Red Hat Mobile Application Platform 4.3
Local Development Guide

For Red Hat Mobile Application Platform 4.3

Red Hat Customer Content Services

Red Hat Mobile Application Platform 4.3 Local Development Guide

For Red Hat Mobile Application Platform 4.3

Legal Notice

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 4.3.

Table of Contents

CHAPTER 1. SETTING UP FHC

1.1. INSTALLING FHC

Overview

FHC is the Red Hat Mobile Application Platform (RHMAP) command line interface. It is a Node.js based CLI for accessing the RHMAP APIs. Almost all functionality which is exposed through the RHMAP Studio is available via FHC.

As FHC is a Node.js module, it can also be included in your own Node.js applications - allowing access to the Platform to be scripted and automated. This allows interaction with the Platform to be integrated into automated processes such as build systems and Continuous Integration systems.

Requirements

To use FHC, you will first need to have Node.js & NPM installed.

To install Node.js, follow the instructions on nodejs.org for your OS. This will add 2 command line applications: **node** and **npm**.

1.1.1. Install FHC

To install FHC, you will need to execute the following in a terminal/command prompt:

```
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 — the central registry of Node.js modules.

The -g flag tells nom 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 zsh, do a **hash -r** to pick it up).

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, including completing app identifiers when performing actions on a single app. To install the FHC bash completion script, execute the following:

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

If you're using an alternative shell to bash, you should append the output of **fhc completion** to the relevant file for example, **~/.zshrc** for zsh.

1.1.2. Usage

To get started, set the target and then login:

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

To list your projects, use:

```
fhc projects
```

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.

See **`fhc help config`** for much much 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

- Adding an SSH Key via FHC
- Working with Projects & Apps
- Local App Development

1.2. SSH KEY SETUP

Overview

Before you can clone a git repo, you must first upload your SSH public key (for more information on Public-key cryptography, check out this Wikipedia article).

This tutorial will show you how to generate a public/private key pair (if required) and upload the SSH Public Key to the Platform via FHC.

Requirements

You should install FHC before completing this tutorial.

1.2.1. Checking for an existing Key

To check whether or not an SSH key has been generated, open **`Terminal`** on Linux or Mac, or **`Git Bash`** on Windows.

Enter **`ls -la ~/.ssh`**

This will list the files in the .ssh directory. If you do not see either of these files, then proceed to Generating an SSH Public Key, as you must generate an SSH public key. If you see files named either *id_rsa.pub* or *id_dsa.pub*, then the key has been generated and you can skip to Adding an SSH Public Key to the Platform via FHC.

1.2.2. Generating an SSH Public Key

In order to generate a key, we will use a tool called ssh-keygen. Open **`Terminal`** on Linux or Mac, or **`Git Bash`** on Windows.

Enter the following:

```
ssh-keygen -t rsa -C "your_email_address@example.com"
```

After you have entered this command you will be asked where to save the key. Press the enter key to use the default location.

You will then be prompted to enter and confirm a password for the private key. When complete, the public & private key should be written to **`~/.ssh/id_rsa.pub`** & **`~/.ssh/id_rsa`**.

1.2.3. Adding an SSH Public Key to the Platform via FHC

Using FHC, log in to the Platform. When adding a public key, you must specify both a name and a key file. To add a key, enter the following command:

```
fhc keys ssh add <label> <key-file>
```

For example:

```
fhc keys ssh add myKey ~/.ssh/id_rsa.pub
```

You can verify that the key has been added correctly by listing out all ssh keys for the user.

