API Docs v1.0

Overview

SparkDash is a dashboard for managing mobile workers, devices and business processes. It's a lightweight, comprehensive notification platform for mobile device management and device analytics.

SparkDash is low cost, easy to implement and open source.

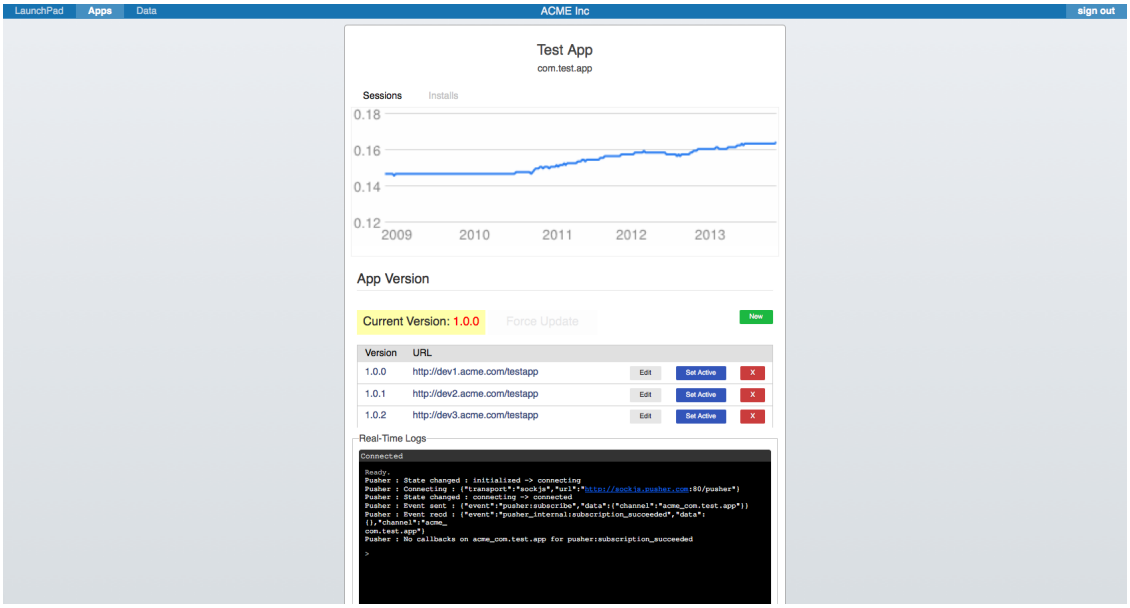
SparkDash allows you to:

- Communicate with your mobile workforce
- See all of their locations on a map
- Pinpoint and assign jobs/tasks to your workforce
- Notify and update your workforce instantly
- Report and review everything that happens

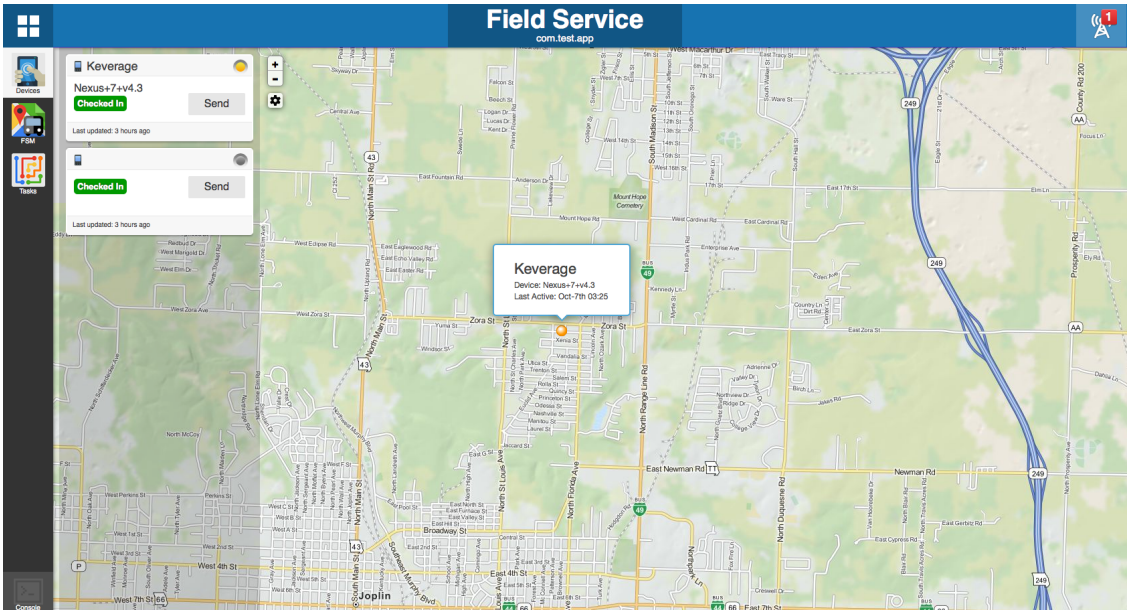
SparkDash is a web application intended to run as a self-hosted (on-premise) application or as a hosted cloud service. It is open source and fully customizable to your needs using a familiar JQuery, CSS and node.js server stack.

