



# **Red Hat Mobile Application Platform 4.7**

## **Local Development Guide**

For Red Hat Mobile Application Platform 4.7



# Red Hat Mobile Application Platform 4.7 Local Development Guide

---

For Red Hat Mobile Application Platform 4.7

## Legal Notice

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## Abstract

This document provides guides for using the RHMAP CLI tool and for setting up a local development environment for developing apps in Red Hat Mobile Application Platform 4.7.

## Table of Contents

<b>CHAPTER 1. SETTING UP FHC</b>	<b>4</b>
1.1. INSTALLING FHC	4
1.1.1. Installing FHC	4
1.1.1.1. Command Completion (Linux and Mac only)	4
1.1.2. Usage	4
1.1.2.1. Configuration	5
1.1.3. Next Steps	5
<b>CHAPTER 2. GETTING STARTED WITH FHC</b>	<b>6</b>
2.1. TARGETING AN RHMAP DOMAIN WITH FHC	6
2.2. CREATING YOUR FIRST PROJECT	6
2.3. DEPLOYING A CLOUD APP	8
2.4. DEPLOYING A CLIENT APP	8
2.5. SUMMARY	8
<b>CHAPTER 3. WORKING WITH PROJECTS AND APPS</b>	<b>10</b>
3.1. LOG IN TO THE PLATFORM	10
3.2. LIST EXISTING PROJECTS	10
3.3. CREATING A PROJECT	11
3.4. ADDING APPS TO A PROJECT	12
3.5. NEXT STEPS	12
<b>CHAPTER 4. DEVELOPING CODE LOCALLY</b>	<b>13</b>
4.1. DEVELOPMENT TOOLS	13
4.1.1. Git	13
4.1.2. Grunt	13
4.2. CLONING THE SOURCE CODE	14
4.3. USING CODE FROM ANOTHER GIT REPOSITORY	14
4.4. ACCESSING CLOUD APPS DURING DEVELOPMENT	15
4.4.1. Using Cloud Runtime	15
4.4.2. Using Local Runtime	15
4.5. RUNNING THE CLIENT LOCALLY	16
4.6. WORKING WITH MBAAS SERVICES LOCALLY	17
4.7. EDITING THE CLOUD CODE	18
4.8. EDITING THE CLIENT CODE	19
4.9. VIEWING CHANGES IN THE APP STUDIO	19
4.10. ADVANCED DEVELOPMENT	20
4.10.1. Using fh.cache	20
4.10.2. Using fh.db	20
<b>CHAPTER 5. WORKING WITH THE PLATFORM USING FHC</b>	<b>21</b>
5.1. MANAGING API KEYS WITH FHC	21
5.1.1. Set User API Keys	21
5.1.2. List User API Keys	21
5.1.3. Create User API Keys	21
5.1.4. Update User API Keys	22
5.1.5. Revoke User API Keys	22
5.2. CREATE A SERVICE USING FHC	23
5.2.1. Log in to the platform	23
5.2.2. Create a Project	23
5.2.3. Create a Service	23
5.3. VIEWING APP STATS WITH FHC	24

<b>CHAPTER 6. WORKING WITH FORMS USING FHC .....</b>	<b>26</b>
6.1. CREATE A FORMS PROJECT USING FHC	26
6.1.1. Create a Forms project	26
6.1.2. Add a form	26
6.1.3. Associate a form with a project	26
6.1.4. Create a theme	27
6.1.5. Associate the theme with the app	27
6.1.6. Add another Forms App	27
6.2. CREATE A FORM USING FHC	28
6.2.1. List available forms	28
6.2.2. Create a form	28
6.2.3. Update form	30
6.3. CREATE A FORM SUBMISSION USING FHC	30
6.3.1. Listing submissions	31
6.3.2. Return a specific submission	31
6.3.3. Submission status	31
6.3.4. Submitting a Submission	31
6.4. CREATE A FORM THEME USING FHC	32
6.4.1. List all themes	32
6.4.2. Create a theme	32
6.4.3. Associating a theme with an app	33
6.4.4. Updating a theme	33



# CHAPTER 1. SETTING UP FHC

## 1.1. INSTALLING FHC

### Overview

**fhc** is the Red Hat Mobile Application Platform (RHMAP) command line interface (CLI) and it is Node.js based. Almost all functionality in the RHMAP Studio is exposed through the RHMAP APIs and is accessed using **fhc**. This allows **fhc** to be included in other Node.js applications thus allowing access to the Platform programmatically. Interaction with the Platform's exposed functionality can be integrated into automated processes such as build systems and continuous integration systems.

### Requirements

To use **fhc**, install Node.js and NPM. To install Node.js, see [nodejs.org](https://nodejs.org).

Once installed, two new command line applications are available, **node** and **npm**.

### 1.1.1. Installing FHC

In a terminal/command prompt, run:

```
npm install -g fh-fhc
```

If installing on Linux, you may need to run this command as a sudoer:

```
sudo npm install -g fh-fhc
```

This will install **fhc** from [npm](https://npmjs.org) — the central registry of Node.js modules.

The **-g** flag tells npm to install **fhc** globally so that it will be available from any directory.

Once installation is complete, **fhc** will be available from your command line. If you use **Z shell (zsh)**, use **hash -r** to reset the hash table for the command to be available.

To test FHC is installed correctly, and to show the version you have installed, use:

```
fhc -v
```

#### 1.1.1.1. Command Completion (Linux and Mac only)

The **fhc** bash completion script allows Tab completion of the various **fhc** commands. Install **fhc** bash completion script:

```
fhc completion >> ~/.bashrc
```



#### NOTE

Append the output of **fhc completion** to a relevant file if you use an alternative shell (not **bash**), for example, **~/.zshrc** for **zsh**.

