



Red Hat Mobile Application Platform ホ スト型 3

ローカル開発ガイド

Red Hat Mobile Application Platform ホスト型 3 向け

Red Hat Mobile Application Platform ホスト型3 ローカル開発ガイド

Red Hat Mobile Application Platform ホスト型 3 向け

法律上の通知

Copyright © 2017 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.

概要

本書は、Red Hat Mobile Application Platform ホスト型 3 で RHMAP CLI ツールを使用し、アプリを開発するためのローカル環境を設定するためのガイドです。

目次

第1章 FHC の設定	4
1.1. FHC のインストール	4
1.1.1. FHC のインストール	4
1.1.1.1. コマンド自動入力 (Linux および Mac のみ)	4
1.1.2. FHC の使用	5
1.1.2.1. 設定	5
1.1.3. 次のステップ	5
第2章 SSH キーの設定	7
2.1. 既存 SSH 公開キーの確認	7
2.2. SSH 公開キーの生成	7
2.3. FHC でプラットフォームに SSH 公開キーを追加する	7
2.4. STUDIO でプラットフォームに SSH 公開キーを追加する	8
第3章 FHC の使用	9
3.1. FHC での RHMAP ドメインのターゲット設定	9
3.2. 新規プロジェクトの作成	9
3.3. クラウドアプリのデプロイ	11
3.4. クライアントアプリのデプロイ	11
3.5. サマリー	12
第4章 プロジェクトおよびアプリに関する作業	13
4.1. プラットフォームへのログイン	13
4.2. 既存プロジェクトの一覧表示	13
4.3. プロジェクトの作成	14
4.4. プロジェクトにアプリを追加する	15
4.5. 次のステップ	16
第5章 ローカルでのコード開発	17
5.1. 開発ツール	17
5.1.1. Git	17
5.1.2. Grunt	17
5.2. ソースコードのクローン	18
5.3. 別の GIT リポジトリからコードを使用する	18
5.4. 開発中のクラウドアプリへのアクセス	19
5.4.1. クラウドランタイムの使用	19
5.4.2. ローカルランタイムの使用	19
5.5. クライアントをローカルで実行する	20
5.6. ローカルでの MBAAS サービスの作業	21
5.7. クラウドコードの編集	22
5.8. クライアントコードの編集	23
5.9. APP STUDIO での変更の確認	23
5.10. 高度な開発	24
5.10.1. fh.cache の使用	24
5.10.2. fh.db の使用	24
5.11. 次のステップ	24
第6章 FHC を使用したプラットフォームでの作業	25
6.1. FHC による API キーの管理	25
6.1.1. ユーザー API キーの設定	25
6.1.2. ユーザー API キーの一覧表示	25
6.1.3. ユーザー API キーの作成	26
6.1.4. ユーザー API キーの更新	26

6.1.5. ユーザー API キーの取り消し	26
6.2. FHC を使用したサービスの作成	27
6.2.1. プラットフォームへのログイン	27
6.2.2. プロジェクトの作成	27
6.2.3. サービスの作成	27
6.3. アプリバイナリーのビルド	28
6.3.1. fhc build コマンド	28
6.3.1.1. アーティファクト	29
6.3.2. 次のステップ	29
6.4. FHC を使用したアプリ統計の表示	29
第7章 FHC を使用したフォームでの作業	32
7.1. FHC を使用したフォームプロジェクトの作成	32
7.1.1. フォームプロジェクトの作成	32
7.1.2. フォームの追加	32
7.1.3. フォームをプロジェクトに関連付ける	32
7.1.4. テーマの作成	33
7.1.5. テーマをアプリに関連付ける	33
7.1.6. 別のフォームアプリの追加	33
7.2. FHC を使用したフォームの作成	34
7.2.1. 利用可能なフォームの一覧表示	34
7.2.2. フォームの作成	34
7.2.3. フォームの更新	36
7.3. FHC を使用したフォームの提出の作成	36
7.3.1. 提出の一覧表示	37
7.3.2. 特定提出の応答	37
7.3.3. 提出のステータス	37
7.3.4. 提出の送信	37
7.4. FHC を使用したフォームテーマの作成	38
7.4.1. テーマの一覧表示	38
7.4.2. テーマの作成	38
7.4.3. テーマをアプリに関連付ける	39
7.4.4. テーマの更新	39
付録A 複数の SSH キーを異なるドメイン/プロジェクトに使用する	40

第1章 FHC の設定

1.1. FHC のインストール

概要

fhc は Red Hat Mobile Application Platform Hosted (RHMAP) のコマンドラインインターフェース (CLI) であり、Node.js に基づいています。RHMAP Studio のほとんどすべての機能は RHMAP API を介して公開され、**fhc** を使用してアクセスできます。このため、**fhc** を他の Node.js アプリケーションに含めて、プログラムを使用して Platform にアクセスできます。Platform の公開された機能との対話は、ビルドシステムや継続的な統合システムなどの自動プロセスに統合できます。

要件

fhc を使用するには、Node.js と NPM をインストールします。Node.js をインストールするには、nodejs.org を参照してください。

インストール後に、**node** と **npm** の 2 つの新しいコマンドラインアプリケーションが利用可能になります。

1.1.1. FHC のインストール

ターミナル/コマンドプロンプトで、以下のコマンドを実行します。

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

Linux にインストールする場合は、以下のように **sudoer** でこのコマンドを実行する必要がある場合があります。

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

これで Node.js モジュールの中央レジストリである **npm** から **fhc** がインストールされます。

-g フラグは、npm に **fhc** をグローバルでインストールするように指示し、どのディレクトリーからも使用できるようにします。

インストールが完了したら、**fhc** はコマンドラインから使用可能となります。**Z shell (zsh)** を使用している場合は、**hash -r** を使用してハッシュテーブルをリセットすることでコマンドが利用可能になります。

FHC が正常にインストールされたことを確認するには、以下のコマンドを使ってインストールしたバージョンを表示します。

```
fhc -v
```

1.1.1.1. コマンド自動入力 (Linux および Mac のみ)

fhc バッシュ補完スクリプトを使用すると、さまざまな **fhc** コマンドで Tab 補完が可能になります。以下の方法で **fhc** バッシュ補完スクリプトをインストールします。

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




注記

別のシェル (**bash** 以外) を使用している場合は、**fhc completion** の出力を該当するファイルに追加します (たとえば、zsh の場合は `~/.zshrc`)。