Screenshots

App Profile



Devices



Technology Stack

SparkDash is an event-driven server, implemented in Node.js, a JavaScript-based server for creating scalable non-blocking I/O and a single-threaded event loops.

The SparkDash client is a modern Web app, implemented in Javascript and uses jQuery, RequireJS and PureCSS to simplify portability across modern browsers. Currently SparkDash only supports modern browsers

(Chrome, Firefox, Safari and Opera).

SparkDash comes bundled with two servers: 1) a mobile-ready API server and 2) a static file server, used to build, compress and deliver the SparkDash Web app. The file server includes a basic proxy service used to route requests to the SparkDash Mobile API server (or third party sites), bypassing cross-domain issues.

SparkDash Mobile APIs are tightly integrated with Spark Mobile to provide ambient access to the notification and analytic APIs. You can also integrate the APIs within any mobile application using simple HTTP requests as well as your own back-end systems.

Note: SparkDash UI uses a proxy to this endpoint to avoid cross domain issues. All proxied requests originating from the browser are denoted with the `_` prefix. For instance, the `/devices` api endpoint is accessible via `/_devices` in the SparkDash Web app.

Endpoints

SparkDash is available as a hosted application, accessible as a subdomain on the Semantic Press website.

	URL
App	http://YOUR_DOMAIN.semanticpress.com
API	http://YOUR_DOMAIN.semanticpress.com/api

Database

SparkDash leverages two types of databases: 1) the filesystem for lightweight persistence of configuration files and 2) Redis, a scalable key-value datastore used for real-time analytics and flexible object store.

Filesystem

The filesystem is used to store JSON-based configuration files. It's accessible through a RESTful interface. Refer to the HTTP endpoint chart below.

App Endpoints	Method	API	Description
`/_db?id=settings`	GET		Returns a JSON object for a given `id`. The `id` is also the name of the file.
`/_db`	POST		Create database object. Pass in an `id` property to set the filename and primary key. Remember to set the header `Content-Type` to `application/json`
`/_db?id=settings`	DELETE		Delete a database object.
`/_db`	OPTIONS		List all JSON documents contained on the filesystem. This will append JSON objects and return a singleton, where each node contains an `id` property for reference.

Redis

Redis is used as the primary storage engine due to it's high availability, fast read/write and publish/subscribe messaging paradigm.

You can configure SparkDash to use a local Redis instance or a remote host provider such as Garantia Data. In either case your data can run within a fully secure, single-tenant environment.

Your Redis database is stored as a single encypted file, which make it easy for cluster replication and portability.

Your Redis instance contains 3 databases used within SparkDash.

Index	Type	Name	Notes
1	Hash	Auth Tokens.	
2	Hash	Devices (clientID / UUID)	
3	Strings	Messages	

Domain Configuration

Each SparkDash instance runs within a specific **domain**, a unique namespace where all apps, databases and accounts reside.

When you register for a new SparkDash account on Semantic Press, your domain will reside as a subdomain on http://semanticpress.com. This is your single-tenent sandbox. You can add logos, color schemes to customize the appearance of your portal.

For on-premise installs, your domain is defined when you run the installation script and can be modified

anytime.