### 1.1.2. Usage



Set the target and login:

```
fhc target https://[your-studio-domain].feedhenry.com
fhc login [email address] [password]
```

To list projects:

```
fhc projects list
```

To list all **fhc** commands:

```
fhc help
```

### 1.1.2.1. Configuration

fhc is extremely configurable. It reads its configuration options from 5 places.

- **Command line switches:**  
Set a config with **-key val**. All keys take a value, even if they are booleans (the config parser doesn't know what the options are at the time of parsing.) If no value is provided, then the option is set to boolean **true**.
- **Environment Variables:**  
Set any config by prefixing the name in an environment variable with **fhc\_config\_**. For example, **export fhc\_config\_key=val**.
- **User Configs:**  
The file at `$HOME/.fhcrc` is an ini-formatted list of configs. If present, it is parsed. If the **userconfig** option is set in the cli or env, then that will be used instead.
- **Global Configs:**  
The file found at `../etc/fhcrc` (from the node executable, by default this resolves to `/usr/local/etc/fhcrc`) will be parsed if it is found. If the **globalconfig** option is set in the cli, env, or user config, then that file is parsed instead.
- **Defaults:**  
fhc's default configuration options are defined in `lib/utils/config-defs.js`. These must not be changed.

Execute **fhc help config** for more information. Use **fhc help** for a list of all commands, or **fhc [command] --help** for help on a specific command.

### 1.1.3. Next Steps

- [Working with Projects & Apps](#)
- [Local App Development](#)

## CHAPTER 2. GETTING STARTED WITH FHC

The following information describes how to create a new Project based on a sample template.

**fhc** is used to clone a Project to a local machine. **npm** is used to install dependencies for local development. **Grunt** is used to start a new Hybrid Client and Node.js Cloud App locally for development within a browser.

### Video Tutorial

- [RHMAP - Create an App](#)

### Prerequisites

- [Section 1.1, “Installing FHC”](#)
- Node.js and Git installed locally

## 2.1. TARGETING AN RHMAP DOMAIN WITH FHC

First you need to target(connect) your RHMAP domain using **fhc**.

Open a command line application, for example, the Terminal on Linux or MacOS, or a command prompt on Windows.

Target your domain by replacing **<domain-url>** with the URL of your domain:

```
fhc target <domain-url>
```

Next you will be required to enter in your RHMAP credentials. Enter the following command and replace **<email or alias>** with your username or email address.

```
fhc login
Username : <email or alias>
```

Enter your password:

```
Password : <enter password>
```

You are now logged into the RHMAP domain you have targeted.

## 2.2. CREATING YOUR FIRST PROJECT

The following information describes how to create a new Project based on the **hello\_world\_project** template.

See the list of available project templates:

```
fhc templates projects
```

Replace the **project name** with the chosen project name and execute:

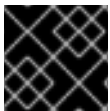
**NOTE**

Make sure to pass the name of the template **hello\_world\_project** as a parameter too.

```
fhc projects create <project name> hello_world_project
```

If successful, a JSON response with all Project information is displayed, for example:

```
{
  "type": "PROJECT",
  "template": null,
  "sysCreated": 1397636395777,
  "guid": "jJHtgZXQPJxXDFJ02pQzMP-d",
  // ...
  "apps": [
    {
      "type": "client_hybrid",
      "description": "",
      "domain": "testing",
      "template": null,
      "email": "joebloggs@feedhenry.com",
      //...
    }
  ],
  "templateId": "default",
  "jsonTemplateId": "hello_world_project"
}
```

**IMPORTANT**

Note down the **guid**, it is required for the following steps.

To review the project information, enter the following command, replacing **project guid** with the **guid** from the previous step.

```
fhc projects read <project guid>
```

This command returns a JSON object shown in the previous step, containing all project information.

Clone the Project to a local machine. Replace the **project name** with the chosen Project name, and also **project guid** with the **guid** from the JSON object:

```
mkdir <project name> ; cd <project name> ; fhc projects clone <project guid>
```

Replace the **project name** with your project name to see the MBaaS example or Cloud App:

```
cd <project name>-Hello-World-MBaaS-Instance ; ls -l
```

Install Cloud App dependencies:

```
npm install
```

Install the **grunt CLI** using the **-g** flag to install the grunt globally:

```
[sudo] npm install -g grunt-cli
```

The Cloud App instance is now ready to be deployed to a local machine.

## 2.3. DEPLOYING A CLOUD APP

To start the cloud server locally:

```
grunt serve
```

The console should output information to indicate that the Cloud App is running on **port 8001** on local machine.

Enter the following curl request on the local machine to test that the Cloud instance is running:

```
curl http://localhost:8001/cloud/hello?hello=world
```

A similar response should be displayed:

```
{  
  "msg": "Hello world"  
}
```

If you received the above response from this curl attempt, your Cloud App is running locally.

## 2.4. DEPLOYING A CLIENT APP

Notice the following command is a chained command. The first part navigates up a level in the directory tree. The second part navigates down a level to the client directory.

Enter the following command, replacing **project name** with the name of your project.

```
cd .. ; cd <project name>-Hello-World-Client ; ls -l
```

Install all dependencies for the Client App:

```
npm install .
```

Run the Client App locally:

```
grunt serve:local
```

If successful, a window opens in your default browser.

## 2.5. SUMMARY

The Platform hosts all Git repositories. Every app within a project will have its own Git repo. Normal Git procedures and processes are performed by changing directory into either the Client App or Cloud App examples.