```
fhc keys ssh
```

You should now be able to see your recently added key.

Note

If you need to use multiple SSH keys, follow our guide for configuring your local SSH to use multiple keys.

1.2.4. Adding an SSH Public Key to the Platform via Studio

Follow these steps to add your public SSH key to the Platform using the Studio:

1. Log into the Studio.
2. Click on the portrait button located on the top right of the screen.
3. Select **`Settings`**.
4. Select **`SSH Key Management`**. A list of previously uploaded SSH Keys will be displayed.
5. Select **`Add New Key`**.
6. Paste your public key into the **`Public Key`** field.
7. Select **`Upload Public Key`**.

You should now be able to see your recently added key in the list of keys.

1.2.5. Using multiple SSH keys for different domains/projects

If you need to access multiple domains in the Red Hat Mobile Application Platform (RHMAP) as different users you will need to use seperate SSH keys when accessing the Git repositories, since each SSH key can only be associated with a single user. You will need to tell Git/SSH which key you want to apply to each host. This is normally done via an SSH config file.

1.2.5.1. Details

SSH will normally attempt to use all the identity files available to it, and if several of your ssh keys are valid in the cluster, it may use one that's valid for the cluster but which doesn't have access to the project that you are trying to clone.

You can update your local SSH config file, normally **`~/.ssh/config`**, to list the identities to use for a particular host, but you should also add an **`IdentitiesOnly yes`** clause, to tell SSH to only use the specified identity file for a particular host, rather than all the identity files.

So if you want to access App repositories as different users, create SSH keys for each of the two users. Upload each SSH key to its corresponding user's account.

In the example below, the SSH key file **`/Users/jbloggs/.ssh/key_for_domain1_id_rsa`** has been uploaded to my user in **`domain1.redhatmobile.com`**, and **`/Users/jbloggs/.ssh/key_for_domain2_id_rsa`** has been uploaded to the user in **`domain2.redhatmobile.com`**

To allow access to the App repositories:

```
git@domain1.redhatmobile.com:domain1/jbAdvTest-mmCord1.git
git@domain2.redhatmobile.com:domain2/jbAdvTest-mmCord2.git
```

Your ssh config file would look something like the following:

```
Host domain1.redhatmobile.com
HostName domain1.redhatmobile.com
IdentitiesOnly yes
IdentityFile /Users/jbloggs/.ssh/key_for_domain1_id_rsa
Host domain2.redhatmobile.com
HostName domain2.redhatmobile.com
IdentitiesOnly yes
IdentityFile /Users/jbloggs/.ssh/key_for_domain2_id_rsa
```

You can then clone the repos using the git commands:

```
git clone git@domain1.redhatmobile.com:domain1/jbAdvTest-mmCord1.git
git clone git@domain2.redhatmobile.com:domain2/jbAdvTest-mmCord2.git
```

Because of the ssh config file, the appropriate SSH key will be used for each domain, for further information on the configuration of ssh, you should check the documentation for your local installation of SSH, depending on your Operating System, it may be available from the command line **`man ssh_config`**.

Note that the above configurations and domain names are examples only and do not refer to actual domains.

CHAPTER 2. GETTING STARTED WITH FHC

This document helps you get started with creating a Project in the Red Hat Mobile Application Platform (RHMAP) via the RHMAP command line tool (fhc). This document will step you through how to create a new Project based on a sample template provided. Once that project has been created you will use fhc to clone the project to your local machine, npm to install dependencies for local development and grunt to start your new Hybrid Client App and Node.js Cloud App locally for development via the browser.

Video Tutorial

- RHMAP - Create an App

Prerequisites

- Section 1.1, “Installing FHC”
- Section 1.2, “SSH Key Setup”
- Node.js and Git installed locally

2.1. TARGETING YOUR DOMAIN

First you will need to target your RHMAP domain via the command line. Open a command line application e.g. Terminal on Linux & MacOS, or a command prompt for Windows. Enter the following command, 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>
```

Next you will be required to enter your **password**.

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

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

2.2. CREATING YOUR PROJECT

Before we create our project, let us take a look at some sample project templates provided by the Platform. Enter the next command to see list of project templates available.

```
fhc templates projects
```

For the purpose of this guide we will be using the Hello World Project template.

Now we are going to create a new Project via command line, based on the **hello_world_project template** provided. Enter the following command to create a new Project in the Platform. Replace the **project name** with a desired name of your choice. Once

successful a JSON response will be print to screen. This JSON object contains all project information. Find the project GUID and copy for next step.

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

Note

Notice we are passing in the name of the template to create the project from as the last param.

