v1/statuses/down",
    },
}
```

### Instance Resource

The Event Instance resource represents an individual event for a given service.

```
/admin/api/v1/services/{service}/events/{sid}
```

#### GET

Returns a service event with the given event sid. The event's status object is also returned as well.

```
GET /admin/api/v1/services/{service}/events/{sid} HTTP/1.1
```

```
{
    "timestamp": "Mon, 28 Jun 2010 22:17:06 GMT",
    "message": "Might be up",
    "sid": "ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "url": "/api/v1/services/example-service/events/ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "status": {
        "id": "down",
        "name": "Down",
        "description": "An explanation of what this status represents",
        "level": "ERROR",
        "image": "/images/status/cross-circle.png",
        "url": "/api/v1/statuses/down",
    }
}
```

#### DELETE

Deletes the given event and returns the deleted event

```
DELETE /admin/api/v1/services/{service}/events/{sid} HTTP/1.1
```

```
{
    "timestamp": "Mon, 28 Jun 2010 22:17:06 GMT",
    "message": "Might be up",
    "sid": "ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "url": "/api/v1/services/example-service/events/ahJpc215d2Vic2VydmljZWRvd25yCwsSBUV2ZW50GA8M",
    "status": {
```

```
        "id": "down",
        "name": "Down",
        "description": "An explanation of what this status represents",
        "level": "ERROR",
        "image": "/images/status/cross-circle.png",
        "url": "/statuses/down",
    },
}
```

### 2.3.4 Statuses

The Status resource represents a possible status for a service.

| Property | Description |
|---|---|
| id | The unique identifier by which to identify the status |
| name | The name of the status, defined by the user |
| description | The description of the status |
| url | The URL of the specific status resource |
| level | The level of this status. Can be any value listed in the Levels List resource |
| image | The URL of the image for this status |

#### List Resource

```
/admin/api/v1/statuses
```

The Status List resource represents all possible systems statuses.

#### GET

Returns all service statuses

```
GET /admin/api/v1/statuses HTTP/1.1
```

```
{
    "statuses": [
        {
            "name": "Available",
            "id": "available",
            "description": "An explanation of what this status represents",
            "level": "NORMAL",
            "image": "/images/status/tick-circle.png",
            "url": "api/v1/statuses/up",
        },
        {
            "name": "Down",
            "id": "down",
            "description": "An explanation of what this status represents",
            "level": "ERROR",
            "image": "/images/status/cross-circle.png",
            "url": "api/v1/statuses/down",
        },
    ]
}
```

#### POST

Creates a new status and returns this newly created status. All parameters are required.

| Param | Description |
|---|---|
| name | The name of the status |
| description | The description of the status |
| level | The level of the status. lues listed in the rce |
| image | The filename of the image, with no extension. See the status-images resource |

```
POST /admin/api/v1/statuses HTTP/1.1 name=Down&description=A%20new%20status&severity=1000&image=cross
```

```
{
    "name": "Down",
    "id": "down"
    "description": "A new status",
    "level": "ERROR",
    "image": "cross-circle",
    "url": "/api/v1/statuses/down",
}
```

### Instance Resource

The Status Instance resource represents a single service status

```
/admin/api/v1/statuses/{name}
```

#### GET

Returns a status object

```
GET /admin/api/v1/services HTTP/1.1
```

```
{
    "name": "Down",
    "id": "down",
    "description": "A new status",
    "level": "ERROR",
    "image": "/images/status/cross-circle.png",
    "url": "/api/v1/statuses/down",
}
```

#### POST

Update the given status. All the following parameters are optional.

| Param | Description |
|---|---|
| name | The name of the status |
| description | The description of the status |
| level | The level of the status. lues listed in the rce |
| image | The filename of the image, with no extension. See the status-images resource |

```
POST /admin/api/v1/statuses HTTP/1.1 description=A%20new%20status&severity=1010&image=cross-circle.pn
```

```
{
    "name": "Down",
    "id": "down",
    "description": "A new status",
    "level": "ERROR",
    "image": "/images/status/cross-circle.png",
    "url": "/api/v1/statuses/down",
}
```

#### DELETE

Delete the given status and return the deleted status

```
DELETE /admin/api/v1/statuses/{name}
```

```
{
    "name": "Down",
    "id": "down",
    "description": "A new status",
    "level": "ERROR",
    "image": "/images/status/cross-circle.png",
    "url": "/api/v1/statuses/down",
}
```

### 2.3.5 Status Levels

The Status Levels resource is a read-only resource which lists the possible levels for a status.

#### List Resource

```
/admin/api/v1/levels
```

#### GET

Returns a list of possible status levels in increasing severity

```
GET /admin/api/v1/levels
```

```
{
    "levels": [
        "NORMAL",
        "WARNING",
        "ERROR",
        "CRITICAL",
    ]
}
```

### 2.3.6 Status Images

The Status Images resource is a read-only resource which lists the icons available to use for statuses

#### List Resource

```
/admin/api/v1/status-images
```

#### GET

Returns a list of status images.