You are now setup to develop your RHMAP project locally.

## CHAPTER 3. WORKING WITH PROJECTS AND APPS

### Overview

In the Platform, a project is used as a container for grouping related apps together and all apps must be created within a project. The following information describes how to create and add projects using **fhc**.

### Requirements

You must have completed the following tutorials:

- [Installing FHC](#)

### 3.1. LOG IN TO THE PLATFORM

Enter **fhc targets** in the terminal to display a list of available target domains:

```
fhc targets list
```

Use the **fhc target** command to select a target domain:

```
fhc target exampleDomain.redhatmobile.com
```



#### NOTE

The code above selects 'exampleDomain.redhatmobile.com' as the target domain.

The target domain has now been selected. To login, use **fhc login** followed by the username and password:

```
fhc login username@example.com password
```



#### NOTE

Once logged in, access restrictions can apply. Team Permissions determine the level of access a User has within the Platform. See [Teams & Collaborations](#) for more information.

### 3.2. LIST EXISTING PROJECTS

List existing projects:

```
fhc projects list
```



#### NOTE

To return the project list in raw JSON format, append **--json** to the end of the command, for example, **fhc projects list --json**.

A list of existing projects is displayed (ordered by date last modified), for example:

```
-----
```

```

-----
| Id                      | Title                      | No. Apps | Last Modified
|
-----
| 1234567890abcdefghijklmn | Hello World                | 3         | 3 hours ago
|
-----
| 9876543210zyxwvutsrqpmn | Welcome to RHMAP           | 2         | 3 hours ago |
|
-----
-----

```

### 3.3. CREATING A PROJECT

It is possible to create a new project that contains no Client or Cloud Apps however a project **must** have a name. Use **fhc projects create** to create a project.

Typically, a new project is created using a pre-defined template. Project templates allow you to bootstrap your development by cloning the project template which typically provides at least one Client App and one Cloud App.

To learn more about Projects, see the [Projects Documentation](#).

In this tutorial, **hello\_world\_project** template is used. This is a basic project with one Client App and one Cloud App, and uses the standard Hello World paradigm to demonstrate basic functionality.

List all the project templates:

```
fhc templates projects
```

Create a new project based on the **hello\_world\_project** template:

```
fhc projects create helloWorld hello_world_project
```

Verify successful creation of the project by searching for the new project:

```
fhc projects list
```

Alternatively, you could search for your specific project by piping the output of the **fhc projects** command to the **grep** command:

```
fhc projects list | grep 'helloWorld'
```

*# Example:*

```
XME5iUr2VoBV3DbXrVF7qApG | helloWorld | 2 | 3 minutes ago
# ProjectId | Title | Number of apps | Minutes since last update
```

The console output will display the recently created project that contains a Client App and a Cloud App.

To list all apps in a project, select a projectId (also known as a 'guid' (Global Unique Identifier)) and use it in conjunction with the **fhc apps**:

```
fhc projects list

fhc apps <the_selected_project_guid>
```

This will list all the apps for the given projectId:

- Id - The guid of the App.
- Title - The title of the app. This will be used as the app name for Client Apps on mobile devices.
- Description - A description for the application. This will be blank by default.
- Type - The type of the app. Term used by Platform to distinguish between different app types.
- Git - The Git URL of the App. This can be used to clone the app for local development
- Branch - The currently selected branch for editing app code in the platform.

### 3.4. ADDING APPS TO A PROJECT

A Project can contain many apps however, apps that have no relationship with each other should be housed in separate projects. Apps are normally added to a project using an existing App Template. There are a selection of App Templates available for both Client and Cloud Apps.

List all available App Templates:

```
fhc templates apps
```

To add a Client App, use the following command, specifying the **<projectId>**, the name for the **<appTitle>**, and the **<appTemplate>** to add an app to a project.

```
fhc app create --project=<projectId> --title=<appTitle> --type=<type> --
template=<appTemplate>
```

*#The following example illustrates the creation of a new project*

```
fhc app create --project=xe2cbz3cky6zfxq2ca66t55u --title='My Native iOS
App' --type=client_native_ios --template=native_ios_swift_blank_app
```



#### NOTE

If no template is specified, a blank Client App is created.

To verify the successful creation of the app, list all apps in the project.

### 3.5. NEXT STEPS

- [Local App Development](#)



## CHAPTER 4. DEVELOPING CODE LOCALLY

### Overview

The following tutorial describes how to set up a local environment for developing a mobile application with the Red Hat Mobile Application Platform (RHMAP).



#### NOTE

See the [Red Hat Mobile Application Platform Supported Configurations](#) page for appropriate versions of software.

### Requirements

You must have completed the following tutorials:

- [Installing FHC](#)
- [Working with Projects & Apps](#)

## 4.1. DEVELOPMENT TOOLS

In addition to **fhc**, Node.js and NPM, other development tools are required.

### 4.1.1. Git

*"Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency."*

- [Download](#)
- [Docs](#)
- [Getting Started](#)

Install git on a specific operating system:

- RHEL / Fedora: see [Git Installation Page](#)
- Debian based distros: **sudo apt-get install git-core**
- Mac OSX: **brew install git** or **sudo port install git-core +doc +bash\_completion** or via the graphical git [installer](#)
- Windows: see [msysgit](#)



#### NOTE

To install **Homebrew** package manager for Mac OSX, see <http://brew.sh/>

### 4.1.2. Grunt

*"The JavaScript Task Runner"*

- [Getting Started](#)
- [Plugins](#)

**grunt** is available from NPM. To install:

```
sudo npm install -g grunt-cli
```



#### NOTE

All RHMAP Template Apps uses **grunt** as the task runner for local development.