This command will take a short time to complete. This is the JSON response that will be returned containing all information about the project you have just created.

If at any time you need to review your project information you can do this by entering the next command, replacing **`project guid`** the project ID you copied in the previous step.

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

This command will return the JSON object shown in the last step, containing all project information. See below.

```
{
  "type": "PROJECT",
  "template": null,
  "sysCreated": 1397636395777,
  "guid": "jJHtgZXQPJxXDFJO2pQzMP-d",
  // ...
  "apps": [
    {
      "type": "client_hybrid",
      "description": "",
      "domain": "testing",
      "template": null,
      "email": "joebloggs@feedhenry.com",
      //...
    }
  ],
  "templateId": "default",
  "jsonTemplateId": "hello_world_project"
}
```

Now that you have created you project in the cloud we are going to clone the entire project to your local machine. This step is chaining 3 commands to copy project to local machine. Replace the **`project name`** with the name you provide in step 3. Replace **`project guid`** with the ID that you copied in step 3.

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

Now, we will have a look at the MBaaS example or Cloud App. Enter the next command replacing the **`project name`** with your project name.

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

Next we will install any dependencies for the Cloud App. Enter the next command to install dependencies via npm, and and the grunt command line interface globally.

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

Your Cloud App instance is now ready to be deployed locally on your machine.

2.3. DEPLOY CLOUD APP

To start your cloud server locally enter the following command.

```
grunt serve
```

You should see some information output to the console to indicated that the Cloud App is running via a **`port 8001`** on your machine. To test that cloud instance is running locally enter the following curl request to your localhost.

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

And you should receive a response similar to…

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

If you received the above response from this curl attempt, your Cloud App is running locally. Now let us get our client side app running locally.

2.4. DEPLOY CLIENT APP

Next we will change directory to the Client App. Notice the following command is also a chained command. The first moves back up a level of the directory tree, the next changes down to the client directory and finally we list contents of directory.

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

Remember to replace **`project name`** with the name of your project.

Next we will install all dependencies for the Client App using npm. To do so, enter the following command.

```
npm install .
```

Your client side app is now ready to be deployed locally, we can run our Client App locally by entering the following command.

```
grunt serve:local
```

A window will open in your default browser once the previous command has been successful.

2.5. SUMMARY

The Platform hosts all Git repositories. This means that when creating any project the apps within this project will have its own Git repo. You can perform normal Git procedures and processes by changing directory into either the Client or Cloud App examples.

You are now setup to develop your RHMAP project locally.

CHAPTER 3. WORKING WITH PROJECTS AND APPS

Overview

This tutorial will cover how to create a project using FHC. It will also cover how to add apps to a project.

Requirements

Before starting this tutorial, you should have completed the following tutorials:

- Installing FHC

3.1. LOG IN TO THE PLATFORM

First open up the **`terminal`**. In order to log in to the Platform, you must first set the target domain that you wish to log in to.

To see a list of target domains available for selection, enter the **`fhc targets`** command in the terminal.

```
fhc targets
```

`fhc targets` displays a list of domains that you can select as your target. To specify which domain you wish to target, use the **`fhc target`** command.

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

This selects the 'exampleDomain.redhatmobile.com' domain as the target domain. Now when you log in to the Platform, you will be logging into this domain.

Now that a domain has been targeted, the next step is to log in. To do this, use the **`fhc login`** command.

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

To login, use the **`fhc login`** command followed by the username and password.

Now that you have logged in, you are free to access the platform within the restrictions of the Teams the User is a Member of. Team Permissions determine the level of access Users have within the Platform. For more information on Teams and Permissions, see Teams & Collaborations.

3.2. LIST EXISTING PROJECTS

Once logged in, you can see a list of existing project by using the **`fhc projects`** command. In the Platform, a project is used as a container for grouping related apps together and all apps must be created within a project.

```
fhc projects
```

This will return a list of existing projects, ordered by lasted modified date - similar to the one below. If you do not yet have any projects, this list will be empty.