1.1.2. FHC の使用

ターゲットを設定し、ログインします。

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

プロジェクトを一覧表示するには、以下のコマンドを実行します。

```
fhc projects list
```

fhc の全コマンドを一覧表示するには、以下を実行します。

```
fhc help
```

1.1.2.1. 設定

fhc は非常に高度な設定が可能で、以下の 5 つの場所から設定オプションを読み取ります。

- コマンドラインのスイッチ:
`--key val` で設定します。すべてのキーは値を取り、これはブール値であっても該当します (設定パーサーは、解析時にはオプションが何かを知りません)。値が提供されないと、そのオプションはブール値 **true** に設定されます。
- 環境変数:
環境変数の名前に **fhc_config_** の接頭辞を付けます。例えば、**export fhc_config_key=val** となります。
- ユーザー設定:
\$HOME/.fhcrc にあるファイルは、初期化済みの設定一覧です。これがあると、解析されます。**userconfig** オプションが cli または環境変数で設定されると、代わりにそちらが使用されます。
- グローバル設定:
../etc/fhcrc (実行可能なノードで、デフォルトでは /usr/local/etc/fhcrc) にファイルがあれば解析されます。**globalconfig** オプションが cli、環境変数、またはユーザー設定で設定されると、代わりにそれらのファイルが解析されます。
- デフォルト:
fhc のデフォルト設定オプションは、**lib/utils/config-defs.js** で定義されます。これらは変更しないでください。

詳細については、**fhc help config** を実行してください。すべてのコマンドの一覧については **fhc help** を使用し、特定のコマンドのヘルプについては **fhc [command] --help** を使用します。

1.1.3. 次のステップ

- [FHC で SSH キーを追加する](#)

- [プロジェクト & アプリに関する作業](#)
- [ローカルでのアプリ開発](#)

第2章 SSH キーの設定

概要

git レポジトリのクローンを実行するには、まず自分の SSH 公開キーをアップロードする必要があります (公開キーの暗号化についての詳細は、[Wikipedia article](#) を参照してください)。

ここでは、公開/秘密キーのペアを生成し (必要な場合)、SSH 公開キーを FHC でプラットフォームにアップロードする方法について解説します。

要件

以下の作業を開始する前に、[FHC のインストール](#) を完了しておく必要があります。

2.1. 既存 SSH 公開キーの確認

SSH 公開キーが既に生成されているかどうかを確認するには、Linux や Mac では **端末** を、Windows では **Git Bash** を開きます。

`ls -la ~/.ssh` を実行します。

これで .ssh ディレクトリーのファイルが一覧表示されます。`id_rsa.pub` または `id_dsa.pub` のいずれかのファイルがある場合はキーが生成済みなので [SSH 公開キーを FHC でプラットフォームに追加する](#) に進みます。これらのファイルがない場合は SSH 公開キーを手動で生成する必要があるので、[SSH 公開キーの生成](#) に進みます。

2.2. SSH 公開キーの生成

キーの生成には、`ssh-keygen` というツールを使用します。Linux や Mac では **端末** を、Windows では **Git Bash** を開きます。

以下を入力します。

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

このコマンドを入力すると、キーの保存場所を質問されます。リターンキーを押して、デフォルトの場所を使用します。

次に、秘密キーのパスワードの入力と確認を求められます。これが完了すると、公開キーと秘密キーがそれぞれ `~/.ssh/id_rsa.pub` と `~/.ssh/id_rsa` に書き込まれます。

2.3. FHC でプラットフォームに SSH 公開キーを追加する

FHC を使用してプラットフォームにログインします。公開キーを追加するには、名前とキーファイルの両方を指定する必要があります。以下のコマンドを入力してキーを追加します。

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

例を示します。

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

以下のようにユーザーのすべての SSH キーを一覧表示すると、該当キーが正常に追加されたことを確認できます。

```
fhc keys ssh
```

これで追加されたキーが表示されます。



注記

複数の SSH キーを使用する必要がある場合は、[複数の SSH キーを異なるドメイン/プロジェクトに使用する](#) を参照してください。

2.4. STUDIO でプラットフォームに SSH 公開キーを追加する

Studio を使用してプラットフォームに公開 SSH キーを追加するには、以下の手順に従います。

1. Studio にログインします。
2. 画面右上のポートレートをクリックします。
3. **設定** を選択します。
4. **SSH キーの管理** を選択します。これまでにアップロードした SSH キーが表示されます。
5. **新規キーを追加** を選択します。
6. **公開キー** のフィールドに公開キーを貼り付けます。
7. **公開キーのアップロード** を選択します。

これで追加されたキーが一覧表示内に表示されます。

第3章 FHC の使用

以下では、同じテンプレートに基づいて新しいプロジェクトを作成する方法について説明します。

fhc は、ローカルマシンにプロジェクトをクローンするために使用されます。**npm** は、ローカル開発向けの依存関係をインストールするために使用されます。**Grunt** は、ブラウザ内での開発のために新しいハイブリッドクライアントと Node.js クラウドアプリをローカルで起動するために使用されます。

動画チュートリアル

- [RHMAP - Create an App](#) (英語)

前提条件

- [「FHC のインストール」](#)
- [2章SSH キーの設定](#)
- ローカルにインストールされた Node.js および Git

3.1. FHC での RHMAP ドメインのターゲット設定

最初に **fhc** を使って RHMAP ドメインにターゲット設定 (接続) する必要があります。

コマンドラインアプリケーション (たとえば、Linux または MacOS の場合はターミナル、Windows の場合はコマンドプロンプト) を開きます。

<domain-url> をご使用のドメインの URL に置き換えてそのドメインをターゲットにします。

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

次に RHMAP 認証情報を入力します。以下のコマンドで <email or alias> をユーザー名もしくはメールアドレスで置換して、実行します。

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

パスワードを入力します。

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

これでターゲットとする RHMAP ドメインにログインできました。

3.2. 新規プロジェクトの作成

以下では、**hello_world_project** テンプレートをベースとした新しいプロジェクトを作成する方法について説明します。

利用可能なプロジェクトテンプレートを一覧表示するには、以下のコマンドを実行します。

```
fhc templates projects
```

project name を、選択したプロジェクト名に置き換えて、以下のコマンドを実行します。

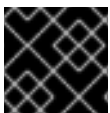
**注記**

テンプレート **hello_world_project** の名前をパラメーターとして渡してください。

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

成功すると、以下のようにすべてのプロジェクト情報とともに JSON 応答が表示されます。

```
{
  "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"
}
```