## 4.2. CLONING THE SOURCE CODE

Create a new directory to house your project development:

```
mkdir helloworld  
cd helloworld
```

Clone the source code for all apps in your project using **fhc projects clone <project-guid>**, for example:

```
fhc projects clone XME5iUr2VoBV3DbXrVF7qApG # <== Replace this projectId  
with your projectId
```

An output message is displayed, for example:

```
Cloning into 'helloworld-Hello-World-Cloud-App'...  
Cloning into 'helloworld-Hello-World-Client'...
```

Once complete, check what new directories were created:

```
ls  
  
helloworld-Hello-World-Client  helloworld-Hello-World-Cloud-App
```

## 4.3. USING CODE FROM ANOTHER GIT REPOSITORY

If you want to use code from a remote repository, for example <https://github.com/evanshortiss/rhmap-express-template>, complete the following procedure:

1. Create an appropriate blank app, for example, `blank_advanced_webapp`
2. Clone the repo you want to use into a temporary directory:

```
git clone git@github.com:evanshortiss/rhmap-express-template.git  
temp_express
```

3. Delete the **.git** directory:

```
cd temp_express
```

```
rm -rf .git
```

4. Copy the contents of the repo to your App repo, for example:

```
cd temp_express
cp -r * <repo-path>
cd <repo-path>
```

You can now develop your app and push the results to RHMAP.

## 4.4. ACCESSING CLOUD APPS DURING DEVELOPMENT

The two methods for accessing Cloud Apps are:

1. Access the *cloud instance* running in an environment on the Platform
2. Run the Cloud App in a *local instance* of Node.js to access it

### 4.4.1. Using Cloud Runtime

Client Apps point to <http://localhost:8001> by default

Determine the URL that is hosting the Cloud App:

```
fhc app hosts --app=<app-guid> --env=<some-env>
```

The response is:

```
{
  "url": "https://testing-3utxgzhawprfqot3tya04v5i-example.feedhenry.com"
}
```

Identify the variable **default\_local\_server\_url** in the **Gruntfile.js** file belonging to the Client App. Set the **default\_local\_server\_url** variable to the value returned by the **fhc app hosts** command.

The locally running Client App will now route requests to the Cloud App running in the Platform.

### 4.4.2. Using Local Runtime



#### NOTE

Your npm version should be at least 1.4.15 to run **grunt serve**.

Run the Cloud Server code locally:

```
cd <PROJECT-DIR>/helloWorld-Hello-World-Cloud-App
```

Install all the dependancies for the Cloud app:

```
npm install
```

Once complete (this may take a number of minutes), start the Node.js server.

```
grunt serve
```

An output message is displayed, for example:

```
Running "env:local" (env) task

Running "concurrent:serve" (concurrent) task
Running "watch" task
Waiting...
Running "nodemon:dev" (nodemon) task
[nodemon] v1.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node application.js`
App started at: Tue Sep 30 2014 15:39:07 GMT+0100 (IST) on port: 8001
```

The Node.js process has a lock on the Terminal Window. The Cloud Server is now running however, to proceed, open another terminal window and verify the Cloud Server is running:

```
curl -X POST http://localhost:8001/hello?hello=world
```

An output message is displayed, for example:

```
{"msg": "Hello world"}
```

Change the input parameter from **world** to another value and observe the updated response.

The **Gruntfile.js** file contains a number of useful commands such as:

```
grunt serve      # run you server locally with 'live reload'
grunt test       # run your tests locally
grunt coverage   # run your tests with code coverage
```

See the Grunt-ReadMe.md file or run **grunt --help** for more information.

## 4.5. RUNNING THE CLIENT LOCALLY

Run the Client App locally:

```
cd <PROJECT-DIR>/helloWorld-Hello-World-Client
```

Install all the dependancies for the Client App - mainly around grunt:

```
npm install
```

Start up a web browser session:

```
grunt serve:local
```

**grunt serve:local** attempts to contact a local Node.js Cloud App on port 8001 by default. A similar message is displayed:

```
Running "serve:local" (serve) task
Running "clean:server" (clean) task
Running "connect:livereload" (connect) task
Started connect web server on http://localhost:9002
Running "watch" task
Waiting...
```

See the local README.md for the 'Hello World Client' Template App to learn more.

## 4.6. WORKING WITH MBAAS SERVICES LOCALLY

MBaaS Services are Node.js applications. MBaaS Services can be used by projects to integrate with back-end systems, for example, an Oracle integration service. MBaaS Services are associated with one or more projects and must be made available to apps otherwise an app in a project cannot access the MBaaS Service. For more information on MBaaS Services, see the [relevant product documentation](#).

Services are invoked from a Cloud App via the [\\$fh.service Cloud API call](#), for example:

```
$fh.service({
  guid: "PFi1ftKRBvlp-qSmgdc0eGe3",
  path: "/hello",
  method: "GET",
  params: {
    "hello": "world"
  }
}, function(err, data) {
  if (err) {
    return console.log(err);
  }
  return console.log(data);
});
```

To interact with an MBaaS Service from a local development environment, a mapping is required between:

1. The service guid (that is, the unique id of the service)
2. The hostname of the running instance of the service, that is being targeted

A new environment variable definition in the Gruntfile.js is required:

```
env : {
  options : {},
  // environment variables - see https://github.com/jsoverson/grunt-env
  for more information
  local: {
    FH_USE_LOCAL_DB: true,
    FH_SERVICE_MAP: function() {
      /*
```

```

    * Define the mappings for your services here - for local
    development.
    * You must provide a mapping for each service you wish to access
    * This can be a mapping to a locally running instance of the
    service (for local development)
    * or a remote instance.
    */
    var serviceMap = {
        'PFi1ftKRBvlp-qSmgdc0eGe3': 'http://127.0.0.1:8010'
    };
    return JSON.stringify(serviceMap);
}
}
},

```

In the above example, **FH\_SERVICE\_MAP** is added to the local environment variable definitions. The new variable is mapped to a function and stores the GUID **PFi1ftKRBvlp-qSmgdc0eGe3** which points to a service running locally on port 8010. To target a hosted instance of the service, use the *Current Host* URL of the service, found on the *Details* page in the Studio. All additional services are added to the **serviceMap** variable.



