



Red Hat Mobile Application Platform

ホスト型 3

サーバーサイド開発者ガイド

Red Hat Mobile Application Platform ホスト型 3 向け

Red Hat Customer Content
Services

Red Hat Mobile Application Platform ホスト型3 サーバーサイド開発者ガイド

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 でクラウドアプリおよびサービスを開発するためのガイドです。

目次

第1章 クラウドアプリの開発	7
1.1. クラウド開発	7
1.1.1. 概要	7
1.1.2. クラウドアプリ構造	7
1.1.2.1. application.js	7
1.1.2.2. package.json	7
1.1.3. サンプル	8
1.2. 環境	8
1.2.1. 環境とは	9
1.2.2. 環境とビジネスオブジェクトの対話	9
1.2.3. 環境がない — 機能が制限される	9
1.2.3.1. チームメンバーシップの更新	9
1.2.3.2. 管理者環境のセットアップ	10
1.2.4. 他のリソース	10
1.3. クラウドアプリでの NODE.JS の使用	10
1.3.1. 概要	10
1.3.2. Node.js モジュールの使用	10
1.3.3. 環境変数	12
1.4. NPM を使用した NODE.JS 依存関係の管理	12
1.4.1. 概要	12
1.4.2. npm とアプリのステージ	13
1.4.3. npm のベストプラクティス	13
1.4.3.1. npm-shrinkwrap ファイルの使用	13
1.4.4. node_modules のアップロード	13
1.5. NODE.JS バージョンの設定	14
1.5.1. fhc の使用	14
1.5.2. Studio の使用	14
第2章 クラウドアプリでの RHMAP 機能の使用	15
2.1. API MAPPER を使用した JSON API 応答の変換	15
2.1.1. 概要	15
2.1.2. セットアップ	15
2.1.3. 使用例	16
2.1.3.1. 要求の作成	16
2.1.3.2. マッピングの追加	17
2.1.3.3. マップされた API の使用	18
2.1.4. カスタムの変換	18
2.1.4.1. カスタムの変換の作成	18
2.1.4.2. サンプル	18
2.2. プッシュ通知を使用したアプリケーションの開発	19
2.2.1. Firebase Cloud Messaging クレデンシャルの取得と google-services.json ファイルのダウンロード	19
2.2.2. Push Notification Hello World テンプレートからのプロジェクトの作成	19
2.2.3. クライアントアプリのパッケージ名の定義	20
2.2.4. プッシュサポートのセットアップ	21
2.2.5. クライアントアプリへの google-services.json ファイルの統合	21
2.2.6. config.xml ファイルの設定	22
2.2.7. クライアントアプリバイナリーのビルド	22
2.2.8. アプリのテスト	22
2.3. MBAAS サービスからフォームフィールドに動的にデータを入力する	23
2.3.1. 概要	23
2.3.2. データソースのサービスの作成	23
2.3.3. データソースの定義	24

2.3.3. フォームフィールドでのデータソースの使用	25
2.4. アプリへの統計の追加	26
2.4.1. 概要	26
2.4.2. カウンターとタイマーの作成	26
2.4.3. 統計の表示	27
第3章 RHMAP データ同期フレームワークの使用	28
3.1. データ同期フレームワーク	28
3.1.1. ハイレベルアーキテクチャー	28
3.1.2. API	29
3.1.3. スタートガイド	29
3.1.3.1. 不必要な同期ループの回避	31
3.1.4. 同期フレームワークの高度な機能の使用	31
3.1.5. さらに詳しく知るために	32
3.1.5.1. データセット	32
3.1.5.2. 競合	32
3.2. 同期に関する用語	33
3.2.1. 同期プロトコル	33
3.2.2. 同期サーバー	34
3.2.3. 同期クライアント	34
3.2.4. 同期サーバーループ	34
3.2.5. 同期クライアントループ	34
3.2.6. 同期周期	34
3.2.7. データセット	34
3.2.8. データセットバックエンド	35
3.2.9. データセットハンドラー	35
3.2.10. データセットクライアント	35
3.2.11. データセットレコード	35
3.2.12. ハッシュ	35
3.3. 同期サーバーのアーキテクチャー	36
3.3.1. アーキテクチャー	36
3.3.1.1. HTTP ハンドラー	36
3.3.1.1.1. 同期 HTTP ハンドラー	36
3.3.1.1.2. 同期レコード HTTP ハンドラー	36
3.3.1.2. キュー	36
3.3.1.3. プロセッサ	37
3.3.1.4. 同期スケジューラー	37
3.4. データ同期設定ガイド	37
3.4.1. 同期周期の設定	37
3.4.2. ワーカーの設定	39
3.4.2.1. 間隔の目的	39
3.4.2.2. ワーカーバックオフ	39
3.5. 同期サーバーアップグレードに関する注記	40
3.5.1. 概要	40
3.5.2. 前提条件	40
3.5.3. データハンドラー関数署名の変更	40
3.5.4. 動作の変更	42
3.5.5. ロガーの変更	42
3.6. 同期サーバーのパフォーマンスとスケーリング	43
3.6.1. 概要	43
3.6.2. パフォーマンスの検査	43
3.6.3. パフォーマンスの理解	44
3.6.3.1. CPU 使用率	44

3.6.3.2. キュー内の残りのジョブ	44
3.6.3.3. API 応答時間	44
3.6.3.4. MongoDB 操作時間	44
3.6.4. 同期サーバーのスケーリング	45
3.6.4.1. ホストされた MBaaS でのスケーリング	45
3.6.4.1.1. Node.js クラスターモジュールの使用	45
3.6.4.1.2. より多くのアプリのデプロイ	45
3.6.4.2. 自己管理された MBaaS でのスケーリング	45
3.7. 同期サーバーにより作成された MONGODB コレクション	46
3.7.1. 概要	46
3.7.2. 同期サーバーコレクション	46
3.7.2.1. fhsync_pending_queue	46
3.7.2.2. fhsync_<datasetId>_updates	46
3.7.2.3. fhsync_ack_queue	47
3.7.2.4. fhsync_datasetClients	47
3.7.2.5. fhsync_<datasetId>_records	47
3.7.2.6. fhsync_queue	47
3.7.2.7. fhsync_locks	47
3.7.3. キューコレクションのプルーニング	48
3.8. 同期サーバーデバッグガイド	48
3.8.1. クライアントの変更は適用されない	48
3.8.2. データセットバックエンドに適用された変更は他のクライアントに伝播されない	49
3.8.3. デバッグログの有効化	50
第4章 RHMAP と他のサービスとの統合	52
4.1. MBAAS サービスの概要	52
4.2. 認証に対する SAML の使用	52
4.2.1. 概要	52
4.2.2. SAML の概要	53
4.2.3. テンプレートのセットアップ	53
4.2.3.1. SAML サービスの作成	53
4.2.3.2. プロジェクトのセットアップ	54
4.2.4. 仕組み	55
4.3. REDHAT OPENSIFT ONLINE PAAS へのクラウドアプリのステージング	58
4.3.1. OpenShift Online とは	58
4.3.1.1. OpenShift Online と RHMAP との統合	58
4.3.1.2. RHMAP による利用可能な OpenShift Online ギアへの影響	59
4.3.1.3. 制限	59
4.3.2. スタートガイド	59
4.3.2.1. 初回ログインおよびセットアップ	59
4.3.2.2. サンプルアプリケーションの作成	61
4.3.2.3. RHMAP を使用した OpenShift Online へのクラウドアプリのデプロイ	61
4.3.2.4. fhc を使用した OpenShift Online へのクラウドアプリのデプロイ	63
4.3.3. どのように連携するか	66
4.3.3.1. ギア	66
4.3.3.2. 環境	67
4.3.3.3. ロギングおよびデバッグ	68
4.3.3.4. OpenShift Online にデプロイされたアプリの削除	69
4.3.3.5. SSH キー	70
4.3.3.6. ドメイン	71
4.4. GOOGLE OAUTH を使用した認証ポリシーのセットアップ	71
4.4.1. Google OAuth アプリ	71
4.4.2. 承認	72

4.4.2.1. ユーザーがプラットフォームに存在することを確認	72
4.4.2.2. ユーザーが認証されているかどうかを確認	72
4.4.3. 認証ポリシーに対するユーザーの追加/削除	73
第5章 データの格納	74
5.1. データブラウザー	74
5.1.1. 概要	74
5.1.2. データブラウザーの使用	74
5.1.2.1. コレクションの表示/追加	74
5.1.2.2. コレクション内のデータ表示	75
5.1.2.2.1. データの並び替え	75
5.1.2.2.2. データのフィルタリング	75
5.1.2.3. データの編集	76
5.1.2.3.1. インラインエディターを使用した編集	76
5.1.2.3.2. 高度なエディターを使用した編集	77
5.1.2.3.2.1. 動的エディターを使用した編集	77
5.1.2.3.2.2. Raw JSON エディターを使用した編集	78
5.1.3. データのエクスポートとインポート	78
5.1.3.1. データのエクスポート	79
5.1.3.2. データのインポート	79
5.1.3.2.1. インポート形式	79
5.1.3.2.2. JSON のインポート	80
5.1.3.2.3. CSV のインポート	80
5.1.3.2.4. BSON または MongoDB 出力のインポート	80
5.1.4. データベースのアップグレード	81
5.2. アプリケーションデータのエクスポート	81
5.2.1. エクスポートデータ形式	82
5.2.2. アプリケーションデータのエクスポート	82
5.2.2.1. 新しいエクスポートジョブの開始	82
5.2.2.2. エクスポートジョブのステータスの問い合わせ	83
5.2.2.3. エクスポートデータのダウンロード	83
5.3. アプリケーションデータのインポート	83
5.3.1. アプリケーションデータのインポート	84
5.3.1.1. 新しいインポートジョブの開始	84
5.3.1.2. インポートジョブのステータスの問い合わせ	84
第6章 プロジェクトへのクライアントアプリとクラウドアプリのインポート	85
6.1. 空のプロジェクトの作成	85
6.2. クライアントアプリのインポート	85
6.3. クラウドアプリのインポート	86
6.4. クライアントアプリの要件	87
6.4.1. Cordova Light アプリ	87
6.4.2. Cordova アプリ	87
6.4.3. Web App	88
6.4.4. ネイティブ Android	88
6.4.5. ネイティブ iOS	88
6.4.6. ネイティブ Windows Phone 8	89
6.4.7. Xamarin	89
6.5. クラウドアプリの要件	89
第7章 古いバージョンの RHMAP からの移行	90
7.1. V2 から V3 への移行の概要	90
7.1.1. アプリとプロジェクト	90
7.1.2. 移行する理由	90

7.1.3. Studio での自動移行	90
7.1.4. fhc を使用した自動移行	94
7.2. V2 から V3 への移行ガイド	94
7.2.1. v2 アプリとは	94
7.2.2. v3 アプリとは	95
7.2.3. プロジェクトとは	96
7.2.4. 新規プロジェクト	96
7.2.5. クラウドの新しい v3 アプリ	96
7.2.5.1. クラウドアプリ	96
7.2.5.2. /cloud フォルダの移行	97
7.2.5.3. /shared フォルダの移行	97
7.2.5.4. (オプション) Node.js SDK の変更	97
7.2.6. クライアント用の新しい v3 アプリ	97
7.2.6.1. Cordova Light アプリ	97
7.2.6.2. /client フォルダの移行	97
7.2.6.3. /shared フォルダの移行	97
7.2.6.4. Javascript SDK インジェクション	97
7.2.6.5. fhconfig.json	98
7.2.6.6. (オプション) レガシー FeedHenry API	98
7.2.6.7. (オプション) v2 アプリクライアントパッケージの移行	99
7.2.6.8. (オプション) v2 アプリの embed/mobile 移行	99
7.3. FH-NODEAPP & FH-WEBAPP/FH-API から FH-MBAAS-API/FH-MBAAS-EXPRESS への移行	99
7.3.1. package.json の変更	100
7.3.2. \$fh を fh に置き換えるためにクラウドコードを変更	100
7.3.3. application.js の変更	100
7.3.4. Grunt の使用	100
7.4. 組み込みアプリから基本的な WEB アプリへの移行	101
7.4.1. 概要	101
7.4.2. 移行	101
7.4.2.1. 移行されたハイブリッドアプリのクローン	101
7.4.2.2. 新しい基本的な Web アプリの作成	101
7.4.2.3. 新しい基本的な Web アプリへの組み込みアプリのコピー	102
7.5. ロールからチームへの移行	102
7.5.1. パーミッションの追加	102
7.5.2. アナリティクス (analytics)	103
7.5.3. フォームエディター (formseditor)	103
7.5.4. フォーム管理者 (formsadmin)	103
7.5.5. 提出ビューワ (submissionsviewer)	104
7.5.6. 提出エディター (submissionseditor)	104
7.5.7. 開発管理者 (devadmin)	105
7.5.8. 開発者 (dev)	105
7.5.9. サービス管理者 (serviceadmin)	106
7.5.10. ドメイン管理者 (portaladmin)	106
7.5.11. 顧客管理者 (customeradmin)	107
7.5.12. リセラー管理者 (reselleradmin)	107
7.5.13. 管理者 (admin)	107

第1章 クラウドアプリの開発

1.1. クラウド開発

1.1.1. 概要

Red Hat Mobile Application Platform ホスト型 (RHMAP) の中心的なコンセプトの 1 つはクラウドアプリです。クラウドアプリは、モバイルデバイスにデプロイされたクライアントアプリとビジネスロジックおよびデータが含まれるバックエンドシステム間の通信を処理するサーバーサイドアプリケーションです。

クラウドアプリは [Node.js](#) を使用して開発されます。**Node.js** は、高速でスケーラブルなネットワークアプリケーションを簡単に構築できるようにする、**Chrome** の **JavaScript** ランタイムに基づいて構築されたプラットフォームです。**Node.js** は軽量さと効率性を実現するイベント駆動の非ブロッキング I/O モデルを使用し、分散されたデバイスで実行するデータ集約型リアルタイムアプリケーションに最適です。

Node.js 開発に精通していない場合は、[Node Beginner Website](#) を参照することをお勧めします。

1.1.2. クラウドアプリ構造

クラウドアプリが RHMAP で作成された場合、その新しく作成されたクラウドアプリは実質的には事前設定された Node.js アプリケーションです。事前設定された対応するクライアントアプリも生成された場合、クラウドアプリにはすべての基本的な設定 (ルート) が含まれるため、クライアントアプリとクラウドアプリは同期されます。また、クラウドアプリをホストするインフラストラクチャーもインプレース状態 (MBaaS) にあり、簡単にアクセスできます。

設定とインフラストラクチャーがすべてインプレース状態にあるため、クラウドアプリは MBaaS にデプロイできます。また、追加コードを必要とせずにクライアントアプリの要求をルーティングできます。上級 Node.js 開発者がサーバーを再設計したり、独自のサーバーを実装したりする場合、これは、**application.js** ファイル内のコードを操作することによって可能になります。



注記

アプリのメインエントリーポイントを表すファイルには必ず **application.js** という名前を付ける必要があります。

クラウドアプリでは次のファイルがデフォルトで提供されます。

1.1.2.1. application.js

このファイルは、アプリケーションが **MBaaS** (クラウド実行環境) にデプロイされたときに呼び出されます。ほとんどの場合は、このファイルを変更する必要はありません。このファイルは、クライアントからの **fh.cloud()** 要求を処理し、適切にルーティングするよう設定されます。

1.1.2.2. package.json

クラウドアプリの設定ファイルは、主に依存関係の管理に使用されます。サードパーティーの Node モジュールを使用している場合、それらのモジュールはこのファイルで指定されます。

関連項目:

- ※ クラウドアプリでの Node.js モジュールの使用
- ※ npm を使用した Node.js 依存関係の管理
- ※ `package.json` の公式ドキュメンテーション

1.1.3. サンプル

クラウドアプリは非常に簡単に準備できます。Studio で新しいプロジェクトを作成するときに **Hello World Project** テンプレートを試したり、GitHub ([feedhenry-templates/helloworld-cloud](https://github.com/feedhenry-templates/helloworld-cloud)) でテンプレートのソースコードを探したりすることにより単純なサンプルを使用できます。以下にテンプレートの簡単な概要を示します。

最初の手順は `application.js` でカスタムルートを設定することです。

```
app.use('/hello', require('./lib/hello.js')());
```

参照される `hello.js` ファイルでは、特定のルートのロジックを定義します。

```
var express = require('express');
var bodyParser = require('body-parser');
var cors = require('cors');

function helloRoute() {
  var hello = new express.Router();
  hello.use(cors()); // enables cross-origin access from all domains
  hello.use(bodyParser()); // parses POST request data into a JS object

  hello.post('/', function(req, res) {
    res.json({msg: 'Hello ' + req.body.hello});
  });
  return hello;
}
module.exports = helloRoute;
```

`$fh.cloud` を使用したこのエンドポイントへのクライアントサイド呼び出しは以下のようになります。

```
$fh.cloud({
  path: 'hello',
  data: {
    hello: 'World'
  }
},
function (res) {
  // response handler
},
function (code, errorprops, params) {
  // error handler
}
);
```

1.2. 環境

1.2.1. 環境とは

環境は、プロジェクト、サービス、フォーム、およびアプリの開発サイクルを個別のステージに論理的に分離する方法です。環境は[ライフサイクル管理](#)の重要な要素です。

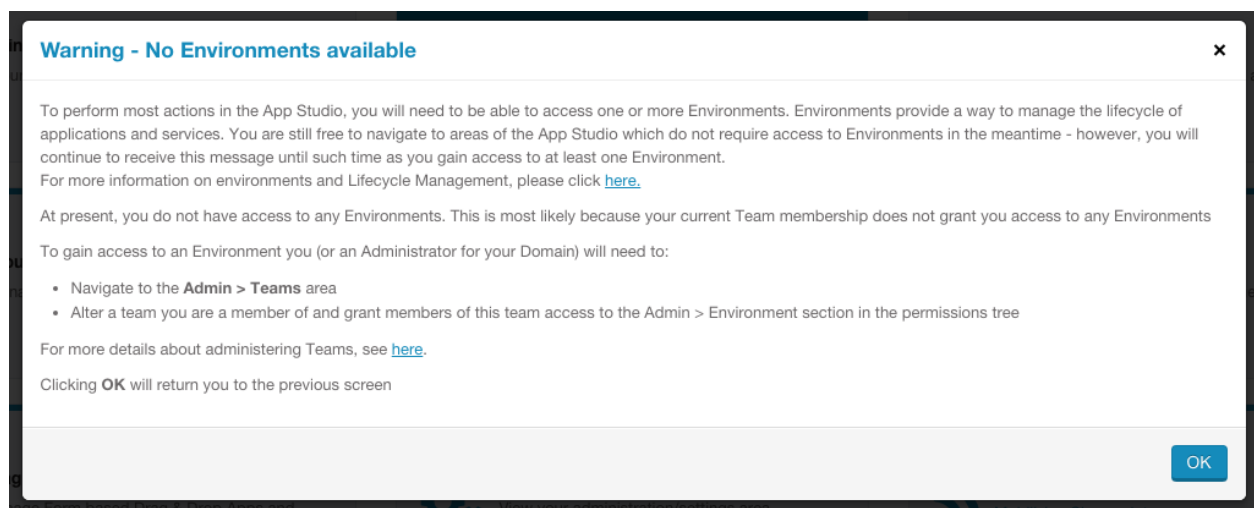
たとえば、プロジェクトには Dev、UAT、および Production の3つのステージが含まれることがあります。各ステージは環境により表されます。環境の数と種類はプラットフォームで設定可能であり、ドメインごとに変更できます。

1.2.2. 環境とビジネスオブジェクトの対話

App Studio で環境が使用される領域は以下のとおりです。

- ※ Projects (Apps を含む)
- ※ Services
- ※ Forms
- ※ Admin > Auth Policies

環境にアクセスせずにこれらの領域にアクセスすると、以下のような警告が表示されます。



1.2.3. 環境がない — 機能が制限される

App Studio でプロジェクト、サービス、フォーム、およびアプリに関連するほとんどのアクションを実行するには、1つまたは複数の環境にアクセスできる必要があります。

現在環境にアクセスできない場合は、以下の理由が考えられます。

- ※ 現在のチームメンバーシップにより環境の使用が許可されない
- ※ 管理者がまだ環境をセットアップしていない

1.2.3.1. チームメンバーシップの更新

環境にアクセスするには、ユーザー (またはドメインの管理者) が以下の操作を実行する必要があります。

- ※ **Admin > Teams** 領域に移動する
- ※ メンバーになっているチームを変更し、そのチームのメンバーに **Admin > Environment** ビ

ジネスオブジェクトへのアクセスを与える

チームの管理の詳細については、[ここ](#)をクリックしてください。

1.2.3.2. 管理者環境のセットアップ

チームがセットアップされ、メンバーシップが適切な場合は、管理者に連絡してください。環境へのアクセスをチームによって与える前に、管理者が環境をセットアップする必要があります。

1.2.4. 他のリソース

※ [チームおよびコラボレーション](#)

※ [ライフサイクル管理](#)

1.3. クラウドアプリでの **NODE.JS** の使用
















1.3.1. 概要

本書では、Node.js モジュールをクラウドアプリに含め、使用方法について説明します。Red Hat Mobile Application Platform (RHMAP) では、クラウドアプリを開発するためにレジストリーで利用可能な Node.js パッケージのモジュールを使用できます。パブリックレジストリー (npmjs.com) では、フレームワーク、コネクタ、およびさまざまなツールを含む数千のパッケージを見つけることができます。

npm is the package manager for **mobile**

 **195,861** total packages  **34,714,562** downloads in the last day  **568,390,822** downloads in the last week  **2,305,601,665** downloads in the last month

packages people 'npm install' a lot

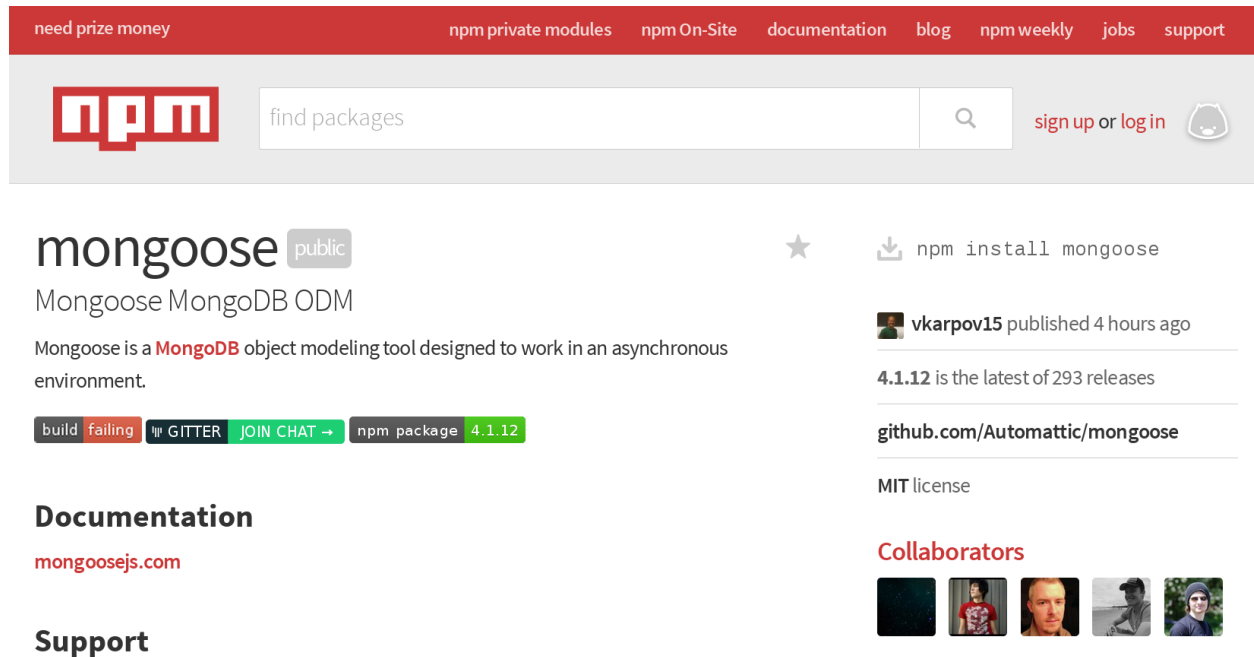
 browserify browser-side require() the node way 102.6 published 3 months ago by substack	 express Fast, unopinionated, minimalist web framew... 4.13.1 published 3 months ago by dougwilson	 pm2 Production process manager for Node.js app... 0.14.3 published 4 months ago by jshkurti
 grunt-cli The grunt command line interface. 0.1.13 published 2 years ago by tkellen	 npm a package manager for JavaScript 2.13.0 published 4 months ago by zkat	 karma Spectacular Test Runner for JavaScript. 0.13.1 published 3 months ago by dignifiedquire
 bower The browser package manager 1.4.1 published 7 months ago by sheerun	 cordova Cordova command line interface tool 5.1.1 published 4 months ago by stevegill	 coffee-script Unfancy JavaScript 1.9.3 published 5 months ago by jashkenas
 gulp The streaming build system 3.9.0 published 5 months ago by phated	 forever A simple CLI tool for ensuring that a given nod... 0.14.2 published 4 months ago by indexzero	 statsd A simple, lightweight network daemon to coll... 0.7.2 published a year ago by pkhzzrd
 grunt The JavaScript Task Runner 0.4.5 published a year ago by cowboy	 less Leaner CSS 2.5.1 published 5 months ago by agatronic	 yo CLI tool for running Yeoman generators 1.4.7 published 5 months ago by sindresorhus

RHMAP 自体の機能の一部はノードモジュールとして公開されます。たとえば、プラットフォームで新しいクラウドアプリを作成する場合、デフォルトでは **package.json** に Mobile Backend As A Service (MBaaS) の基盤となる **fh-mbaas-api** という名前のモードモジュールが含まれます。

1.3.2. Node.js モジュールの使用

アプリにある機能が必要であり、一から作成したくない場合は、npmjs.com で既存のモジュールを探すことをお勧めします。

たとえば、**Mongoose** ODM フレームワークを使用する場合は、レジストリーの**mongoose** パッケージの公式ページ (npmjs.com/package/mongoose) を参照してください。



プロジェクトでパッケージの最新バージョンを使用する場合は、クラウドアプリのディレクトリーで以下のコマンドを実行してパッケージをローカルにインストールし、依存関係としてクラウドアプリの **package.json** ファイルに追加します。

```
npm install --save mongoose
```

または、以下の例のように、**dependencies** オブジェクトの最後で依存関係を追加することにより依存関係を **package.json** に手動で追加します。

```
{
  "name": "my-fh-app",
  "version": "0.2.0",
  "dependencies": {
    "body-parser": "~1.0.2",
    "cors": "~2.2.0",
    "express": "~4.0.0",
    "fh-mbaas-api": "~4.9.0",
    "mocha": "^2.1.0",
    "request": "~2.40.0",
    "mongoose": "~4.1.12" // added mongoose dependency
  }
}
```

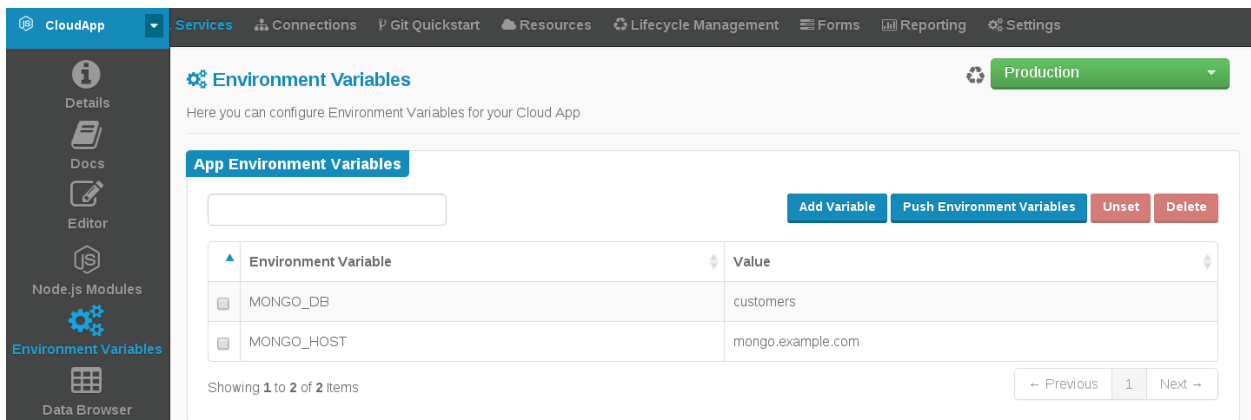
この時点で、パッケージで提供されるモジュールを使用できます。多くの場合、npmjs.com のパッケージページには、使用方法とドキュメンテーションへのリンクが含まれます。すべてのモジュールで共通な次の手順では、コードでモジュールを**必須**にして使用できるようにします。

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/my_database');
...
```

この例では、データベース URL はソースでハードコーディングされます。ただし、より一般的なシナリオでは開発ライフサイクルの異なるステージ (開発、テスト、本番稼働) に応じて異なるデータベースを使用します。これは、環境変数で設定値を定義することにより実現できます。

1.3.3. 環境変数

ホスト名、ユーザー名、パスワードなどのクラウドアプリの設定の部分がライフサイクルステージで異なる場合は、環境変数を使用してこのような設定値を設定できます。これにより、コードを変更せずに設定を変更できるようになります。Studio において各クラウドアプリには **Environment Variables** セクションがあり、変数を作成または削除したり、値を設定したり、アプリの実行中インスタンスに変数をプッシュしたりできます。



環境変数は、異なる値を異なる環境にプッシュする機能と組み合わせると特に有用です。たとえば、テスト用と本番稼働用に異なるデータベースホストを使用できます。他の利点は、機密性がある設定情報をアプリケーションのコードベース外に格納できることです。

クラウドアプリでは、環境変数は **process.env** オブジェクトを介してアクセスできます。環境変数を使用するよう変更された Mongoose 接続呼び出しの前の例は以下のようになります。