```
GET /admin/api/v1/status-images
```

```
{
    "images": [
        {
            "name": "sample-image",
            "url": "/status-images/sample-image.png",
        },
        {
            "name": "sample-image",
            "url": "/status-images/sample-image.png",
        },
    ]
}
```

## 2.4 REST API Examples

### 2.4.1 PHP

```
<?php

require_once('OAuth.php');

$consumer_key = 'anonymous';
$consumer_secret = 'anonymous';
$oauth_key = 'ACCESS_TOKEN';
$oauth_secret = 'ACCESS_SECRET';
$app_id = "stashboard";

$consumer = new OAuthConsumer($consumer_key, $consumer_secret);
$token = new OAuthToken($oauth_key, $oauth_secret);

// Set up a request function
function request($consumer, $token, $url, $method = "GET", $data = null) {

        $sign = new OAuthSignatureMethod_HMAC_SHA1();
        $request = OAuthRequest::from_consumer_and_token(
                $consumer, $token, $method, $url, $data
        );

        $request->sign_request($sign, $consumer, $token);
        $ch = curl_init($request->get_normalized_http_url());

        if ($method == "POST") {
                curl_setopt($ch, CURLOPT_POST ,1);
                curl_setopt($ch, CURLOPT_POSTFIELDS , $request->to_postdata());
```

```
        }

        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
        curl_setopt($ch, CURLOPT_FOLLOWLOCATION ,1);
        curl_setopt($ch, CURLOPT_HEADER ,0);  // DO NOT RETURN HTTP HEADERS

        return curl_exec($ch);
}

// Fill in your website
$base_url = "https://$app_id.appspot.com/admin/api/v1";

$data = array(
                                "name" => "An Example Service",
                                "description" => "An example service, created "
                                        . "using the StashBoard API",
                        );

$r = request($consumer, $token, $base_url . "/services", "POST", $data);
$service = json_decode($r);

// GET the list of possible status images
$r = request($consumer, $token, $base_url . "/status-images", "GET");
$data = json_decode($r);
$images = $data->images;

// Pick a the first image
$image = $images[0];

// POST to the Statuses Resources to create a new Status
$data = array(
                                "name" => "Example Status",
                                "description" => "An example status, means nothing",
                                "level" => "NORMAL",
                                "image" => $image->name,
                        );

$r = request($consumer, $token, $base_url . "/statuses", "POST", $data);
$status = json_decode($r);

// Create a new event with the given status and given service
$data = array(
                                "message" => "Our first event! So exciting",
                                "status" => $status->id,
                        );

$r = request($consumer, $token, $service->url . "/events", "POST", $data);
$event = json_decode($r);

print_r($event);
```

### 2.4.2 Ruby

```
require 'rubygems'
require 'oauth'
```

```ruby
require 'json'

oauth_key = 'XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX'
oauth_secret = 'YYYYYYYYYYYYYYYYYYYYYYY'

# Fill in your website
base = "https://stashboard.appspot.com"

@consumer=OAuth::Consumer.new "anonymous",
                              "anonymous",
                              {:site=>base}

@token = OAuth::AccessToken.new(@consumer, oauth_key, oauth_secret)

# POST to the Services Resource to create a new service. Save the response for
# later
@response = @token.post("/admin/api/v1/services", {
    :name => "An Example Service",
    :description => "An example service, created using the StashBoard API",
})
srvice = JSON.parse(@response.body)

# GET the list of possible status images
@response = @token.get("/admin/api/v1/status-images")
data = JSON.parse(@response.body)
images = data['images']

# Pick a random image for our status
image = images[rand(images.length)]


# POST to the Statuses Resources to create a new Status
@response = @token.post("/admin/api/v1/statuses", {
    :name => "Maintenance",
    :description => "The web service is under-going maintenance",
    :level => "WARNING",
    :image => image["name"],
})

status = JSON.parse(@response.body)

@response = @token.post("/admin/api/v1/services/" + srvice["id"] + "/events", {
    :message => "Our first event! So exciting",
    :status => status["id"],
})
event = JSON.parse(@response.body)

puts event
```

### 2.4.3 Python

```python
import oauth2 as oauth
import json
import urllib
import unittest
import random
```

```
# Stashboard application id
app_id = "stashboard-hrd"

# These keys can be found at /admin/credentials
consumer_key = 'anonymous'
consumer_secret = 'anonymous'
oauth_key = 'ACCESS_TOKEN'
oauth_secret = 'ACCESS_SECRET'

# Create your consumer with the proper key/secret.
# If you register your application with google, these values won't be
# anonymous and anonymous.
consumer = oauth.Consumer(key=consumer_key, secret=consumer_secret)
token = oauth.Token(oauth_key, oauth_secret)

# Create our client.
client = oauth.Client(consumer, token=token)

# Base url for all rest requests
base_url = "https://%s.appspot.com/admin/api/v1" % app_id


# CREATE a new service
data = urllib.urlencode({
        "name": "Generic Web Service",
        "description": "An example web service or REST API",
        })

resp, content = client.request(base_url + "/services",
                               "POST", body=data)
service = json.loads(content)

# GET the list of possible status images
resp, content = client.request(base_url + "/status-images", "GET")
data = json.loads(content)
images = data["images"]

# Pick a random image for our status
image = random.choice(images)

# POST to the Statuses Resources to create a new Status
data = urllib.urlencode({
    "name": "Maintenance",
    "description": "The web service is under-going maintenance",
    "image": image["name"],
    "level": "WARNING",
})

resp, content = client.request(base_url + "/statuses", "POST", body=data)
status = json.loads(content)

# Create a new event with the given status and given service
data = urllib.urlencode({
    "message": "Our first event! So exciting",
    "status": status["id"],
})

resp, content = client.request(service["url"] + "/events", "POST", body=data)
```

```
print json.loads(content)
```