#### NOTE

**FH\_SERVICE\_MAP** environment variable is defined as a function rather than a simple string.

Users can define the service mappings as a standard JSON object rather than as a stringified JSON object. **FH\_SERVICE\_MAP** environment variable is set using a standard JSON object in the following example:

```
FH_SERVICE_MAP: '{"PFi1ftKRBvlp-qSmgdc0eGe3":"http://127.0.0.1:8010"}'
```

A working example can be seen [here](#).

## 4.7. EDITING THE CLOUD CODE



#### NOTE

When the Node.js Cloud API server is running, **grunt** is monitoring for all changes. **grunt** applies changes in real time.

Navigate to the Cloud App directory and open **./lib/hello.js**.

To add an additional parameter that sets the current time in the response, input a new parameter to both the GET and POST request. Replace both occurrences of:

```
res.json({msg: 'Hello ' + world});
```

with:

```
res.json({msg: 'Hello ' + world, 'timestamp': new Date().getTime() });
```

The will add a timestamp with the current time in milliseconds to the response. The reason there are two occurrences is that we have a route handler for both GET and POST requests.

Your Cloud App should automatically restart as soon as the file is saved. After saving the file, re execute the previous curl command:

```
curl -X POST http://localhost:8001/hello?hello=world
```

The response now includes a timestamp:

```
{"msg": "Hello World", "timestamp": 1412111429480}
```

## 4.8. EDITING THE CLIENT CODE

Navigate to your Client App directory and open `./www/index.html`. This is a basic web-page with an input field for a "name" and a button to call the Cloud. There is also a `cloudResponse` div that is used to populate the response from the Cloud Call.

Directly under the code "cloudResponse":

```
<div id="cloudResponse"></div>
```

Add a timestamp div:

```
<div id="timestamp"></div>
```

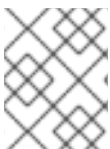
In `./www/hello.js`, a click handler is assigned to the "Say Hello" button. When triggered, the Cloud App's `/hello` endpoint is called using the `$fh.cloud` API.

To cater for the timestamp, open `./www/hello.js` and directly under the code:

```
document.getElementById('cloudResponse').innerHTML = "<p>" + res.msg + "  
</p>";
```

Add this code:

```
document.getElementById('timestamp').innerHTML = "<p>" + new  
Date(res.timestamp) + "</p>";
```



### NOTE

A new Date Object is created to accept the timestamp value returned from the cloud for the purposes of displaying the value.

## 4.9. VIEWING CHANGES IN THE APP STUDIO

Execute the following command in the client directory and also in the cloud directory:

```
git commit -am"Add timestamp parameter"  
git push origin master
```

The changes made locally are now visible within the App Studio.

## 4.10. ADVANCED DEVELOPMENT

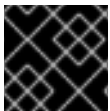
For advanced development that makes use of MBaaS APIs, for example, `fh.db()` and `fh.cache()`, additional software is required.

### 4.10.1. Using `fh.cache`

To develop locally using `fh.cache`, install [Redis](#) locally:

- RHEL / Fedora: see [Redis Download Page](#)
- Debian based distros: **`apt-get install redis-server`**
- Mac: **`brew install redis`** or **`sudo port install redis`**
- Windows: see [MSOpenTech Redis](#)

Once installed and running, calls to `fh.cache` in the Cloud Server will operate in the same manner as the RHMAP Cloud.



#### IMPORTANT

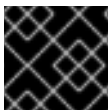
The default Redis port is used therefore there is no need to modify the Redis config file.

### 4.10.2. Using `fh.db`

To develop locally using `fh.db`, install [Mongo](#) locally:

- RHEL / Fedora: follow the instructions [here](#)
- Debian based distros: follow the instructions [here](#)
- Mac: follow the instructions [here](#) or **`brew install mongod`**
- Windows: follow the instructions [here](#)

Once installed and running, calls to `fh.db` in the Cloud Server will operate in the same manner as the RHMAP Cloud.



#### IMPORTANT

The default Mongo port is used therefore there is no need to modify the Mongo config file.



## CHAPTER 5. WORKING WITH THE PLATFORM USING FHC

### 5.1. MANAGING API KEYS WITH FHC

When certain FHC commands are executed, it needs to communicate with the platform using the user's identity. This requires the user to login to the platform using the **fhc login** command. User API keys provides an alternative way to use FHC. You don't have to run the login command anymore if you have set the user API key in FHC. FHC can use the user's API key to authenticate when communicating with the platform.

#### 5.1.1. Set User API Keys

In FHC, the command related to API keys management is **fhc keys**. If you run this command without any additional argument, you will be shown the command help.

To set a user API key for the current FHC target, you should enter the following command:

```
fhc keys user target eb8d9b9ea050b9b23fea59e50ba281c67f3715e5
```



#### NOTE

This requires you to have previously set the target using **fhc target**.