```
mongoose.connect('mongodb://' + process.env.MONGO_HOST + '/' +
  process.env.MONGO_DB);
```

1.4. NPM を使用した NODE.JS 依存関係の管理

1.4.1. 概要

npm (ノードパッケージマネージャー) は Node.js 用の依存関係管理ツールであり、Node.js アプリケーションを開発するために必要になります。原則として、npm は Maven、Cocoapods、NuGet などの他の依存関係管理システムに似ています。

- ✳ プロジェクトでは標準化されたファイル (Node.js の場合は **package.json**) で依存関係が宣言される
- ✳ パッケージは 1 か所 (registry.npmjs.org にある npm レジストリー) でホストされる
- ✳ 依存関係バージョンは、ワイルドカードまたは範囲を使用して明示的に指定できる
- ✳ 推移的な依存関係のバージョンは自動的に決定または修正できる ([npm-shrinkwrap ファイルの使用](#)を参照)
- ✳ インストール後にパッケージがローカルでキャッシュされる (**node_modules** フォルダー)

クラウドアプリでの Node.js モジュールの使用については、[クラウドアプリでの Node.js モジュールの使用](#)を参照してください。

1.4.2. npm とアプリのステージ

クラウドアプリをデプロイする場合は、アプリの環境で **npm** コマンドが実行され、アプリの依存関係がダウンロードおよびインストールされます。以下のことに注意してください。

- ※ 初めてクラウドアプリがデプロイされる場合は、完全な **npm install** が実行されます。このため、アプリの最初のデプロイメントには非常に時間がかかることがあります。
- ※ 以降の各デプロイにおいて、**package.json** ファイルが以前のデプロイから変更された場合は、**npm update** が実行されます。
- ※ **package.json** に変更が加えられない場合、**npm** コマンドは実行されません (**update** と **install** も実行されません)。
- ※ **npm** を完全に再実行する場合は、以下のように削除ステージを実行します。
 - Studio でクラウドアプリの **Deploy** セクションの **Clean Stage** チェックボックスをオンにします。
 - **fhc** を使用している場合は、**--clean** オプション (つまり、**fhc app stage <app-id> <env-name> --clean**) を使します。

1.4.3. npm のベストプラクティス

npm を使用した基本的なベストプラクティスは、ノードモジュールでバージョン管理がどのように機能するかを理解し ([The semantic versioner for npm](#) を参照)、**package.json** の依存関係に対して ***** を使用しないことです。

ローカルで開発する場合に役に立つヒントは、**npm** (**npm install request --save** など) を使用してモジュールをインストールするときに **--save** フラグを使用することです。**--save** フラグを使用すると、インストールされたバージョンの **package.json** の依存関係セクションに新しいモジュールが追加されます。また、**shrinkwrap** ファイルの使用もお勧めします。

1.4.3.1. npm-shrinkwrap ファイルの使用

npm-shrinkwrap.json ファイルは、パッケージの推移的な依存関係のバージョンをロックダウンするため、パッケージのインストール時に使用する各依存関係の適切なバージョンを制御できます。

shrinkwrap ファイルを作成するには、以下のコマンドを実行します。

```
npm shrinkwrap
```

生成された **npm-shrinkwrap.json** ファイルをクラウドアプリのルートディレクトリーに配置します。RHMAP のアプリで **npm install** が呼び出されると、**shrinkwrap** ファイルで定義されたバージョンが使用されます。

shrinkwrap の詳細については、公式な **npm** ドキュメンテーション ([Lock down dependency versions](#)) を参照してください。

1.4.4. node_modules のアップロード

RHMAP では、**node_modules** ディレクトリーをコミットし、クラウドアプリで使用できます (ただし、これは推奨される方法ではありません)。この場合、アプリがステージングされても npm はまったく実行されず (**clean** オプションが指定されている場合であっても)、アップロードしたモジュールはアプリを実行するために使用されます。ただし、ネイティブのノードモジュールは実行するのと同じアーキテクチャー (RHMAP のインスタンスによって異なる可能性があります) でコンパイルする必要があります。

1.5. NODE.JS バージョンの設定

1.5.1. fhc の使用

最初に、fhc の最新バージョンを用意します。

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

次に fhc ランタイムコマンドを使用して、環境で利用可能なランタイムを確認します。

```
fhc runtimes --env=dev
```

このコマンドを実行すると、**4.4.7** などのバージョンが出力されます。

左側にはランタイムの名前が示され、右側には特定のバージョンが示されます。

```
fhc app stage --app=<APP_GUID> --runtime=<node version>
```

ステージ中にアプリのランタイムを設定するには、<node version> (**4.4.7** など) を追加します。

これにより、ノードのアプリのランタイムが v4.4.7 に設定されます。

別のランタイムに変更するには、ランタイムの引数をステージコマンドに再び追加します。

1.5.2. Studio の使用

Studio のデプロイ画面では、アプリが設定された現在のランタイムを確認できます。また、ここで新しいランタイムを設定し、アプリをデプロイすることもできます。

第2章 クラウドアプリでの RHMAP 機能の使用

2.1. API MAPPER を使用した JSON API 応答の変換

2.1.1. 概要

サードパーティー API からデータをアクセスするときは、多くの場合、アプリケーションの受信データの構造または形式を変更する必要があります。たとえば、値を異なる形式に変換し、データを事前処理して、新しい値を派生させたり、アプリケーションのモデルを適合させるためにフィールドの名前変更または削除を行ったりします。

RHMAP には、データ変換ロジックをカプセル化し、主要なアプリケーションロジックから抽象化することを可能にする **API Mapper** という名前の視覚的なツールが含まれます。API Mapper を使用すると、以下のことを行って JSON API の応答を簡単に変換できるようになります。

- ※ フィールド名を変更する。
- ※ 必要でないフィールドを除外する。
- ※ 組み込みまたはカスタムの変換を使用してフィールド値を変換する。

2.1.2. セットアップ

API Mapper はクラウドサービステンプレートとして提供されます。API Mapper の 1 つのインスタンスは複数のクラウドアプリとクラウドサービスで使用でき、複数の異なる API を変換するためにも使用できます。ただし、必要な場合は、任意の数の API Mapper インスタンスをデプロイできます。

API Mapper を使用するには以下の手順に従ってください。

1. **API Mapper** サービスをデプロイします。
 - a. **Services & APIs** セクションで、**Provision MBaaS Service/API** をクリックします。
 - b. リストで **API Mapper** を見つけ、**Choose** をクリックします。
 - c. **API Mapper** などの名前を入力し、**Next** をクリックします。
 - d. API Mapper が作成後にデプロイされる環境を選択し、**Next** をクリックします。
 - e. **Finish** をクリックし、デプロイメントが完了するまで待機します。
2. サービスをパブリックにします。

新しく作成された API Mapper サービスの **Service Details** ページにある **Access Control** セクションで、**Make this Service Public to all Projects and Services** チェックボックスをオンにします。次に **Save Service** をクリックします。

Security

Access Control

Here you can control which Projects and Services have permission to call this Service

Select Projects

Select Services

☒ Make this Service Public to all Projects and Services (Public to all projects)

Save Service

3. サービスのデータベースをアップグレードします。

API Mapper には、**アップグレードされたデータベース**が必要です。作業を進める前に、API Mapper サービスが完全にデプロイされ、実行中であることを確認します。データベースをアップグレードするには、以下の手順を実行します。

- a. API Mapper サービスの **Data Browser** セクションで、右上隅にある **Upgrade Database** ボタンをクリックし、**Upgrade Now** をクリックして確定します。アップグレードプロセスが完了するまで待機します。
- b. **Deploy** セクションの **Deploy Cloud App** をクリックして API Mapper サービスを再デプロイします。

この時点で、サービスの **Preview** セクションから API Mapper の管理インターフェースにアクセスできます。

データベースのアップグレードが完了したら、プレフィックス **fhsync_** の付いた新たなコレクションが作成され、同期機能が有効になります。Red Hat では、同期機能を使う予定がない場合でも、これらのコレクションを保持しておくことを推奨しています。

2.1.3. 使用例

この例は、要求、マッピング、およびカスタム変換を作成する方法を示しています。デモのために、この手順では変換するソース API の例としてパブリックな Github API を使用します。

2.1.3.1. 要求の作成

feedhenry-templates/fh-api-mapper リポジトリに関する情報を取得するために、Github API エンドポイントの要求を作成します。

1. API Mapper のホームページから、**New Request** をクリックします。
2. URL フィールドに API 要求の完全な URL を貼り付けます。

`https://api.github.com/repos/feedhenry-templates/fh-api-mapper`

3. Github API を使用するために、各要求で **User-Agent** ヘッダーが送信されるようにします。以下の値を入力します。

‣ **Header Key:** User-Agent

‣ **Header Value:** FHApiMapper

4. 次に、この要求の内部マウントパスを選択します。**Mount Path** タブを選択し、パスを入力します。この例では、マウントパスとして **/thisrepo** を使用します。マップされた要求はこのパスで利用できます (パスはこの場合以下の URL になります)。

`http://<serviceURL>/thisrepo`

5. **Create Request** をクリックしてデータベースに要求を格納します。
6. **Send Request** をクリックして要求を送信し、応答を取得します。
7. **Response** セクションで、**Response Headers** セクションと **Response Body** セクションが期待したように表示されることを確認します。この時点で、変換の前に元の応答本文を確認できます。

2.1.3.2. マッピングの追加

要求を保存し、内部パスに対してマップした後で、応答の各フィールドに適用する変換を選択します。この手順は、フィールドの破棄および名前変更を行い、フィールド値を変換する方法を示しています。

1. **Response Mapping** セクションで、**Add a Mapping** をクリックします。

マッピングが作成されたら、応答で返されたフィールドのリストが左側に表示されます。

2. **owner** フィールドを破棄します。

owner フィールドをクリックして **Field Mapping** パネルで編集するためにそのフィールドを選択します。**Use this field** のチェックをオフにして **owner** フィールドとそのすべての子供を破棄します。

3. **id** フィールドの名前を **_id** に変更します。

左側にある **id** フィールドを選択します。**Rename Field** ボックスで、**_id** と入力します。

4. **private** フィールドの名前を **public** に変更し、値を反転します。

前の手順のように、フィールド **private** の名前を **public** に変更します。次に、**Transform** ドロップダウンリストの **invert** という名前の変換を選択します。



注記

Field Mapping セクションで行われたすべての変更は自動的に保存されます。

変換をセットアップした後は、**Response** セクションの **Response Body** タブで元の応答本文とマップされた応答本文を比較できます。元の応答は左側に表示され、マップされた応答は右側に表示されます。

マップされた応答には、値が **true** の **public** フィールドと **_id** フィールド (**id** ではない) が含まれ、**owner** フィールドは含まれません。

2.1.3.3. マップされた API の使用

Response セクションには、以下のものを含むさまざまなクラウドからマップ済み要求を呼び出すコード例を含む **Sample Code** タブが含まれます。

✳ [\\$fh.service API](#)

✳ Node.js **request** モジュール

✳ cURL

たとえば、コマンドラインから要求を呼び出すには、**cURL Request** コードをコピーし、コマンドラインに貼り付け、そのコマンドを実行します。マップされた応答はコンソールに表示されます。

2.1.4. カスタムの変換

組み込みのフィールド変換以外に、カスタムの変換関数を作成することもできます。

2.1.4.1. カスタムの変換の作成

カスタムの変換関数を登録するには、以下のコード例で示されているように、API Mapper の **application.js** ファイルの **/** ルートに対する関数に渡される設定オブジェクトでその変換関数を宣言します。

```
app.use('/', require('./lib/api')({
  transformations : {
    <transformation name> : {
      type: <'array' | 'number' | 'string' | 'boolean'>,
      transform: <unary function>
    }
  }
}));
```

カスタム変換は、**transformations** オブジェクトのプロパティとして定義されます。プロパティの名前は変換の名前に対応します。プロパティの値は **transformation definition** オブジェクトです。

各 **transformation definition** オブジェクトには以下の 2 つのプロパティがあります。

- ✳ **type**: 入力値の JavaScript 型を表す文字列。可能な値は、**array**、**number**、**string**、**boolean** です。
- ✳ **transform**: 変換関数。この関数は元のフィールドを唯一の引数として取得し、変換された値を返します。

2.1.4.2. サンプル

この例は、偶数を **0** に、奇数を **1** に変更する **even** という名前の変換を作成する方法を示しています。

1. API Mapper サービスの **Editor** に移動し、**application.js** ファイルを開きます。

2. **mixedArrayTransform** という名前の既存の 1 つのカスタム変換を含む **transformations** オブジェクトの宣言を探します。
3. **transformations** プロパティの値を以下のオブジェクトに置き換えます。

```
{
  even: require('./transformations/even.js')
}
```

4. 以下の内容で新しい **transformations/even.js** ファイルを作成します。

```
// First, tell the mapper it operates on numbers
exports.type = 'number';
// then, implement the function.
exports.transform = function(n) {
  return n%2;
};
```

even という名前の新しい変換が、数値型用の API Mapper UI で利用可能になります。

2.2. プッシュ通知を使用したアプリケーションの開発

概要

このチュートリアルでは、プラットフォームの組み込みのプッシュ通知サーバーから送信されたプッシュ通知を受け取るサンプルアプリケーションを構築するプロセスについて説明します。この例で作成するアプリケーションは Cordova に基づき、Android と関連する Firebase Cloud Messaging (旧名は Google Cloud Messaging) プラットフォームを対象にしています。ただし、他のすべてのサポート対象プラットフォームに対する手順は類似しています。

Red Hat Mobile Application Platform ホスト型でのプッシュ通知サポートの詳細については、[Push Notifications](#) を参照してください。

2.2.1. Firebase Cloud Messaging クレデンシャルの取得と **google-services.json** ファイルのダウンロード

組み込みの UnifiedPush Server (UPS) から Google のプッシュ通知ネットワークにアクセスできるようにするには、最初にサーバーキーを取得し、Firebase コンソールでプロジェクトを確立する必要があります。クレデンシャルを取得する詳細な手順については、[Obtaining Firebase Cloud Messaging Credentials](#) を参照してください。これらの手順はネイティブ Android アプリ向けであり、すべてのプロセスを完了する必要はありません。セットアップを正常に完了するには、以下の 3 つのアイテムを取得する必要があります。

- ※ サーバーキー (旧名は API キー)
- ※ 送信者 ID (旧名はプロジェクト番号)
- ※ Firebase コンソールから生成された **google-services.json** ファイル

他のプッシュネットワークについては、[Obtaining Credentials for Push Networks](#) を参照してください。

2.2.2. Push Notification Hello World テンプレートからのプロジェクトの作成

Push Notification Hello World
An example project using the Unified Push Service built-into the platform

Client Apps: 4 | Cloud Apps: 1 | Category: Samples

Name your Project *

Name

Create

Below are the Apps that make up this Template - you can uncheck them if you'd rather not include them in your Project

App Templates

App Name: Cloud App
Type: @ Node.js Cloud App
Template Source: <https://github.com/feedhenry-templates/helloworld-cloud.git>
Documentation: /templateapps/static/hello_world_mbaas_instance/README.md
Description: Hello World Node.js Express App which echos a username

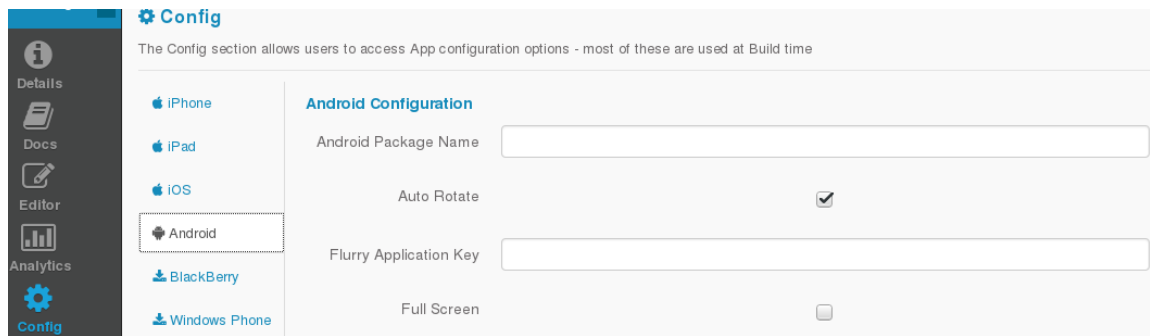
App Name: Simple Push Windows App
Type: Native Windows Store app
Template Source: <https://github.com/feedhenry-templates/pushstarter-windows-app.git>

1. RHMAP Studio で **Projects** に移動し、**New Project** をクリックします。
2. **Push Notification Hello World** テンプレートを探し、右側にある **Choose** をクリックします。
3. プロジェクトの名前を **Name your Project** フィールドに入力します。
4. **App Templates** セクションで **Simple Cordova Push App** を探します。このアプリが選択されていることを確認し、プロジェクトテンプレート内で選択解除できる他のすべてのアプリを選択解除します。チェックボックスは各アプリテンプレートの右上隅にあります。
5. プロジェクトテンプレートの右上にある **Create** をクリックします。プロジェクトの作成が完了するまで待機します。次に、画面の下部にある **Finish** をクリックします。

2.2.3. クライアントアプリのパッケージ名の定義

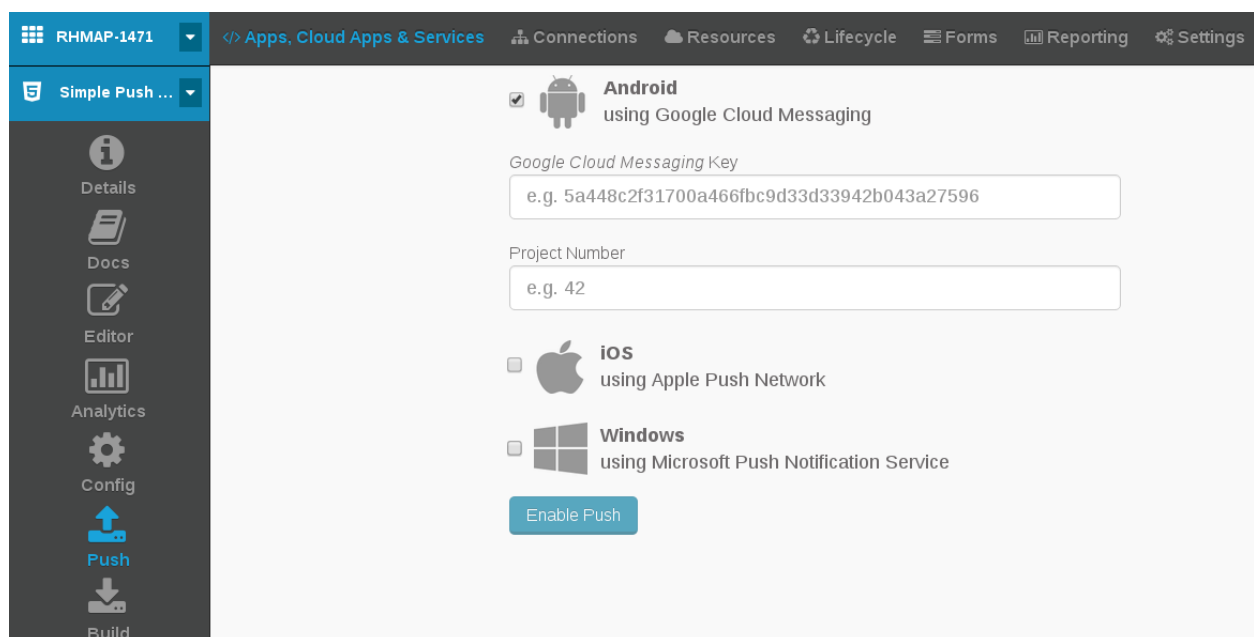
1. プロジェクトの **Apps, Cloud Apps & Services** セクションで **Simple Cordova Push App** をクリックします。
2. 画面の左側にある **Config** をクリックします。
3. Config メニューの **Android** オプションを選択します。
4. **Android Package Name** 下にあるクライアントアプリのパッケージ名を定義します。

この名前は、Firebase コンソールでのアプリの登録時に必要です。パッケージの名前を定義するときは、Java パッケージの命名規則に従います。



2.2.4. プッシュサポートのセットアップ

プッシュサポートは、各クライアントアプリに対して明示的に有効にする必要があります。また、最初の手順 ([Firebase Cloud Messaging クレデンシャルの取得](#)と [google-services.json ファイルのダウンロード](#)) で取得したプッシュネットワーククレデンシャルを提供する必要もあります。



1. プロジェクトの **Apps, Cloud Apps & Services** セクションで **Simple Cordova Push App** をクリックします。
2. 画面の左側にある **Push** をクリックします。
3. **Enable Push** をクリックします。
4. **Android** オプションを選択し、最初の手順で取得した **Server key** と **Sender ID** を入力します。 **Enable Push** をクリックします。

2.2.5. クライアントアプリへのgoogle-services.json ファイルの統合

1. 画面の左側にある **Editor** をクリックします。
2. Editor で **www** フォルダーを選択し、ツールバーの **+** 記号をクリックして新しいファイルを作成して **google-services.json** という名前を付けます。

3. Firebase コンソールから以前にダウンロードした **google-services.json** ファイルを開き、Editor でその内容を **google-services.json** にコピーします。
4. **File** をクリックし、次に Editor ツールバーの **Save** をクリックして、変更内容を保存します。

2.2.6. config.xml ファイルの設定

Firebase コンソールで定義されたパッケージ名に合わせて config.xml ファイルの **widget id** 設定を変更します。

2.2.7. クライアントアプリバイナリーのビルド

The screenshot shows the Red Hat Mobile Application Platform interface. On the left is a sidebar with navigation icons: Config, Push, Build, Export, Credentials, and Integrate. The main area is titled 'Cloud App Connection' and contains a 'Select Cloud App' dropdown menu and a 'Connection Tag' input field set to '0.0.5'. Below this is a 'Build' button. The 'Building for android' section shows a green progress bar labeled 'Completed'. The 'Artifact History' section shows a table with two items, both for Android platform, version 4 and 3, with a status of 'Success'.

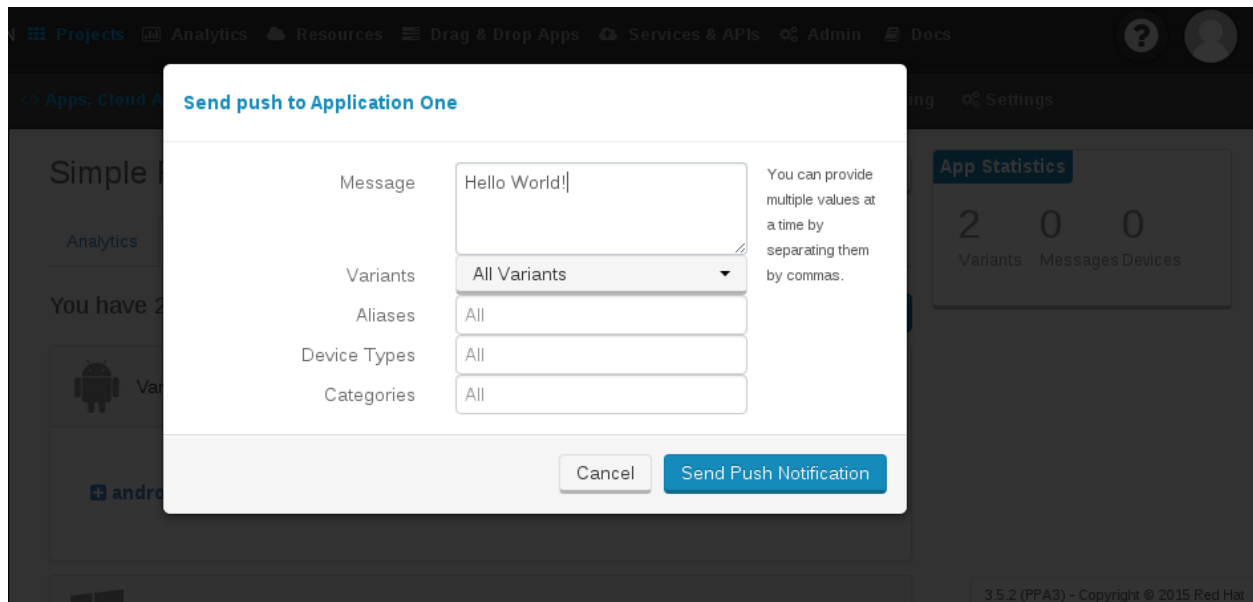
Platform	App Version	Date	Status	Type	Credential	Git Branch/Tag	Git Commit	View Logs	Download
Android	4	Oct 20th 2015 11:44:24 pm	Success	Debug		Branch : master	9841a2dd5f595c57	View Logs	Download
Android	3	Oct 15th 2015 10:00:18 pm	Success	Debug		Branch : master	9841a2dd5f595c57	View Logs	Download

Build artifacts can be retrieved for up to 90 days.

特別なセットアップなしで簡単にビルドできる **Debug** タイプの Android バイナリーをビルドします。

1. 画面の左側にある **Build** をクリックします。
2. **Client Binary** 画面で、**Android** を選択します。
3. 画面の下部にある **Build** をクリックし、アプリがビルドされるまで待機します。**Artifact History** 画面の下部にある進捗状況を監視できます。
4. ビルドが完了したら、ダウンロードリンクと QR コードがあるダイアログが表示されます。または、**Artifact History** テーブルの最初の行にある **Download** をクリックします。
5. モバイルデバイスで、表示された QR コードをスキャンし、アプリをインストールします。

2.2.8. アプリのテスト



1. モバイルデバイスで、新しくインストールされた **Simple Cordova Push App** を開きます。
2. RHMAP で、プロジェクトを開き、**Apps, Clouds & Services** で **Simple Cordova Push App** を開きます。
3. 画面の左側にある **Push** をクリックします。
4. 画面の上部にある **Send Notification to this app** をクリックします。
5. **Message** に必要な情報を入力し、他のすべてのフィールドは変更せずに、**Send Push Notification** をクリックします。
6. すぐにモバイルデバイスが、提供されたメッセージとともにプッシュ通知を受け取ります。

2.3. MBAAS サービスからフォームフィールドに動的にデータを入力する

2.3.1. 概要

フォームビルダーで、フォームフィールドのデータソースとして MBaaS サービスのエンドポイントを使用できます。本書では、MBaaS サービスの作成、データソースの定義、およびデータソースを使用したフォームフィールドへのデータの入力を実行する方法について説明します。

フォームのデータソースの詳細については、[Using Data Sources](#) を参照してください。

2.3.2. データソースのサービスの作成

データソースを定義するには、データソースの有効な JSON 応答を提供する MBaaS サービスエンドポイントを提供する必要があります。この例では、JSON ファイルから静的なデータを提供する新しい MBaaS サービスを作成します。

1. Studio で、**Services & APIs** に移動します。**Provision MBaaS Service/API** をクリックします。

2. 左側にある **Data Sources** カテゴリーを選択します。
3. テンプレート (たとえば、**Static Data Source**) を選択します。
4. サービスの名前を選択し、そのサービスを環境にデプロイします。

2.3.3. データソースの定義

データを提供する MBaaS サービスを作成したら、データソースで使用できます。

1. Studio で **Drag & Drop Apps > Data Sources** に移動します。 **New Data Source** をクリックします。
2. 名前と説明を設定します。両方とも必須です。
3. 以前に作成された MBaaS サービスと、データソースとして使用するエンドポイントを選択します。

この例で使用する **Static Data Source** テンプレートはエンドポイント `/static_ds/months` でデータを提供します。

Validate ボタンを使用して、サービスがデータソースとして使用できるデータを返すかどうかを確認できます。確認の前に、右上隅にある環境セクターを使用して適切な環境が選択されるようにします。

データが有効な場合は、**"Success Data Source Is Valid"** というメッセージが表示されます。データが有効でない場合、またはサービスに到達できない場合は、問題を示すエラーメッセージが表示されます。

有効な場合は、**View** ボタンを使用して返されたデータを表示できます。

3 Choose A Service

Your Data Source retrieves it's data from an MBaaS Service. Select the service and path to use below.

Pick A Service

InventoryService ▼

[View service details](#)

HTTP path to call via HTTP GET (e.g /countries)

/current *

Validate

Success Data Source Is Valid

View

4 Update Schedule

We keep your Data Source up to date by periodically refreshing its data from the MBaaS Service - by default, Sources are updated once every 24 hours - you can adjust this below

*

a day

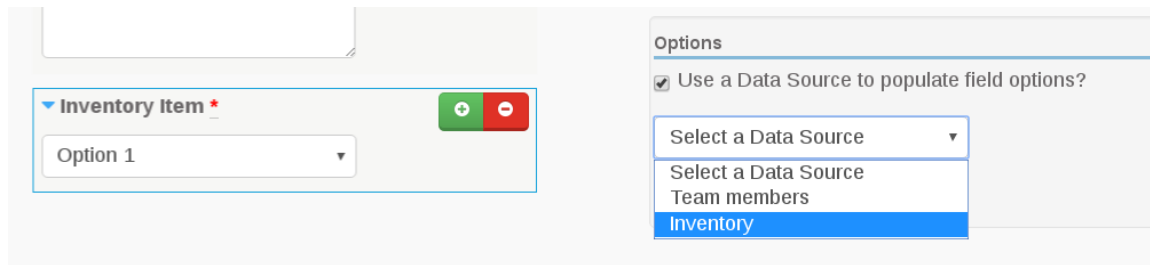
- 更新頻度 (新しいデータを取得するために、設定された MBaaS サービスエンドポイントを呼び出す頻度) を選択します。更新頻度は 1 分から 7 日間までの範囲の値です。
- 監査ログに保持する MBaaS サービスエンドポイントの呼び出しからの応答の数を選択します。
- Create Data Source** をクリックします。**Data Source Created Successfully** というメッセージが表示されます。

この時点で、フォームフィールドにデータを入力するためにフォームビルダーでデータソースを使用できます。

2.3.4. フォームフィールドでのデータソースの使用

データソースの定義後に、フォームビルダーでそのデータソースを使用できるようになります。

- Studio で、**Drag & Drop Apps > Forms Builder** に移動します。
- 既存のフォームを開くか、新しいフォームを作成します。**Edit Form** に移動します。
- サポートされるいずれかのタイプ (たとえば、**Dropdown** フィールド) をフォームにドラッグします。作成されたフィールドを選択します。
- Options** セクションで **Use a Data Source to populate field options?** をチェックします。
- 前の手順で作成されたデータソースを選択します。



選択されたデータソースから以前に一度でもデータがロードされた場合 (作成中に検証する場合など) は、プレビューとして、データの利用可能な最新バージョンがフィールドオプションに表示されます。

右上隅にある環境セクターを使用して同じ環境が選択されるようにします (データソースの MBaaS サービスがデプロイされます)。

6. フォームを保存し、デプロイします。

Dashboard に移動すると、プレビューの右側にあるデータソースからデータが入力されたフィールドを確認できます。

2.4. アプリへの統計の追加

2.4.1. 概要

プラットフォームには、ユーザーがアクセスできるカウンターとタイマーが保持されます。一部はプラットフォームにより提供され、一部はアプリケーションの開発者が指定できます。

カウンターは、特定の機能の使用を追跡するために使用でき、増分または減分できます。プラットフォームでは、現在アクティブなサーバーサイドアクションを示すカウンター (つまり、現在実行中のモバイルアプリケーションにより呼び出されたサーバーサイド機能の数) が提供されます。

タイマーは、指定されたアクションを実行するのにかかった時間を追跡するために使用できます。プラットフォームでは、サーバーサイドアクションの実行時間を追跡するタイマーが提供されます。定期的 (プラットフォームで設定された間隔) に、現在のカウンターとタイマーが履歴にプッシュされ、ゼロに設定されます。

プラットフォームから統計を要求するときは、データを返す最近の間隔の数と破棄の間隔の長さ (ミリ秒単位) がデータとともに返されます。

タイマーデータは、間隔で発生したタイミングイベントの数 (上限値とか下限値を含む) として返されます。また、90 パーセンタイルの上限値と平均値もあります。

2.4.2. カウンターとタイマーの作成

アプリ開発者は以下のコードを使用して独自のカウンターとタイマーを作成できます。