**重要**

guid を書き留めてください、以下のステップで使用します。

プロジェクト情報を見直す必要がある場合は、以下のコマンドを入力します。**project guid** を前のステップの **guid** で置き換えます。

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

このコマンドは、前のステップで示された JSON オブジェクトを返し、これにはプロジェクトの全情報が含まれます。

プロジェクトをローカルマシンにクローンします。**project name** を、選択したプロジェクト名に置き換え、**project guid** を JSON オブジェクトからの **guid** で置き換えます。

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

MBaaS の例またはクラウドアプリを確認するために、**project name** をご使用のプロジェクト名に置き換えます。

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

クラウドアプリの依存関係をインストールします。

```
npm install
```

-g フラグを使って **grunt CLI** をインストールし、grunt をグローバルでインストールします。

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

これでクラウドアプリのインスタンスをローカルマシンにデプロイする準備ができました。

3.3. クラウドアプリのデプロイ

クラウドサーバーをローカルで起動するには、以下のコマンドを実行します。

```
grunt serve
```

コンソールに、クラウドアプリがローカルマシンの **port 8001** で実行されていることを示す情報が表示されます。

ローカルマシンで以下の curl 要求を入力して、クラウドインスタンスが実行中であることを確認します。

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

以下のような応答が表示されるはずです。

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

この curl で上記のような応答が返されると、クラウドアプリがローカルで実行されていることが確認できます。

3.4. クライアントアプリのデプロイ

以下のコマンドはチェーンコマンドであることに注意してください。最初の部分では、ディレクトリツリーの 1 つ上のレベルに移動します。2 番目の部分では、1 つ下のレベルのクライアントディレクトリに移動します。

project name をプロジェクト名で置き換え、以下のコマンドを実行します。

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

クライアントアプリ用のすべての依存関係をインストールします。

```
npm install .
```

クライアントアプリをローカルで実行します。

```
grunt serve:local
```

成功すると、ウィンドウがデフォルトのブラウザで開きます。

3.5. サマリー

Platform ではすべての Git レポジトリがホストされます。プロジェクト内の各アプリには独自の Git レポジトリがあります。通常の Git 作業とプロセスは、クライアントアプリまたはクラウドアプリの例に移動することにより実行されます。

これで RHMAP プロジェクトをローカルで開発する設定が整いました。

第4章 プロジェクトおよびアプリに関する作業

概要

プラットフォームでは、プロジェクトは関連アプリをまとめるコンテナとして使用されており、アプリはすべてプロジェクト内で作成する必要があります。以下では、**fhc** を使用してプロジェクトを作成、追加する方法について説明します。

要件

以下のチュートリアルを終了していることを前提としています。

- [FHC のインストール](#)

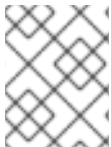
4.1. プラットフォームへのログイン

端末に **fhc targets** を入力して利用可能なターゲットドメインを一覧表示します。

```
fhc targets list
```

fhc target コマンドを使用してターゲットドメインを選択します。

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



注記

上記のコードでは、'exampleDomain.redhatmobile.com' がターゲットドメインとして選択されます。

これでターゲットドメインが選択されました。ログインするには、**fhc login** の後にユーザー名とパスワードを続けます。

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



注記

ログインが完了したら、アクセス制限が適用されます。チームパーミッションでプラットフォーム内のユーザーのアクセスレベルが決定されます。詳細は、[チームおよびコラボレーション](#) を参照してください。

4.2. 既存プロジェクトの一覧表示

既存のプロジェクトを一覧表示します。

```
fhc projects list
```



注記

プロジェクト一覧を raw JSON 形式としたい場合は、**fhc projects list --json** のようにコマンドの最後に **--json** を追記します。

既存プロジェクトが最新更新日の順番で表示されます。例を示します。

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

4.3. プロジェクトの作成

クライアントアプリやクラウドアプリを含まない新規プロジェクトを作成することは可能ですが、プロジェクトには名前を付けることが必要です。**fhc projects create** を使用してプロジェクトを作成します。

通常、新規プロジェクトは事前定義のテンプレートを使用して作成します。プロジェクトテンプレートを使用すると、普通は 1 つのクライアントアプリと 1 つのクラウドアプリを提供するプロジェクトテンプレートをクローンすることで開発をブーストラップすることができるようになります。

プロジェクトについての詳細は、[プロジェクト](#) を参照してください。

このチュートリアルでは、**hello_world_project** テンプレートを使用します。これは 1 つのクライアントアプリと 1 つのクラウドアプリを提供する基本的なプロジェクトで、標準の Hello World パラダイムを使用して基本的な機能が説明されます。

全プロジェクトテンプレートを一覧表示するには、以下を実行します。

```
fhc templates projects
```

hello_world_project テンプレートをベースにした新規プロジェクトを作成するには、以下を実行します。

```
fhc projects create helloWorld hello_world_project
```

以下を実行してプロジェクトが正常に作成されたことを確認します。

```
fhc projects list
```

別の方法では、以下のように **fhc projects** コマンドの出力を **grep** コマンドでパイプすることで、特定のプロジェクトを検索することもできます。

```
fhc projects list | grep 'helloWorld'

# Example:

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

■
コンソールの出力にクライアントアプリとクラウドアプリが含まれる、最近作成されたプロジェクトが表示されます。

プロジェクト内の全アプリを一覧表示するには、projectId ('guid' (Global Unique Identifier) と呼ぶ) を選択し、**fhc apps** と使用します。

```
fhc projects # lists all projects. Next, select a projectId
fhc apps <the_selected_project_guid>
```

これで指定された projectId のアプリすべてが表示されます。

- Id - アプリの guid。
- Title - アプリのタイトル。これはモバイルデバイス上でクライアントアプリのアプリ名として使用されます。
- Description - アプリの説明。デフォルトでは空白になります。
- Type - アプリのタイプ。プラットフォームが異なるアプリタイプを区別するために使用します。
- Git - アプリの Git URL。これはローカル開発用にアプリをクローンする際に使用できます。
- Branch - プラットフォームでアプリコードを編集するために現在選択されているブランチ。

4.4. プロジェクトにアプリを追加する

プロジェクトには複数のアプリを含めることが可能ですが、相互に関係の内アプリは別個のプロジェクトに格納するようにします。アプリは通常、既存のアプリテンプレートを使ってプロジェクトに追加されます。クライアントおよびクラウドアプリの両方に多くのテンプレートが用意されています。

以下を実行して利用可能なアプリテンプレートを一覧表示します。