To check which API key is currently used for FHC, you can run this command

```
fhc keys user target
```

After setting a user API key in FHC, you can run other FHC commands as usual.

For example, you can list, create, update and revoke user API keys using FHC.

#### 5.1.2. List User API Keys

If you have logged in or set the user API key using FHC, you can see all the API keys associated with the current user by running this command:

```
fhc keys user list
```

If the output format is set to json in your FHC config, each key object will look like this:

```
{
  "key": "eb8d9b9ea050b9b23fea59e50ba281c67f3715e5",
  "keyReference": "1j6qcwiziRkbYy6sUUHDooAY",
  "keyType": "user",
  "label": "test 1",
  "revoked": "2012-04-20T15:18:17.258Z",
  "revokedBy": "1j6qcwiziRkbYy6sUUHDooAY",
  "revokedEmail": "dev@example.com"
}
```

#### 5.1.3. Create User API Keys

To create a user API key, run this command:

```
fhc keys user create test2
```

You should get a response similar to:

```
{
  "key": "22d0c0ac5990e8fd2c466c26db1a9eff8a171511",
  "keyReference": "1j6qcwiziRkbYy6sUUHDooAY",
  "keyType": "user",
  "label": "test2",
  "revoked": "",
  "revokedBy": "",
  "revokedEmail": ""
}
```

#### 5.1.4. Update User API Keys

To update a user API key using FHC, you can run this command:

```
fhc keys user update 22d0c0ac5990e8fd2c466c26db1a9eff8a171511 test3
```

You should get a response similar to:

```
{
  "key": "22d0c0ac5990e8fd2c466c26db1a9eff8a171511",
  "keyReference": "1j6qcwiziRkbYy6sUUHDooAY",
  "keyType": "user",
  "label": "test3",
  "revoked": "",
  "revokedBy": "",
  "revokedEmail": ""
}
```

#### 5.1.5. Revoke User API Keys

To revoke a user API key using FHC, you can run this command:

```
fhc keys user revoke 22d0c0ac5990e8fd2c466c26db1a9eff8a171511
```

You will be prompted to confirm the revoke action. After confirmation, you should see something similar to:

```
{
  "key": "22d0c0ac5990e8fd2c466c26db1a9eff8a171511",
  "keyReference": "1j6qcwiziRkbYy6sUUHDooAY",
  "keyType": "user",
  "label": "test3",
  "revoked": "2012-04-20T16:14:11.702Z",
  "revokedBy": "1j6qcwiziRkbYy6sUUHDooAY",
  "revokedEmail": "dev@example.com"
}
```

## 5.2. CREATE A SERVICE USING FHC

### Overview

This tutorial shows you how to create a service and associate it with a project. For more information regarding services, and their functionality, see the [Cloud Services](#) documentation.

### Requirements

The user must be a member of one or more teams with the following permissions:

- Project (View & Edit)
- Service (View & Edit)

For more information on Permissions, see [Teams & Collaborations](#).

#### 5.2.1. Log in to the platform

```
fhc login myUsername myPassword
```

#### 5.2.2. Create a Project

Now that you have successfully logged in to the platform, we will create a project that we will then associate a service with. In order to create the project, enter the following command:

```
fhc projects create serviceProject
```

To verify that the project has successfully been created, enter the following command to search for projects on the domain:

```
fhc projects | grep 'serviceProject'
```

This should return the newly created project. Notice that there are currently two apps in this project. A Client App, and a Cloud App.

#### 5.2.3. Create a Service

Services can be added to a project to allow the Client to access back-end functionality they would otherwise not have access to. Before we create a service, we will view a list of available templates. To view the service templates, enter the following command:

```
fhc templates services
```

This will output a list of all available service templates. We will add the PayPal service to the project. To do this, copy the service id of the PayPal service.

When creating a service, we must also specify a name. Enter the following command:

```
fhc services create paypalService paypal
```

This has successfully created a new service that can be associated with Projects.

### 5.3. VIEWING APP STATS WITH FHC

FHC can be used to access raw, JSON stats data for your apps.

To access your app stats (timers and counters) for a particular app, use the following command:

```
fhc stats <APP_ID> app <NUMBER_OF_RESULTS>
```

Where **APP\_ID** is the ID of your app as listed in an **fhc apps** command, and **NUMBER\_OF\_RESULTS** is the maximum number of results you wish to see returned. See below for a sample Stats response.

```
{
  interval: 10000,
  results: [
    {
      "ts": 1333107734000,
      "numStats": 6,
      "counters": [
        {
          "key": "DOMAIN_APPID_api_FUNCTIONNAME_active_requests",
          "value": {
            "value": 3,
            "valuePerSecond": 0.3
          }
        },
        {
          "key": "DOMAIN_APPID_api_FUNCTIONNAME2_active_requests",
          "value": {
            "value": 27,
            "valuePerSecond": 2.7
          }
        },
        {
          "key": "DOMAIN_APPID_api_FUNCTIONNAME3_active_requests",
          "value": {
            "value": 456,
            "valuePerSecond": 45.6
          }
        }
      ],
      "timers": [
        {
          "key": "DOMAIN_APPID_api_FUNCTIONNAME_request_times",
          "value": {
            "upper": 86,
            "lower": 63,
            "count": 17,
            "pcts": [
              {
                "pct": "90",
                "value": {
                  "mean": 76,
                  "upper": 86
                }
              }
            ]
          }
        }
      ]
    }
  ]
}
```

```

    ]
  },
  {
    "key": "DOMAIN_APPID_api_FUNCTIONNAME2_request_times",
    "value": {
      "upper": 99,
      "lower": 82,
      "count": 41,
      "pcts": [
        {
          "pct": "90",
          "value": {
            "mean": 88.66666666666667,
            "upper": 99
          }
        }
      ]
    }
  },
  {
    "key": "DOMAIN_APPID_api_FUNCTIONNAME3_request_times",
    "value": {
      "upper": 99,
      "lower": 75,
      "count": 2783,
      "pcts": [
        {
          "pct": "90",
          "value": {
            "mean": 79.1254547827,
            "upper": 99
          }
        }
      ]
    }
  }
]
}

```

## CHAPTER 6. WORKING WITH FORMS USING FHC

### 6.1. CREATE A FORMS PROJECT USING FHC

#### Overview

This tutorial will show you how to build an Forms Project using FHC. This project will consist of an Forms Client App, along with a Cloud App.