```
$fh.stats.inc('COUNTERNAME'); // Increments a counter
$fh.stats.dec('COUNTERNAME'); // Decrements a counter
$fh.stats.timing('TIMERNAME', timeInMillis); // Store a timer value
```

クラウドアプリの統計 API の詳細については、以下のドキュメントを参照してください。

» [\\$fh.stats API Reference](#)

"app" の統計タイプを指定することにより、開発者のカウンターとタイマーはプラットフォームから要求できます。

カウンターのキー名はフォーム **DOMAIN_APPID_app_COUNTERNAME** に基づきます。ここで、**DOMAIN_APPID** はドメインの名前であり、**APPID** はアプリの ID であり、**COUNTERNAME** はカウンターに開発者が付けた名前です。

タイマーのキー名はフォーム **DOMAIN_APPID_app_TIMERNAME** に基づきます。ここで、**DOMAIN_APPID** はドメインの名前であり、**APPID** はアプリの ID であり、**TIMERNAME** はタイマーに開発者が付けた名前です。

2.4.3. 統計の表示

ユーザーは、Studio から統計データにアクセスしたり、コマンドラインクライアントである [FHC](#) から生データ自体にアクセスしたりできます。

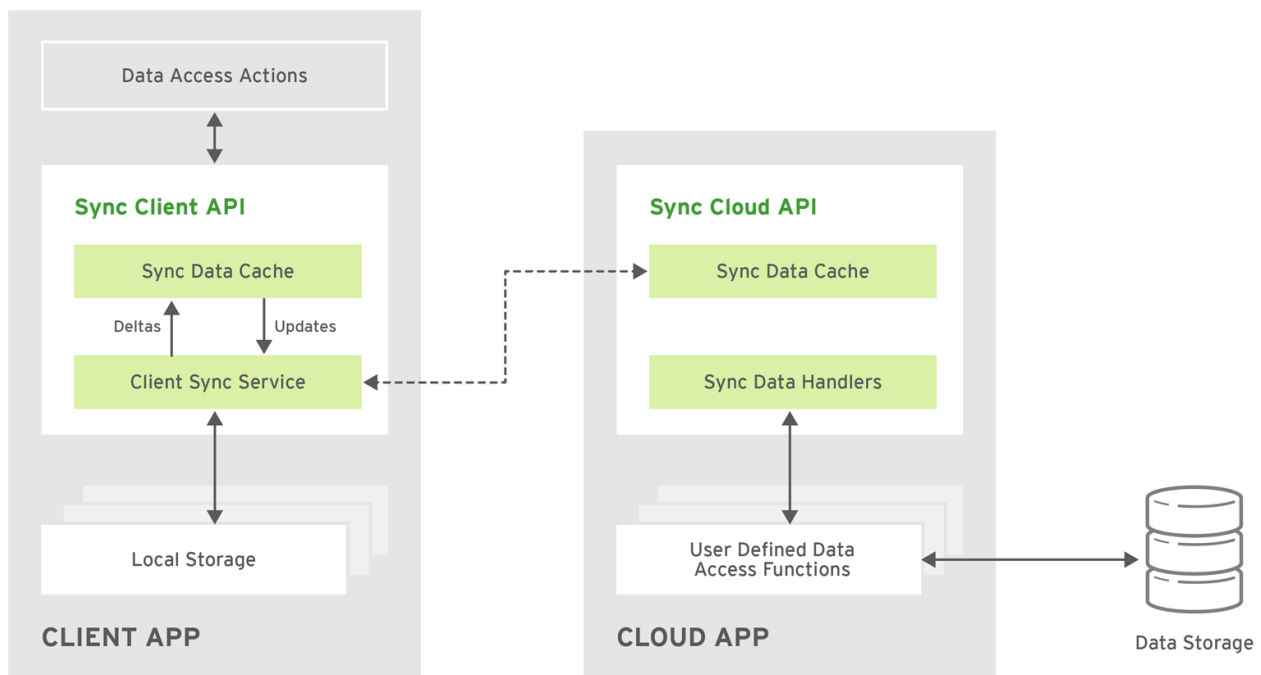
第3章 RHMAP データ同期フレームワークの使用

3.1. データ同期フレームワーク

RHMAP モバイルデータ同期フレームワークには、以下の機能が含まれます。

- ✳ モバイルアプリがオフラインでデータを使用および更新できる (ローカルキャッシュ)
- ✳ クラウドアプリ経由で複数のクライアントアプリから双方向のデータ同期を管理し、バックエンドデータストアに戻すメカニズムを提供する
- ✳ データアップデート (つまり、デルタ) をクラウドアプリから接続済みクライアントに分散できる
- ✳ クラウドにある複数のアップデートからデータ競合管理を有効にする
- ✳ ネットワーク接続が失われたときに RHMAP アプリはシームレスで稼働し続けることができ、ネットワーク接続が回復したときに復帰できます。

3.1.1. ハイレベルアーキテクチャー



FEEDHENRY_434425_0117

同期フレームワークは、クライアントアプリと Node.js クラウドアプリ API のセットで構成されます。

クライアントアプリは、バックエンドデータに直接アクセスしません。代わりに、同期クライアント API を使用して、デバイスに格納されたデータの読み取りと一覧表示を行い、変更 (作成、更新、および削除) をクラウドアプリに送信します。データにローカルで行われた変更は、クラウドアプリに送信される前にローカルの同期データキャッシュに格納されます。クライアントアプリは、同期クライアント API からリモートデータセットへの変更の通知を受け取ります。

クライアントアプリはクライアント同期サービスを使用して、リモートデータセットに行われた変更 (デルタ) をクラウドアプリから受け取り、同期データキャッシュに格納します。また、クライアントアプリはクライアント同期サービスを使用してローカルで行われた更新をクラウドアプリに送信します。クライアントアプリがオフラインのときは、キャッシュされた更新がデバイス上のローカルストレージにフラッシュされ、ネットワーク接続が再確立される前にクライアントアプリ

が閉じられる場合に変更を永続化できます。変更は、クライアントアプリが次回オフラインになったときにクラウドアプリにプッシュされます。

クラウドアプリはバックエンドデータに直接アクセスせず、同期クラウド API を介してのみアクセスします。クラウドアプリは、クライアント同期サービスを使用してクライアントアプリからアップデートを受け取るために同期クラウド API を使用します。これらのアップデートはクラウドアプリ同期データキャッシュに格納されます。クラウドアプリは、標準的な CRUDL (create, read, update, delete and list) および **collisionHandler** 関数を使用して、ホストされたストレージでバックエンドデータを管理するために同期クラウド API を使用します。

標準的なデータハンドラー関数以外に、クラウドアプリにはユーザー定義データアクセス関数も使用できます。

3.1.2. API

同期用のクライアントおよび Node.js API 呼び出しは、以下のガイドで文書化されています。

✎ [JavaScript SDK API](#) の項「Sync」

✎ [Node.js API](#) の項「Sync」

✎ [iOS SDK ドキュメント](#)

✎ [Android SDK Docs](#)

3.1.3. スタートガイド

アプリで同期フレームワークを使用するには、以下の手順を実行します。

1. クライアントサイドで \$fh.sync を初期化します。

```
//See [JavaScript SDK API](../api/app_api.html#app_api-_fh_sync)
for the details of the APIs used here

var datasetId = "myShoppingList";

//provide sync init options
$fh.sync.init({
  "sync_frequency": 10,
  "do_console_log": true,
  "storage_strategy": "dom"
});

//provide listeners for notifications.
$fh.sync.notify(function(notification){
  var code = notification.code
  if('sync_complete' === code){
    //a sync loop completed successfully, list the update data
    $fh.sync.doList(datasetId,
      function (res) {
        console.log('Successful result from list:',
JSON.stringify(res));
      },
      function (err) {
        console.log('Error result from list:',
JSON.stringify(err));
      }
    );
  }
});
```

```

    });
  } else {
    //choose other notifications the app is interested in and
    provide callbacks
  }
});

//manage the data set, repeat this if the app needs to manage
multiple datasets
var query_params = {}; //or something like this: {"eq":
{"field1": "value"}}
var meta_data = {};
$fh.sync.manage(datasetId, {}, query_params, meta_data,
function(){

});

```

通知について

同期フレームワークは、同期ライフサイクル中にさまざまな種類の通知を提供します。使用しているアプリの要件に応じて、アプリがリスンする通知の種類を選択し、コールバックを追加できます。ただし、これは必須ではありません。通知リスナーがなくても同期フレームワークにより同期が実行されます。

適切な通知リスナーを追加すると、アプリのユーザーエクスペリエンスが向上します。

- ✳ 同期フレームワークエラーが発生した状況で重大なエラーメッセージをユーザーに表示します (**client_storage_failed** など)。
- ✳ デバッグを支援するためにエラーと障害をコンソールに表示します (**remote_update_failed** や **sync_failed** など)。
- ✳ デルタを受け取った場合は、同期データに関連する UI を更新します。たとえば、データに変更がある場合は、**delta_received** と **record_delta_received** を使用できます。
- ✳ 競合を監視します。

\$fh.sync API を使用してクライアントで CRUDL 操作を実行します。

2. クラウドサイドで \$fh.sync を初期化する

この手順はオプションであり、データセットバックエンドとの同期ループ周期を変更するなど、サーバーでデータセットオプションを上書きする場合のみ必要です。デフォルトの同期周期を変更する場合は、以下の項「[留意事項](#)」を参照してください。

```

var fhapi = require("fh-mbaas-api");
var datasetId = "myShoppingList";

var options = {
  "syncFrequency": 10
};

fhapi.sync.init(datasetId, options, function(err) {
  if (err) {
    console.error(err);
  }
});

```

```

    } else {
      console.log('sync initied');
    }
  });
};

```

この時点でアプリで同期フレームワークを使用するか、サンプルアプリを使用して基本的な使用方法を学ぶことができます ([クライアントアプリ](#)と[クラウドアプリ](#))。

デフォルトのデータアクセス実装が要件を満たさない場合は、オーバーライド関数を提供できます。

3.1.3.1. 不必要な同期ループの回避

クライアントとサーバーの同期頻度は別々に設定されるため、サーバーサイドの同期頻度がクライアントサイドの同期頻度と異なる場合は 1 つの同期ごとに 2 つの同期ループが実行されることがあります。クライアントで長い周期を設定しても、サーバーの同期周期は変更されません。2 つの同期ループを回避するには、サーバーのデータセットの `syncFrequency` 値をクライアントの対応するデータセットの `sync_frequency` 値に設定します。

例

- ※ サーバーサイドデータセットの `syncFrequency` は 120 秒に設定されます。
- ※ クライアントサイドデータセットの `sync_frequency` は 120 秒に設定されます。

ただし、クライアントとサーバーで異なる周期が必要な場合は、異なる値を設定できます。

3.1.4. 同期フレームワークの高度な機能の使用

同期フレームワークは、アプリ開発者がデータセットのソースデータを定義することを可能にするフックを提供します。通常、ソースデータは外部のデータベース (MySQL、Oracle、MongoDB など) ですが、これは要件ではありません。データセットのソースデータはどんな種類でもかまいません (csv ファイル、FTP メタデータ、複数のデータベーステーブルから取得されたデータなど)。同期フレームワークの唯一の要件は、ソースデータの各レコードが一意的 ID を持ち、データが同期フレームワークに JSON オブジェクトとして提供されることです。

バックエンドデータソースと同期するために、アプリ開発者は同期のコードを実装できます。

たとえば、データベースからデータをロードする代わりにバックエンドからデータをリストするときに、ハードコーディングされたデータを返すことができます。

1. クライアントサイドで `$fh.sync` を初期化します。

これは [Getting Started](#) の手順 1 と同じです。

2. クラウドサイドで `$fh.sync` を初期化し、オーバーライドを提供します。

```

var fhapi = require("fh-mbaas-api");
var datasetId = "myShoppingList";

var options = {
  "syncFrequency": 10
};

//provide hard coded data list
var datalistHandler = function(dataset_id, query_params, cb,

```

```

meta_data){
  var data = {
    '00001': {
      'item': 'item1'
    },
    '00002': {
      'item': 'item2'
    },
    '00003': {
      'item': 'item3'
    }
  }
  return cb(null, data);
}

fhapi.sync.init(datasetId, options, function(err) {
  if (err) {
    console.error(err);
  } else {
    $fh.sync.handleList(datasetId, datalistHandler);
  }
});

```

他のオーバーライドの提供方法については、[Node.js API](#) の項「Sync」を参照してください。

3.1.5. さらに詳しく知るために

興味がある場合は、同期フレームワークを理解するのに役に立つ以下の情報をお読みください。

3.1.5.1. データセット

データセットはアプリクライアントとアプリクラウド間で同期するデータを表す JSON オブジェクトです。データセットの構造は以下のとおりです。

```

{
  record_uid_1 : {<JSON Object of data>},
  record_uid_2 : {<JSON Object of data>},
  record_uid_3 : {<JSON Object of data>},
  ...
}

```

データセットの各レコードは一意的 ID (UID) を持つ必要があります。この UID はデータセットのレコードに対してキーとして使用されます。

同期フレームワークは複数のデータセットを管理できます (各データセットは個別に設定できます)。

各データセットは、同期 API と通信するときに (アプリクライアントとアプリクラウドの両方で) 使用する必要がある一意の名前を持ちます。

3.1.5.2. 競合

競合は、クライアントがアップデートをレコードに送信しようとしたときに、レコードのクライアントのバージョンが古い場合に発生します。通常、競合は、クライアントがオフラインであり、レコードのローカルバージョンに対してアップデートを実行したときに発生します。

競合を処理するには、以下のハンドラーを使用します。

- ※ **handleCollision()** - 競合が発生したときに同期フレームワークによって呼び出されます。デフォルトの実装により、"`<dataset_id>_collision`" という名前のコレクションにデータレコードが保存されます。
- ※ **listCollision()** - データ競合のリストを返します。デフォルトの実装により、"`<dataset_id>_collision`" という名前のコレクションからすべての競合レコードがリストされます。
- ※ **removeCollision()** - 競合のリストから競合レコードを削除します。デフォルトの実装により、"`<dataset_id>_collision`" という名前のコレクションからハッシュ値に基づいた競合レコードが削除されます。

データ競合を処理するハンドラー関数オーバーライドを提供できます。オプションは以下のとおりです。

- ※ データ管理者があとで手動で解決するために競合レコードを格納します。
- ※ 競合を引き起こしたアップデートを破棄します。これを行うために、**handleCollision()** 関数は渡された競合レコードに何も処理を行いません。

警告

これにより、競合を引き起こしたアップデートがクラウドアプリにより破棄されるため、データが失われる可能性があります。

- ※ 競合を引き起こしたアップデートを適用します。これを行うには、**handleCollision()** 関数が、データセットに対して定義された **handleCreate()** 関数を呼び出す必要があります。

警告

この結果、競合を引き起こしたアップデートはデータの古いバージョンに基づき、一部のフィールドは古い値に戻されることがあるため、データが失われる可能性があります。

ネイティブ同期クライアントは、類似したインターフェースを使用します。API とサンプルコードは、[iOS Github repo](#) と [Android Github repo](#) で確認できます。

3.2. 同期に関する用語

3.2.1. 同期プロトコル

同期クライアントと同期サーバー間の通信用プロトコル。詳細については、[同期プロトコル](#)を参照してください。

3.2.2. 同期サーバー

同期サーバーは、[同期プロトコル](#)のサーバー部分であり、fh-mbaas-api モジュールに含まれます。以下のことを行います。

- ※ [データセットバックエンド](#)と統合するために[同期サーバー API](#)を公開する
- ※ [同期サーバーループ](#)を実行して、データセットバックエンドのアップデートが検出されたときに[データセット](#)を更新する

3.2.3. 同期クライアント

同期クライアントは、[同期プロトコル](#)のクライアント部分です。3つの同期クライアント実装があります。

- ※ Javascript ([FeedHenry Javascript SDK](#))
- ※ Objective C ([FeedHenry iOS SDK](#))
- ※ Java ([FeedHenry Android SDK](#))

同期クライアント:

- ※ [データセット](#)に対する CRUDL アクションのために[同期クライアント API](#)をフロントエンドに公開する
- ※ [同期クライアントループ](#)を実行して、特定の[データセット](#)に対して指定された[同期周期](#)で[同期サーバー](#)を呼び出す

3.2.4. 同期サーバーループ

同期サーバーループは、[同期サーバー](#)で継続的に実行される機能です (各実行の間に 500ms の待機が発生)。

各実行中は、すべての[データセットクライアント](#)に対して反復処理が行われ、[データセット](#)を[データセットバックエンド](#)から同期する必要があるかどうかを確認できます。

3.2.5. 同期クライアントループ

同期クライアントループは、[同期クライアント](#)で継続的に実行される機能です (各実行の間に 500ms の待機が発生)。

各実行中は、すべての[データセットクライアント](#)に対して反復処理が行われ、[データセット](#)を[同期サーバー](#)と同期する必要があるかどうかを確認できます。

3.2.6. 同期周期

[同期クライアント](#)では、これは特定の[データセット](#)用の[同期サーバー](#)からのアップデートをチェックする間隔です。

[同期サーバー](#)では、これは特定の[データセット](#)用[データセットバックエンド](#)からのアップデートをチェックする間隔です。

詳細については、[同期周期の設定](#)を参照してください。

3.2.7. データセット

データセットは、1つ以上の[同期クライアント](#)、[同期サーバー](#)、および[データセットバックエンド](#)間で同期されるレコードのコレクションです。

3.2.8. データセットバックエンド

[同期クライアント](#)と[同期サーバー](#)間で同期されたデータ用レコードのシステム。

API を提供する任意のシステムであり、[同期サーバー](#)から mysql データベースや SOAP サービスなどと統合できます。

同期サーバーは、データセットバックエンドと統合するために[データセットハンドラー](#)を使用して[同期サーバー API](#) を公開します。

3.2.9. データセットハンドラー

データセットハンドラーは、[同期サーバー](#)を[データセットバックエンド](#)に統合する機能です。

[データセット](#)に対して CRUDL アクションを実行したり、[データセットレコード](#)間で競合を管理したりする多くのハンドラーがあります。

これらのハンドラーのデフォルトの実装では fh.db (RHMAP MBaaS で支援される MongoDB) を使用します。

これらの各ハンドラーはオーバーライドできます。詳細については、[Sync Server API](#) を参照してください。

重要: ハンドラーをオーバーライドする場合、Red Hat は、デフォルトの実装を使用している一部のハンドラーとオーバーライドされた実装を使用している他のハンドラーでの異常な動作を回避するために、**すべてのハンドラーをオーバーライドすることをお勧めします。**

3.2.10. データセットクライアント

データセットクライアントは、クライアントとサーバー間で積極的に同期されている各[データセット](#)に対する[同期クライアント](#)と[同期サーバー](#)に格納された設定です。

以下のようなデータが含まれます。

- ※ [データセットの同期周期](#)
- ※ [データセットバックエンド](#)を呼び出すときに含めるクエリーパラメーター
- ※ [データセットレコードの最新のハッシュ](#)
- ※ [データセットバックエンド](#)との同期が現在実行中であるかどうかに関するデータ

3.2.11. データセットレコード

データセットレコードは[データセット](#)内の個別レコードです。

以下のものが含まれます。

- ※ このレコードが表すローデータ (MySQL テーブルの行値など)
- ※ ローデータの[ハッシュ](#)

3.2.12. ハッシュ

同期プロトコルで使用されるハッシュには2つの種類があります。

- ※ 個別レコードを比較してそれらが異なるかどうかを確認するために使用される個別[データセットレコード](#)のハッシュ。

- ※ すべてのレコードに対して反復処理を行わずにクライアントのレコードセットとサーバーのレコードセットを比較するために使用される特定の**データセットクライアント**に対するすべての**データセットレコード**のハッシュ。

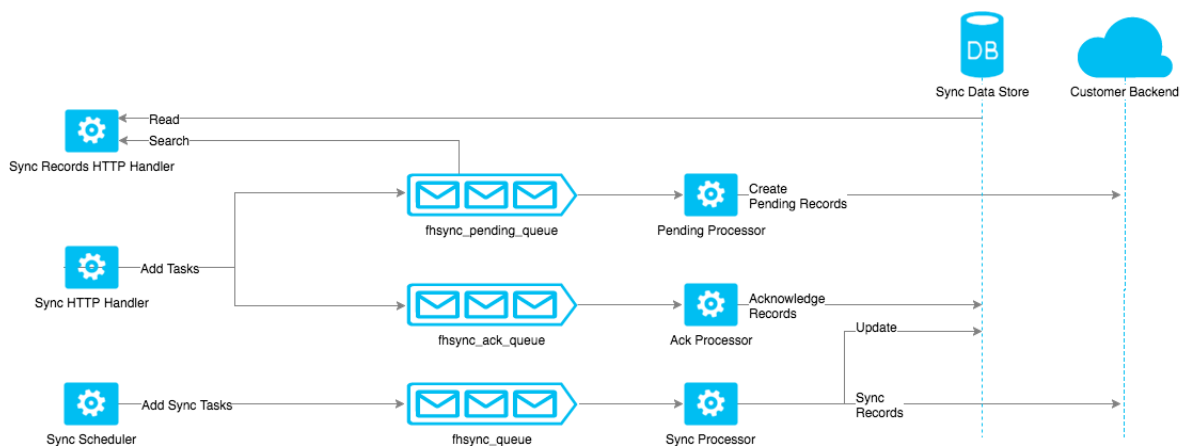
3.3. 同期サーバーのアーキテクチャー

同期フレームワークの一般的な概要については、[同期に関する概要](#)と[同期に関する用語](#)を参照してください。

3.3.1. アーキテクチャー

同期サーバーアーキテクチャーに含まれるもの: * HTTP ハンドラー * キューおよびプロセッサー * 同期スケジューラー

これらの各コンポーネントによりデータは MongoDB で永続化されます。



3.3.1.1. HTTP ハンドラー

これらのハンドラーは、同期クライアントからの同期要求を処理します。

3.3.1.1.1. 同期 HTTP ハンドラー

データセットクライアントを作成または更新し、保留中のレコードと確認を処理のために適切なキューにプッシュします。

3.3.1.1.2. 同期レコード HTTP ハンドラー

最新のデータをクライアントの状態と比較します。デルタの取得後に、処理されているがまだ同期されていないアップデートがチェックされます。このハンドラーはデルタ内のすべてのレコードを反復処理します。レコードが保留中のキューにある場合や適用された場合、レコードはこのハンドラーによってデルタから削除され、更新されたデルタはクライアントに返されます。

3.3.1.2. キュー

同期フレームワークでは以下のキューが使用されます。

- ※ **fhsync_queue** - 同期が必要なデータセット向けのジョブ

- ※ **fhsync_ack_queue** - 確認が必要な保留中の変更向けのジョブ
- ※ **fhsync_pending_queue** - 処理が必要な保留中の変更向けのジョブ

メッセージはこれらのキューに格納され、プロセッサによって消費されます。

3.3.1.3. プロセッサ

各キューには対応するプロセッサがあります。

- ※ 同期プロセッサ - **fhsync_queue** からジョブを取得し、これらのジョブを処理します。
- ※ 確認プロセッサ - **fhsync_ack_queue** から確認を取得し、MongoDB からこれらの確認を削除します。
- ※ 保留中プロセッサ - **fhsync_pending_queue** から保留中のアイテムを取得し、変更をデータセットバックエンドに適用します。

同期サーバーの各ワーカーは、タスクを分散することを可能にするこれらの各プロセッサの 1 つのインスタンスを持ちます。

3.3.1.4. 同期スケジューラー

水平的にスケールされた場合、各同期ワーカーは固定された間隔で同期スケジューラーになろうとします。各ワーカーは MongoDB にあるロックを取得しようとしています。同期スケジューラーのロックがあるワーカーは、データセットの最後の同期のタイムスタンプと同期周期を確認して同期する必要があるデータセットを決定します。データセットを同期する必要がある場合、ジョブは **fhsync_queue** に追加されます。

3.4. データ同期設定ガイド

データ同期設定は、クライアントサイドとクラウド (サーバーサイド) に適用できます。

クライアントサイドの同期周期は、**sync_frequency** 変数を使用して設定されます。**sync_frequency** の設定の例については、ドキュメンテーションの [この項](#) を参照してください。

クラウドの同期周期は、**syncFrequency** 変数を使用して設定されます。**syncFrequency** の設定の例については、ドキュメンテーションの [この項](#) を参照してください。

3.4.1. 同期周期の設定

同期周期は、システムが 2 つの同期プロセスの間に待機する期間です。

重要: クライアントとサーバーで別々に周期を設定できます。ただし、Red Hat は以下のシナリオを回避するために同じ設定を使用することをお勧めします。

- ※ サーバーが [データセットバックエンド](#) からのアップデートをチェックするよりも高い頻度でクライアントが呼び出しを行うため、クライアントから不必要なトラフィックが発生します。
- ※ サーバーが [データセットバックエンド](#) からのアップデートをチェックするよりも低い頻度でクライアントが呼び出しを行うため、アクティビティーがないことが原因でサーバーがキャッシュから [データセット](#) を破棄します。

サーバーの同期周期の値により、同期プロセッサが実行される頻度が決定されます。同期プロセッサが実行されるたびに、データセットバックエンドに対してリスト操作が実行され、データとローカルコピーが同期されます。使用しているアプリケーション向けの同期周期の最適な値を決定するために、以下のセクションをお読みください。

- ※ 使用しているクライアントが他のクライアントの変更をどれくらい早く認識するか？

クライアントが変更を送信すると、それらの変更はデータセットバックエンドに直接適用されます。パフォーマンスを向上させるために、他のクライアントはローカルコピーからデータを取得します。したがって、他のクライアントは次の同期プロセッサの実行後まで新しい変更を取得できません。他のクライアントができるだけ早く変更を取得する必要がある場合は、同期周期に低い値を設定することを検討してください。

- ※ 同期プロセッサの実行にどれくらいかかるか？

同期周期の値により、システムが同期プロセッサの実行の間に待機する時間が決定されます。つまり、同期周期は 1 つの実行が完了してから次の実行が開始されるまでの時間です。したがって、2 つの同期プロセッサが同時に実行される状況は発生しません。計算式は以下のとおりです。

実際の同期周期 = 同期プロセッサの実行時間 + 同期周期

これにより、システムがデータセットバックエンドに行く要求の数を簡単に計算できるようになります。

同期プロセッサの各実行にかかる時間を調べるには、同期統計エンドポイントを問い合わせる **sync_worker** が完了するのにかかる平均 **Job Process Time** (ジョブプロセス時間) を確認します。

- ※ データセットバックエンドサービスはどれくらいの負荷を処理できるか？

同期プロセッサが実行されるたびに、データセットバックエンドに対するリスト操作が実行されます。同期周期を設定する場合は、バックエンドで生成される要求の数を予測し、バックエンドがその負荷を処理できるようにする必要があります。

たとえば、データセットの同期周期を 100ms に設定し、各同期プロセッサの実行に 100ms かかる場合は、サーバーによって約 5 要求/秒がバックエンドに生成されます。ただし、同じバックエンドを使用する同期周期が 100ms の別のデータセットがある場合、バックエンドに対する負荷は 10 要求/秒になります。バックエンドに対して負荷テストを行うことにより、バックエンドがその負荷を処理できるかどうかを調べることができます。

ただし、アプリをスケールするとき、この値は大きくなりません。たとえば、サーバーに複数のワーカーが存在する場合、同期プロセッサの実行は、ワーカーで繰り返されず、分散されます。この設計により、アプリの負荷が大きいときにバックエンドが保護されます。

- ※ サーバーの負荷がどれくらい増えるか？

データがバックエンドから返されると、サーバーはデータをローカルストレージに保存する必要があります (MongoDB)。システムは変更がある場合のみアップデートを実行します。ただし、ローカルストレージの現在のデータを取得するには、読み取り操作を最初に実行する必要があります。同期プロセッサの実行がたくさんあるときは、サーバー自体の負荷が増えることがあります。場合によっては、このことを考慮する必要があります (特にデータセットが大きい場合)。

サーバーのパフォーマンスを理解するために、同期統計エンドポイントを使用して CPU 使用率と MongoDB 処理時間を確認できます。

同期周期値を使用して、サーバーがバックエンドに生成する要求の数を制御できます。この値は、バックエンドが負荷を処理でき、サーバー自体が過負荷の状態でない限り、0ms に設定できます。

3.4.2. ワーカーの設定

同期アーキテクチャーで説明されているように、同期データを格納するために使用するさまざまなキューがあります。データを処理するために、各キューに対応するワーカーが作成されます。この唯一のタスクは、キューからジョブを一度に 1 つずつ取得し、処理することです。ただし、あるジョブを完了してから利用可能な次のジョブを取得するまでの時間を表す間隔値があります。この値は、ワーカーのパフォーマンスを最大化するために設定できます。

3.4.2.1. 間隔の目的

キューからジョブを取得する要求は非ブロック操作です。キューにジョブが残っていないと、要求が返され、ワーカーはジョブを再び取得しようとします。

この場合、またはジョブが非常に早く完了する場合は、ワーカーによって主要なイベントループが過負荷の状態になり、他のコード実行が遅くなることがあります。このシナリオを回避するために、各ワーカーには以下の間隔値設定項目があります。

✧ **pendingWorkerInterval**

✧ **ackWorkerInterval**

✧ **syncWorkerInterval**

デフォルトの間隔値は非常に低い (1ms) ですが、設定可能です。このデフォルト値では、ジョブを実行するのに時間がかかり、主要なイベントループで他の操作を完了することを可能にするいくつかの非ブロック I/O 操作 (リモート HTTP 呼び出しや DB 呼び出しなど) があることを前提とします。この低いデフォルトの間隔によって、ジョブをできるだけ速く処理できるようになり、CPU をより効率的に使用できるようになります。ジョブがない場合は、ワーカーによってリソースが不必要に過負荷の状態にならないようにバックオフメカニズムが呼び出されます。

デフォルト値によってデータセットバックエンドに対して発生する要求が多すぎる場合、またはデフォルトの間隔値を変更する必要がある場合は、1 つ以上のワーカーの設定オプションを上書きできます。

3.4.2.2. ワーカーバックオフ

キューにジョブが残っていない場合、各ワーカーはバックオフストラテジーを持ちます。これにより、ワーカーが不必要な CPU サイクルを使用したり、キューに対する不必要な呼び出しを行ったりすることが防がれます。新しいジョブがキューに格納されている場合、ワーカーは次にキューを確認するときに間隔をリセットします。

各ワーカーの動作は以下の設定オプションで上書きできます。

✧ **pendingWorkerBackoff**

✧ **ackWorkerBackoff**

✧ **syncWorkerBackoff**

デフォルトでは、すべてのワーカーは、最大遅延値で指数ストラテジーを使用します。最小間隔が 1ms に設定されている場合、ワーカーはジョブを処理してから別のジョブをキューから取得するまで 1ms 待機します。このパターンはキューにアイテムが存在する限り続行します。キューが空になると、間隔は最大間隔 (たとえば、60 秒) に到達するまで指数的 (2ms、4ms、8ms、16ms、... ~16s、~32s) に増えます。次に、ワーカーはジョブがあるかどうかを調べるためにキューを 60 秒ごとに確認します。将来キューでジョブが見つかったら、ワーカーは再び 1ms ごとにキューを確認します。

詳細については、[Sync API Doc](#) を参照してください。

3.5. 同期サーバーアップグレードに関する注記

3.5.1. 概要

本項では以下のことを行う開発者を対象とします。

- ※ アプリケーションで同期サーバーを使用する
- ※ **fh-mbaas-api** のバージョンを **<7.0.0** から **>=7.0.0** にアップグレードする

fh-mbaas-api@>=7.0.0 をすでに使用している場合は、本項のどの手順も実行しないでください。



注記

このアップグレードでは、同期クライアントに変更は加えられません。

3.5.2. 前提条件

7.0.0 より前は、同期サーバーが **fh.db** API を使用して同期操作データを MongoDB に格納していました。**fh.db** は、中間 http API (fh-ditch) を介することがある MongoDB のラッパーです。この結果、同期操作データに対して実行できるアクションは制限されます。また、MongoDB に直接接続されるモジュールの使用も制限されます。**fh-mbaas-api@7.0.0** では、同期サーバーで MongoDB への直接接続が必要です。

条件は以下のとおりです。

- ※ ホストされた MBaaS の場合は、アプリデータベースを「アップグレード」する必要があります。
- ※ 自己管理された MBaaS の場合は、デフォルトですべてのアプリが MongoDB の独自のデータベースを取得するため、何も行う必要はありません。

3.5.3. データハンドラー関数署名の変更

同期データハンドラー向けのメソッド署名は、新しい同期フレームワークでは異なります。データハンドラーを実装した場合は、パラメーターの順序を変更する必要があります。これらの変更は、javascript のパラメーター順序規則 (つまり、コールバックが最後のパラメーター) に準拠します。

重要 パラメーターとして各ハンドラーに渡されたコールバック関数が各呼び出しに対して実行されるようにします。これにより、ハンドラーの完了後もワーカーを維持できます。

7.0.0 よりも前のバージョンとそのバージョンのデータハンドラーおよび署名は以下のとおりです。