```
fhc templates apps
```

クライアントアプリをプロジェクトに追加するには、以下のコマンドを使用して、<projectId>、<appTitle> の名前、および <appTemplate> を指定します。

```
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
```



注記

テンプレートが指定されない場合は、空のクライアントアプリが作成されます。

プロジェクト内の全アプリを一覧表示して、アプリが正常に作成されたことを確認します。

4.5. 次のステップ

- [ローカルでのアプリ開発](#)
- [アプリバイナリーのビルド](#)

第5章 ローカルでのコード開発

概要

本章では、Red Hat Mobile Application Platform ホスト型 (RHMAP) 内でのモバイルアプリケーション開発用のローカル環境を設定する方法について説明します。[アプリ](#) と [Node.js クラウド](#) サーバー開発の両方が対象となります。

要件

以下のチュートリアルを終了していることを前提としています。

- [FHC のインストール](#)
- [SSH キーの設定](#)
- [プロジェクト & アプリに関する作業](#)

5.1. 開発ツール

`fhc`、Node.js、および NPM 以外に、他の開発ツールが必要です。

5.1.1. Git

"Git は無償のオープンソースの分散型バージョン管理システムで、スピードと高い効率性で小さいものから大型のものまですべてのプロジェクトに対処できます。"

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

特定のオペレーティングシステムに git をインストールします。

- RHEL / Fedora: [Git Installation Page](#) を参照してください。
- Debian ベースのディストリビューション: `sudo apt-get install git-core`
- Mac OSX: `brew install git` または `sudo port install git-core +doc +bash_completion`。もしくは、グラフィカルな git [installer](#) を使用します。
- Windows: [msysgit](#) を参照してください。



注記

Mac OSX に **Homebrew** パッケージマネージャーをインストールするには、<http://brew.sh/> を参照してください。

5.1.2. Grunt

"JavaScript タスクランナー"

- [Getting Started](#)

- [Plugins](#)

grunt は NPM で利用可能です。インストールするには、以下のコマンドを実行します。

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



注記

すべての RHMAP テンプレートアプリは **grunt** をローカル開発用のタスクランナーとして使用します。

5.2. ソースコードのクローン

プロジェクト開発を含める新しいディレクトリーを作成します。

```
mkdir helloworld  
cd helloworld
```

fhc projects clone <project-guid> を使用してプロジェクト内の全アプリのソースコードをクローンします。以下に例を示します。

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

以下のような出力メッセージが表示されます。

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

完了したら、作成された新しいディレクトリーを確認します。

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

5.3. 別の GIT リポジトリからコードを使用する

<https://github.com/evanshortiss/rhmap-express-template> などのリモートのリポジトリからコードを使用するには、以下の手順を実行します。

1. `blank_advanced_webapp` といった空白のアプリを作成します。
2. 使用するリポジトリを一時ディレクトリーにクローンします。

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

3. **.git** ディレクトリーを削除します。

```
cd temp_express  
rm -rf .git
```

-
4. 以下のように、リポジトリのコンテンツをアプリリポジトリにコピーします。

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

これでアプリを開発してその結果を RHMAP にプッシュすることができます。

5.4. 開発中のクラウドアプリへのアクセス

クラウドアプリにアクセスするには以下の 2 つの方法があります。

1. Platform 上の環境で実行中のクラウドインスタンスにアクセスする。
2. Node.js のローカルインスタンスでクラウドアプリを実行してアクセスする。

5.4.1. クラウドランタイムの使用

クラウドアプリはデフォルトで <http://localhost:8001> を参照します。

クラウドアプリをホストしている URL を調べます。

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

応答は以下のようになります。

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

クラウドアプリに属する **Gruntfile.js** ファイルで変数 **default_local_server_url** を見つけます。**default_local_server_url** 変数を、**fhc app hosts** コマンドで返された値に設定します。

ローカルで実行されているクライアントアプリによって、Platform で実行中のクラウドアプリに要求がルーティングされます。

5.4.2. ローカルランタイムの使用



注記

grunt serve を実行するには、npm のバージョンが 1.4.15 以上である必要があります。

クラウドサーバーコードをローカルで実行します。

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

クラウドアプリ向けのすべての依存関係をインストールします。

```
npm install
```

完了したら (数分かかることがあります)、Node.js サーバーを起動します。

```
grunt serve
```

以下のような出力メッセージが表示されます。

```
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
```

Node.js プロセスにより、ターミナルウィンドウでロックがかかります。クラウドサーバーは実行中ですが、作業を進めるために、別のターミナルウィンドウを開き、クラウドサーバーが実行中であることを確認します。

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

以下のような出力メッセージが表示されます。

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

入力パラメーターを **world** から別の値に変更し、更新された応答を確認します。

Gruntfile.js ファイルには、以下のような複数の有用なコマンドが含まれます。

```
grunt serve      # 'live reload' でサーバーをローカル実行
grunt test       # ローカルでテストを実行
grunt coverage   # コードカバレッジでテストを実行
```

詳細については、Grunt-ReadMe.md ファイルを参照するか、**grunt --help** を実行してください。

5.5. クライアントをローカルで実行する

クライアントアプリをローカルで実行します。

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

クライアントアプリ向けのすべての依存関係 (主に grunt 関連) をインストールします。

```
npm install
```

Web ブラウザーセッションを開始します。

```
grunt serve:local
```


grunt serve:local により、デフォルトでポート 8001 でのローカル Node.js クラウドアプリへの接続が試行されます。以下のようなメッセージが表示されます。

```
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...
```

詳細については、'Hello World Client' テンプレートアプリ向けの README.md を参照してください。

5.6. ローカルでの MBAAS サービスの作業

MBaaS サービスは Node.js アプリケーションです。MBaaS サービスは、Oracle 統合サービスのようなバックエンドシステムと統合するためにプロジェクトで使用できます。MBaaS サービスは 1 つ以上のプロジェクトと関連付けてアプリで利用できるようにする必要があります。そのようにしないと、プロジェクト内のアプリは MBaaS サービスにアクセスできません。MBaaS サービスの詳細については、[relevant product documentation](#) を参照してください。

サービスは、クラウドアプリから [\\$fh.service クラウド API コール](#) で呼び出されます。以下に例を示します。