#### Requirements

The user must be a member of one or more teams with the following permissions:

- Project (View & Edit)
- Drag & Drop Apps (View & Edit)

For more information on Permissions, see [Teams & Collaborations](#).

#### 6.1.1. Create a Forms project

First we must create an App Forms Project. This can be done by specifying which template to use when creating a project. This will create a project containing a Hybrid Forms App.

```
fhc projects create FormsProject appforms_project
```

#### 6.1.2. Add a form

Now that the Project containing the Forms App has been created, the next step is to associate a form with that App. We will do this by specifying the path to the form file we wish to use.

```
fhc forms create <form.json>
```

For example:

```
fhc forms create ~/Desktop/forms/smallForm.json
```

#### 6.1.3. Associate a form with a project

Now that a form has been created on the Platform, we must now associate it with the app inside our project.

```
fhc forms apps update <project-id> <form-id>
```

For example:

```
fhc forms apps update 534572a3fb19a3983fd92c15 5579624572fb19ajsfig398c
```

Your App now has a form associated with it. Use the **fhc preview** command to view the current state of your form.

```
fhc preview <app-id>
```

This will load a preview of your app in the browser. You can see that the form has been successfully associated with the app, however it appears rather bare, as no theme has yet been assigned to make the form more visually appealing.

#### 6.1.4. Create a theme

A theme is created in a similar way to that of a form. You create the `theme.json` file and then specify the path to that file using the **fhc themes create** command.

```
fhc themes create <theme.json>
```

For example:

```
fhc themes create ~/Desktop/themes/CustomTheme.json
```

#### 6.1.5. Associate the theme with the app

Now that the theme has been added to the Platform, you must apply it to the project you created.



#### NOTE

Themes are associated with Projects, not individual Apps.

```
fhc themes app set <project-id> <theme-id>
```

For example:

```
fhc themes app set HQgUjokQi7jizH8T-7TD4SQQ 534578160663a687117a4bd1
```

At this stage you have a functioning Forms Project that contains a Client Forms App, and an Cloud App. The Client App has a form and a theme associated with it. Once again, you can preview using **fhc preview <app-id>**. You will notice that the form is a lot more visually appealing now that a theme has been associated with the project.

#### 6.1.6. Add another Forms App

If you wanted to add a forms based app to a project, you would do as follows:

```
fhc apps create <project-id> <app-title> [<app-template-id>]
```

For the template-id you specify the 'appforms\_client' template. This creates a Hybrid Forms App.



#### NOTE

To view all available App Templates, use the 'fhc templates apps' command.

So to add the Forms App to the project, enter the following into the terminal:

```
fhc apps create HQgUjokQi7jizH8T-7TD4SQQ Second_Form_App appforms_client
```

This has now successfully created another Form based App within your project. You can associate a form and theme with your app exactly as you did before.

## 6.2. CREATE A FORM USING FHC

### Overview

This tutorial explains how to create a form for use with an App Forms apps.

### Requirements

The user must be a member of one or more teams with the following permissions:

- Domain — Drag & Drop Apps
- Form (View & Edit)

For more information on Permissions, see [Teams & Collaborations](#).

#### 6.2.1. List available forms

Target the Platform domain using **fhc target**.

```
fhc target exampleDomain.feedhenry.com
```

Then log in using the **fhc login** command.

```
fhc login testUser@example.com password
```

To list all forms on a domain, use the **fhc forms** command.

```
fhc forms
```

This command will list all forms that you have access to based on the Permissions assigned to the logged-in User.

#### 6.2.2. Create a form

In order to create a form, you need to pass in a **form.json** object. Create a **form.json** object and take note of its path. The example **form.json** file used in this tutorial can be seen below.

```
{
  "description": "This is a test form",
  "name": "Test Form",
  "updatedBy": "testing-admin@example.com",
  "pageRules": [

  ],
  "fieldRules": [

  ],
  "pages": [
    {
      "name": "Page 1",
      "fields": [
        {
          "fieldOptions": {
```



```

        "definition":{
            "defaultValue":""
        },
        "required":false,
        "type":"text",
        "name":"Text",
        "helpText":"Text",
        "repeating":false
    },
    {
        "required":false,
        "type":"file",
        "name":"File",
        "helpText":"File",
        "repeating":false
    }
]
},
{
    "name":"Page 2",
    "fields":[
        {
            "required":false,
            "type":"text",
            "name":"Page 2 Text",
            "helpText":"Page 2 Text",
            "repeating":false
        }
    ]
}
],
"lastUpdated":"2014-01-22T16:51:53.725Z",
"dateCreated":"2014-01-22T14:43:21.806Z",
"pageRef":{
    "52dfd909a926eb2e3f000001":0,
    "52dff729e02b762d3f000004":1
},
"fieldRef":{
    "52dfd93ee02b762d3f000001":{
        "page":0,
        "field":0
    },
    "52dfd93ee02b762d3f000002":{
        "page":0,
        "field":1
    },
    "52dff729e02b762d3f000003":{
        "page":1,
        "field":0
    }
},
"lastUpdatedTimestamp":1390409513725,
"appsUsingForm":123,