```
// <7.0.0
sync.handleList(dataset_id, function(dataset_id, params, callback,
meta_data) {});
sync.globalHandleList(function(dataset_id, params, callback, meta_data)
{});
// >=7.0.0
```

```
sync.handleList(dataset_id, function(dataset_id, params, meta_data,
callback) {});
sync.globalHandleList(function(dataset_id, params, meta_data, callback)
{});

// <7.0.0
sync.handleCreate(dataset_id, function(dataset_id, data, callback,
meta_data) {});
sync.globalHandleCreate(function(dataset_id, data, callback, meta_data)
{});
// >=7.0.0
sync.handleCreate(dataset_id, function(dataset_id, data, meta_data,
callback) {});
sync.globalHandleCreate(function(dataset_id, data, meta_data, callback)
{});

// <7.0.0
sync.handleRead(dataset_id, function(dataset_id, uid, callback,
meta_data) {});
sync.globalHandleRead(function(dataset_id, uid, callback, meta_data) {});
// >=7.0.0
sync.handleRead(dataset_id, function(dataset_id, uid, meta_data,
callback) {});
sync.globalHandleRead(function(dataset_id, uid, meta_data, callback) {});

// <7.0.0
sync.handleUpdate(dataset_id, function(dataset_id, uid, data, callback,
meta_data) {});
sync.globalHandleUpdate(function(dataset_id, uid, data, callback,
meta_data) {});
// >=7.0.0
sync.handleUpdate(dataset_id, function(dataset_id, uid, data, meta_data,
callback) {});
sync.globalHandleUpdate(function(dataset_id, uid, data, meta_data,
callback) {});

// <7.0.0
sync.handleDelete(dataset_id, function(dataset_id, uid, callback,
meta_data) {});
sync.globalHandleDelete(function(dataset_id, uid, callback, meta_data)
{});
// >=7.0.0
sync.handleDelete(dataset_id, function(dataset_id, uid, meta_data,
callback) {});
sync.globalHandleDelete(function(dataset_id, uid, meta_data, callback)
{});

// <7.0.0
sync.listCollisions(dataset_id, function(dataset_id, callback, meta_data)
{});
sync.globalListCollisions(function(dataset_id, callback, meta_data) {});
```

```
// >=7.0.0
sync.listCollisions(dataset_id, function(dataset_id, meta_data, callback)
{});
sync.globalListCollisions(function(dataset_id, meta_data, callback) {});

// <7.0.0
sync.removeCollision(dataset_id, function(dataset_id, collision_hash,
callback, meta_data) {});
sync.globalRemoveCollision(function(dataset_id, collision_hash, callback,
meta_data) {});
// >=7.0.0
sync.removeCollision(dataset_id, function(dataset_id, collision_hash,
meta_data, callback) {});
sync.globalRemoveCollision(function(dataset_id, collision_hash,
meta_data, callback) {});
```

3.5.4. 動作の変更

同期サーバーが MongoDB に直接接続するようになったため、起動時にセットアップ時間が必要です。**sync.init()** を現在使用している場合は、これらの呼び出しを**sync:ready** イベントハンドラーでラップします。たとえば、以下のコードを使用する場合は、

```
fh.sync.init('mydataset', options, callback);
```

イベントハンドラーに配置するよう変更します。

```
fh.events.on('sync:ready', function syncReady() {
  sync.init('mydataset', options, callback);
});
```

または、同期 API からイベントエミッターを使用できます。

```
fh.sync.getEventEmitter().on('sync:ready', function syncReady() {
  sync.init('mydataset', options, callback);
});
[source,json]
```

3.5.5. ロガーの変更

fh.sync.init() に渡された **logLevel** オプションは利用できなくなりました。デフォルトでは、新しい同期サーバーは何もログに記録しません。すべてのロギングでは [debug] (<https://www.npmjs.com/package/debug>) モジュールが使用されます。同期サーバーからの出力をログに記録する場合は、以下のように **DEBUG** 環境変数を設定できます。

```
DEBUG=fh-mbaas-api:sync
```

SDK 全体からすべてのログを参照する場合は、以下のように設定します。

```
DEBUG=fh-mbaas-api:*
```

debug モジュールの他のすべての環境変数と動作機能が利用可能です。

3.6. 同期サーバーのパフォーマンスとスケーリング

3.6.1. 概要

同期サーバーはスケーラブルとして設計されています。

本項では、同期サーバーのパフォーマンスとスケーリングのオプションについて説明します。

3.6.2. パフォーマンスの検査

同期サーバーのパフォーマンスを検査するには 2 つのオプションがあります。

1. `/mbaas/sync/stats` エンドポイントを問い合わせます。

デフォルトでは、同期フレームワークによって (実行中の場合) メトリクスデータは Redis に保存されます。また、HTTP GET 要求を `/mbaas/sync/stats` エンドポイントに送信してこれらのメトリクスデータの概要を表示できます。

このエンドポイントからは以下の情報が利用可能です。

- ✧ すべてのワーカーの CPU とメモリーの使用状況
- ✧ さまざまなジョブを処理するのにかかる時間
- ✧ さまざまなジョブキュー内の残りのジョブ数
- ✧ さまざまな API 呼び出しにかかる時間
- ✧ さまざまな MongoDB 操作にかかる時間

これらの各メトリクスに対して、サンプル値、現在値、最大値、最小値、および平均値の合計数を確認できます。

デフォルトでは、各メトリクスに対して最後の 1000 サンプルが収集されますが、その値は `statsRecordsToKeep` 設定オプションを使用して制御できます。

このエンドポイントは使いやすく、同期サーバーの現在のパフォーマンスを理解するのに十分な情報を提供します。

2. InfluxDB と Grafana を使用してメトリクスデータを視覚化します。

現在および過去のメトリクスデータを視覚化する場合は、メトリクスデータを InfluxDB に送信するよう同期サーバーに指示し、Grafana でグラフを表示できます。

InfluxDB と Grafana をセットアップするのに役に立つ多くのオンラインチュートリアルがあります。以下に例を示します。

- ✧ [How to setup InfluxDB and Grafana on OpenShift](#)

InfluxDB が実行されたら、以下の設定を更新して同期サーバーがメトリクスデータを送信するよう指示します。

- ✧ `metricsInfluxdbHost`
- ✧ `metricsInfluxdbPort`



注記

metricsInfluxdbPort が UDP ポートであることを確認します。

Grafana でメトリクスデータグラフを表示するには、グラフ付きの新しいダッシュボードを作成する必要があります。これを行う最も簡単な方法は、[この Grafana ダッシュボードファイル](#)をインポートすることです。アプリが実行されたら、メトリクスデータを Grafana ダッシュボードに表示できます。

Grafana グラフの設定方法の詳細については、[Grafana ドキュメンテーション](#)を参照してください。

3.6.3. パフォーマンスの理解

同期サーバーのパフォーマンスを理解するために確認する必要がある主要なメトリクスは以下のとおりです。

3.6.3.1. CPU 使用率

これは最も重要なメトリクスです。CPU が過負荷である場合は、同期サーバーがクライアント要求に応答できませんが、できるだけ CPU 使用率を上げたいことがあります。

この問題を解決するには、しきい値を確立して同期サーバーをスケールするタイミングを決定します。推奨値は 80% です。

CPU 使用率がこの値未満である場合は、同期サーバーをスケールする必要はなく、ワーカー間隔設定値を少し小さくして CPU 使用率を上げることができます。ただし、CPU 使用率がこのしきい値を超える場合は、同期サーバーのスケールリングを検討してください。

3.6.3.2. キュー内の残りのジョブ

同期サーバーはさまざまなジョブをキューに保存して後で処理します。

キュー内のジョブの数が増加し続け、CPU 使用率が比較的低い場合は、ジョブをより速く処理するためにワーカー間隔設定値を小さくします。

同期サーバーの負荷がすでに大きい場合は、新しいワーカーを作成してジョブを処理できるよう同期サーバーをスケールリングすることを検討してください。

3.6.3.3. API 応答時間

さまざまな同期 API の応答時間の増加が確認され、CPU 使用率が上昇している場合は、同期サーバーに負荷がある状態を意味します。同期サーバーのスケールリングを検討してください。

ただし、CPU 使用率がそれほど変わらない場合は、何かほかのことによってその問題が引き起こされていることを意味するため、その問題を調査する必要があります。

3.6.3.4. MongoDB 操作時間

本番稼働環境では、さまざまな MongoDB 操作の時間は比較的短く、一定です。これらの操作の時間が増え始めた場合は、同期サーバーにより MongoDB に対して生成される操作の数が多すぎる状態であり、MongoDB の制限に到達しようとしていることを意味します。

この場合は、ボトルネックが MongoDB に存在するため、同期サーバーをスケーリングしても問題は解決されません。以下の選択肢を検討します。

- ※ **useCache** フラグを `true` に設定してキャッシュを有効にします。これにより、データセットレコードを読み取るデータベース要求の数が減少します。
- ※ さまざまなワーカー間隔と同期周期を増加します。
- ※ 可能な場合は、MongoDB をスケーリングします。

3.6.4. 同期サーバーのスケーリング

同期サーバーをスケールする場合は、以下のことを検討します。

3.6.4.1. ホストされた MBaaS でのスケーリング

RHAMP SaaS プラットフォームで同期サーバーをスケールするには以下の 2 つの選択肢があります。

3.6.4.1.1. Node.js クラスターモジュールの使用

単一のアプリ内部でスケールするには、[Nodejs Clustering](#) を使用してさらにワーカーを作成します。

3.6.4.1.2. より多くのアプリのデプロイ

別の選択肢は、より多くのアプリをデプロイし、同じ MongoDB に対して既存のアプリとして指定することです。これにより、同期サーバーをさらにスケールすることが可能になります。

より多くのアプリをデプロイする手順は以下のとおりです。

- ※ 同じコードのより多くのアプリを既存のアプリとしてデプロイします。
- ※ 既存のアプリの MongoDB 接続文字列を見つけます。

これは、App Studio の **Environment Variable (環境変数)** 画面にリストされています。**FH_MONGODB_CONN_URL** という名前のシステム環境変数を探してください。

- ※ 値をコピーし、新しく作成されたアプリで **SYNC_MONGODB_URL** という名前の新しい環境変数を作成して、値として MongoDB url を貼り付けます。
- ※ アプリを再デプロイします。

この方法では、HTTP 要求処理と同期データ処理を完全に分離できます。

たとえば、このように設定された 2 つのアプリ、App 1 と App 2 が存在し、App 1 が HTTP 要求を受け取るクラウドアプリとします。この場合は、以下のことを行えます。

- ※ ワーカーの同時実行数を 0 に設定して App 1 のすべての同期ワーカーを無効にします。この結果、App 1 は HTTP 要求の処理のみを行うことになります。
- ※ App 2 の同期ワーカーの同時実行数を増加し、同期間隔値を減少します。

ワーカーの同時実行数の設定方法については、[\\$fh.sync.setConfig](#) を参照してください。

3.6.4.2. 自己管理された MBaaS でのスケーリング

自動スケーリング機能を使用して OpenShift でアプリケーションをスケールします。

メトリクスが OpenShift クラスターで有効であることを確認し、**oc autoscale** コマンドを使用してアプリケーションのスケール方法を設定します。

たとえば、OpenShift 3.2 の場合は、[クラスターメトリクスを有効にする方法](#)と[アプリケーションをスケールする方法](#)を参照してください。

3.7. 同期サーバーにより作成された MONGODB コレクション

3.7.1. 概要

同期サーバーは、実行中にさまざまなコレクションを MongoDB に保持します。

本書では、同期サーバーが作成するコレクションとその目的について説明します。



注記

データ損失が起こる可能性があるため、これらのコレクションは変更しないでください。

3.7.2. 同期サーバーコレクション

同期サーバーにより作成されたすべてのコレクションには接頭辞 **fhsync** が付けられます。

3.7.2.1. fhsync_pending_queue

このコレクションは、すべてのデータセットに対してすべてのクライアントから送信された変更を保存するために使用されます。

デバッグに役に立つフィールドは以下のとおりです。

- ※ **tries**: 値が 0 よりも大きい場合は、変更が同期サーバーによってすでに処理されたことを意味します。
- ※ **payload.hash**: 保留中の変更の一意の ID。
- ※ **payload.cuid**: クライアントの一意の ID。
- ※ **payload.action**: 変更の種類 (**create** や **update** など)。
- ※ **payload.pre**: 変更が行われる前のデータ。
- ※ **payload.post**: 変更が行われたあとのデータ。
- ※ **payload.timestamp**: 変更がクライアントで行われた日時。

3.7.2.2. fhsync_<datasetId>_updates

fhsync_pending_queue コレクションからの保留中の変更が処理された場合、結果はこのコレクションに保存されます。クライアントは、次の同期のときに結果を取得し、該当するクライアント通知をトリガーします。

デバッグに役に立つフィールドは以下のとおりです。

- ※ **hash**: 上記コレクションからの保留中の変更の一意な ID。
- ※ **type**: 変更が正常に適用された場合。可能な値は **applied**、**failed**、または **collision** です。

3.7.2.3. fhsync_ack_queue

クライアントは送信された変更の結果を取得したあとに (**fhsync_<datasetId>_updates** コレクションに保存)、サーバーと受信を確認してサーバーがその受信確認を削除できるようにします。このコレクションは、クライアントにより送信された確認を保存するために使用されます。

デバッグに役に立つフィールドは以下のとおりです。

- ※ **payload.hash**: **fhsync_pending_queue** コレクションからの保留中の変更の一意な ID。

3.7.2.4. fhsync_datasetClients

このコレクションは、同期サーバーによって管理されたすべてのデータセットクライアントを永続化するために使用されます。

デバッグに役に立つフィールドは以下のとおりです。

- ※ **globalHash**: データセットクライアントの現在のハッシュ値。
- ※ **queryParam**: データセットクライアントに関連付けられたクエリパラメーター。
- ※ **metaData**: データセットクライアントに関連付けられたメタデータ。
- ※ **recordUids**: データセットクライアントに属するすべてのレコードの一意な ID。
- ※ **syncLoopEnd**: データセットクライアントに対する最後の同期ループが完了した日時。

3.7.2.5. fhsync_<datasetId>_records

このコレクション内のこのデータは、データセットバックエンドからのデータのローカルコピーです。これにより、クライアントからの同期要求が速く処理され、データセットバックエンドに対する要求の数も減少します。

デバッグに役に立つフィールドは以下のとおりです。

- ※ **data**: データセットバックエンドから返されたレコードの実際のデータ。
- ※ **uid**: レコードの一意な ID。
- ※ **refs**: このレコードを含むすべてのデータセットクライアントの ID。

3.7.2.6. fhsync_queue

このコレクションは、**fhsync_<datasetId>_records** をデータセットバックエンドと同期する要求を保存するために使用されます。

デバッグに役に立つフィールドは以下のとおりです。

- ※ **tries**: 0 よりも大きい場合は、要求が同期サーバーによってすでに処理されたことを意味します。

3.7.2.7. fhsync_locks

データセットバックエンドと同期できるのは一度に 1 つのワーカーだけです。このためにロックが使用されます。このコレクションは、ロックを永続化するために使用されます。ロックメカニズムを使用して問題をデバッグしない限り、このコレクションを使用する必要性はほとんどありません。

3.7.3. キューコレクションのプルーニング

各キューコレクションに対して、ドキュメントは処理後すぐに削除されません。代わりに、ドキュメントは **deleted** と示されます。これにより、開発者はドキュメントを監査ログとして使用し、デバッグに役立てたりすることができるようになります。

これらのキューが容量を使用しすぎないようにするために、これらのメッセージには TTL (time to live) を設定できます。TTL 値に到達すると、これらのメッセージはデータベースから削除されます。

詳細については、[\\$fh.sync.setConfig](#) の「queueMessagesTTL」オプションを参照してください。

3.8. 同期サーバーデバッグガイド

3.8.1. クライアントの変更は適用されない

この問題のデバッグを容易に行うために、以下の質問に回答してください。

クライアントはサーバーに変更を送信しましたか？

クライアントが変更を行った後に同期要求の要求本文を確認して変更が送信されたかどうかを調べます。変更は **pending** アレイにある必要があります。以下に例を示します。

```
{
  "fn": "sync",
  "dataset_id": "myShoppingList",
  /*...*/
  "pending": [{
    "inFlight": true,
    "action": "update",
    "post": {
      "name": "Modified Name",
      "created": 1495123790928
    },
    "postHash": "2e90b858164184b9ff31e0937cef8ddf4a959ac5",
    "timestamp": 1495799747404,
    "uid": "591dc768a95300322eee1d1f",
    "pre": {
      "name": "Original Name",
      "created": 1495123790928
    },
    "preHash": "421932b23f05f8aef528d73fff3cbf5aa00786a4",
    "hash": "f98f595974f7e7e1f07aed6220fab04446f459c9",
    "inFlightDate": 1495799747850
  }]
}
```

変更が保留中のアレイにない場合は、デバイスがオンラインであることを検証します。クライアントアプリにエラーがないか確認し、該当する同期アクションを呼び出していることを検証します(たとえば、更新の場合は `sync.doUpdate()`)。クライアントアプリで、変更を行うコードの周辺でデバッグしたり、ロギングを追加したりすると役に立ちます。

この変更のレコードが `fhsync_pending_queue` コレクションにありますか？

「いいえ」の場合は、サーバーが変更を受け取っていないか、受け取るときにエラーが発生しました。

- ※ アプリが変更を正常に送信したことを検証します。送信しなかった場合は、アプリをデバッグして問題を理解してください。アプリにエラーがあるか、サーバーからの応答にエラーがあることが考えられます。
- ※ アプリから変更を受け取ったときにサーバーログでエラーを確認します。エラーがない場合は、[デバッグログの有効化](#)を参照してください。

レコードが `fhsync_pending_queue` コレクションに存在したが、レコードに対するキューの Time To Live (TTL) 期間が経過したため、レコードが削除された可能性があります。これに該当する場合は、TTL を増やすと、デバッグが有効になります。

レコードの `deleted` フィールドにタイムスタンプがありますか？

「いいえ」の場合は、そのアイテムがまだ処理されていません。

- ※ 通常は、保留中のワーカーが、そのアイテムよりも先にキュー内の他のアイテムを処理しています。そのアイテムが処理のためにキューの最上位に到達するまで待機してください。
- ※ しばらく時間がたってもアイテムが処理されない場合や、キューにこのアイテムしかない場合は、サーバーログでエラーを確認します。エラーがない場合は、[デバッグログの有効化](#)を参照してください。

この更新のレコードが `fhsync_pending_queue` コレクションにありますか？

「いいえ」の場合は、更新の処理中にエラーが発生した可能性があります。

- ※ サーバーログでエラーを確認します。エラーがない場合は、[デバッグログの有効化](#)を参照してください。

レコードの `type` フィールドが `failed` または `collision` に設定されていますか？

「はい」の場合は、更新をデータセットバックエンドに適用できない可能性があります。

- ※ 「競合」により、競合ハンドラーが呼び出されたことが考えられます。競合を解決する必要があります。
- ※ 更新の「失敗」により、障害の理由とともに通知がクライアントに送られたことが考えられます。理由はレコードの `msg` フィールドに記載されます。

「いいえ」であり、タイプが `applied` の場合は、作成または更新ハンドラーをデバッグして、なぜそのハンドラーで変更がデータセットバックエンドに適用されたと見なされたのかを確認する必要があります。

`type` フィールドは、`collision`、`failed`、または `applied` のいずれかである必要があります。

3.8.2. データセットバックエンドに適用された変更は他のクライアントに伝播されない

変更がデータセットバックエンドからクライアントに同期されるまでに十分な時間が経過しましたか？

変更がデータセットバックエンドに適用され、他のクライアントがその変更を取得する前に、以下の2つのことが起こる必要があります。

- ※ サーバー上の同期ループはレコードキャッシュの更新を完了する必要があります。つまり、そのデータセットに対してリストハンドラーが呼び出されます。
- ※ クライアント上の同期ループは、サーバーレコードキャッシュからのクライアントローカルレコードキャッシュの更新を完了する必要があります。

十分な時間が経過したら、データセットバックエンドとの同期中のエラーをサーバーログで確認してください。

データセット向けの `fhsync_queue` コレクションに最近のレコードがありますか？

「いいえ」の場合は、レコードの TTL 値が経過し、レコードが削除された可能性があります。この場合は、TTL 値を増やしてさらにデバッグすることができます。

別の可能性としては、同期スケジューラーでそのデータセットの同期がスケジュールされなかったことが挙げられます。この理由としては、そのデータセットに対して現在アクティブなクライアントがなく、そのデータセットに対してクライアントが最後にアクティブであったときから `clientSyncTimeout` が経過したことが考えられます。

レコードの `deleted` フィールドにタイムスタンプがありますか？

「いいえ」の場合は、同期がまだ処理されていないことを意味します。

- ※ 通常は、同期ワーカーが、そのアイテムよりも先にキュー内の他のアイテムを処理しています。そのアイテムがキューの最上位に到達するまで待機してください。
- ※ しばらく時間がたってもアイテムが処理されない場合や、キューにこのアイテムしかない場合は、サーバーログでエラーを確認します。エラーがない場合は、[デバッグログの有効化](#)を参照してください。

`fhsync_<datasetid>_records` キャッシュ内のレコードは最新ですか？

リストハンドラーが呼び出され、結果がレコードキャッシュに追加されている必要があります。レコードキャッシュが更新されたことを検証するには、更新されたレコードの `fhsync_<datasetid>_records` コレクションを確認します。このレコード内のデータはデータセットバックエンドのデータに一致する必要があります。一致しない場合は、サーバーログでリストハンドラーのエラーと動作を確認します。リストハンドラーにロギングを追加すると役に立つことがあります。

クライアント同期呼び出しが正常に実行されましたか？

クライアントが同期呼び出しを行う場合に、サーバーからの有効な応答があることを確認します。呼び出しが正常に行われた場合は、クライアントが更新済みレコードを取得していることを検証します。サーバーキャッシュに更新済みレコードが含まれるのにクライアントが更新済みレコードを受け取らない場合は、クエリーパラメーターとサーバーに送信されたメタデータが正しいことを確認します。デバッグログを有効にすると、正しくないデータがどのようにクライアントに送信されたのかを簡単に調べることができる場合があります。

3.8.3. デバッグログの有効化

デバッグのために同期ログを有効にするには、サーバーで以下の環境変数を設定します。

DEBUG=fh-mbaas-api:sync

この処理により、大量のログが生成されます。各ログエントリーには、そのログメッセージのコンテキストでアクションを実行する特定のデータセット ID がタグ付けされます (可能な場合)。これらのログは理解しにくいことがありますが、クライアントからの更新とそれらの更新のさまざまな段階を追跡できます。成功したシナリオのログと不成功のシナリオのログを比較し、障害が発生した段階を特定すると役に立つことがあります。

この問題の原因は、(特にエッジケースに関連する) カスタムハンドラー実装にあることが考えられます。カスタムハンドラーにログを追加すると役に立つことがあります。

データセットバックエンド接続の問題 (特に断続的な問題) はデバッグおよび特定が困難なことがあります。データセットバックエンドを外部的に監視または確認すると、役に立つことがあります。

第4章 RHMAP と他のサービスとの統合

4.1. MBaaS サービスの概要

MBaaS サービスは、複数のプロジェクト内の複数のクラウドアプリケーションが使用できる共有機能を提供するクラウド/Web アプリケーションです。通常のユースケースは以下のとおりです。

- ※ サードパーティ製のサービスにアクセスするために API を提供します。たとえば、複数のプロジェクトに SMS メッセージを送信する要件が含まれるとします。この場合は、SMS メッセージを送信する API を提供するためにサービスを作成し、複数のプロジェクトでそのサービスにアクセスすることができます。
- ※ レガシーカスタマーバックエンドシステムに API を提供します。たとえば、ユーザー認証を実行するレガシーシステムがあるとします。この場合は、プロジェクトで使用する一貫性のある新しい API セットを提供するサービスを作成し、レガシーシステムへのアクセス方法に関するすべての詳細を隠すことができます。



注記

MBaaS サービスは、クライアントアプリで直接使用するよう設計されていません。たとえば、iOS アプリは MBaaS サービスを直接呼び出しません。クライアントアプリは関連するクラウドアプリを呼び出し、クラウドアプリは要求を 1 つ以上の MBaaS サービスに送信します。

MBaaS サービスを使用する利点は以下のとおりです。

- ※ コードの再利用性が向上します。これは、外部サービスへのアクセスが複雑な場合に特に重要です。
- ※ 機密情報を保護します。場合によっては、外部サービスにアクセスするためにユーザークレデンシャルが必要です。サービスを使用することにより、複数のプロジェクト開発者がクレデンシャルを共有する必要がなくなります。

4.2. 認証に対する SAML の使用

4.2.1. 概要

このチュートリアルでは、モバイルアプリケーションで SAML 認証を使用する例を示します。SAML は現在 Red Hat Mobile Application Platform ホスト型 (RHMAP) では認証ポリシーとして利用できませんが、ユーザーの独自のソリューションの土台として使用できるテンプレートが提供されます。

SAML 認証の同じソリューションは、サーバーサイドロジックとクライアントアプリの両方で構成されます。

- ※ **SAML サービス** — SAML 2.0 認証を実行するために、サービスプロバイダーとして機能し、[passport-saml](#) とともに [Passport.js](#) を使用するクラウドサービス
- ※ **SAML Cloud App** — クライアントアプリから SAML サービスへの要求を代理するクライアントアプリ

以下の各プラットフォーム向けのクライアントアプリが提供されます。

- ✧ [Cordova](#)
- ✧ [Android](#)
- ✧ [iOS](#)
- ✧ [Windows Phone](#)

4.2.2. SAML の概要

提供されたテンプレートの構造について説明するために、最初に SAML のコンセプトを簡単に説明します。SAML 認証シナリオには以下の 3 つの要素があります。

- ✧ **プリンシパル**—サービスプロバイダーにより提供された保護済みリソースにアクセスしようとするユーザー
- ✧ **サービスプロバイダー (SP)**—プリンシパルにサービスを提供する Web アプリケーション
- ✧ **ID プロバイダー (IdP)**—プリンシパルの ID を検証し、サービスプロバイダーに ID アサーションを提供することが唯一の目的の Web サービス

これらの要素は、複数の前提に基づいて認証の処理で対話します。

- ✧ プリンシパルはサービスプロバイダーとパスワードまたは他のシークレットを共有しません。
- ✧ ID プロバイダーはプリンシパルとサービスプロバイダーの両方により信頼されています。
- ✧ ID プロバイダーは、ID アサーションを多くのサービスプロバイダーに提供して **Single Sign-On (SSO)** を有効にできます。

4.2.3. テンプレートのセットアップ

最初に、提供されたテンプレートから **SAML Service** クラウドサービスを作成します。このサービスはソリューションの中心的なコンポーネントであり、サービスプロバイダーとして機能します。この例で提供される実際のサービスでは、エンドポイント **/session/valid** でユーザーの詳細が表示されています。SAML 認証プロセスをサポートするエンドポイントは他に複数あります。

- ✧ **/login**—IdP のログイン画面にリダイレクトします。
- ✧ **/session/login_host**—IdP のログイン画面の URL を返します。
- ✧ **/login/callback**—IdP が ID アサーションをこのエンドポイントにポストします (ログインの成功または失敗が示されます)。
- ✧ **/login/ok**—ログイン成功を示すために、クライアントアプリがこの URL にリダイレクトされます。

サービスは IdP として **Active Directory Federation Services 2.0 (ADFS)** を使用するよう設定されていますが、少しだけ調整して他の任意の SAML 2.0 準拠 IdP を使用するよう設定できます。

4.2.3.1. SAML サービスの作成

1. Studio で、**Services & APIs** に移動し、**Provision MBaaS Service/API** をクリックします。
2. **SAML Service** テンプレートを見つけ、**Choose** をクリックし、任意の名前 (たとえば、"SAML Service") を入力して **Create** をクリックします。

3. 以下のように設定の詳細を入力します。

- ✳ **SAML_ISSUER** — 現時点では空白のままにします。
- ✳ **SAML_CALLBACK_URL** — 現時点では空白のままにします。
- ✳ **SAML_ENTRY_POINT** — ADFS エンドポイントに URL を入力します。
- ✳ **SAML_AUTHN_CONTEXT** — 値 `urn:federation:authentication:windows` を入力します。
- ✳ **SAML_CERT** — このデモの場合は、空白のままにできます。

4. **Next** をクリックし、サービスが作成されるまで待機します。次に、**Finish** をクリックします。

サービスが作成およびデプロイされたら、**SAML_ISSUER** と **SAML_CALLBACK_URL** を設定する必要があります。

1. **Service Details** ページで **Current Host** フィールドを確認し、その値をメモします。この値はこの例では `<mbaas-host>` と示されます。また、**Service ID** の値もメモします。
2. 左側のメニューで **Environment Variables** をクリックします。
3. **App Environment Variables** セクションで、**SAML_ISSUER** と **SAML_CALLBACK_URL** の両方の変数に以下の値を設定します (`<mbaas-host>` は手順 1 で見つけた値に置き換えます)。