```
$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);
});
```

ローカル開発環境から MBaaS サービスと対話するには、以下の間でマッピングが必要です。

1. サービス guid (つまり、サービスの一意な id)
2. ターゲットとなるサービスの実行中インスタンスのホスト名

Gruntfile.js での新しい環境変数の定義が必要です。

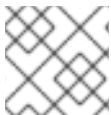
```
env : {
  options : {},
  // environment variables - see https://github.com/jsoverson/grunt-env
  for more information
  local: {
    FH_USE_LOCAL_DB: true,
    FH_SERVICE_MAP: function() {
```

```

/*
 * ローカル開発用のサービスのマッピングをここで定義します。
 * アクセスしたい各サービスでマッピングを提供する必要があります。
 * これは、ローカルで実行中のサービスのインスタンスとする（ローカル開発用）
 * もしくはリモートインスタンスとすることができます。
 */
var serviceMap = {
  'PFi1ftKRBvlp-qSmgdc0eGe3': 'http://127.0.0.1:8010'
};
return JSON.stringify(serviceMap);
}
},
},

```

上記の例では、**FH_SERVICE_MAP** がローカル環境変数の定義に追加されます。新しい変数は関数に対してマップされ、ポート 8010 でローカルで実行中のサービスを参照する GUID **PFi1ftKRBvlp-qSmgdc0eGe3** を格納します。サービスのホスト済みインスタンスをターゲットにするには、Studio の **Details** ページにあるサービスの **Current Host** URL を使用してください。すべての追加サービスは **serviceMap** 変数に追加されます。



注記

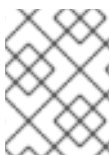
FH_SERVICE_MAP 環境変数は、単純な文字列ではなく関数として定義されます。

ユーザーは、文字列化された JSON オブジェクトではなく標準的な JSON オブジェクトとしてサービスマッピングを定義できます。以下の例では、**FH_SERVICE_MAP** 環境変数は標準的な JSON オブジェクトを使用して設定されます。

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

利用可能なサンプルは、[ここ](#)で確認できます。

5.7. クラウドコードの編集



注記

Node.js クラウド API サーバーが実行されている場合は、**grunt** がすべての変更を監視します。**grunt** はリアルタイムで変更を反映します。

クラウドアプリのディレクトリに移動し、**./lib/hello.js** を開きます。

応答で現在の時間を設定するパラメーターを追加するには、新しいパラメーターを GET と POST の両方の要求に入力します。

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

上記の内容を以下の内容に置き換えます。

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

これで応答にミリ秒単位の現在の時間がタイムスタンプに追加されます。これが 2 つあるのは、GET および POST リクエストの両方用にルートハンドラーがあるためです。

このファイルが保存されると、クラウドアプリは自動的に再起動します。ファイルの保存後に、以前実行した curl コマンドを再実行します。

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

応答にタイムスタンプが含まれるようになります。

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

5.8. クライアントコードの編集

クライアントアプリのディレクトリーに移動し、`./www/index.html` を開きます。これは、"name" 用の入力フィールドとクラウドを呼び出すボタンがある基本的な Web ページです。また、クラウドコールから応答に値を入力するために使用する `cloudResponse` div もあります。

コード "cloudResponse" の直下:

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

timestamp div の追加:

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

`./www/hello.js` で、クリックハンドラーは "Say Hello" ボタンに割り当てられます。トリガーされると、`$fh.cloud` API を使用してクラウドアプリの `/hello` エンドポイントが呼び出されます。

タイムスタンプを提供するには、`./www/hello.js` を開き、

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

上記のコードの直下に以下のコードを追加します。

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



注記

値を表示するために、クラウドから返されたタイムスタンプ値を受け入れる新しい日付オブジェクトが作成されます。

5.9. APP STUDIO での変更の確認

クライアントディレクトリーとクラウドディレクトリーで以下のコマンドを実行します。

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

これで、ローカルで行われた変更が App Studio に表示されるようになります。

5.10. 高度な開発

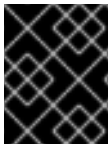
`fh.db()` や `fh.cache()` といった MBaaS API を使用した高度な開発には、別のソフトウェアが必要です。

5.10.1. fh.cache の使用

`fh.cache` を使用してローカルで開発するには、[Redis](#) をローカルでインストールします。

- RHEL / Fedora: [Redis Download Page](#) を参照してください。
- Debian ベースのディストリビューション: `apt-get install redis-server`
- Mac: `brew install redis` または `sudo port install redis`
- Windows: [MSOpenTech Redis](#) を参照してください。

Redis がインストールおよび実行されると、クラウドサーバーの `fh.cache` への呼び出しは RHMAP クラウドの場合と同様に動作します。



重要

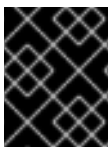
デフォルトの Redis ポートが使用されるため、Redis 設定ファイルを変更する必要はありません。

5.10.2. fh.db の使用

`fh.db` を使用してローカルで開発するには、[Mongo](#) をローカルでインストールします。

- RHEL / Fedora: [こちら](#) (英語) の手順を実行します。
- Debian ベースのディストリビューション: [こちら](#) (英語) の手順を実行します。
- Mac: [こちら](#) (英語) の手順を実行する、または `brew install mongodb`
- Windows: [こちら](#) (英語) の手順を実行します。

インストールされて実行されると、クラウドサーバーの `fh.db` への呼び出しは RHMAP クラウドの場合と同様に動作します。



重要

デフォルトの Mongo ポートが使用されるため、Mongo 設定ファイルを変更する必要はありません。

5.11. 次のステップ

- [アプリバイナリーのビルド](#)
- [FHC CLI](#)

第6章 FHC を使用したプラットフォームでの作業

6.1. FHC による API キーの管理

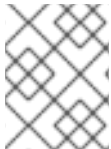
特定のFHC コマンド実行時には、ユーザー ID を使用してプラットフォームと通信する必要があります。これには、ユーザーが **fhc login** コマンドを使用してプラットフォームにログインする必要があります。ユーザー API キーは FHC の使用で別の方法を提供します。ユーザー API キーを FHC で設定していれば、ログインコマンドを実行する必要がなくなります。プラットフォームとの通信時に、FHC は API キーを使用して認証することが可能になります。

6.1.1. ユーザー API キーの設定

FHC での API キー管理に関連するコマンドは、**fhc keys** になります。引数なしでこのコマンドを実行すると、コマンドのヘルプが表示されます。

現行の FHC ターゲットにユーザー API キーを設定するには、以下のコマンドを入力します。

```
fhc keys user target eb8d9b9ea050b9b23fea59e50ba281c67f3715e5
```



注記

これを実行するには、**fhc target** を使用して事前にターゲットを設定している必要があります。

現在 FHC に使用されている API キーを確認するには、以下のコマンドを実行します。