Domain Data Model

```
{
  _id: 5247635f1f97fdff77000001,
  username: 'terryzmartin@gmail.com',
  id: 'acme',
  title: 'ACME Inc',
  logo: null,
  icon: null,
  favicon: null,
  database: {
     engine: 'redis',
     port: 17438,
     host: 'pub-redis-17438.us-west-1.1.azure.garantiadata.com',
     pass: 'blahblahblah'
  }
}
```

Security

SparkDash is designed to be fully secure, from the transport layer, to session, application and data persistence. Each API request requires a valid `Authorization` token that is derived by hashing (SHA-1) your Spark license with a secret key. To send a secure request to SparkDash, simply add the `Authorization` string to the HTTP header.

Note: You can generate a new auth token from the SparkDash admin panel.

Example:

```
Authorization: 90ajf09hr93hrhsdfhsoifasofi0902j309j
```

Error Response

An error will occur if the token is eitehr missing or invalid. A `response` property will determine if an error occurred.

Example:

```
{
  "response": "error",
  "message": "Invalid authorization token."
}
```

Note: Security tokens are stored as a Redis hash table, referenced in database 1 (see Redis section). The key is the authorization token. The value is a JSON object that contains the `domain` and `appPackage`.

Creating a SparkDash App

Before you can use SparkDash, you must register for a Semantic Press user account and create a new SparkDash project. From there, you can generate a Spark Mobile auth key and API gateway for your Spark Mobile apps to connect to. All SparkDash hosted apps will reside as a subdomain on http://semanticpress.com.

For example, if you are the admin for **Acme, Inc.**, you can manage your apps at http://acme.semanticpress.com and obtain your auth token for API requests.

Your API gateway would be http://acme.semanticpress.com/api

Note: If you are a developer, you can use the Semantic Press API to integrate the SparkDash provisioning process.

Name	Endpoint	Description
Health	http://api.semanticpress.com/health	View server statistics
Register App	http://api.semanticpress.com/app/register	Create a new SparkDash app
Validate App	http://api.semanticpress.com/app/app/validate	
Validate Spark Key	http://api.semanticpress.com/app/spark/validate/key	
Validate Spark Token	http://api.semanticpress.com/app/spark/validate/token	
Generate Spark Token	http://api.semanticpress.com/app/spark/generate/token	Requires account authorization

SparkDash API

API overview:

Name	Endpoint	Description
Beacon	http://SUBDOMAIN.semanticpress.com/api/beacon	Main receiving endpoint.
Devices	http://SUBDOMAIN.semanticpress.com/api/devices	List of devices.
Messages	http://SUBDOMAIN.semanticpress.com/api/messages	Message notification endpoint
Health	http://SUBDOMAIN.semanticpress.com/api/health	View server statistics

/api/beacon

Beacon is the primary method call for send device-specific analytics. Arbitrary data is stored as a hash map of key-value pairs, which can be used to store physical location of each device, crash analytics, user info, lifecycle events, or virtually any other property that Spark Mobile application wants to track.

There are two concepts to learn with Beacon: **Client Events** and **Event Callbacks**. Because SparkDash is an event-driven, bi-directional notification platform, all requests are asynchronous. This means a device will send a request to SparkDash and then wait to receive notifications. Devices connected to SparkDash can communicate to 1:1 or 1:many with other devices running the same application.

Request Data Model

The following are property names for all Beacon requests.

Field	Required	Description
eventType	**Required**	Determines the type of request
eventName	**Required**	Determines the name of the request type
appID	**Required**	Used to associate analytics to a specific content handler. To reference the main app ID, use `packageID`.
appBuild	**Required**	The android:versionCode field in the Manifest
enabled	*Optional*	Boolean. Set geolocation monitoring true/false. Used for eventName: **Location**.
clientID	*Optional*	Primary key for devices
timestamp	*Optional*	Unix timestamp. Default: Auto generated
groups	*Optional*	Array of string names assigned to a user. Used for filtering.
expires	*Optional*	Seconds. Used to delete devices, time to live.
latitude	*Optional*	Needed for geocoding
longitude	*Optional*	Needed for geocoding

Response Data Model

All Beaconrequests will return a `response` value of either **ok** or **error** along with a `message` object that will be the payload from the callback event. This `message` object is provided as a convenience. All clients subscribed to the callback event will receive this payload object.