```
<mbaas-host>/login/callback
```

4.2.3.2. プロジェクトのセットアップ

最初にプロジェクトを作成します。

1. Studio で、**Projects** に移動し、**New Project** をクリックします。
2. **SAML Example Project** を選択し、**Choose** をクリックして、任意の名前 (たとえば、"SAML Example") を入力して **Create** をクリックします。
3. プロジェクトが作成されたら、**Finish** をクリックします。

次に、以前に作成した SAML サービスをプロジェクトに関連付ける必要があります。

1. **SAML Example** プロジェクトの **MBaaS Services** 列で、**+** をクリックして新しいサービスを追加します。
2. 以前に作成した **SAML サービス** を探し、クリックして、画面下部で **Associate Services** をクリックします。

クラウドアプリには SAML サービスへの参照が必要です。SAML サービスを参照する環境変数を作成します。

1. **SAML Cloud** クラウドアプリに移動し、左側の **Environment Variables** セクションにアクセスします。
2. **App Environment Variables** セクションで **Add Variable** をクリックし、以下の値を入力

して、**Create** をクリックします。

✳ 名前: **SAML_SERVICE**

✳ 値: 以前にメモした SAML サービスの**サービス ID**

3. **Push Environment Variables** をクリックして環境変数をランタイム環境に伝播します。

アプリをビルド、デプロイ、およびテストすることができます。

4.2.4. 仕組み

認証プロセス中に起こったことを見ていきます。

Sign In ボタンをクリックすると、クライアントアプリが **sso/session/login_host** エンドポイントを呼び出し IdP のログイン画面の URL を取得します。この URL はアプリ内ブラウザで開き、IdP と認証できます。

JavaScript

```
$('.sign-in-button').on('click', function(e) {
  $fh.cloud({
    path: 'sso/session/login_host',
    method: "POST",
    data: {
      "token": $fh._getDeviceId()
    }
  },
  function(res) {
    console.log('sso host:' + res.sso);
    var browser = cordova.InAppBrowser.open(res.sso, '_blank',
    'location=yes');
    ...
  }
});
```

Android

```
JSONObject params = new JSONObject();
params.put(TOKEN, Device.getDeviceId(getApplicationContext()));

FHCloudRequest request = FH.buildCloudRequest(
  "sso/session/login_host", "POST", null, params);
request.executeAsync(new FHActCallback() {
  @Override
  public void success(FHResponse res) {
    Log.d(TAG, "FHCloudRequest (login_host) - success");
    String ssoStringURL = res.getJson().getString("sso");
    Log.d(TAG, "SSO URL = " + ssoStringURL);
    SAMLActivity.this.displayWebView(ssoStringURL);
  }
}
...
}
```

iOS

```

NSString* deviceId = [[FHConfig sharedInstance] uuid];
NSMutableDictionary __params = [NSMutableDictionary dictionary];
params[@"token"] = deviceId;
FHCloudRequest __cloudReq = [FH
buildCloudRequest:@"sso/session/login_host" WithMethod:@"POST"
AndHeaders:nil AndArgs: params];

// Initiate the SSO call to the cloud
[cloudReq execWithSuccess:^(FHResponse _success) {
    NSLog(@"EXEC SUCCESS =%@", success);
    NSDictionary_ response = [success parsedResponse];
    NSString* urlString = response[@"sso"];
    NSURL* loginUrl = [[NSURL alloc] initWithString:urlString];
    // Display WebView
    FHSAMLViewController *controller = [[FHSAMLViewController alloc]
initWithURL:loginUrl];
    [[[[UIApplication sharedApplication] keyWindow] rootViewController]
presentViewController:controller animated:YES completion:nil];
    ...
}

```

Windows

```

private async void Button_Click(object sender, RoutedEventArgs e)
{
    var response = await
    FHClient.GetCloudRequest("sso/session/login_host", "POST", null,
    GetRequestParams()).ExecAsync();

    var resData = response.GetResponseAsJsonObject();
    var sso = (string)resData["sso"];
    if (!string.IsNullOrEmpty(sso))
    {
        webView.Visibility = Visibility.Visible;
        webView.Navigate(new Uri(sso));
    }
}

```

IdP で必要なクレデンシャルを入力し、ログインフォームを送信したら、IdP は ID アサーションを SAML サービスの **/login/callback** エンドポイントにポストし直します。

SAML サービスは以下のことを行います。

- ※ 受け取った SAML アサーションをユーザーのトークンに関連付けます (HTTP セッションで渡されます)。
- ※ SAML アサーションからのデータを保持します。
- ※ ユーザーを **/login/ok** にリダイレクトします (認証の成功がクライアントアプリに通知されます)。

ユーザーが正常にログインしたら、アプリ内ブラウザが閉じられ、クライアントアプリはサービスプロバイダーにより提供されたサービスを使用できます (**sso/session/valid** を呼び出しユーザーの詳細を取得します)。

JavaScript

```

$fh.cloud({
  path: 'sso/session/valid',
  method: "POST",
  data: {
    "token": $fh._getDeviceId()
  }
},
function(details) {
  $('first_name').text(details.first_name);
  $('last_name').text(details.last_name);
  $('email').text(details.email);
  $('expires').text(details.expires);
},
...

```

Android

```

JSONObject params = new JSONObject();
params.put(TOKEN, Device.getDeviceId(getApplicationContext()));

FHCloudRequest request = FH.buildCloudRequest(
  "sso/session/valid", "POST", null, params);
request.executeAsync(new FHActCallback() {
  @Override
  public void success(FHResponse res) {
    Log.d(TAG, "FHCloudRequest (valid) - success");
    User user = new User();
    user.setFirstName(res.getJson().getString("first_name"));
    user.setLastName(res.getJson().getString("last_name"));
    user.setEmail(res.getJson().getString("email"));
    user.setExpires(res.getJson().getString("expires"));

    Log.d(TAG, user.toString());

    SAMLActivity.this.displayUserData(user);
  }
});
...

```

iOS

```

NSString* deviceID = [[FHConfig sharedInstance] uuid];
NSMutableDictionary __params = [NSMutableDictionary dictionary];
params[@"token"] = deviceID;
FHCloudRequest __cloudReq = [FH buildCloudRequest:@"sso/session/valid"
  WithMethod:@"POST" AndHeaders:nil AndArgs: params];

// Initiate the SSO call to the cloud
[cloudReq execWithSuccess:^(FHResponse _success) {
  NSDictionary_ response = [success parsedResponse];
  // Manage next UI view controller
  [self performSegueWithIdentifier:@"showLoggedIn" sender: response];
} AndFailure:^(FHResponse *failed) {
  NSLog(@"Request name failure =%@", failed);
}];

```

Windows

```
var response = await FHClient.GetCloudRequest("sso/session/valid",
"POST", null, GetRequestParams()).ExecAsync();
if (response.Error == null)
{
    var data = response.GetResponseAsDictionary();
    name.Text = string.Format("{0} {1}", data["first_name"],
data["last_name"]);
    email.Text = (string) data["email"];
    expires.Text = ((DateTime) data["expires"]).ToString();
}
else
{
    await new MessageDialog(response.Error.ToString()).ShowAsync();
}
```

4.3. REDHAT OPENSIFT ONLINE PAAS へのクラウドアプリのステージング

概要

このガイドの目的は Red Hat Mobile Application Platform ホスト型 (RHMAP) で OpenShift Online を MBaaS ターゲットとして使用し、アプリケーションをデプロイする手順を提供することです。

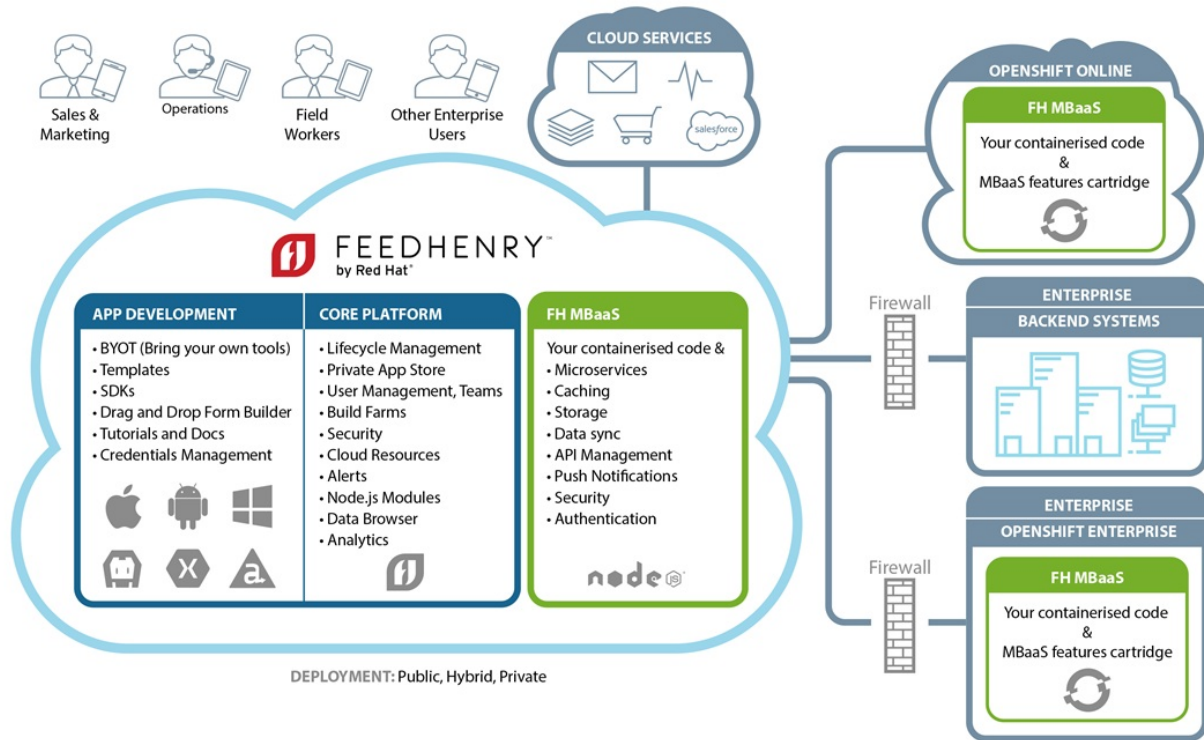
4.3.1. OpenShift Online とは

[OpenShift Online](#) は、アプリケーションのプロビジョニング、管理、およびスケーリングを自動化する、Red Hat のパブリッククラウドアプリケーションの開発およびホスティングプラットフォームです。

4.3.1.1. OpenShift Online と RHMAP との統合

OpenShift Online ユーザーは OpenShift Online クレデンシャルを使用して RHMAP にログインし、クラウドアプリとサービスの MBaaS ターゲットとして利用可能な OpenShift Online ギアを使用できます。**MBaaS ターゲット**は、クラウドアプリとその関連するサービスをデプロイできるコンテナ化されたインフラストラクチャーです。また、1 つまたは複数の MBaaS ターゲットを環境に収集することもできます。ユーザーは複数の環境を作成して (各環境に 1 つまたは複数の MBaaS ターゲットが含まれる状態で)、開発の異なるステージ (たとえば、Dev、UAT、および Production 環境) とプロジェクトライフサイクル管理を有効にできます。

また、MBaaS サービスに関連するプロビジョニング、ログイン、エンドポイントセキュリティなどのすべての操作は OpenShift Online MBaaS にも適用されます。これらの操作の詳細については、[MBaaS Services Product Features page](#) を参照してください。



4.3.1.2. RHMAP による利用可能な OpenShift Online ギアへの影響

RHMAP アプリケーションは、OpenShift Online にデプロイする他のすべてのアプリケーションと同じように扱われます。利用可能なギアの一部またはすべてを使用して RHMAP クラウドと Web アプリケーションをデプロイできますが、利用可能なギアの数によって制限されます。OpenShift Online で実行されているすべてのアプリケーションと同様に、利用可能なギアと機能の数は[現在のプラン](#)によって異なります。RHMAP で OpenShift Online ギアを使用する方法の詳細については、項「[ギア](#)」を参照してください。

4.3.1.3. 制限

RHMAP で OpenShift Online を MBaaS ターゲットとして使用することは現時点では開発者レビューとして可能であり、本番稼働環境には推奨されません。

4.3.2. スタートガイド

最初に必要なものは OpenShift Online アカウントです。既存のアカウントを使用するか、[新しいアカウントを作成](#)することができます。



注記

これは OpenShift Online アカウントを使用して RHMAP にログインする場合のみ可能です。この機能は既存の RHMAP アカウントでは利用できません。

4.3.2.1. 初回ログインおよびセットアップ

OpenShift Online アカウントを取得したら、これらのクレデンシャルを使用して RHMAP にログインできます。OpenShift Online アカウントを使用して初めてログインする場合は、RHMAP によってセットアッププロセスが示されます。それ以降のログインでは、セットアッププロセスなしで RHMAP にアクセスできます。



注記

要求に対応するために、アクセスは招待状によってのみ提供しています。ログインページに提供されたリンクを使用して興味があることを登録すると、スペースが利用可能になり次第招待状メールが送信されます。招待状メールには、アクセスを提供する適切なログインリンク (アクセストークンを含む) が含まれます。ログインするためにログインページに初めてアクセスする場合はこのリンクを使用する必要があります。

セットアッププロセスを開始するには、以下の手順に従ってください。

1. [RHMAP OpenShift Online ログインページ](#)に移動し、**Request an Invite** をクリックします。
2. サインアップフォームに適切な詳細情報 (利用可能な場合は プロモーションコードを含む) を入力し、**Sign Up** をクリックします。

OPENS SHIFT

SIGN UP TO FEEDHENRY

Company Name

Full Name

Email

Promotion Code (Optional)

☐ Agree to [Terms and Conditions](#)

SIGN UP

Already have an account? [Login Here](#)

3. 招待状メールを受け取ったら、招待状メールに提供されたリンクから RHMAP に移動し、OpenShift Online クレデンシャルを使用してログインします。
4. 正常にログインしたら、RHMAP によってドメイン名を作成するよう求められます。希望するドメイン名を入力し、**Create** をクリックします。
5. RHMAP によってドメインが作成され、必要なセットアッププロセスが自動的に開始され

ます。基本的に RHMAP により承認トークンとパブリックキーが作成され、RHMAP が OpenShift Online ギアと通信し、アプリケーションをユーザーの代わりにデプロイできるようになります。

自動セットアッププロセスが完了したら、RHMAP が OpenShift Online アカウントに接続されます。この時点で OpenShift Online は、クラウドアプリケーションをデプロイしたときに MBaaS ターゲットとして現れます。



OpenShift Online を MBaaS ターゲットとして使用方法を示すために、テンプレートから新しいサンプルアプリケーションを作成し、クラウド部分を OpenShift Online にデプロイする手順について詳細に説明します。

4.3.2.2. サンプルアプリケーションの作成

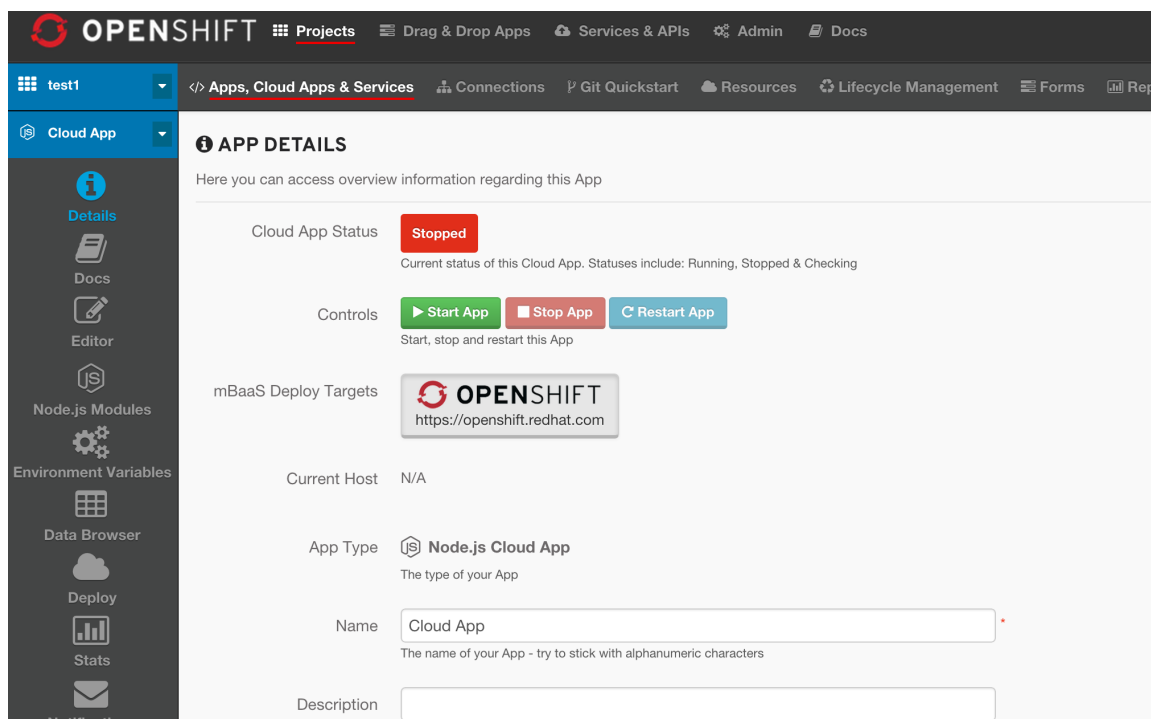
新しいアプリケーションを作成するには、以下の手順に従ってください。

1. 画面の左上にある **Projects** をクリックします。
2. 画面の左側にある **New Project** ボタンをクリックします。
3. 少なくとも 1 つのクラウドコンポーネントを含むプロジェクト (たとえば、**Hello World Project**) を選択します。
4. プロジェクトの名前を入力し、画面の右側にある **Create** ボタンをクリックします。
5. プロジェクトが作成されたら、画面の下部にある **Finish** ボタンをクリックします。
6. プロジェクトの詳細ページが表示されます。

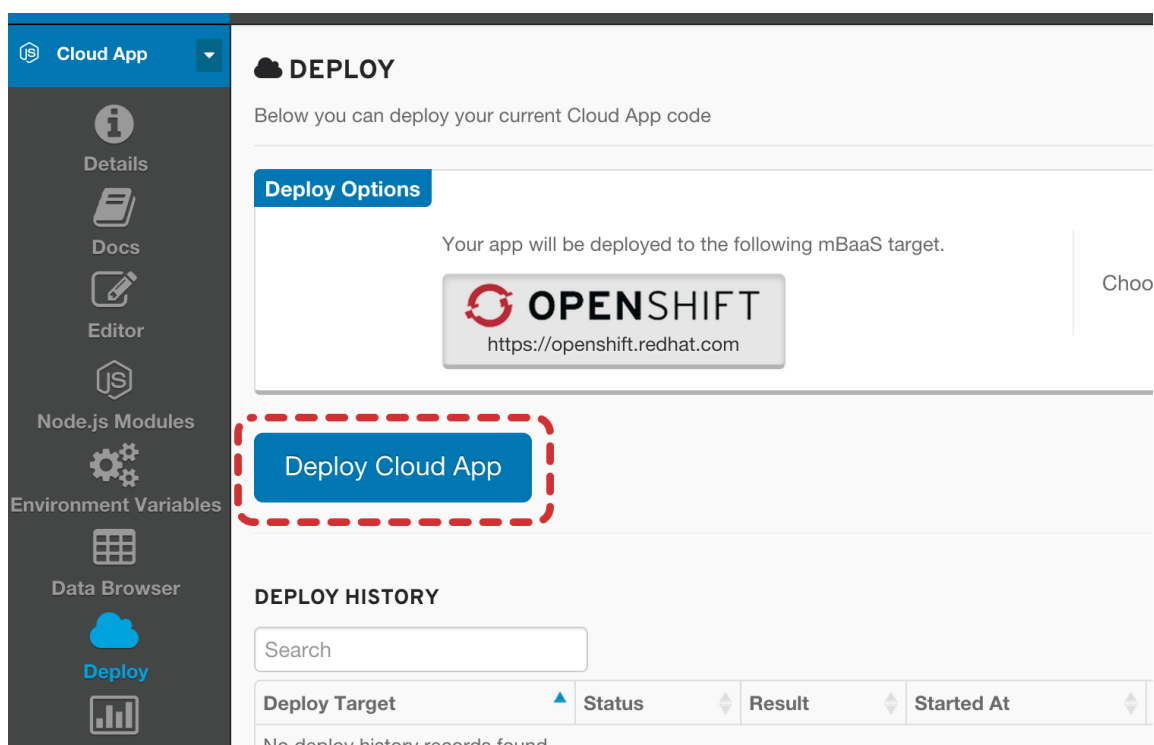
4.3.2.3. RHMAP を使用した OpenShift Online へのクラウドアプリのデプロイ

クラウドアプリを OpenShift Online にデプロイするには、以下の手順に従ってください。

1. クラウドアプリの主要な詳細セクションが表示されていない場合は、そのセクションに移動します。
2. OpenShift Online にデプロイするクラウドまたは Web アプリ (クラウドアプリなど) を選択します。
3. アプリの詳細ページで OpenShift が **MBaaS Deploy Targets** の隣にリストされていることを確認します。



4. 画面の左側にある **Deploy** をクリックします。
5. 画面の下部にある **Deploy Cloud App** ボタンをクリックします。



6. これによりデプロイメントが開始され、ステータスが表示されます。



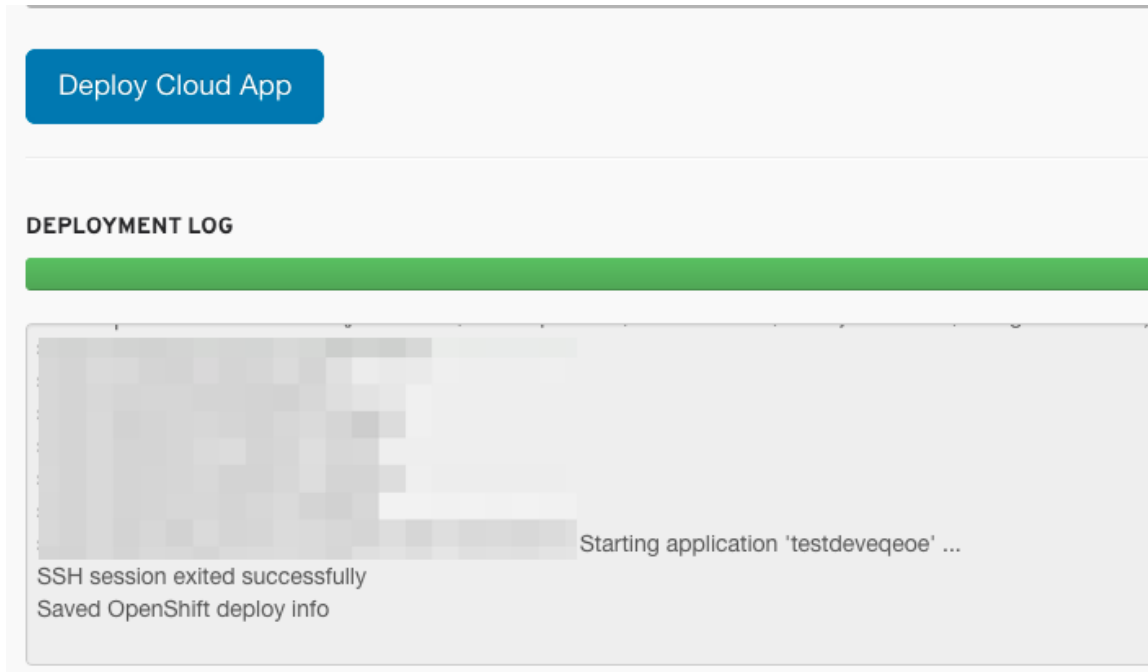
注記

初めてのデプロイメントの場合、OpenShift Online へのデプロイにはしばらく（数分）時間がかかることがあります。それ以降の同じアプリのデプロイメントでは時間が大幅に短縮されます。デプロイメントが開始され、ステータスに従わない場合は、ページから離れることができ、アプリが引き続き環境にデプロイされます。

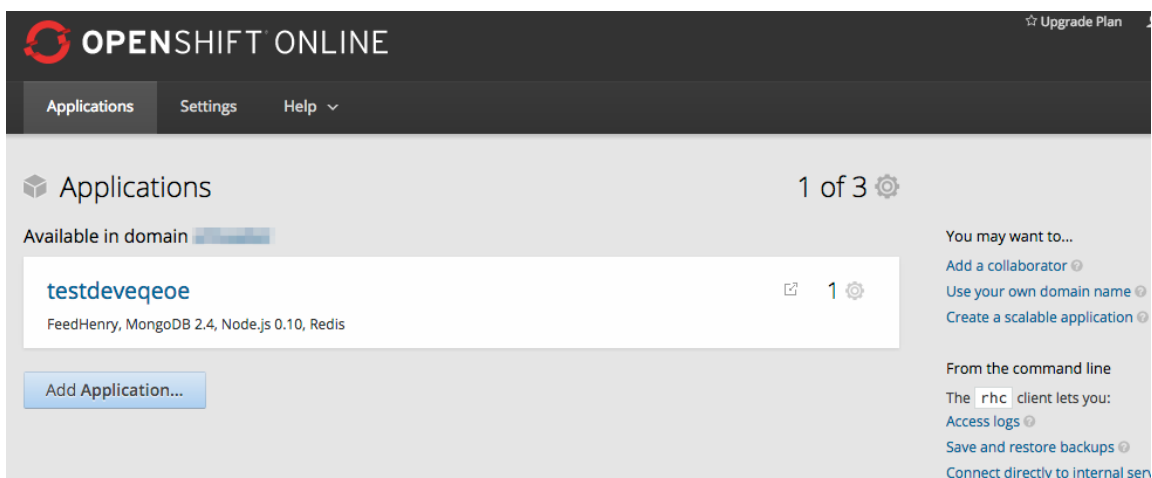


注記

ギアの不足に関連する警告を受け取ることがあります。これは、OpenShift Online アカウントに利用可能なギアが残っていない場合に起こります。OpenShift Online アカウントから他のアプリケーションを削除してギアを解放する必要があることがあります。



7. デプロイメントが正常に完了したら、画面の左側にある **Details** をクリックしてアプリの詳細ページに戻ります。
8. アプリがデプロイされたことを確認するには、**Current Host** の隣にあるリンクをクリックします。
9. また、OpenShift Online アカウントにログインして、アプリがそこにデプロイされていることを確認することもできます。



4.3.2.4. fhc を使用した OpenShift Online へのクラウドアプリのデプロイ



注記

本項では、**fhc** コマンドラインツールを使用します。**fhc** のインストールおよび使用に関する詳細については、[Local Development ドキュメンテーション](#)を参照してください。

fhc を使用してクラウドアプリを OpenShift Online にデプロイするには、以下の手順を実行する必要があります。

1. ターゲットを設定し、ログインします。
2. OpenShift Online MBaaS ターゲットを確認します。
3. デプロイする環境の **ID** を取得します。
4. デプロイするアプリの **ID** を取得します。
5. **stage** 操作を使用してアプリをデプロイします。

ターゲットを設定し、ログインするには、以下の手順に従ってください。

```
$ fhc target https://{YOUR-DOMAIN}.openshift.feedhenry.com
```

```
$ fhc login
```

OpenShift Online MBaaS ターゲットを検証するには、以下のコマンドを実行して、現在設定されている MBaaS サービスをリストします。

```
$ fhc admin mbaas list
```

この結果、以下のような情報が表示されます。

ID	URL	サービスキー	ユーザー名	パスワード	変更時間
openshift-jsmith-email-com-enJo	https://openshift.redhat.com	undefined	jsmith@email.com	undefined	数秒前

デプロイ先の環境の **ID** を取得するには、以下のコマンドを実行して利用可能な環境をリストします。