```
fhc keys user target
```

FHC で API キーを設定したら、他の FHC コマンドを通常通り実行できます。

例えば、FHC を使用してユーザー API キーの一覧表示、作成、更新、および取り消しができます。

6.1.2. ユーザー API キーの一覧表示

FHC を使用してログインしている、またはユーザー API キーを設定している場合は、以下のコマンドを実行すると現行ユーザーに関連するすべての API キーが表示されます。

```
fhc keys user list
```

FHC config で出力形式が json に設定されていれば、各キーオブジェクトは以下のようになります。

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

6.1.3. ユーザー API キーの作成

ユーザー API キーを作成するには、以下のコマンドを実行します。

```
fhc keys user create test2
```

以下のような応答が返されます。

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

6.1.4. ユーザー API キーの更新

FHC を使用してユーザー API キーを更新するには、以下のコマンドを実行します。

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

以下のような応答が返されます。

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

6.1.5. ユーザー API キーの取り消し

FHC を使用してユーザー API キーを取り消すには、以下のコマンドを実行します。

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

取り消しアクションの確認が求められます。確認すると、以下のような応答が返されます。

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

6.2. FHC を使用したサービスの作成

概要

本チュートリアルでは、サービスの作成と、それをプロジェクトに関連付ける方法について説明します。サービスおよびその機能に関する情報については、[Cloud Services](#) のドキュメントを参照してください。

要件

ユーザーは、以下のパーミッションを持つ 1 つ以上のチームメンバーである必要があります。

- プロジェクト (表示 & 編集)
- サービス (表示 & 編集)

パーミッションについての詳細は、「[チームおよびコラボレーション](#)」を参照してください。

6.2.1. プラットフォームへのログイン

```
fhc login myUsername myPassword
```

6.2.2. プロジェクトの作成

プラットフォームにログインしたら、サービスを関連付けるプロジェクトを作成します。プロジェクトを作成するには、以下のコマンドを実行します。

```
fhc projects create serviceProject
```

プロジェクトが正常に作成されたことを確認するには、以下のコマンドを実行してドメイン上でプロジェクトを検索します。

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

これで新規に作成されたプロジェクトが返されます。このプロジェクトには、クライアントアプリとクラウドアプリという 2 つのアプリがあることに留意してください。

6.2.3. サービスの作成

サービスをプロジェクトに追加すると、この追加なしではアクセスできなかったバックエンド機能にクライアントがアクセスできるようになります。サービスを作成する前に、利用可能なテンプレートを確認してみましょう。サービステンプレートを一覧表示するには、以下のコマンドを入力します。

```
fhc templates services
```

これで利用可能なサービステンプレートすべてが一覧表示されます。ここでは PayPal サービスをプロジェクトに追加します。これを実行するには、PayPal サービスの ID をコピーします。

サービスの作成時には、名前も指定する必要があります。以下のコマンドを実行します。

```
fhc services create paypalService paypal
```


これでプロジェクトに関連付けられる新規サービスが作成されました。

6.3. アプリバイナリーのビルド

概要

本チュートリアルでは、FHC を使用したモバイルアプリバイナリーのビルド方法について説明します。Red Hat Mobile Application Platform ホスト型 (RHMAP) でのアプリバイナリーのビルドには、クラウドでホストされている "Build Farm" を使用します。これは、ソースコードをモバイルアプリに変換します。(Xcode や ADT) といったローカルの開発ツールがインストールされていれば、ご自身でローカルにアプリバイナリーを構築できます。ただし、RHMAP Build Farm を使用すると、ビルドプロセスの自動化、これまでのビルド履歴の維持、および開発ツールがないプラットフォーム用のアプリのビルド (例えば、Linux OS から iOS バイナリーをビルドする) などができるようになります。

要件

本チュートリアルを開始する前に、以下のチュートリアルを完了してください。

- [FHC のインストール](#)
- [プロジェクト & アプリに関する作業](#)

6.3.1. fhc build コマンド

fhc を使用したビルドでは、多くのパラメーターを指定する必要があります。これはビルドのタイプや使用先となるプラットフォームによって異なります。[プロジェクトおよびアプリに関する作業](#) のチュートリアルでは、プロジェクトの一部として cordova ハイブリッドアプリを作成しました。ここでは、このアプリを使用して Android 用のビルドを実行します。

fhc build コマンドは以下のパラメーターを取ります。

- **project=<project-id>** — プロジェクトの guid。
- **app=<app-id>** — アプリをビルドするクライアントアプリの guid。
- **cloud_app=<cloud-app-id>** — クライアントアプリの通信先となるクラウドアプリの guid。
- **tag=<tag>** — 本ビルドに使用する [Semver](#) タグ。詳細は、[Connections](#) を参照してください。
- **destination=<destination>** — ビルドのタイプ—例、android、iOS、windowsphone。
- **version=<version>** — ターゲットとなるビルドタイプの特定の OS バージョン。
- **config=<config>** — 実行するビルドタイプ—'debug' (デフォルト)、'distribution' または 'release'。
- **keypass=<private-key-password>** — ビルドに使用される秘密キーのパスワード (release ビルドの場合のみ必要)。
- **certpass=<certificate-password>** — ビルドに使用される証明書のパスワード (release ビルドの場合のみ必要)。
- **download=<true|false>** — 生成されたバイナリーをダウンロードするかどうか。

- **provisioning=<path-to-provisioning-profile>** — ビルドに使用するプロビジョニング
グプロファイルへのパス。
- **cordova_version=<cordova-version>** — 使用する cordova のバージョン
- **environment=<environment>** — ターゲット環境の ID。例: dev。

すべてのビルドタイプにおいてこれらの全パラメーターが必要なわけではありません。詳細は、**fhc build --help** コマンドを実行してください。

アプリのビルド時には、アプリ ID に加えてアプリが格納されているプロジェクトを fhc に指定する必要があります。その後にクライアントアプリの通信先となるクラウドアプリの ID を入力します。クライアントアプリは、接続によりクラウドアプリに向けられます。クライアントは色々な接続によって異なるクラウドアプリと通信することが可能なため、ビルド実行時には使用する接続タグを指定する必要があります。

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

ビルドが完了すると、以下のような確認の出力が表示されます。