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

If you wish to return the project list in raw JSON format, append the --json flag to the end of the command. This flag works for most FHC commands and is useful for piping output from FHC into other tools or into files. The JSON format typically returns significantly more data that the standard tabular format, so use with caution.

```
fhc projects --json
```

3.3. CREATING A PROJECT

Before we can create any apps, we must first create a Project to contain them. To learn more about Projects, see the Projects Documentation. In order to create a project, use the **`fhc projects create`** command. When using this command, you must specify a project name.

While it is possible to create bare projects (that is, projects with no apps inside them), it is often prefereable to create a new project from one of the pre-defined templates. Project templates allow you to bootstrap your development by cloning the project template. This typically provides at least one Client App and one Cloud App.

If you wish to use a template from the list of available templates, specify the *template id* after the project name. Use the **`fhc templates projects`** command to view a list of project templates.

```
fhc templates projects
```

You can then use a template as a starting point for a project. In this tutorial, we are going to use the **`hello_world_project`** template as our starting point. This is a basic project with one Client App and one Cloud App, which uses the standard Hello World paradigm to demonstrate basic functionality.

```
fhc projects create helloWorld hello_world_project
```

The example above creates a new project based on the **`hello_world_project`** template. The response from the **`fhc projects create`** command is a JSON Object which represents the project and apps which have just been created. This output can be ignored for now.

To verify that the project has been successfully created, enter the following command:

```
fhc projects
```

This will list all projects on the domain. Alternatively, you could just search for your specific project

by piping the output of the **`fhc projects`** command to the grep command:

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

The response from the above command (when the output is piped through grep) will be similar to this:

```
| XME5iUr2VoBV3DbXrVF7qApG | helloWorld  | 2  | 3 minutes ago |
```

Since we used grep, the table heading has been removed. The headings associated with this output (as described above in 'Creating a Project') are: projectId, title, number of apps, and when it was last updated.

From the console output you will have seen that your recently created project contains 2 apps. It contains both a Client App, and a Cloud App for it to communicate with. In order to list all apps within a given project, you can first use the **`fhc projects`** command to list the projects, select a projectId - or **`guid`** (Global Unique IDentifier) as it is commonly referred to - and then use that guid in conjunction with the **`fhc apps`** command to list all apps for a specific project.

```
fhc projects # lists all projects. From here you can select a projects guid

fhc apps <the_selected_project_guid>
```

This will list all the apps in the project whose guid you specified. The output will be include the following information:

- 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 - They type of the app. Used by the Platform to distinguish 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

As you saw earlier, there are two apps in your newly created project, both a Client App, and a Cloud App. Typically, apps in the same project will be related to each other - for example, a native iOS and a native Android app, a hybrid mobile app and a Web Portal may all co-exist within the same project. Apps which have no relationship with each other should be housed in separate projects.

You can add as many apps as you like to a project. Apps are normally added from an existing template (similar to how we created a project from a template). To see a list of App Templates, enter the following command:

```
fhc templates apps
```

This will display a list of all available app templates. There are a selection of templates available for both Client and Cloud apps. In order to add an app to a project, you will need to specify the project id, the name for the app, and the app template. If no template is specified, a blank Client App will be selected as default.

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

```
fhc app create --project=XME5iUr2VoBV3DbXrVF7qApG --title='My Native iOS
App' --template=blank_native_ios_client
```

This will successfully create a new app within the specified project. If you once again list all apps in your project, you will now see the newly created third app listed.

3.5. NEXT STEPS

- Local App Development
- Building App Binaries

CHAPTER 4. DEVELOPING CODE LOCALLY

Overview

This tutorial describes how to set up your local environment for developing a mobile application with the Red Hat Mobile Application Platform (RHMAP), both the Mobile App and the Node.js Cloud Server development.

Requirements

Before starting this tutorial, you should have completed the following tutorials:

- Installing FHC
- SSH Key Setup
- Working with Projects & Apps

4.1. DEVELOPMENT TOOLS

There are a few additional tools required for local development - over and above FHC, Node.js and NPM (as described in Installing FHC). This tutorial assumes you have already installed these basic tools via the **`Installing FHC`** tutorial.