```
$ fhc admin environments list
```

この結果、以下のような情報が表示されます。

ID	ラベル	作成時のデプロイ	更新時のデプロイ	MBaaS ターゲット	変更時間
production-openshift-jsmith-email-com-enjo	Production	undefined	undefined	openshift-jsmith-email-com-enJo	7 日前
development-openshift-jsmith-email-com-enjo	Development	undefined	undefined	openshift-jsmith-email-com-enJo	7 日前

デプロイするアプリの **ID** を取得するには、アプリが含まれるプロジェクトの**ID** を最初を取得する必要があります。現在のプロジェクトは以下のコマンドを実行してリストできます。

\$ fhc projects

この結果、以下のような情報が表示されます。

ID	タイトル	アプリの数	最終更新日時
wp3nlgt2jr5lvymx62tayp5z	project1	2	2 時間前
piilhyeyqpad4nlgyveo6a43	project2	2	1 日前

この時点でプロジェクトの **ID** を使用して **apps** コマンドでアプリの**ID** を取得できます。

fhc apps <projID>

たとえば、アプリが **project1** に含まれる場合は、以下のコマンドを実行します。

\$ fhc apps wp3nlgt2jr5lvymx62tayp5z

この結果、以下のような情報が表示されます。

ID	タイトル	説明	タイプ	Git	ブランチ
wp3nlgrxlyzbvgwvfjnmulnat	クラウドアプリ		cloud_nodejs	git@example.feedhenry.net:openshift/project1-Cloud-App.git	master

ID	タイトル	説明	タイプ	Git	ブランチ
wp3nlgudwgejhnz buj5twsus	Cordova アプリ リ		client_hybrid	git@example. feedhenry.net :openshift/.git	

最後に、アプリと環境の ID を使用して **stage** コマンドでアプリを OpenShift Online にデプロイできます。

```
fhc app stage --app=APP-ID --env=ENV-ID
```

たとえば、クラウドアプリを **project1** から **Development** にデプロイする場合は、以下のコマンドを実行します。

```
fhc app stage --app=wp3nlgxrlyzbgvwfjnmulnat --env=development-openshift-jsmith-example-com-enjo
```

実行後に、以下のような情報が出力されます。

```
{
  "action": {},
  "cacheKey": "CloudAppdevelnat-openshiftdeploy",
  "error": "",
  "log": [],
  "status": "complete"
}
```

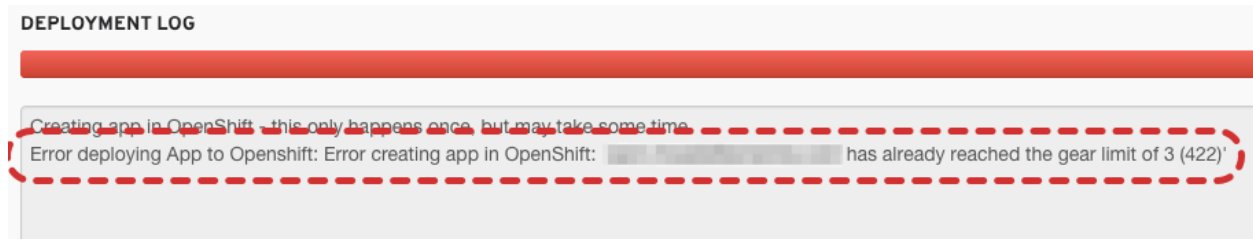
4.3.3. どのように連携するか

RHMAP は、OpenShift Online ギアを MBaaS ターゲットとして扱うために、最初に OpenShift Online への接続を確立する必要があります。この接続は OpenShift Online クレデンシャルを使用して RHMAP に初めてログインする場合にセットアップされます。RHMAP はユーザーのクレデンシャルを使用してユーザーの代わりに OpenShift Online にログインし、承認トークンを交換して、RHMAP と OpenShift Online 間の今後の通信を可能にする SSH キーをセットアップします。次に、RHMAP は自動的に OpenShift Online を MBaaS ターゲットとして設定し、プロジェクトで利用できるようにします。

初期設定が完了し、RHMAP プロジェクトで OpenShift MBaaS ターゲットを使用できるようになったら、RHMAP クラウドアプリを OpenShift Online にデプロイできます。これらのアプリは他の OpenShift Online アプリケーションと同様に扱われ、特別な利点として RHMAP に統合されます。たとえば、デプロイメント、アプリケーションロギング、および他の操作は引き続き RHMAP から実行できますが、OpenShift Online で実行されているアプリケーションに SSH 経由で直接接続することもできます。また、アプリケーションが OpenShift Online コンソールから直接デプロイされた場合は、そのアプリケーションの詳細を表示することもできます。

4.3.3.1. ギア

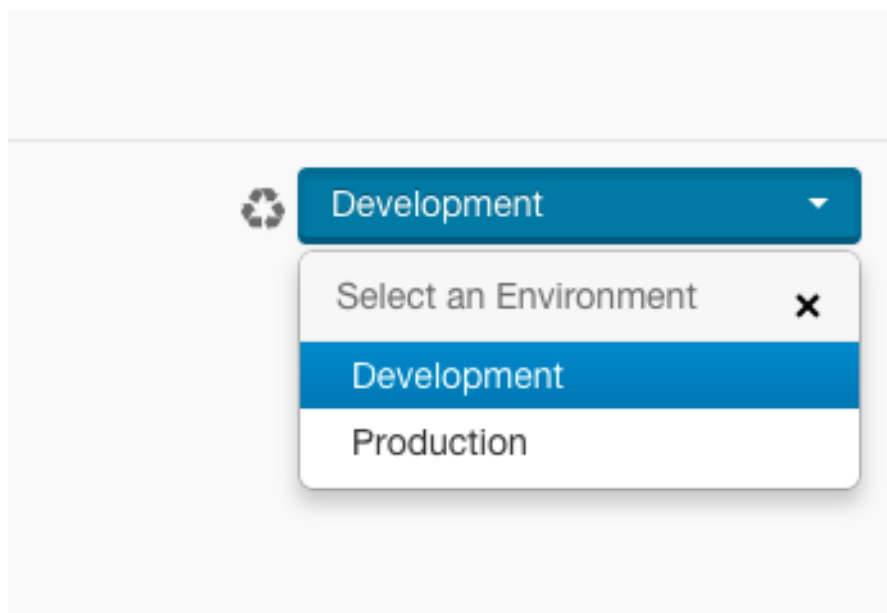
OpenShift Online にデプロイされた RHMAP クラウドアプリは OpenShift Online にデプロイされた他のアプリケーションと同様に扱われるため、利用可能なギアの数に制限されます。ギア数は **プラン**によって異なることがあります。RHMAP からクラウドアプリをデプロイしようとし、利用可能なギアが不足している場合は、以下のメッセージが表示されます。



この問題を解決するには、[アカウントのアップグレード](#)または OpenShift Online で実行されている別のアプリケーションの削除を検討してください。

4.3.3.2. 環境

OpenShift Online クレデンシャルを使用して RHMAP にログインすると、**Development** と **Production** の 2 つの環境が設定されています。これらの環境は、クラウドアプリケーションの詳細を表示するときに画面の右上にあるメニューを使用して切り替えることができます。



すべての環境でのすべてのクラウドアプリ開発の概要については、画面の上部のプロジェクトの **Lifecycle Management** セクションをクリックしてください。

各環境の各アプリケーションは独自のギアを受け取ることに注意してください。たとえば、プロジェクトに **App1** と **App2** の2つのクラウドアプリがあり、各アプリケーションに1つのギアが必要な場合は、**Development** と **Production** で **App1** と **App2** を実行するために4つのギアが必要になります。



注記

この時点で、OpenShift Online で RHMAP を使用する場合は、提供された **Development** と **Production** 以外の他の環境を設定することはできません。

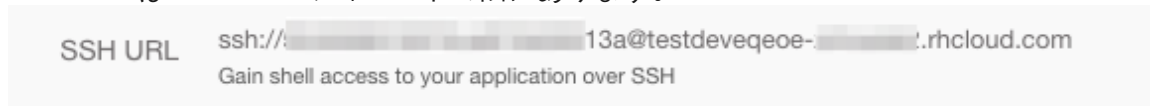
4.3.3.3. ロギングおよびデバッグ

OpenShift Online にデプロイされたクラウドアプリのログ参照とトラブルシューティングは、RHMAP または OpenShift Online ツールから実行できます。

OpenShift ツール

OpenShift では、SSH 経由でログインしてアプリケーションをトラブルシューティングする機能が提供されます。クラウドアプリの SSH URL を調べるには以下の手順に従ってください。

1. プロジェクトに移動します。
2. 希望するクラウドアプリをクリックします。
3. 画面の左側にある **Details** をクリックします。
4. SSH URL は **SSH URL** フィールドの隣にあります。



注記

SSH URL フィールドは、OpenShift Online にデプロイされたクラウドアプリでのみ利用可能です。SSH を使用した OpenShift への接続の詳細については、[OpenShift SSH ヘルプページ](#)を参照してください。

SSH と他の OpenShift Online ツールの詳細については、[公式ドキュメンテーション](#)を参照してください。

RHMAP ツール

RHMAP からクラウドアプリのログを参照するには、以下の手順に従ってください。

1. プロジェクトに移動します。
2. 希望するクラウドアプリをクリックします。
3. 画面の左側にある **Logs** をクリックします。

RHMAP でのアプリケーションのデバッグの詳細については、[デバッグガイド](#)を参照してください。

4.3.3.4. OpenShift Online にデプロイされたアプリの削除

クラウドアプリは RHMAP 側から削除する必要があります。RHMAP では、OpenShift Online からのクラウドアプリの削除が自動的に実行されます。

RHMAP からクラウドアプリを削除するには、以下の手順に従ってください。

1. 希望するプロジェクトに移動します。
2. 削除するクラウドアプリを選択します。
3. **Details** セクションで、画面下部にある **Delete App** をクリックします。



注記

アプリケーションを RHMAP で削除せずに OpenShift Online で削除し、OpenShift Online に再デプロイしようとする、**error: "Error deploying App to OpenShift: Error disabling auto deploy: Application 'XXXXXXXXXXXXXXXXXXXXXXX' not found. (404)"** というメッセージが表示されることがあります。

4.3.3.5. SSH キー

OpenShift Online クレデンシャルを使用して初めて RHMAP にログインすると、RHMAP により OpenShift Online との認証および承認のやり取りが実行され、OpenShift Online アカウントにパブリックキーが追加されます。次に、パブリックキーを使用して SSH 経由で OpenShift Online にクラウドアプリがデプロイされます。OpenShift Online アカウントに関連付けられたすべてのキーは、[アカウントの設定ページ](#)で参照できます。

RHMAP により生成されたパブリックキーが失効したり、OpenShift Online アカウントから削除されたりした場合は、以下のメッセージが表示されることがあります。

DEPLOYMENT LOG

```
Mirroring latest App repo changes to Openshift Repo
{"message":"git fetch -p origin\n#####\n\nUNAUTHORIZED ACCESS IS STRICTLY PROHIBITED AND MAY BE PUNISHABLE #\n# UNDER THE COMPUTER FRAUD AN\nSYSTEM, DISCONNECT NOW. BY #\n# CONTINUING, YOU CONSENT TO YOUR KEYSTROKES AND DATA CONTENT BE\nCONSTITUTES CONSENT TO MONITORING AND AUDITING. #\n#####\nssh://[redacted]13a@testdeveqoe-[redacted].rhcloud.com/~/.git/testdeveqoe.git\nPermission denied (publickey)\nhave the correct access rights\nand the repository exists.\nError: Command failed: Permission denied (publickey,gssapi-keyex\naccess rights\nand the repository exists.\n\nError pushing mirror of repo to ssh://[redacted]13a@testdeveqoe-[redacted].rhcloud.com\nInitialising SSH connection :[redacted]13a@testdeveqoe-[redacted].rhcloud.com
```


この問題を解決するには、RHMAP からログアウトし、再びログインします。RHMAP により、新しいパブリックキーが作成され、デプロイメントに使用されます。

また、デプロイメントの実行前に、アクセストークンを再生成するよう要求されることもあります。

RE-GENERATE KEYPAIR

×

We've detected that our existing access tokens which enable communication between FeedHenry and OpenShift have expired or have been revoked. We will need your username and password to re-generate them. We will only need these credentials once, and don't store them.



OPENSIFT

Username

Password

☐ Dismiss for rest of session

Cancel

Re-Generate Access Tokens

OpenShift Online クレデンシャルを入力し、**Re-Generate Access Tokens** をクリックします。

4.3.3.6. ドメイン

ドメインは OpenShift Online アプリケーション URL の一部 (たとえば、**myApplication-domain.example.com**) であり、多くのアプリケーションは単一のドメインにデプロイできます。ユーザーが定義できるドメインの数は[現在のプラン](#)によって異なります。ドメインの詳細と管理方法については、[OpenShift Online ドキュメンテーション](#)を参照してください。

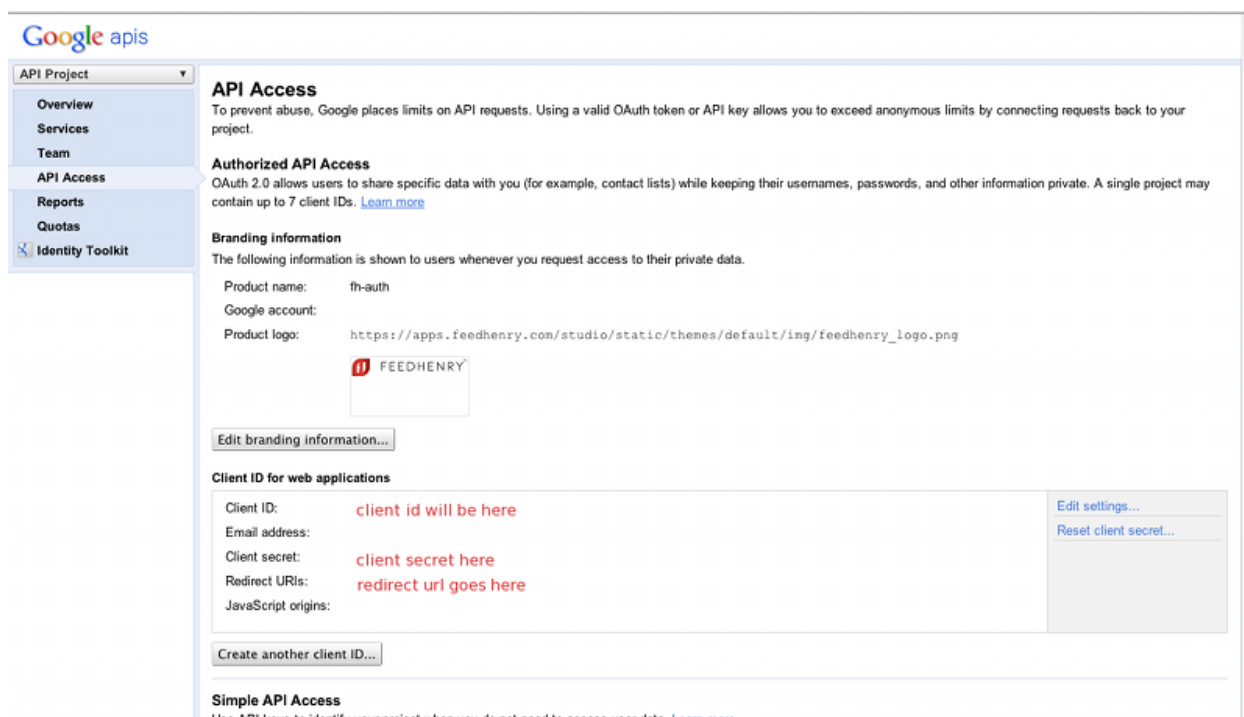
4.4. GOOGLE OAUTH を使用した認証ポリシーのセットアップ

4.4.1. Google OAuth アプリ

アプリのユーザーの認証を Google の OAuth サービスに対して実行するには、最初に複数の手順を実行する必要があります。

- ※ 最初に、[Google API コンソール](#)を使用して Google でアプリをセットアップする必要があります。アプリの作成時に Web アプリケーションを選択します。詳細は、後で更新されることがあるため、それほど重要ではありません。アプリが作成されたら、Google により提供されたクライアント ID とクライアントシークレットをメモしてください。
- ※ クライアント ID は 000000000000.apps.googleusercontent.com のようになります。
- ※ シークレットは数字および数値のハッシュです。

以下の図では、Google のコンソールでこれらの項目が赤で示されています。



- ※ ドメイン管理者権限を持つチームのユーザーとして Studio にログインし、Auth Policies タブをクリックします。次に Create ボタンをクリックして新しいポリシーを作成します。OAuth2 のタイプを選択し、Google によって提供された詳細を入力します。



注記

Studio で **Auth Policies** タブにアクセスするには、ユーザーは **Authorization Policy** に対して **View & Edit** が設定されたチームのメンバーである必要があります。

- ※ Auth Policy 作成ページで提供されたコールバック URL をアプリコンソールの Redirect URI 下の Google アプリに追加します。

4.4.2. 承認

承認パネルには 2 つのオプションがあります。

- ※ ユーザーがプラットフォームに存在することを確認します。
- ※ ユーザーが認証されているかどうかを確認します。

これらいずれかのオプションを選択しない場合は、適切な Google アカウントを持つすべてのユーザーがアプリにアクセスすることを許可すると見なされます。

4.4.2.1. ユーザーがプラットフォームに存在することを確認

このオプションでは、ユーザーが適切な Google アカウントを持ち、ユーザーが Google によって返された ID (例: "someuser@gmail.com") でプラットフォームで登録された場合、ユーザーはアプリケーションにアクセスすることが許可されます。

4.4.2.2. ユーザーが認証されているかどうかを確認

このオプションでは、ユーザーが適切な Google アカウントを持ち、Google によって返されたユーザー ID がこの認証ポリシーに関連付けられている場合、ユーザーはこの認証ポリシーに関連

付けられたアプリケーションを使用することが許可されます。

4.4.3. 認証ポリシーに対するユーザーの追加/削除

既存のユーザーを認証ポリシーに追加するには、check if user is approved for Auth オプションをクリックします。スワップ選択が、利用可能なユーザーが入力された状態で表示されます。これらいずれかのユーザーをポリシーに追加するには、ユーザーを選択し、承認済み列の方向を向いた矢印を押します。ユーザーを削除するには、承認済み列でユーザーを選択し、利用可能な列の方向を向いた矢印をクリックします。

認証ポリシーの設定が完了したら、"Update Policy" ボタンをクリックします。

第5章 データの格納

5.1. データブラウザー

5.1.1. 概要

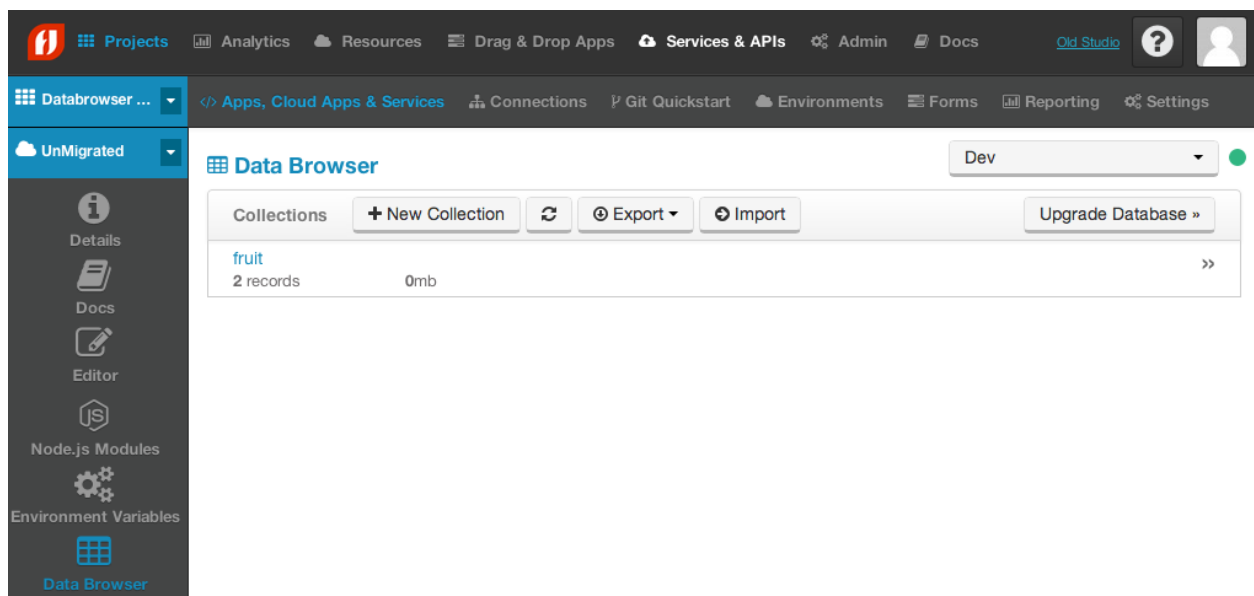
App Studio の Data Browser セクションでは、開発者は以下のことを行えます。

- ※ アプリに関連するデータを視覚的かつ対話的に表示する。
- ※ コレクションを表示、作成、および削除する。
- ※ コレクション内のデータを修正する。

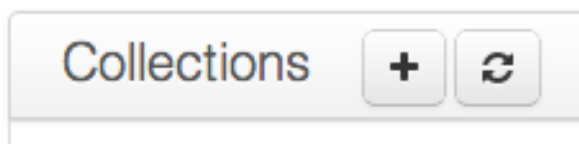
5.1.2. データブラウザーの使用

5.1.2.1. コレクションの表示/追加

Studio のクラウド管理セクションにあるデータブラウザータブを選択すると、アプリに関連付けられているコレクションが表示されます。



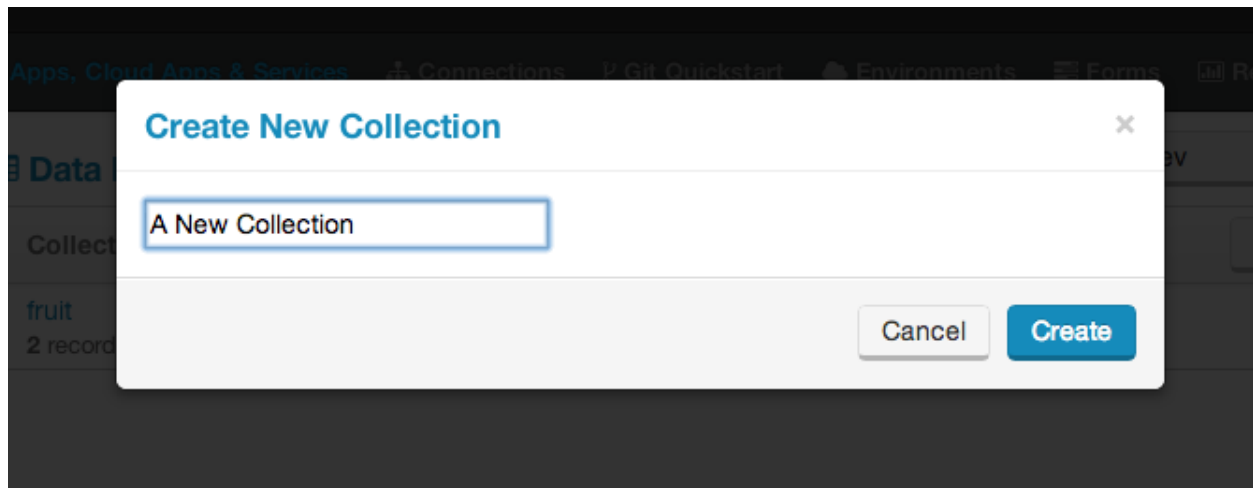
この図では、コレクションリストの最上部に 2 つのコントロールが示されています。



これらのボタンを使用して以下のことができます。

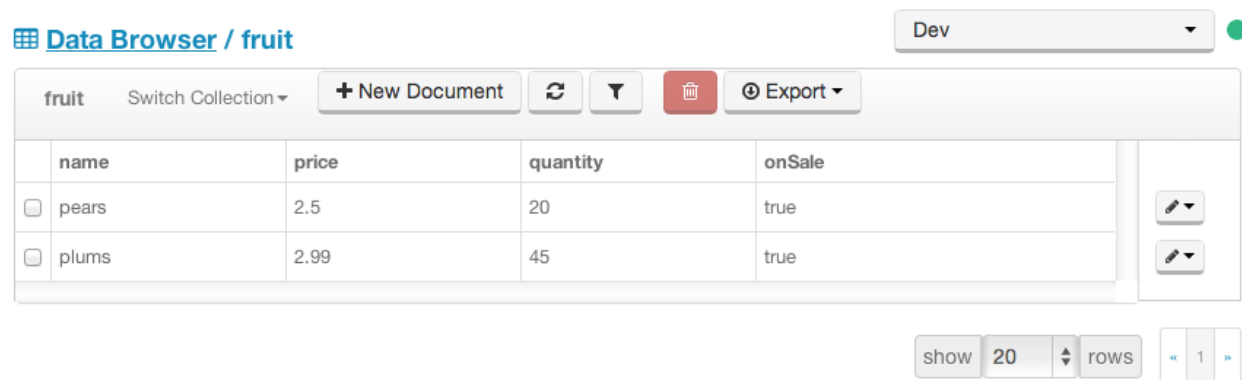
- ※ コレクションを追加します。
- ※ コレクションのリストを更新します。

コレクション追加のボタンをクリックすると、コレクション名の入力を求められます。作成ボタンをクリックしてコレクションを作成します。

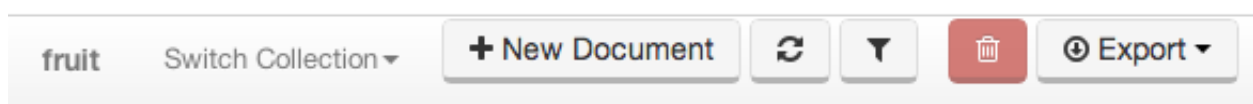


5.1.2.2. コレクション内のデータ表示

コレクションに格納されたデータを表示するには、データブラウザーにリストされたコレクションのいずれかをクリックします。このビューは、コレクションに関連付けられたデータを示しています。



画面の最上部には主要なリスト機能が示されます。



これらのボタンを使用すると、以下のことが行えます。

- ✳ コレクションを切り替えます。このオプションを選択すると、アプリ用のコレクションのリストが表示されます。コレクションをクリックしてそのコレクションのデータをリストします。
- ✳ コレクションにエントリーを追加します。
- ✳ データをインポートおよびエクスポートします (後述)。

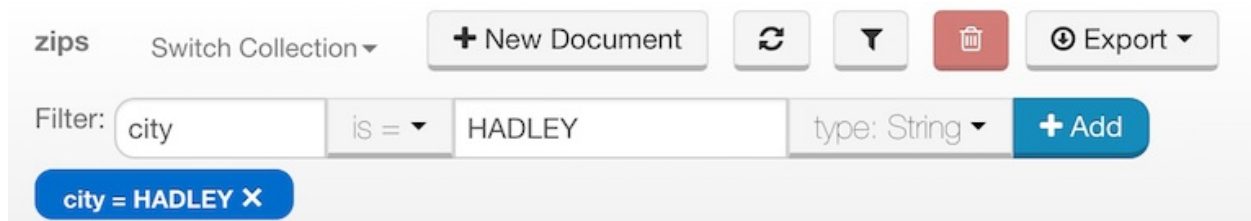
5.1.2.2.1. データの並び替え

特定のフィールドでデータを並び替えるには、一覧上部のフィールド名をクリックします。並び替えは昇順と降順で切り替わります。

5.1.2.2.2. データのフィルタリング

表示されたデータをフィルタリングするには、データブラウザー画面上部にある「フィルター」ボタンをクリックします。このボタンをクリックすると、フィルタリングオプションが表示されます。これらのオプションを使用すると、1つ以上のフィールドで表示されたデータをフィルタリングできます。フィルタリングでは、以下の JSON データ型がサポートされます。

- ※ **String** - テキストベースのフィールドをフィルタリングできます。
- ※ **Number** - 数値をフィルタリングします。
- ※ **Boolean** - true 値と false 値を受け取ります。



注記

'.' 文字を使用して入れ子オブジェクト内部でフィルタリングできます。たとえば、**author.name** をフィルターキーとして使用すると、以下の構造でドキュメントのコレクション内部の **author** オブジェクトの **name** プロパティの値によってフィルターされます。