```
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
```

6.3.1.1. アーティファクト

ビルドが成功したら、そのアーティファクトがアーティファクト履歴に追加されます。アーティファクトを使用すると、ビルドを再実行せずにアプリを再度ダウンロードすることができます。

特定アプリの全アーティファクトを一覧表示するには、**fhc artefacts <projectId> <appId>** コマンドを使用します。

```
fhc artefacts XME5iUr2VoBV3DbXrVF7qApG XME5iNsoeRA2xvbmaDYMCdsi
```

これで指定されたアプリの全アーティファクトが表示されます。

6.3.2. 次のステップ

- [FHC CLI](#)

6.4. FHC を使用したアプリ統計の表示

FHC を使用すると、アプリの raw の JSON 統計情報データにアクセスできます。

特定アプリの統計情報 (タイマーおよびカウンター) にアクセスするには、以下のコマンドを使用します。

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

APP_ID は **fhc apps** コマンドで表示されるアプリの ID で、**NUMBER_OF_RESULTS** は表示する結果の最大数の数字で置き換えます。サンプルの統計情報は以下のようになります。

```
{
  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
          }
        }
      ]
    }
  }
]
}
```

第7章 FHC を使用したフォームでの作業

7.1. FHC を使用したフォームプロジェクトの作成

概要

本チュートリアルでは、FHC を使用してフォームプロジェクトを構築する方法を説明します。このプロジェクトは、フォームクライアントアプリとクラウドアプリで構成されています。

要件

ユーザーは、以下のパーミッションを持つ 1 つ以上のチームメンバーである必要があります。

- プロジェクト (表示 & 編集)
- Drag & Drop App (表示 & 編集)

パーミッションについての詳細は、「[チームおよびコラボレーション](#)」を参照してください。

7.1.1. フォームプロジェクトの作成

最初にアプリフォームプロジェクトを作成します。これは、プロジェクト作成時に使用するテンプレートを指定することで行います。これでハイブリッドフォームアプリを含むプロジェクトが作成されます。

```
fhc projects create FormsProject forms_project
```

7.1.2. フォームの追加

フォームアプリを含むプロジェクトが作成されたら、次にフォームをそのアプリに関連付けます。これは、使用するフォームファイルへのパスを指定することで行います。

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

例を示します。

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

7.1.3. フォームをプロジェクトに関連付ける

プラットフォーム上でフォームが作成されたら、これをプロジェクト内のアプリに関連付けます。

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

例を示します。

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

これでフォームがアプリに関連付けられました。**fhc preview** コマンドを実行すると、フォームの現在のステータスが確認できます。

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

これでブラウザーにアプリのプレビューが読み込まれます。フォームがアプリに関連づけられていることが確認できますが、フォームには外観をよくするテーマが割り当てられていないため、見た感じはまだ最小限のものになっています。

7.1.4. テーマの作成

テーマはフォームと同様の方法で作成されます。**fhc themes create** コマンドを使用して theme.json ファイルを作成し、そのファイルへのパスを指定します。

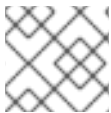
fhc themes create <theme.json>

例を示します。

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

7.1.5. テーマをアプリに関連付ける

プラットフォームにテーマが追加されたら、作成済みのプロジェクトにこれを適用します。



注記

適用が関連付けられるのは個別のアプリではなく、プロジェクトになります。

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

例を示します。

```
fhc themes app set HQgUjokQi7jjizH8T-7TD4SQQ 534578160663a687117a4bd1
```

この時点では、機能するフォームプロジェクトにクライアントフォームアプリとクラウドアプリが含まれることになります。クライアントアプリには、フォームとテーマが関連付けられています。**fhc preview <app-id>** を使用するとプレビューが確認できます。プロジェクトにテーマが関連付けられたので、フォームの外観がより魅力的なものになっていることが確認できます。

7.1.6. 別のフォームアプリの追加

フォームベースのアプリをプロジェクトに追加したい場合は、以下のコマンドを実行します。

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

template-id では、'appforms_client' テンプレートを指定します。これでハイブリッドフォームアプリが作成されます。



注記

利用可能なアプリテンプレートを確認するには、'fhc templates apps' コマンドを使用します。

プロジェクトにフォームアプリを追加するには、以下のコマンドを実行します。

```
fhc apps create HQgUjokQi7jjizH8T-7TD4SQQ Second_Form_App appforms_client
```

これで別のフォームベースのアプリがプロジェクト内に作成されました。アプリには、最初に行った方法でフォームとテーマを関連付けることができます。

7.2. FHC を使用したフォームの作成

概要

本チュートリアルでは、アプリフォームのアプリで使用するフォームの作成方法を説明します。

要件

ユーザーは、以下のパーミッションを持つ 1 つ以上のチームメンバーである必要があります。

- ドメイン — Drag & Drop Apps
- フォーム (表示 & 編集)

パーミッションについての詳細は、「[チームおよびコラボレーション](#)」を参照してください。

7.2.1. 利用可能なフォームの一覧表示

fhc target を使用して希望するドメインをターゲットにします。

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

fhc login コマンドを使用してログインします。

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

ドメイン上の全フォームを一覧表示するには、**fhc forms** コマンドを使用します。

```
fhc forms
```

このコマンドで、ログインユーザーに割り当てられているパーミッションに基づくアクセスがある全フォームが一覧表示されます。

7.2.2. フォームの作成

フォームを作成するには、**form.json** オブジェクトを渡す必要があります。**form.json** オブジェクトを作成し、そのパスを書き留めます。本チュートリアルで使用する **form.json** ファイルの例は、以下の通りです。