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

To Install git, follow the instructions for your preferred 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, visit http://brew.sh/

4.1.2. Grunt

"The JavaScript Task Runner"

- Getting Started
- Plugins

Grunt is available via NPM. To install, simply enter the following command

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

All FeedHenry Template Apps use grunt for local development and as the task runner.

4.2. CLONING THE SOURCE CODE

The following assumes you have completed the SSH Key Setup tutorial as well as the Working with Projects and Apps tutorial and have created the 'Hello World' Project.

First, you should create a new directory to house your project development - for example,:

```
mkdir helloworld
cd helloworld
```

The easiest way to clone the source code for all apps in your project is to use the **`fhc projects clone <project-guid>`** command - for example,:

```
fhc projects clone XME5iUr2VoBV3DbXrVF7qApG  # <== Replace this guid with
your own one
```

This will clone the apps within the specified project. Output similar to the following should be seen:

```
Cloning into 'helloWorld-Hello-World-Cloud-App'...

Cloning into 'helloWorld-Hello-World-Client'...
```

Once complete, execute the **`ls`** command to see the new directories which have been created:

```
$ ls
helloWorld-Hello-World-Client   helloWorld-Hello-World-Cloud-App
```

4.3. ACCESSING CLOUD APPS DURING DEVELOPMENT

There are two ways to access Cloud Apps during development:

- access the *cloud instance* running in an environment on the Platform, or
- run the Cloud App in a *local instance* of Node.js.

4.3.1. Using Cloud Runtime

By default, Client Apps are pointed to **`http://localhost:8001`** as the server for local development. In order to develop against the Cloud App running on the Platform, follow these steps.

Find out the current host of the Cloud App running in a particular environment:

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

This command will return a response similar to the following:

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

Open **`Gruntfile.js`** of the Client App in development and look for the variable **`default_local_server_url`**. Change its value to that returned by the **`fhc app hosts`** command in the previous step.

Requests from a locally executed Client App will now be sent to the Cloud App running in the Platform.

4.3.2. Using Local Runtime

Note

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

To run the Cloud Server code locally:

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

This will bring us to the root of our Cloud App.

```
npm install
```

This will install all the dependancies for running our Cloud app. This may take up to a few minutes time to complete the first time, but will be much quicker on subsequent runs.

```
grunt serve
```

This will start up the Node.js server. Output similar to the following should be displayed in the console:

```
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 terminal window is now being held by the Node.js process, so you will need to open another terminal window to proceed.

Your Node.js Cloud Server is now running. you can verify this with curl:

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

If the request is successful, you should see a response similar to this:

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

Try changing the input parameter from **`world`** to your own name and see that the response also changes.

The Gruntfile.js has several useful commands out of the box - take a look through the **`Gruntfile.js`** file to get familiar with the additional options, 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
```

For more information on Grunt, see the **`Grunt-ReadMe.md`** file, or run **`grunt --help`**.

4.4. RUNNING THE CLIENT LOCALLY

To run the Client App locally, and have it talk to our locally running Cloud Server

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

This will bring us into the root of our Client App

```
npm install
```

This will install the client dependancies - mainly around grunt

```
grunt serve:local
```

This will start up a web browser session and tell it to try and contact a local Node.js Cloud App on port 8001 by default, so it should be able to hit our locally running server without having to make any modifications. Output similar to this should be observed:

```
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...
```

For more information on Grunt, see the local README.md for the 'Hello World Client' Template App.

4.5. WORKING WITH MBAAS SERVICES LOCALLY

MBaaS Services are Node.js applications which can be used by projects to integrate with back-end systems for example, an Oracle integration service. Services are associated with one or more projects to make them available to Apps in that project. For more information on service, check out

the relevant product documentation

Services can be invoked from a Cloud App via the $fh.service Cloud API call - e.g:

```
$fh.service({
  guid: "PFi1ftKRBvlp-qSmgdcOeGe3",
  path: "/hello",
  method: "GET",
  params: {
    "hello": "world"
  }
}, function(err, data) {
  if (err) {
    return console.log(err);
  }
  return console.log(data);
});
```

When developing locally, it is often useful to be able to work against a local version of a service. Equally, it can sometimes be useful to be able to target a remote instance of a service from a local development version of a Cloud App.