```
{
  "title": "",
  "author": {
    "name": "John"
  }
}
```

5.1.2.3. データの編集

データブラウザーでは、インラインまたは高度なエディターを使用してデータを編集できます。

- ※ インラインエディターは、コレクション内の簡単なデータの編集に使用します (例: 単一フィールド内でのテキスト変更)。
- ※ 複雑なデータ型の編集には高度なエディターが使用されます。これは、対話型の動的エディターや Raw JSON エディターを使用して行われます。

5.1.2.3.1. インラインエディターを使用した編集

インラインエディターを使用してエンTRIESを編集するには、データエンTRIES右側の Edit オプションを選択してから Edit Inline を選択します。以下の図で示されているように、オプションが緑色のチェックマークと黒矢印のアイコンに変わります。

A New Collection		Switch Collection ▾	+	↺	▼	🗑
FullName	Address					
<input type="checkbox"/> David Ryan	Advanced editor only					
<input type="checkbox"/> Stephen Ryan	[object Object]					

フィールドが複雑すぎてインラインエディターで編集できない場合は、"Advanced Editor Only" というテキストが表示されます。このフィールドは高度なエディターでのみ編集可能です。

エントリーの更新が終わったら、緑色のチェックを選択して変更をデータにコミットするか、黒矢印を選択して変更をキャンセルします。

5.1.2.3.2. 高度なエディターを使用した編集

複雑なデータ型の編集には高度なエディターを使用します (例: フィールドが複数の入れ子フィールドで構成されている場合)。

高度なエディターを開くには、データエントリー右側の Edit オプションを選択し、Advanced Editor を選びます。

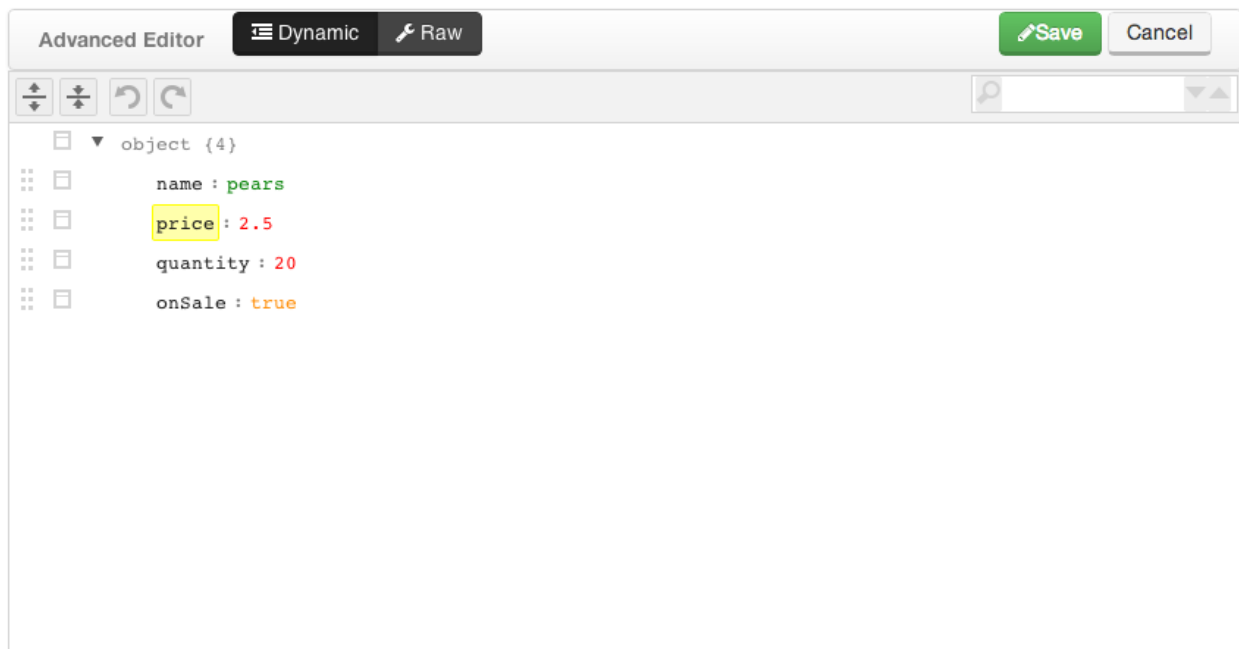


高度なエディターには以下の2つのモードがあります。

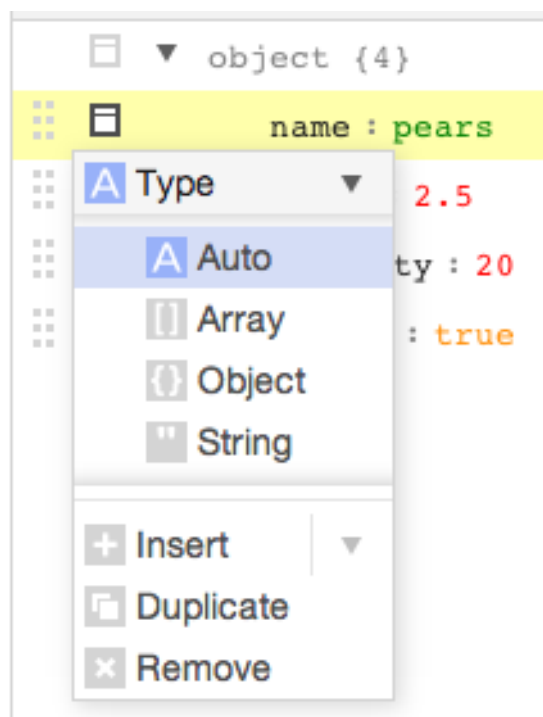
- ✎ フィールドを対話型で追加、編集する動的エディター。
- ✎ JSON 形式でデータを直接編集する Raw JSON エディター。

5.1.2.3.2.1. 動的エディターを使用した編集

動的エディターは、JSON データ用の対話型エディターです。各フィールドの構造化ビューが提供され、複雑なデータ型を追加または編集できます。



アクションメニューは、エントリーの複雑なフィールドを管理するのに必要なすべての機能を提供します。



ここで利用可能なオプションは以下のとおりです。

- ※ **Type:** このオプションでは、フィールドのデータ型をアレイ、JSON オブジェクト、または文字列に変更します。またフィールドを自動的に設定することも可能で、その場合、データ型は入力されたデータから自動的に選択されます。
- ※ **Sort:** このオプションは、複雑なタイプのサブフィールドを昇順または降順で並び替えます。
- ※ **Append:** このオプションは、選択したオブジェクトの後にフィールドを追加します。
- ※ **Insert:** このオプションは、選択したオブジェクトの前にフィールドを追加します。
- ※ **Duplicate:** このオプションを使用すると、選択したオブジェクトがコピーされ、選択したオブジェクトの最後にそのオブジェクトが追加されます。
- ※ **Remove:** エントリーからフィールドを削除します。

5.1.2.3.2.2. Raw JSON エディターを使用した編集

Raw Editor を使用すると、JSON 構文のデータを編集できます。入力されるデータが有効な JSON フォーマットであることを確認してください。JSON データは、フォーマット化されたフォームまたはコンパクトなフォームのいずれかで表示できます。



5.1.3. データのエクスポートとインポート

5.1.3.1. データのエクスポート



注記

データブラウザーインターフェースに組み込まれたエクスポート機能は、テストとレビューのみを目的としています。クラウドアプリまたはサービスからデータコレクションをエクスポートするには、[FHC](#) を使用します。

データは、'Export' ドロップダウンメニューを使用してエクスポートされます。以下の 3 つの形式が利用可能です。

※ JSON

※ CSV

※ BSON (Mongo Dump)

選択した形式のエクスポートボタンをクリックしたら、**.zip** ファイルがダウンロードされます。この内容はユーザーのデータです。

アプリ内に含まれるすべてのコレクションをエクスポートするには、コレクションリスト画面のツールバーの 'Export' を使用します。個々のコレクションのデータをエクスポートするには、コレクションのデータリスト内から 'Export' ドロップダウンを使用します。

インポート用の形式もデータをエクスポートする場合と似ています。各形式のこのスキーマの詳細を以下に記載します。

5.1.3.2. データのインポート



注記

データブラウザーインターフェースに組み込まれたインポート機能は、テストとレビューのみを目的としています。クラウドアプリまたはサービスからデータコレクションをインポートするには、[FHC](#) を使用します。

コレクションリスト画面の 'Import' ボタンをクリックすることにより、データをデータブラウザーにインポートできます。サポートされる形式は以下のとおりです。

※ JSON

※ CSV

※ BSON (Mongo Dump)

※ 上記の 3 つのいずれかの形式を含む ZIP アーカイブ

各ファイルはインポートされるコレクションに対応します。ファイルの名前はデータがインポートされるコレクションの名前に対応します。

コレクションが存在しない場合は、コレクションを作成します。コレクションが存在する場合は、インポートされるドキュメントが既存の内容に追加されます。

5.1.3.2.1. インポート形式

以下では、さまざまな種類のインポートの既定の形式について説明します。各ケースで、フルーツ

の架空のデータセットをインポートします。インポートされたコレクションの名前は **fruit** になります。

各例には、文字列、数値、ブール値などの、インポートでサポートされる各データ型が含まれます。複雑なオブジェクト型はサポートされないことに注意してください。

5.1.3.2.2. JSON のインポート

JSON 形式のインポートは単にドキュメントの JSON アレイです。

fruit.json:

```
[
  {
    "name": "plums",
    "price": 2.99,
    "quantity": 45,
    "onSale": true,
    "_id": "53767254db8fc14837000002"
  },
  {
    "name": "pears",
    "price": 2.5,
    "quantity": 20,
    "onSale": true,
    "_id": "53767254db8fc14837000003"
  }
]
```

5.1.3.2.3. CSV のインポート

CSV をインポートするには、プラットフォームで使用されるセパレーター、区切り記号、および改行の設定に注意してください。

以下にサンプルファイルを示します。

fruit.csv:

```
name,price,quantity,onSale,_id
"plums",2.99,45,true,53767254db8fc14837000002
"pears",2.5,20,true,53767254db8fc14837000003
```

5.1.3.2.4. BSON または MongoDB Dump 出力のインポート

mongodump ツールを実行すると、既存の MongoDB データベースのデータを簡単にエクスポートできます。このツールにより、**dump** という名前のディレクトリーが作成され、データベースが含まれる一連のフォルダーとサブフォルダー、さらにコレクション名が出力されます。これらのコレクションを RMAP データベースにインポートするには、出力された **.bson** ファイルを取得し、直接インポートします。ディレクトリー構造または出力されたメタデータ **.json** ファイルは必要ありません。BSON はバイナリー形式であるため、ここでは例を示しません。代わりに[ファイルをダウンロード](#)することができます。

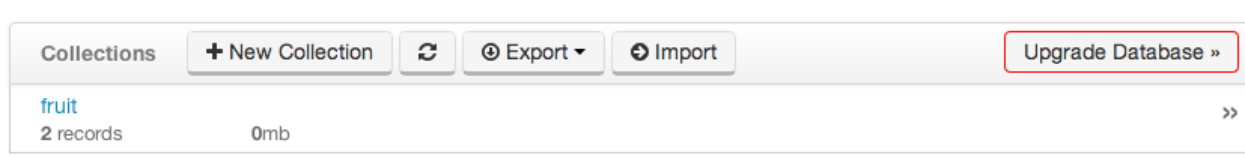
また、任意の mongodb のインストールで提供された **bsondump** ツールを使用して、**.bson** ファイル内部のデータを表示することもできます (**bsondump fruit.bson**)。

```
{ "name" : "plums", "price" : 2.99, "quantity" : 45, "onSale" : true,
  "_id" : ObjectId( "53767254db8fc14837000002" ) }
{ "name" : "pears", "price" : 2.5, "quantity" : 20, "onSale" : true,
  "_id" : ObjectId( "53767254db8fc14837000003" ) }
2 objects found
```

5.1.4. データベースのアップグレード

\$fh.db API で提供されるもの以外のデータベース操作を実行する必要がある場合は、MongoDB ドライバーを直接使用してデータベースにアクセスできます。データベースへの直接アクセスを有効にするには、最初にデータベースをアップグレードし、専用インスタンスに移行する必要があります。

アプリのデータベースをアップグレードするには、データブラウザー画面の右上隅にある **Upgrade Database** ボタンをクリックします。



アップグレード中にプラットフォームにより以下の手順が実行されます。

- ※ アプリが停止されます。
- ※ アプリ用に特別に新規データベースが作成されます。
- ※ アプリに環境変数 **FH_MONGODB_CONNURL** が設定されます。この環境変数には、MongoDB ドライバーに渡すことができるデータベース接続文字列が含まれます。

データベースにすでにデータが含まれる場合は、以下の手順が実行されます。

- ※ データが古いデータベースから新しいデータベースに移行されます。
- ※ すべての処理が正常に実行されると、データは古いデータベースから削除されます。
- ※ データが新しいデータベースで検証されます。

注記: **application.js** ファイルと **package.json** ファイルの内容を更新する必要がある場合があります。この場合は、移行画面で通知されます。

すべてのデータ移行手順が完了したら、アプリを再デプロイする必要があります。

データベースのアップグレードが完了したら、プレフィックス **fhsync_** の付いた新たなコレクションが作成され、同期機能が有効になります。Red Hat では、同期機能を使う予定がない場合でも、これらのコレクションを保持しておくことを推奨しています。

5.2. アプリケーションデータのエクスポート

概要

多くの場合、バックアップを作成したり、既存のデータを使用して他のホスト済みデータベースをテストまたはセットアップしたりするためにクラウドアプリのデータベースからデータをエクスポートすることが必要です。**fhc** を使用すると、クラウドアプリまたはサービスに関連付けられたホスト済みデータベースからすべてのデータをエクスポートできます。

完全なリファレンスについては、**fhc help appdata export** を実行して利用可能なコマンドのリストを参照してください。特定のコマンドのリファレンスについては、**fhc help appdata export <command>** を実行してください。

要件

- ※ RHMAP バージョン 3.11.0 以降
- ※ fhc バージョン 2.9.0 以降

5.2.1. エクスポートデータ形式

クラウドアプリまたはサービスに関連するすべてのコレクションは、単一の **tar** アーカイブにエクスポートされ、圧縮されたバイナリ JSON (BSON) ファイルとして個別のコレクションから構成されます。各 BSON ファイルの名前は元のコレクションの名前と一致します。

例:

```
export.tar
|__ <COLLECTION_1_NAME>.bson.gz
|__ <COLLECTION_2_NAME>.bson.gz
```

BSON ファイルは、標準的な MongoDB バックアップおよび復元ツールと互換性があります。詳細については、[Back Up and Restore with MongoDB Tools](#) を参照してください。

5.2.2. アプリケーションデータのエクスポート

アプリケーションデータのエクスポートプロセスは、主に以下の 3 つの手順から構成されます。

1. [新しいエクスポートジョブを開始する](#)
2. [エクスポートジョブのステータスを問い合わせる](#)
3. [エクスポートされたデータをダウンロードする](#)

5.2.2.1. 新しいエクスポートジョブの開始

新しいエクスポートジョブを開始するには、以下のコマンドを入力します。

```
fhc appdata export start --appId=<APP_ID> --envId=<ENV_ID> [--stopApp=
[y/n>]]
```

- ※ **APP_ID** - クラウドアプリまたはサービスの ID
- ※ **ENV_ID** - アプリが実行されているデプロイ環境の ID

実行後に、データエクスポート中にアプリを停止するかどうかを尋ねるプロンプトが表示されます。

- ※ **n** を選択すると、エクスポートジョブの処理中にクラウドアプリが実行されたままになります。エクスポート中に新しいデータがコレクションのいずれかに追加された場合、データは現在のエクスポートジョブに含まれません。
- ※ **y** を選択すると、エクスポートジョブが開始したときにクラウドアプリが停止します。エクスポートジョブが完了したら、アプリを手動で再起動する必要があります。

コマンド実行後 (たとえば、`fhc` をスクリプト化する場合) にプロンプトを省略する場合は、実行前にコマンドでオプションの **--stopApp** フラグを提供します。

同じ環境の同じアプリに対して別のエクスポートジョブがすでに実行されている場合は、アクションを実行せずに `start` コマンドが終了します。

5.2.2.2. エクスポートジョブのステータスの問い合わせ

ジョブが開始されたら、コマンドラインツールによって、該当するすべてのフィールドに情報が記入された状態で以下の `status` コマンドが出力されます。

```
fhc appdata export status --appId=<APP_ID> --envId=<ENV_ID> --jobId=<JOB_ID> [--interval=<MILLISECONDS>]
```

エクスポートジョブのステータスを問い合わせるには、このコマンドをシェルにコピーアンドペーストします。

コマンドを実行したままにし、定期的にジョブステータスを報告するには、オプションの **--interval** フラグを含め、ジョブのステータスを問い合わせる間隔を指定します。

ジョブが完了したら、`status` コマンドによってジョブステータスが **complete** と返されます。

5.2.2.3. エクスポートデータのダウンロード

ジョブが完了したらエクスポート済みデータをダウンロードするために、以下のように **download** コマンドを実行します。

```
fhc appdata export download --appId=<APP_ID> --envId=<ENV_ID> --jobId=<JOB_ID> --file=<FILENAME>
```

- ※ **APP_ID** - クラウドアプリまたはサービスの ID
- ※ **ENV_ID** - アプリが実行されているデプロイ環境の ID
- ※ **JOB_ID** - エクスポートジョブの ID と対応するエクスポートファイル
- ※ **FILENAME** - エクスポートされたデータを格納するファイルのパス

指定された場所にファイルがすでに存在する場合は、アクションを実行せずに `download` コマンドが終了します。

5.3. アプリケーションデータのインポート

概要

fhc appdata export を使用してアプリケーションデータをエクスポートしたら、**fhc appdata import** を使用してデータをファイルシステムからクラウドアプリまたはサービスに関連するホスト済みデータベースにインポートできます。

完全なリファレンスについては、**fhc help appdata import** を実行して利用可能なコマンドのリストを参照してください。特定のコマンドのリファレンスについては、**fhc help appdata import <command>** を実行してください。

要件

- ※ RMAP バージョン 3.12.0 以降
- ※ fhc バージョン 2.10.0 以降
- ※ ターゲットアプリケーションにはアップグレードされたデータベースが必要です。詳細については、[データベースのアップグレード](#)を参照してください。
- ※ インポートするファイルの形式は **fhc appdata export** で作成されたのと同じである必要があります。詳細については、[エクスポートデータ形式](#)を参照してください。

5.3.1. アプリケーションデータのインポート

アプリケーションデータのインポートプロセスは、主に以下の 2 つの手順から構成されます。

1. [新しいインポートジョブを開始する](#)
2. [インポートジョブのステータスを問い合わせる](#)

5.3.1.1. 新しいインポートジョブの開始

新しいインポートジョブを開始するには、以下のコマンドを入力します。

```
fhc appdata import start --appId=<APP_ID> --envId=<ENV_ID> --path=
<FILE_PATH>
```

- ※ **APP_ID** - クラウドアプリまたはサービスの ID
- ※ **ENV_ID** - アプリが実行されているデプロイ環境の ID
- ※ **FILE_PATH** - インポートするファイルのパス

実行後に、提供されたファイルのアップロードが開始され、アップロードが完了するまでメッセージの出力なしでコマンドが実行されたままになります。アップロードが完了したら、コマンドが終了し、インポートジョブが開始されます。

同じ環境の同じアプリに対して別のインポートジョブがすでに実行されている場合は、アクションを実行せずに start コマンドが終了します。

5.3.1.2. インポートジョブのステータスの問い合わせ

ジョブが開始されたら、コマンドラインツールによって、該当するすべてのフィールドに情報が記入された状態で以下の status コマンドが出力されます。

```
fhc appdata import status --appId=<APP_ID> --envId=<ENV_ID> --jobId=
<JOB_ID> [--interval=<MILLISECONDS>]
```

インポートジョブのステータスを問い合わせるには、このコマンドをシェルにコピーアンドペーストします。

コマンドを実行したままにし、定期的にジョブステータスを報告するには、オプションの **--interval** フラグを含め、ジョブのステータスを問い合わせる間隔を指定します。

ジョブが完了したら、status コマンドによってジョブステータスが **complete** と返されます。

第6章 プロジェクトへのクライアントアプリとクラウドアプリのインポート

概要

既存のクライアントアプリとクラウドアプリをプロジェクトにインポートできます。インポートされる各アプリに対して RHMAPP で git リポジトリが作成されます。

インポートされるアプリは、本書で概説された以下の要件を満たす必要があります。

※ [クライアントアプリの要件](#)

※ [クラウドアプリの要件](#)

要件

- ※ RHMAPP でホストされた Git リポジトリからクローン、プッシュ、またはプルするには、プラットフォームで SSH キーを設定する必要があります。SSH キーをまだ追加していない場合は、[SSH Key Setup](#) を参照してください。

6.1. 空のプロジェクトの作成

本書では、現在 Studio にプロジェクトがないことを前提とします。ベアプロジェクトを作成するには、以下の手順に従ってください。

1. Studio にログインします。
2. **Projects** に移動します。
3. **+ New Project** をクリックします。
4. **Empty Project (空のプロジェクト)** テンプレートを選択し、新しいプロジェクトに名前を付けます。
5. **Create** ボタンをクリックします。
6. プロジェクトが作成されたら、**Finish** をクリックします。

6.2. クライアントアプリのインポート

クライアントアプリを作成する前に、プロジェクトで少なくとも 1 つのクライアントアプリがすでに作成されていることを確認します。既存のアプリをインポートする場合は、[クラウドアプリのインポート](#)を参照してください。

1. ベアプロジェクトで、**Apps, Cloud Apps & Services** に移動します。
2. **Apps** ボックスで **+** をクリックします。
3. **Import Existing App** をクリックします。
4. 作成する **App Type** を選択します。アプリをインポートできるようにするには、[クライアントアプリの要件](#)を参照してください。
5. **Next** をクリックし、インポートされたアプリに名前を付けます。

6. アプリのインポート方法を選択します。

- ※ **パブリック Git リポジトリ**

クライアントアプリが含まれるパブリック Git リポジトリの URL とブランチ名を入力します。デフォルト値は **master** です。

- ※ **Zip ファイル**

コンピューターで、クライアントアプリの内容の単一の ZIP アーカイブを作成します。

```
zip -r app.zip ./ -x *.git*
```

Studio で、アップロードする ZIP ファイルを選択します。

- ※ **Bare Repo**

アプリのコードをプッシュできる空のリポジトリが作成されます。

7. **Import & move on to Integration** をクリックします。クライアントアプリがプロジェクトにインポートされます。

Bare Repo オプションを使用してインポートしたら、**Already have a git repo** をクリックし、表示された手順を実行します。

Zip File または **Bare Repo** オプションを使用してインポートしたら、RHMMap SDK の統合に関する手順が表示されます。

8. **Finished - Take me to My App!** をクリックします。

6.3. クラウドアプリのインポート

1. ベアプロジェクトで、**Apps, Cloud Apps & Services** に移動します。
2. **Cloud Code Apps** ボックスで **+** をクリックします。
3. **Import Existing App** をクリックします。
4. 作成する **App Type** を選択します。アプリをインポートできるようにするには、[クラウドアプリの要件](#)を参照してください。
5. **Next** をクリックし、インポートされたアプリに名前を付けます。
6. 初期デプロイメント環境を選択します。

クラウドアプリは、インポート後すぐに選択された環境にデプロイされます。初期デプロイメントを省略する場合は、**None** を選択します。

7. アプリのインポート方法を選択します。

- ※ **パブリック Git リポジトリ**

クラウドアプリが含まれるパブリック Git リポジトリの URL とブランチ名を入力します。デフォルト値は **master** です。

- ※ **Zip ファイル**

コンピューターで、クラウドアプリの内容の単一の ZIP アーカイブを作成します。

```
zip -r app.zip ./ -x *.git*
```

Studio で、アップロードする ZIP ファイルを選択します。

» Bare Repo

アプリのコードをプッシュできる空のリポジトリが作成されます。

8. **Import & move on to Integration** をクリックします。クラウドアプリがプロジェクトにインポートされます。

Bare Repo オプションを使用してインポートしたら、**Already have a git repo** をクリックし、表示された手順を実行します。

9. **Finished - Take me to My App!** をクリックします。

6.4. クライアントアプリの要件

クライアントアプリをプロジェクトにインポートする前提条件は、作成するクライアントアプリの種類によって異なります。

6.4.1. Cordova Light アプリ

"www" ディレクトリのみが公開された標準的な Cordova 3.x アプリ。これらのアプリでは、標準的な Web テクノロジー (HTML5、CSS、JavaScript) を使用します。ネイティブコードは使用しません。

これらは、プラットフォームの使用方法を学ぶための最も単純なアプリですが、開発者は標準的な Cordova プラグインのみを使用するよう制限されます。これは、**config.json** ファイルを使用して実現されます ([Cordova プラグイン](#))。

要件

ソースファイル (たとえば、HTML、CSS、および JavaScript) は、**www** ディレクトリに存在する必要があります。

6.4.2. Cordova アプリ

標準的な Cordova 3.x モバイルアプリケーション。これらのアプリは Web テクノロジーとネイティブコードの組み合わせで構成されます。基礎となるネイティブプロジェクトと Cordova ライブラリーは開発者に公開され、アプリ (Cordova プラグインとサードパーティー製 SDK を含む) の完全なカスタマイズが可能になります。

通常、これらのアプリのネイティブコードを記述または変更するのにかかる時間は比較的短く、小規模な開発チームのみがネイティブコードを扱う必要があります。開発チームはネイティブコードを開発、設定、および管理し、標準的な Cordova プラグインの手法を使用して追加のプラグインまたは SDK を Web 層に公開する必要があります。

開発チームは、ネイティブコードについてほとんど心配する必要がありません。開発チームは、引き続き純粋な Web 技術を使用して開発を行えますが、ネイティブ層から公開された追加の機能も利用できます。

要件

- » サポートされる最小 PhoneGap バージョン: 3.0

- ※ PhoneGap 3.x の標準的な構造に準拠し、以下のファイル/フォルダーを含む必要があります。

- `www/index.html`
- `www/config.xml`
- `platforms/`
- `plugins/`
- `merges/`
- `.cordova/`

6.4.3. Web App

Node.js + Express Web アプリケーション。これらのアプリは、高度なモバイル Web サイトとデスクトップブラウザ Web ポータルを提供します。この場合、非常に強力な Node.js (Jade などのテンプレートエンジンを使用したサーバーサイドテンプレートなどの機能を含む) を使用して Web アプリ開発を行えます。また、標準的な HTML5、CSS、および JavaScript 向けの静的ファイルの提供もサポートされます。

要件

- ※ 以下のファイルが / にある Node.js アプリである必要があります。

- `package.json`
- `application.js` - Node.js アプリがデプロイされたときに開始されるスクリプト

- ※ 以下のように、アプリが `FH_PORT` 環境変数を介して指定されたポートをリッスンする必要があります。

```
app.listen(process.env.FH_PORT)
```

6.4.4. ネイティブ Android

ネイティブの Android アプリ。

要件

- ※ Ant または Gradle ベースの有効な Android プロジェクト (Eclipse または IDEA に基づいて Android Studio で作成) である必要があります。Ant ベースのプロジェクトでは、/ (プロジェクトルート) に `AndroidManifest.xml` ファイルが含まれる必要があります。
- ※ 最小 Android プラットフォームバージョン: 2.3 (API レベル 9)。
- ※ すべての依存関係は、Android SDK または Maven リポジトリのいずれかで利用可能である必要があります。

6.4.5. ネイティブ iOS

ネイティブ iOS アプリケーション。

要件

- ※ ローカルで適切にビルドおよびコンパイルされる有効な Xcode プロジェクトである必要があります。
- ※ 最小 iOS OS ターゲットバージョン: 7.0。

6.4.6. ネイティブ Windows Phone 8

ネイティブ Windows Phone 8 アプリ。

要件

- ※ Visual Studio 2012 Express で作成された有効な Windows Phone 8 プロジェクトである必要があります。
- ※ Windows Phone SDK バージョンが Windows Phone 8 以上である必要があります。

6.4.7. Xamarin

Xamarin アプリ。

要件

Xamarin Studio または Visual Studio のいずれかで作成された有効な Xamarin ソリューションである必要があります。

6.5. クラウドアプリの要件

- ※ 以下のファイルが / にある Node.js アプリケーションである必要があります。
 - **package.json**
 - **application.js** - Node.js アプリがデプロイされたときに開始されるスクリプト
- ※ 以下のように、アプリが **FH_PORT** 環境変数を介して指定されたポートをリッスンする必要があります。