```
{
  "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
  }

```

fhc を使用してフォームを作成するには、**fhc forms create** コマンドを以下のように使用します。

fhc forms create <form-file.json>

このコマンドで、JSON フォームオブジェクトをパラメーターとして渡す必要があります。ファイルのパスを以下のように指定します。

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

これでプラットフォーム上にフォームが再作成されます。フォームが作成されたら、**fhc forms get** コマンドでフォームを表示させます。このコードを好きなディレクトリーにコピーしておくと、今後このフォームを編集したい場合に使用できます。別の方法では、以下のように **fhc forms get** コマンドの結果をファイルにプリントしておくこともできます。

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

7.2.3. フォームの更新

開発中のある段階で、フォームの更新が必要になる場合もあるでしょう。これを行うには、フォームを保存したファイルのコンテンツを編集します。例えば、新たなテキストフィールドを追加する、またはあるルールをページに適用するといった場合、**fhc forms update** コマンドを使用すると既存フォームを更新することができます。

fhc forms update <path_to_form>

例を示します。

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



注記

この更新コマンドで最初にフォームを作成した際に使用したフォームファイルへのパスを渡すと、このファイルには ID が割り当てられていないので、フォームを複製することになります。つまり、フォームを更新するのではなく、新しいインスタンスが作成され、新たな ID が割り当てられます。

これで変更が適用されます。

プロジェクトにフォームを関連付ける方法については、[アプリフォームプロジェクトの作成](#) チュートリアル [プロジェクトにアプリフォームを関連付ける](#) セクションを参照してください。

7.3. FHC を使用したフォームの提出の作成

概要

本チュートリアルでは、フォームの提出の一覧表示、取得、生成の方法を説明します。

要件

ユーザーは、以下のパーミッションを持つ 1 つ以上のチームメンバーである必要があります。

- ドメイン — Drag & Drop Apps
- フォーム (表示 & 編集) — 提出 (表示 & 編集)

パーミッションについての詳細は、「[チームおよびコラボレーション](#)」を参照してください。

このチュートリアルを行うには、フォームが作成されている必要があります。新規フォームの作成については、[フォームの作成](#) を参照してください。

7.3.1. 提出の一覧表示

フォームの提出を一覧表示するには、**fhc submissions** または **fhc submissions list** コマンドを使用します。

```
fhc submissions list
```

これでアクセス可能な全フォームの提出が一覧表示されます。

7.3.2. 特定提出の応答

fhc submissions get コマンドを使用すると、ID に基づいて特定の提出を返すことができます。

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

これで特定の提出が返されます。**.pdf** 接尾辞をファイル名に加えると、提出を PDF 形式で保存することもできます。

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

例を示します。

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

7.3.3. 提出のステータス

fhc submissions status コマンドを特定の提出の ID と実行すると、その提出のステータスを確認できます。

```
fhc submissions status 5335bc00d4598fdb5cae04f7
```

7.3.4. 提出の送信

fhc submissions submitdata コマンドで **submission.json** ファイルを渡すと、提出のデータを送信することができます。

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

例を示します。

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

提出に完了の push mark を行うには、**fhc submissions complete <submission-id>** コマンドを使用します。

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

PDF として保存した場合と同様に、提出は zip ファイルとして保存することも可能です。これは、あるプロジェクトにいくつもの提出がある場合に便利です。

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



注記

プロジェクトのパラメーターは、**app=<project-id>** のアプリフィールドに渡されます。

```
fhc submissions export file=submissions.zip app=H_DNbFNJWS9uZI7LJ0kut20
```

7.4. FHC を使用したフォームテーマの作成

概要

このチュートリアルでは、FHC を使用してフォームにテーマを作成する方法を説明します。

要件

ユーザーは、以下のパーミッションを持つ 1 つ以上のチームメンバーである必要があります。

- ドメイン — Drag & Drop Apps
- テーマ (表示 & 編集)

パーミッションについての詳細は、「[チームおよびコラボレーション](#)」を参照してください。

7.4.1. テーマの一覧表示

fhc target コマンドを使用してドメインをターゲットにします。**fhc login** を使用してログインします。

ドメイン上の全テーマを一覧表示するには、**fhc themes** コマンドを使用します。

7.4.2. テーマの作成

テーマの作成には、**fhc themes create** コマンドを使用します。このコマンドでは、パラメーターに json ファイルを渡す必要があります。

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

例を示します。

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

これでプラットフォーム上に指定されたテーマが作成されます。テーマには ID が割り当てられます。

テーマが正常に作成されると、コンソールに出力されます。

出力をファイルに保存し、パスを書き留めます。このパスは、テーマ更新の際に使用されます。

7.4.3. テーマをアプリに関連付ける

テーマをアプリに追加する前に、まずアプリをプロジェクトに関連付ける必要があります。アプリをプロジェクトに追加する方法については、[プロジェクトおよびアプリに関する作業](#)を参照してください。

テーマをアプリに関連付けるには、**fhc themes app set** コマンドを使用します。

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

例を示します。

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

7.4.4. テーマの更新

ある段階でテーマを更新するには、**fhc themes update** コマンドを使用します。テーマを作成した後に保存した場所に移動します。そこで変更を加えてから、ファイルを保存します。これでテーマを更新することができます。

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

これで変更が読み込まれます。更新されたテーマは、再度コンソールに出力されます。

付録A 複数の SSH キーを異なるドメイン/プロジェクトに使用する

Red Hat Mobile Application Platform ホスト型 (RHMAP) 内の複数のドメインに異なるユーザーとしてアクセスする必要がある場合は、Git レポジトリにアクセスする際に別の SSH キーを使用する必要があります。各 SSH キーは単一ユーザーとの関連付けのみが可能であるためです。各ホストに適用するキーを Git/SSH に指示する必要があります。これは通常、SSH 設定ファイルを使用してセットアップします。

SSH は通常、利用可能なすべての ID ファイルの使用を試み、クラスター内でいくつかの ssh キーが有効である場合は、そのクラスターで有効でかつクローンしようとしているプロジェクトにアクセスがないキーを使用する可能性があります。

ローカルの SSH 設定ファイルは通常 `~/.ssh/config` で、これを更新すると特定ホストに使用する ID を一覧表示できますが、**IdentitiesOnly yes** 句を追加して、すべての ID ファイルを使用するのではなく、特定ホストに指定された ID ファイルのみを使用するよう SSH に指示する必要があります。

たとえば、アプリのレポジトリに 2 人の別のユーザーとしてアクセスしたい場合は、それぞれのユーザーに SSH キーを作成し、それに対応するユーザーアカウントにアップロードします。

下記の例では、SSH キーファイル `/Users/jbloggs/.ssh/key_for_domain1_id_rsa` が `domain1.redhatmobile.com` のユーザーに、`/Users/jbloggs/.ssh/key_for_domain2_id_rsa` が `domain2.redhatmobile.com` のユーザーにアップロードされています。

アプリのレポジトリへのアクセスを許可するには、以下のようにします。

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

ssh 設定ファイルは以下のようになります。

```
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
```

これで以下の git コマンドを使用してレポジトリをクローンできます。

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

ssh 設定ファイルにより、適切な SSH キーが各ドメインに使用されます。ssh 設定に関する詳細情報は、SSH のローカルインストールについてのドキュメントを参照してください。お使いのオペレーティングシステムによっては、**man ssh_config** のコマンドで入手できる場合があります。



注記

上記の設定およびドメイン名は例なので、実際のドメインとして使用しないでください。