The message payload object may contain additional fields that are relevant to the request `eventType` and `eventName`.

Beacon Required Fields

All beacon requests must have the following properties. Some **Beacon Client Events** may require additional

fields. If any are missing the request will throw an error.

- `eventType`
- `eventName`
- `clientID`

Device Expiration (seconds)

A device can expire (auto-delete) from the monitoring list after a specified amount of seconds has elapsed, also known as time to live (TTL). By default device will live forever on the monitoring list, unless it is removed in an explicit way.

To set an expiration, use the `expiration` property and pass an integer representing the number of seconds to elapse.

Example

For example, assume we want to monitor the location status of a single device based on the `clientID`. The `update_client_geo@beacon` is the event callback handle that will notify subscribers that this particular device has change it's location.

The `enabled` field determines the monitoring state of a device (boolean), which provides a real-time perspective to the list of devices. If a device is set to `false` the device icon will visually change to a disabled state and the device location will be locked in the last known geolocation.

```
POST http://acme.semanticpress.com/api/beacon
Content-Type: application/json

{
    "clientID":"ef12155c-a286-3253-bfb1-24dbe403a1fas",
    "eventType":"internal",
    "eventName":"Location",
    "userID": "Keverage",
    "enabled":true,
    "longitude": "-94.481894",
    "latitude": "37.1142284"
}
```

Example: Beacon Response

```
{
  "response": "ok",
  "message": {
    "event": "update_client_geo@beacon",
    "payload": {
      "clientID": "ef12155c-a286-3253-bfb1-24dbe403a1fas",
      "device_state": 1,
      "timestamp": 1382228312
    }
  }
}
```

Callback Events

You can subscribe to **Callback Event** notification events that respond to various SparkDash API requests.

Beacon Events

eventName	Pusher Event	Push Payload	Description
Location	**new_client@beacon**	{clientID: "ef12155c-a286-3253-bfb1-24dbe403a1fa",enabled:true,lat: "37.1142284",lng: "-94.481894"}	Notify a client that a new device exists. Used to register a new device object.
Location	**update_client@beacon**	{clientID: "ef12155c-a286-3253-bfb1-24dbe403a1fa",enabled:true,lat: "37.1142284",lng: "-94.481894"}	Updates an existing device object with new data.
Remove	**remove_client@beacon**	{clientID: "ef12155c-a286-3253-bfb1-24dbe403a1fa"}	Remove a client from the SparkDash database.
Internal	**internal_startapp@beacon**	{"current_appBuild": "1.0.0"}	Callback event for **Start App**. This will include the `current_appBuild` for enforcing the Version Policy.
Internal	**internal_undefined@beacon**		This is an event callback handler for undefined `eventName`

Main Events

eventName	Pusher Event	Data	Description
	reset_app@main	{clientID: "ef12155c-a286-3253-bfb1-24dbe403a1fa"}	Force the device to wipe the app and start over.
	heartbeat@main	{}	Listen for beats to keep the socket connection open.
	message@main	{clientID: "ef12155c-a286-3253-bfb1-24dbe403a1fa"}	Send a message to the main channel.
	log@main	{clientID: "ef12155c-a286-3253-bfb1-24dbe403a1fa"}	Log statements for the packageID.
	update@main	{'type':'current_appBuild','data': {'appBuild':'','url':'http://'}}	Send an update request on the main channel. The `type` field determines what action to perform.
	custom@main	{"data":{}}	A custom event callback handler that the developer may pass data

These eventName properties are currently mapped to the following Pusher notification events.

Client Events: Extended

Some events may need specific data to be return as part of the response, rather than a simple "ok". This may be required for certain lifecycle events or transactions. In this case, the `{message:{payload:{}}` object will contain the response

Start App

Lifecycle request that notifies Beacon of a new app launch (start) state.