In order to be able to interact with an MBaaS Service from a local development environment, we need to set up a mapping between the service guid (that is, the unique id of the service) and the hostname of the running instance of the service we wish to target. To do this, we will add a new environment variable definition to our Gruntfile.js for local development. For example:

```
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-qSmgdcOeGe3': 'http://127.0.0.1:8010'
      };
      return JSON.stringify(serviceMap);
    }
  }
},
```

In the above example, the **FH_SERVICE_MAP** variable is being added to the local environment variable definitions. For a working example of this, take a look at the hello world cloud app Gruntfile.js. This new variable is mapped to a simple function, within which we are declaring that the service with the GUID **PFi1ftKRBvlp-qSmgdcOeGe3** is running locally on port 8010. If we wanted to target a hosted instance of the service, we could use the *Current Host* URL of the service, found on the *Details* page in the Studio.

You can add as many services as you wish to the **`serviceMap`** variable.

Note that the **`FH_SERVICE_MAP`** environment variable is defined as a function rather than a simple string to allow use to define the service mappings as a standard JSON object rather than as a stringified JSON object. That is, the following definition for the **`FH_SERVICE_MAP`** environment variable would also work:

```
FH_SERVICE_MAP: '{"PFi1ftKRBvlp-qSmgdcOeGe3":"http://127.0.0.1:8010"}'
```

4.6. EDITING THE CLOUD CODE

Now that we have our Node.js Cloud API server running, let's make some minor changes to the code. Note that the Node.js code is being monitored by grunt for any changes, so we do not need to restart our app to see our changes. We will add an extra parameter to the response with the current time.

First, navigate back to your Cloud App directory and open the file **`./lib/hello.js`**

Within this file, replace both occurrences of:

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

with the following:

```
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 should now include the timestamp, similar to this:

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

4.7. EDITING THE CLIENT CODE

We will now modify the client code to display the new timestamp parameter being returned from the cloud code.

First, navigate to your Client App directory and open the file **`./www/index.html`**. It's a pretty basic web-page, with an input field for a "name", and a button to call the Cloud. There's also a cloudResponse div which we populate with the response from the Cloud Call.

We will make one small edit to this file to add a new div beneath the "cloudResponse" div to populate with the new timestamp parameter which our Cloud Call is now returning. Directly below:

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

And add a new timestamp div:

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

We will need to make another small tweak to our app to populate this timestamp area. Open the **./www/hello.js** file.

In this file, we can see a click handler is being setup for the "Say Hello" button - when the button is clicked, we call the Cloud App's /hello endpoint by using our $fh.cloud API. We've now modified our Cloud App to return a timestamp in this response, so lets make the following change to populate our new div with this timestamp. Directly below:

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

Add the following code:

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

Note that we create a new Date Object from the timestamp returned from the cloud for display purposes.

4.8. VIEWING CHANGES IN THE APP STUDIO

In order to view the changes made locally within the App Studio, we will need to commit and push the updates using git.

From within both the client and cloud directories, execute the following commands:

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

The changes made locally should now be visible within the App Studio.

4.9. ADVANCED DEVELOPMENT

For more advanced development, which makes use of MBaaS APIs such as **fh.db()** and **fh.cache()** you will need to install additional software.

4.9.1. Using fh.cache

As mentioned above, if you want to develop locally using fh.cache, you first need to 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 Redis is installed and running, calls to fh.cache in your Cloud Server code will work locally exactly as they would when running in the RHMAP Cloud. Note the default Redis port is used automatically, there is no need to modify Redis config.

4.9.2. Using fh.db

As mentioned above, if you want to develop locally using fh.db, you first need to install Mongo locally:

- RHEL / Fedora: follow the instructions here
- Debian based distros: follow the instructions here
- Mac: follow the instructions here or **`brew install mongodb`**
- Windows: follow the instructions here

Once Mongo is installed and running, calls to fh.db in your Cloud Server code will work locally exactly as they would when running in the RHMAP Cloud. Note that the default Mongo port is used automatically, there is no need to modify Mongo config.

4.10. NEXT STEPS

- Building App Binaries
- FHC CLI

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. BUILDING AN APP BINARY

Overview

This tutorial will cover how to build a mobile app binary using FHC. Building App Binaries with the Red Hat Mobile Application Platform (RHMAP) makes use of the cloud hosted "Build Farm" which is used to turn source code into mobile apps. If you have local developer tools installed (such as Xcode or ADT), you are free to build app binaries locally yourself. However, using the RHMAP Build Farm allows you to automate the build process, maintain a history of previous builds and build apps for platforms which you may not have developer tools for (for example, building iOS binaries from a Linux OS).

Requirements

Before starting this tutorial, you should have completed the following tutorials:

- Installing FHC
- Working with Projects & Apps

5.3.1. fhc build command

There are a number of parameters that need to be specified to perform a build using fhc. This will vary depending on the type of build, and the platform for which you are building. During the Working with Projects and Apps tutorial, we created a cordova hybrid app as part of the project. We will now use that app to perform a build for Android.

The fhc build command takes the following parameters:

- **`project=<project-id>`** — the guid of the project.
- **`app=<app-id>`** — the guid of the Client App to build app.
- **`cloud_app=<cloud-app-id>`** — the guid of the Cloud App which the Client App should connect to.
- **`tag=<tag>`** — the Semver tag to use for this build. See Connections for more details.
- **`destination=<destination>`** — the type of build — e.g android, iOS, windowsphone.
- **`version=<version>`** — the specific OS version to target for the destination.
- **`config=<config>`** — the type of build to perform — 'debug' (default), 'distribution' or 'release'.
- **`keypass=<private-key-password>`** — the password for the private key used for building (only needed for release builds).
- **`certpass=<certificate-password>`** — the password for the certificate used for building (only needed for release builds).

- **`download=<true|false>`** — whether or not to download the generated binary.
- **`provisioning=<path-to-provisioning-profile>`** — the path to the provisioning profile to be used for the build.
- **`cordova_version=<cordova-version>`** — the version of cordova to use.
- **`environment=<environment>`** — the id of a target environment — e.g dev.

Not all of these parameters are required for all build types. For more information use the **`fhc build --help`** command.

When building an app, we must tell fhc the project the app is in as well as the app id. You must then enter the id of the Cloud App that the Client App speaks to. Client Apps are pointed towards Cloud apps via a connection. Clients can communicate with various Cloud apps via a number of different connections, and so when performing a build you must also specify the connection tag you wish to use.

```
fhc build project=XME5iUr2VoBV3DbXrVF7qApG app=XME5iNsoeRA2xvbmaDYMCdsi
cloud_app=XME5iHzfwW9hcGw6_F7Eiwvf tag=0.0.3 destination=android
config=debug keypass= certpass= download=true
```

When the build has been completed, you will get confirmation output similar to what is displayed below.

```
Download URL: Download URL: https://ngui-
demo.sandbox.feedhenry.com/digman/android-v3/dist/3e0837f2-76b6-4512-
881c-bd3bf427929d/android~4.0~7~HelloWorldClient.apk?
digger=diggers.digger206u
```

5.3.1.1. Artefacts

After a build has been successful, a build artefact is added to the build history. These artefacts allow users to re-download an app without having to perform the build again.

To list all artefacts for a given app, you must use the **`fhc artefacts <projectId> <appId>`** command.

```
fhc artefacts XME5iUr2VoBV3DbXrVF7qApG XME5iNsoeRA2xvbmaDYMCdsi
```

A list will be displayed containing all artefacts for the specified app.

5.3.2. Next Steps

- FHC CLI

5.4. 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 forms_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. Create a form

Target your desired 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 Creatation 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 submissons 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_DNbFNJWS9uZIt7LJOkut20
```

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

To add a theme to an app, you must first have first associated an app with a project. If you do not yet know how to add an app to a project, you can learn how to do so here.

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 x5KmSy0gXPZOvpD_hHqCOwZe 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.