```

```
{
  "submissionsToday":1234,
  "submissionsTotal":124125
}
```

To create the form above using **fhc**, we use the **fhc forms create** command. The command operates as follows:

**fhc forms create <form-file.json>**

A JSON form object needs to be passed as a parameter for the command. The path of the file is to be specified.

```
fhc forms create ~/Desktop/forms/form.js
```

This will recreate the given form on the Platform. After the form has been created, use the **fhc forms get** command to display the form. Then copy and paste the code into your desired directory in case you want to edit the form in future. Alternatively, you could print the result of the **fhc forms get** command to file by issuing the following command:

```
fhc forms get <app-id> > ~/Desktop/formFile.txt
```

### 6.2.3. Update form

At some stage during development, you may want to update a form. To do this, edit whatever content you like in the file where you saved your form earlier. For example, you may wish to add another text field, or apply a rule to a page. You can then use the **fhc forms update** command to update an existing form.

**fhc forms update <path\_to\_form>**

For example:

```
fhc forms update ~/Desktop/formFiles/form
```



#### NOTE

If you pass in the path to the form file you used to create the form initially, the update command will actually duplicate the form, as that file has no id assigned to it. This means that instead of updating the form, a new instance will be created, and a new id will be assigned.

The new changes will now have taken effect.

To learn how to associate a form with a project, see the [Associate a form with a project](#) section of the [Create an App Forms project](#) tutorial.

## 6.3. CREATE A FORM SUBMISSION USING FHC

### Overview

This tutorial will inform you of how to list, retrieve, and generate form submissions.

### Requirements

The user must be a member of one or more teams with the following permissions:

- Domain — Drag & Drop Apps
- Form (View & Edit) — Submission (View & Edit)

For more information on Permissions, see [Teams & Collaborations](#).

In order to complete this tutorial, you will have to already have created a form. See [Form Creation](#) for a guide to creating a new form.

### 6.3.1. Listing submissions

In order to list submissions for a form, you can use either the **fhc submissions** or the **fhc submissions list** command.

```
fhc submissions list
```

This will return a list of all submissions for all forms you have access to.

### 6.3.2. Return a specific submission

The **fhc submissions get** command can be used to return a specific submission based on its id.

```
fhc submissions get <submission-id>
```

This will return that specific submission. You can also save the submission in .pdf form simply by appending a filename with the **.pdf** suffix.

```
fhc submissions get <submission-id> <filename>.pdf
```

For example:

```
fhc submissions get 534cf14de078a2c47291e5b3 savedSubmission.pdf
```

### 6.3.3. Submission status

You can check the status of a submission by using the **fhc submissions status** command in conjunction with a specific submission id.

```
fhc submissions status 5335bc00d4598fdb5cae04f7
```

### 6.3.4. Submitting a Submission

You can submit data for a submission by passing a submission.json file into the **fhc submissions submitdata** command.

```
fhc submissions submitdata <submission.json>
```

For example:

```
fhc submissions submitdata ~/Desktop/templateForSubmission.json
```

You can complete a submission by using the **fhc submissions complete <submission-id>** command to push mark the submission as finished.

```
fhc complete submissions complete 5335bc00d4598fdb5cae04f7
```

Just as you can save submissions as PDF's, you can also save submissions as zip files. This can be useful if a particular project has a number of submissions.

```
fhc submissions export file=<zip-file> app=<project-id> || form=<form-id>
```



#### NOTE

The project parameter is passed into the app field **app=<project-id>**.

```
fhc submissions export file=submissions.zip app=H_DNbFNJWS9uZIt7LJ0kut20
```

## 6.4. CREATE A FORM THEME USING FHC

### Overview

This tutorial will teach you how to create a theme for a form using FHC.

### Requirements

The user must be a member of one or more teams with the following permissions:

- Domain — Drag & Drop Apps
- Theme (View & Edit)

For more information on Permissions, see [Teams & Collaborations](#).

### 6.4.1. List all themes

Target your domain using the **fhc target** command, and log in using **fhc login**.

To list all themes on a domain, you can use the **fhc themes** command.

### 6.4.2. Create a theme

To create a theme, you must use the **fhc themes create** command. This command requires you to pass a json file as a paramter.

```
fhc themes create <theme-file.json>
```

For example:

```
fhc themes create ~/Desktop/themes/theme.json
```

This will create the specified theme on the Platform. It will also assign an id to the theme.

When the theme is successfully created, it will be output to the console.

Save the output to a file and take note of its path. This will be used to update a theme. To save the output to file

### 6.4.3. Associating a theme with an app

Before adding a theme to an app, you must have an app associated with a project. If you do not know how to add an app to a project, please refer to guide [Working with Projects and Apps](#).

Themes are associated with apps using the **fhc themes app set** command.

```
fhc themes app set <app-id> <theme-id>
```

For example:

```
fhc themes app set x5KmSy0gXPZ0vpD_hHqC0wZe zRdUezAD3l11huedCk-WJswZ
```

### 6.4.4. Updating a theme

If you want to update a theme at any stage, you can do so via the **fhc themes update** command. Navigate to where you saved the theme after you created it. Make whatever changes you wish, and then save the file. You can now update the theme.

```
fhc themes update ~/Desktop/themes/theme.json
```

This will load in the changes you have made. The newly updated theme will once again be output to the console.