Callback Event:

internal_startapp@beacon

Required fields:

- `appPackage`

Request

```
{
    "eventType":"internal",
    "eventName":"start app",
    "clientID":"ef12155c-a286-3253-bfb1-24dbe403a1fas",
    "appPackage":"com.test.app"
}
```

Response

```
{
  "response": "ok",
  "message": {
    "event": "internal_startapp@beacon",
    "payload": {
      "current_appBuild": "1.0.0"
    }
  }
}
```

Custom Events

You can track custom **Callback Events** where you define the `eventName` by using the `'eventType':'custom'`:

```
{
    "eventType":"custom",
    "eventName":"SomeCustomEvent",
    "clientID":"ef12155c-a286-3253-bfb1-24dbe403a1fa"
    ... more
}
```

Pusher event: `custom_SomeCustomEvent@beacon`

/api/devices

GET

Returns an array of registered devices and corresponding JSON object representing device analytics.

Properties

Query	Description	Example
appPackage	This is app package id.	`com.test.app`
app_oid	This is the internal app Object ID assigned by SparkDash.	`5249aaf5fa77d45eb3000001`

Example

```
GET http://acme.semanticpress.com/devices?appPackage=com.test.app
```

```
[
  {
    "deviceModel": "Nexus+7+v4.3",
    "clientID": "09a8sdf98asd0f98-0as0a9sdf08-asdf",
    "appTitle": "Service+Mobile+1.0",
    "userID": "Keverage",
    "longitude": "-94.481894",
    "latitude": "37.1142284"
  },
  {
    "deviceModel": "Nexus+7+v4.3",
    "clientID": "ef12155c-a286-3253-bfb1-24dbe403a1fa",
    "appTitle": "Service+Mobile+1.0",
    "userID": "Keverage",
    "longitude": "-94.481894",
    "latitude": "37.1142284"
  }
]
```

/api/messages

POST

Send a message to one or all devices subscribed to the main channel. Use `socketID` to directly target a device, otherwise Spark Mobile will manage the message filter logic based on `userID` and `clientID`.

If you pass a `clientID` string, then the Callback Event will also contain this field.

Note: You must pass an Authorization string in the header. The database is storing a temporary copy of each message as a Redis string, using the Auth string and a unique identifier as the key. Each message will expire after 24 hours. You can change this setting in the SparkDash admin panel.

Request

```
{
    "appPackage":"com.test.app",
    "clientID":"1234567",
    "userID":"Tom",
    "socketID":"",
    "data":{}
}
```

Response

```
{
  "response": "ok",
  "message": {
    "event": "message@main",
    "payload": "HELLO"
  }
}
```

Event:

message@main

```
{
  "data": "HELLO"
}
```

/health

GET

Returns a JSON object of the server health, including CPU profiling and heap profiling.

Read more about server monitoring: http://stackoverflow.com/questions/5580776/monitoring-a-node-js-server

```
{
  "pid": 13592,
  "memory": {
    "rss": 42119168,
    "heapTotal": 20608760,
    "heapUsed": 11413688
  },
  "uptime": 99
}
```

HTML Form Processing

Each form must have a hidden field with a name of `id` and a value that is mapped to the properties file. This will persist form data to the filesystem using `node-store`.

Each form field will be parsed into a JSON object and then persisted to the filesystem. You must follow a convention of defining an `id` property to the form.

Example

```
<form id="form-c3" action="" class="pure-form pure-form-aligned">
  <fieldset>
    <div class="pure-control-group">
         <label for="name">Idle Timeout (minutes)</label>
         <input name="id" value="map" type="hidden">
         <input name="idleTimeout" id="name" type="text" placeholder="Enter num of min
     </div>
   </fieldset>
   <div class="clearfix" style="text-align: center;">
     <button type="submit" class="pure-button pure-button-primary" data="submit-c3">Su
   </div>
</form>
```

Key	Value
Offline	0
Active	1
Idle	2
Pause/Processing	3

- Offline/Active value of 0/1 would be handy for quick conditional checks for false/true.

Misc

- https://github.com/smeijer/L.GeoSearch
- https://github.com/smeijer/leaflet-locatecontrol
- https://github.com/Leaflet/Leaflet.draw
- https://github.com/aratcliffe/Leaflet.contextmenu
- https://github.com/makinacorpus/Leaflet.FileLayer
- https://github.com/dwilhelm89/LeafletSlider
- https://github.com/tmcw/leaflet-pip
- https://github.com/tommoor/tinycon

http://kami-design.com/kato/index.html

Todo:

Tabs

- Device locating
- Basic app analytics, user metrics
- Bug reporting