```
app.listen(process.env.FH_PORT)
```

第7章 古いバージョンの RHMAP からの移行

7.1. V2 から V3 への移行の概要

7.1.1. アプリとプロジェクト

v2 Studio では、アプリはハイブリッドアプリ、Web アプリ、クラウドアプリ、および場合によってはネイティブ iOS または Android アプリすべてを表します。v3 Studio では、アプリはこれらのコンセプトの1つを表します。アプリはクラウドアプリまたはネイティブ iOS アプリのいずれかであり、両方ではありません。v3 Studio では、機能とプラットフォームを分けるためにさまざまなアプリの種類が提供されます (ただし、これらのアプリはプロジェクトでグループ化できます)。

7.1.2. 移行する理由

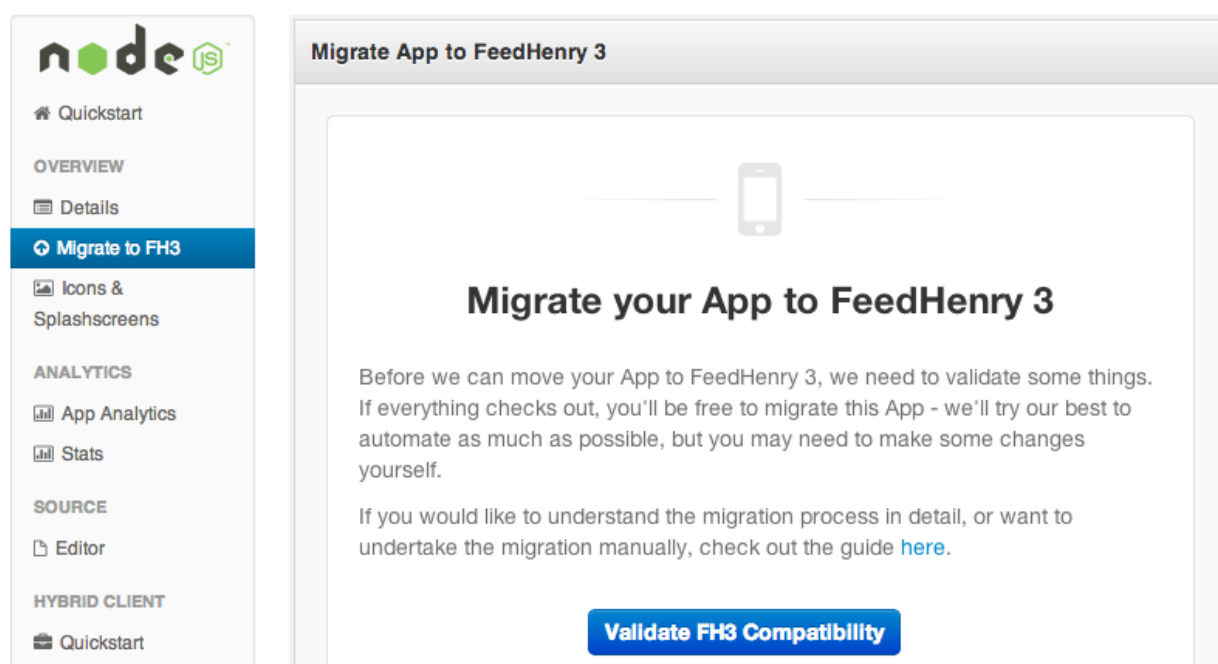
v2 から v3 の Studio に移行する最大の利点は、新しい複数の機能を利用できることです。プロジェクトワークフローにより、開発ライフサイクル全体でプロジェクト内の複数のアプリを簡単に管理できるようになります。v3 の各アプリは、ホストされた git リポジトリによって支持されます。これらのアプリは、[SSH キーをアップロードする](#)だけで使用できます。

v2 Studio は引き続きサポートされますが、v2 Studio では新しいプラットフォームの機能は使用できません。

7.1.3. Studio での自動移行

v2 アプリを個別に v3 プロジェクトに移行するために、v2 Studio では自動移行機能が利用できます。この移行プロセスの詳細を知りたい場合、またはアプリを手動で移行する場合は、[詳細なアプリ移行ガイド](#)を参照してください。

Studio アプリ移行機能はドメイン内のすべてのアプリで利用可能です。ただし、アプリによっては移行が簡単でない場合があります。Studio は、検証チェックを実行して自動移行がアプリに適しているかどうかを判断しようとします。このチェックは非破壊的であり、アプリにはまったく変更が加えられません。この機能を実行して移行しないことを決定することができます。



アプリのサイズに応じて、検証チェックには数秒から最大 1 分かかります。検証チェック後に、移

行を自動的に行えるかどうかを通知する視覚的なレポートが生成されます。このレポートには、アプリで検出されたことと移行で注意する必要がある潜在的な問題 (警告) に関する情報が示されます。作業を円滑に進めるために、移行前にすべての問題を解決する (または問題の解決を計画する) ことが重要です。

✔ Validate FH3 Compatibility

App is compatible with auto migration. See report for details of the migration.

✔ App instances check OK. The existing App will be converted into a Project

✔ Hybrid app detected

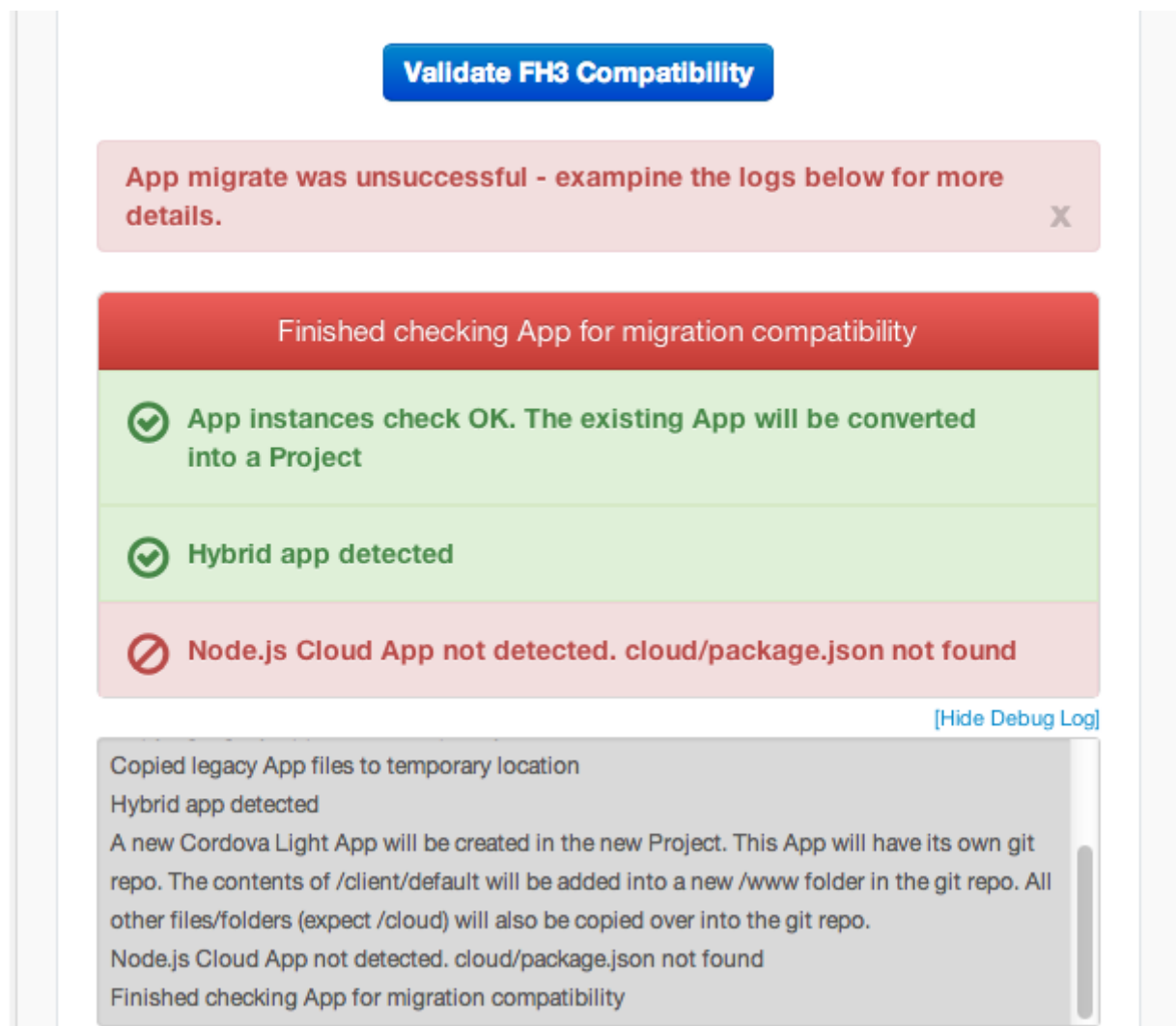
✔ Node.js app type detected.

⚠ Detected fh-webapp feedhenry module in cloud/package.json. It is encouraged to update your Node.js app to use the latest version of fh-mbaas-express to ensure all v3 features are available.

[\[Show Debug Log\]](#)

➡ Migrate this App

検証チェックが何らかの理由で失敗した場合は、エラーが表示されます。可能な場合は、移行のためにアプリを再び検証しようとする前にエラーを修正する必要があります。手動による移行が唯一の選択肢となることもあります。不確かな移行の問題またはサポートが必要な移行の問題は、<https://access.redhat.com> から報告できます。



検証チェックが成功した場合は、**Migrate** ボタンが表示されます。このボタンを押すと、移行が開始され、進捗のフィードバックがリアルタイムで提供されます。移行が成功すると、小さなレポートが作業途中で検出されたすべての潜在的な問題とともに表示されます。強調表示された問題は、v3 Studio を最大に活用するのに重要である場合があるため、完全に理解することをお勧めします。

➡ Migrate this App

Finished migration of legacy App to Project

- ✔ App instances check OK. The existing App will be converted into a Project
- ✔ Hybrid app detected
- ✔ Node.js app type detected.
- ✔ Converted legacy App to Project
- ✔ Created new Repo for Cloud app: git@testing.feedhenry.me:testing/Test-App-3-Test-App-3---Cloud.git
- ✔ Created new client App in Project: git@testing.feedhenry.me:testing/Test-App-3-Test-App-3---Client.git
- ✔ Added fhconfig.json to client app repo for connectivity to cloud app
- ✔ Added v2_to_v3_migrate.md migration guide to Client repo

⚠ Detected fh-webapp feedhenry module in cloud/package.json. It is encouraged to update your Node.js app to use the latest version of fh-mbaas-express to ensure all v3 features are available.

[\[Show Debug Log\]](#)

This app has now been migrated to a new FeedHenry 3 Project. To view your new Project, click the link below:

https://testing.feedhenry.me/#projects/cUbUilB9nTvHhJKKmDNblFe_

Note: This will redirect you to the FeedHenry 3 App Studio - you can switch back by clicking the "Old Studio" link

移行中に問題が発生した場合は、プラットフォームにより、すべての変更がロールバックされ、アプリが移行前の状態に復元されます。繰り返しますが、この移行ステージで発生したすべての問題は、<https://access.redhat.com> で報告できます。

✔ Validate FH3 Compatibility

App migrate was unsuccessful - examine the logs below for more details. ✕

➡ Migrate this App

Finished rolling back migration

- ✔ App instances check OK. The existing App will be converted into a Project
- ✔ Hybrid app detected
- ✔ Node is app type detected

7.1.4. fhc を使用した自動移行

複数のアプリを移行する場合は、移行をスクリプト化すると便利です。これは、**fhc migrate** コマンドを使用して行えます。以下に例を示します。

```
fhc migrate <appid>
```

migrate コマンドにより、最初に検証チェックが実行され、問題がない場合に、続行するか尋ねられます。

ドメイン内のすべてのアプリを自動的に移行する場合は、以下のスクリプトを土台として使用できます。

```
#!/usr/bin/env node
var fh = require('fh-fhc');
var async = require('async');

// Init fhc
fh.fhc.load(function (err) {
  if (err) return console.error(err);
  // Target my domain and login

fh.target(['https://testing.feedhenry.me', 'testing@example.com', 'password'], function(err, ok) {
  if (err) return console.error(err);
  console.log('ok:', ok);
  // List all apps
  fh.apps([], function(err, apps) {
    if (err) return console.error(err);
    console.log('apps:', apps);
    // Iterate over apps, calling migrate for each
    async.mapSeries(apps.list, function(app, cb) {
      // Passing 'silent' here will bypass prompt after validation
      check if check is ok
      fh.migrate([app.id, 'silent'], function(err, data) {
        if (err) return cb(err);
        console.log('data:', data);
        return cb(null, data);
      });
    }, function(err, results) {
      if (err) return console.error(err);
      // all done
      console.log('results:', results);
    });
  });
});
});
```

この例は、[gist](#) (`package.json` を含む) としても利用できます。

7.2. V2 から V3 への移行ガイド

7.2.1. v2 アプリとは

v2 アプリは v2 Studio でのみ確認できるアプリです。Web アプリとしてホストし、デバイスに対してビルドおよびデプロイして、実行されているサーバーサイドコンポーネントを Node.js (または従来の Rhino) 環境にデプロイできます。

これらすべての機能は、一意の ID である GUID (Globally Unique Identifier) を持つ単一のアプリに割り当てられます。単一のコードベースはプラットフォームに格納できます (Studio と API を介してのみアクセスできます)。また、git リポジトリ (パブリックまたはプライベート) に格納して、Studio からリンクすることもできます。上記の各機能を単一のコードベースからサポートすると、厳密なディレクトリ構造が強制されます。この構造は以下のとおりです。

※ **/client** には、以下のように 1 つまたは複数のフォルダー (パッケージ) が含まれます。

- **/default** には、Web アプリまたは Cordova アプリの土台として機能するデフォルトの **index.html** ファイルが含まれます。
- **/<other>** には、特定のプラットフォームで優先されるように設定できる他のファイルが含まれます (たとえば、iOS Cordova アプリをビルドするときに異なる **index.html** を使用)。

※ **/cloud** には以下のものが含まれます。

- **application.js** の土台となるファイルを含む Node.js アプリ
- または **main.js** の土台となるファイルを含む Rhino アプリ

※ **/shared** には、以下のコードで利用できるファイルが含まれます。

- 実行時のクラウドコード
- 配信時またはビルド時のクライアントコード

すべての機能は単一のアプリ guid に割り当てられるため、このアプリにより表されるさまざまな Web アプリまたは Cordova アプリはクラウドコードに密接に結合されます。つまり、このアプリの実行中のクラウドコードとのみ対話します。

7.2.2. v3 アプリとは

v3 アプリは v3 Studio でのみ確認できるアプリです。すべての v3 アプリはプロジェクトの一部です。v3 アプリの機能はアプリの種類 (Cordova アプリ、クラウドアプリ、ネイティブ iOS など) によって異なります。一般的に v2 アプリの個別機能は v3 アプリタイプにマップされます。たとえば、v3 アプリタイプが**クラウドアプリ**の場合は、Node.js コードをクラウド環境にデプロイできます。タイプが**Cordova Light**の場合は、Web コンテンツ (html、css、js) を使用してさまざまなプラットフォーム用のネイティブアプリバイナリーをビルドできます。各アプリタイプは特定の機能を考慮して設計されているため、特にチームで開発を容易に管理できます (懸念事項の切り分けが可能)。



注記

一般的に、v2 アプリは v2 Studio のみで確認できます (v3 Studio の v3 アプリ/プロジェクトと同様)。自動的に移行された v2 アプリは両方の Studio で確認できますが、v3 でのみ変更できます (v2 Studio では **migrated** アプリとしてフラグ付けされます)。

v3 Studio では、確認するアプリタイプで利用できる機能のみが表示されます。たとえば、確認するアプリが**Cordova Light** アプリタイプである場合は、Build オプションが表示されますが、Deploy オプションは表示されません。(新しい) **ネイティブ iOS** アプリを確認する場合は、Build オ

プッシュが表示されますが、Config オプションと Push オプションは表示されません (これらは Cordova と類似したアプリであるため)。

各 v3 アプリには、独自の git リポジトリ (プラットフォームでホストされる) により支持された独自のコードベースがあります。つまり、厳密なディレクトリ構造はありません。たとえば、クラウドアプリの場合は、git リポジトリのルートにファイル package.json と application.js があります。一般的に、ルートにコードを配置することが適切な場合、プラットフォームはビルド時またはデプロイ時にその場所でコードを探します。この唯一の例外は **Cordova Light** アプリです。このアプリでは、すべての Web コンテンツ (HTML、CSS、JS) を /www フォルダーに保持する Cordova の規則に従います。これにより、ビルドスクリプトと他の非本番稼働コンテンツをビルドされるアプリに含めずに保守することが簡単になります。

7.2.3. プロジェクトとは

v3 Studio では、プロジェクトはアプリの論理的なグループです。プロジェクトのクライアントアプリが (接続を介して) 同じプロジェクトのクラウドアプリまたはデプロイ済みアプリに API 呼び出しを行うよう設定できます。プラットフォームの一部の機能 ([フォーム](#)、[接続](#)、[レポート](#) など) はプロジェクトに関連付けられます。また、プロジェクトには [サービス](#) も関連付けられます。サービスは、サードパーティー製システムとの事前定義された統合です。プロジェクトにはコードベースと独自の特別な機能がないため、プロジェクトをビルドまたはデプロイすることはできません。

7.2.4. 新規プロジェクト

v3 Studio では、アプリは独自に存在することはできません。アプリはプロジェクトに属する必要があります。v2 アプリを v3 Studio に移行する場合は、ベアプロジェクトを選択することをお勧めします。これにより、アプリがない状態で新しいプロジェクトが作成されます。

警告

v3 Studio で新しいプロジェクトを作成して手動で v2 アプリを移行する場合、既存のアプリの一意の ID (guid) は v3 で異なります。新しいクライアントアプリを該当するアプリストアに公開する必要があります。これは、v2 Studio での自動移行プロセスの場合は該当しません。

7.2.5. クラウドの新しい v3 アプリ

7.2.5.1. クラウドアプリ

v2 アプリは、サーバーサイド Rhino または Node.js としてデプロイできます。この場合は、**/cloud** のコンテンツが取得され、環境にデプロイされます。v3 でこの機能を取得するには、新しいクラウドアプリを作成します。ただし、Rhino サーバーサイドコードは v3 Studio ではサポートされません。自動的に移行される Rhino コードは限定的にサポートされますが、新しい Rhino アプリは利用できません。

警告

Rhino サーバーサイドコードは v2 アプリから v3 **Rhino** アプリに自動的に移行できませんが、サポートは制限されます。v3 Studio を最大限活用するには、Rhino コードを Node.js に移行することが推奨されます。

7.2.5.2. /cloud フォルダの移行

クラウドアプリで、Studio エディターまたは git リポジトリのローカルクローン (こちらの方が容易) を使用して、ルートフォルダのすべての内容 (隠しファイルを除く) を削除します。/cloud の内容は、ルートフォルダに直接ドロップできます。

7.2.5.3. /shared フォルダの移行

/shared フォルダが v2 アプリに存在する場合、その内容は / に直接ドロップできます。これは、v2 アプリの /shared フォルダの動作に似ています。

7.2.5.4. (オプション) Node.js SDK の変更

ほとんどのレガシー v2 アプリでは、クラウドコードで **fh-nodeapp** または **fh-webapp/fh-api** node.js モジュールが使用されます。/cloud から移行されたコードは引き続きデプロイされ、機能しますが、新しい **fh-mbaas-express** ミドルウェアと **fh-mbaas-api** モジュールを使用することが推奨されます。これらを使用する方法の例はクラウドアプリテンプレートと v3 Studio で利用可能なさまざまなプロジェクトテンプレートで利用できます。**fh-nodeapp** または **fh-webapp/fh-api** から **fh-mbaas-express** への移行に関するガイドは、[ここで参照](#)できます。

7.2.6. クライアント用の新しい v3 アプリ

7.2.6.1. Cordova Light アプリ

v2 アプリはハイブリッドアプリ (Cordova) としてビルドできます。この場合は、/client/default からすべてのコンテンツが取得され、Cordova/www フォルダに配置され (自動的な html ラッピングと v2 パッケージのための追加の手順を実行)、ネイティブアプリバイナリーが生成されます。v3 でこの機能を取得するには、新しい **Cordova Light** アプリを作成します。

7.2.6.2. /client フォルダの移行

/client/default フォルダを取得し、/client 下の他のすべてのフォルダを無視する場合、この部分の移行はそれほど難しくありません。**Cordova Light** アプリで、Studio エディターまたは git リポジトリのローカルクローン (こちらの方が容易) を使用して、/www フォルダのすべての内容 (存在する場合) を削除します。/client/default の内容は、/www フォルダに直接ドロップできます。

7.2.6.3. /shared フォルダの移行

/shared フォルダが v2 アプリに存在する場合、その内容は /www に直接ドロップできます。これは、v2 アプリの /shared フォルダの動作に似ています。

7.2.6.4. Javascript SDK インジェクション

v2 アプリでは、**index.html** の内容が html で自動的にラップされ、RHMAP javascript SDK と初期化 javascript が挿入されます。v2 index.html ファイルのテンプレートは以下のようになります。

```
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
```

```

    <!-- mobile meta tags go here -->
    <script>
      var fh_app_props = {}; // for example, guid, host
      var fh_destination_code = ''; // for example, ios, android, mobile,
embed
    </script>
    <link href="legacy-styles.css" rel="stylesheet"/>
    <script src="legacy-feedhenry-sdk.js"
type="text/javascript"></script>
    <script src="legacy-frameworks.js" type="text/javascript"></script>

  </head>
  <body style="margin:0px;padding:0px">
    <!-- index.html contents go here -->
  </body>
</html>

```

v3 アプリでは、**index.html** の html ラッピング、SDK、またはコードインジェクションがありません。つまり、**Cordova Light** アプリでは、**index.html** で、任意の doctype を定義したり、任意のメタタグを設定したり、任意のスタイルシートまたはスクリプトを好きな場所に配置したりすることができます。v3 アプリでは、**index.html** の内容はまったく変更されません。これは、**index.html** ファイルを v2 アプリから v3 アプリに移行するときに、一部の変更が必要になることを意味します。

最初に、**index.html** の内容は、必要な html 構文 (doctype、html タグ、body タグなど) を含むよう記述する必要があります。次に、Javascript SDK を含める必要があります。つまり、[githubj](#) から最新バージョンの **feedhenry.js** を取得します。このファイルは、**/www** フォルダに追加された状態で以下のように **index.html** に含めることができます。

```
<script src="feedhenry.js"></script>
```

7.2.6.5. fhconfig.json

Javascript SDK はアプリの起動時に自動的に初期化されますが、初期化呼び出しを行う場所を認識している必要があります。これは、設定ファイル **fhconfig.json** を読み取ることにより実行されます。この内容はプロジェクトの **Connections** タブから取得できます。特定の接続に対して **Configure** オプションを使用すると、JSON オブジェクトが示されます。これらの内容は **/www/fhconfig.json** にコピーできます。以下に例を示します。

```

{
  "appid": "00000000000000000000000000000000",
  "appkey": "aaaaaaaaabbbbbbbcccccccccdddddddeeeeeeee",
  "connectiontag": "0.0.1",
  "host": "https://feedhenry.example.com",
  "projectid": "11111111111111111111111111111111"
}

```

このファイルが所定の場所にある場合、アプリは起動時に初期化呼び出しを行います。初期化呼び出しが成功すると、ホスト情報が返され、アプリはすでにプロジェクトにあるクラウドアプリに対して **\$fh.cloud** 呼び出しを行えるようになります。

7.2.6.6. (オプション) レガシー FeedHenry API

RHMAP SDK は FeedHenry v2 よりも大幅に小型化されています。最も大きな違いは \$fh.acc、\$fh.audio、\$fh.ca などのすべての Cordova ラッパー関数が削除されていることです。アプリでこれらの機能が必要な場合は、2つのオプションがあります。

これらのレガシー API は隔離され、別の js ファイル <https://github.com/fheng/fh-js-api-v2> に移動されました。Cordova アプリをビルドする場合は、レガシープラグインを <https://github.com/fheng/fh-cordova-plugins-api> から除外できます。

これらのレガシー API により提供される機能の各プラグインも利用できます。外部ストレージや Webview などの機能は FeedHenry github ページ <https://github.com/feedhenry?query=cordova> から利用できます。

7.2.6.7. (オプション) v2 アプリクライアントパッケージの移行

v2 アプリでは、パッケージはディレクトリ継承の形を取ります。たとえば、特定の css ファイルを iOS ビルド用の別のバージョンで上書きすることが可能になります。この例では、使用するデフォルトの css ファイルは **/client/default/style.css** です。iOS 固有のファイルは **/client/myiospackage/style.css** です。iOS 用にビルドする場合は、**/client/default** 内のすべてのファイルを **/client/myiospackage** 内のファイルで上書きするようパッケージ設定を指定できます。

v3 アプリでは、パッケージを使用しなくなりました。類似の機能を得るには 2つの推奨アプローチがあります。最初のアプローチは、以下の組み合わせです。

- ※ デバイスの解像度に基づいてスタイルのメディアクエリーを使用 (https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries)
- ※ 期待される機能のサポートを追加/開発するためにライブラリーを使用 (たとえば、<http://modernizr.com/>)
- ※ 安全な JavaScript を実現するための機能またはユーザーエージェント検出

2番目のアプローチは **Cordova** アプリにある **merges** フォルダ (**Cordova Light** アプリでは利用不可) を使用することです。merges フォルダは v2 パッケージに似ています。Cordova の公式ドキュメントは、<http://cordova.apache.org/docs/en/3.3.0/guide/cli/index.html#using-merges-to-customize-each-platform> と <https://github.com/apache/cordova-cli/tree/3.3.0-0.1.0#merges> で参照できます。

7.2.6.8. (オプション) v2 アプリの embed/mobile 移行

v2 では、**embed** 宛先を使用して、プラットフォームでクライアントコードを Web アプリとしてホストできます。**mobile** 宛先を使用すると、プラットフォームでクライアントコードをモバイル Web アプリとしてホストできます。**mobile** と **embed** の主な違いは html ラッピングで追加されたメタタグと、アプリの分析/レポートのタグ付け方法です。

v3 Studio では、index.html を完全に制御しながら同じ機能を得るために**基本的な web アプリ**を作成することが推奨されます。組み込みアプリの移行ガイドについては、[ここ](#)をクリックしてください。

7.3. FH-NODEAPP & FH-WEBAPP/FH-API から FH-MBAAS-API/FH-MBAAS-EXPRESS への移行

fh-nodeapp & **fh-webapp/fh-api** は非推奨になり、**fh-mbaas-api/fh-mbaas-express** に置き換えられました。**fh-mbaas-api** と **fh-mbaas-express** には、**fh-nodeapp** および **fh-webapp/fh-api** にある既存のすべての機能と、データブラウザーおよびフォームのサポートなどの新しい追加機能が含まれます。

以下では、**fh-nodeapp** または **fh-webapp/fh-api** から **fh-mbaas-api** と **fh-mbaas-express** に移行する手順について説明します。[Hello World Cloud](#) テンプレートは最小の **fh-mbaas-api/fh-mbaas-express** アプリケーションの優れた例であり、アプリを **fh-mbaas-api/fh-mbaas-express** に移行するときの参考対象として優れています。

7.3.1. package.json の変更

package.json を更新するには、以下の手順に従ってください。

1. **fh-nodeapp** または **fh-webapp/fh-api** を削除し、[The Hello World Package.js file](#) にある最新バージョンの **fh-mbaas-api** に置き換えます。以下に例を示します。

```
"fh-mbaas-api" : "~4.5.0"
```

2. [Express](#) を明示的に含める必要もあります。

```
"express": "~4.0.0",  
"body-parser": "~1.0.2",  
"cors": "~2.2.0"
```

3. 最後に、**npm install** を実行します。

7.3.2. \$fh を fh に置き換えるためにクラウドコードを変更

グローバルの **\$fh** は使用されなくなりました。必要な場合は作成できますが、代わりに **fh** に置き換えることが推奨されます。

1. **lib/main.js** で、**var fh = require('fh-mbaas-api');** を追加します。
2. **\$fh** の箇所を **fh** に置き換えます。
3. **\$fh** を使用するすべてのファイルに対して作業を繰り返します。

7.3.3. application.js の変更

fh-mbaas-api/fh-mbaas-express の場合、**application.js** ファイルは大幅に異なります。既存の **application.js** を [Hello World テンプレートアプリの最新の application.js ファイル](#) に置き換えることが推奨されます。

7.3.4. Grunt の使用

これはオプションですが、使用することを強く推奨します。新しい **FeedHenry** テンプレートは、[ローカル開発](#) とテスト向けのベストプラクティスの支援のために **Grunt** を頻繁に使用します。

アプリで **Grunt** を使用するには、[Hello World Cloud](#) テンプレートの該当する **Grunt** 部分をコピーすることをお勧めします。

1. Gruntfile である <https://github.com/feedhenry-templates/helloworld-cloud/blob/master/Gruntfile.js> を独自のアプリのルートにコピーします。
2. 'devDependencies' をこの package.json から独自の package.json である <https://github.com/feedhenry-templates/helloworld-cloud/blob/master/package.json> にコピーします。コピーが完了したら、`npm install` を実行します。

7.4. 組み込みアプリから基本的な WEB アプリへの移行

7.4.1. 概要

FeedHenry v2 では、**Embed** 宛先を使用して、プラットフォームでクライアントコードを Web アプリとしてホストできました。これは既存のクライアントアプリの単純な HTML ベース Web ポータルを作成する場合に便利でした。

RHMAP Studio では、組み込みアプリが非推奨になり、基本的な Web アプリに置き換えられました。基本的な Web アプリは、アプリを組み込む以前の FeedHenry v2 オプションと同じ機能を提供します。また、ライフサイクル管理のために index.html、アプリの実行方法と実行場所、および個別の Git リポジトリを完全に制御することもできます。

アプリを RHMAP プロジェクトに移行した後で、組み込みアプリは引き続き実行されますが、既存の組み込みアプリを再デプロイすることはできなくなります。本書では、RHMAP プロジェクトで既存の組み込みアプリを新しい基本的な Web アプリに移行する手順について説明します。

7.4.2. 移行

移行を開始するために、既存の FeedHenry v2 アプリを RHMAP プロジェクトにすでに移行したことを前提とします。まだ移行していない場合は、[ここ](#)にあるガイドを参照してください。

7.4.2.1. 移行されたハイブリッドアプリのクローン

- ※ 最初に、移行されたハイブリッドアプリの Git リポジトリをローカルでクローンする必要があります。これを行うには、**Apps, Cloud Apps & Services** 画面で **Cordova Light** アプリのリポジトリ URL の隣にある **Copy** ボタンを押します。ターミナルにおいて好きなフォルダーで以下のコマンドを入力します。

```
git clone __Cordova_Light_Git_URL__
```

- ※ これにより、Cordova アプリのソースがローカルでクローンされます。このフォルダーを後で使用するので (古い組み込みアプリのソースコードが含まれるため)、ターミナルを開いたままにし、App Studio に切り替えます。

7.4.2.2. 新しい基本的な Web アプリの作成

次に、使用する新しい基本的な Web アプリまたは移行される組み込みアプリを作成する必要があります。これを行うには、**Apps, Cloud Apps & Services** 画面で **Apps** 領域の **+** をクリックします。Add a new App Client 画面で **Create New App** を選択し、**Blank Basic Web App** テンプレート (Web アプリの新しい名前を入力) と **Create new App** を選択します。これにより、空の基本的な Web アプリが作成されます。

FeedHenry v2 では、組み込みアプリのソースコードはハイブリッドおよびクラウドコードと同じ場所に存在していました (**client/default/** のコードがコピーおよびデプロイされていました)。組み込みパッケージを使用している場合は、ソースが **client/** フォルダ下の別のフォルダーに

存在することがあります。FeedHenry v2 アプリを移行した場合、これらのフォルダーは若干変更されています。**www/** はソースフォルダーであり、以前に存在したすべてのパッケージフォルダーは **legacy_packages/** フォルダーに現れます。

7.4.2.3. 新しい基本的な Web アプリへの組み込みアプリのコピー

- ※ 新しい基本的な Web アプリを作成したら、**Details** 画面が表示されます。新しい基本的な Web アプリのソースをクローンし、以前にクローンした移行済み Cordova Light アプリから既存の組み込みアプリのソースをコピーします。**Git SSH Clone URL** の隣にある **Copy** ボタンを押し、このリポジトリをローカルでクローンします。以下に例を示します。

```
git clone git@yourdomain.feedhenry.me:yourdomain/your-new-basic-web-app.git
```

- ※ この時点で、新しい基本的な Web アプリと移行済み Cordova Light アプリの両方のローカルコピーが存在します。古い組み込みアプリのソースを新しい基本的な Web アプリにコピーします。
- ※ **Basic Web App** で、**www/css/** フォルダーと **index.html** ファイルを削除できます。**feedhenry.js** (JS SDK) と **fhconfig.json** (JS SDK の設定ファイル) は保持します。
- ※ **Cordova Light App** で、古い組み込みアプリに使用したパッケージを特定する必要があります。通常は、古いアプリの内容である **client/default** フォルダーが使用されますが、**client/your_embed_package** などの組み込みアプリ用の別のパッケージを使用した可能性もあります。組み込みアプリのソースコードを含むフォルダーの内容を新しい **Basic Web App** にコピーします。
- ※ 古い組み込みアプリのファイルがコピーされたら、変更をプッシュします。**Basic Web App** で以下のようなコマンドを入力します。

```
git commit -am "Copied embed app"
git push origin master
```

- ※ 最後に、**Basic Web App** の **Deploy** 領域に移動し、デプロイします。基本的な Web アプリの新しい URL は **Details** 画面に表示されます。

7.5. ロールからチームへの移行

チームは、プラットフォームの機能領域へのアクセスを制御する新しい強力なシステムです。

本書では、ロールベースの制限と新しいパーミッションベースの制限を比較します。各ロールは、同等の制限を作成するのに必要なパーミッションと比較されます。

7.5.1. パーミッションの追加

ロールベースのパーミッションシステムでは、ロールは以下の 3 つのレベルのユーザーに割り当てられます。

- ※ **このドメイン**: ロールは、現在のホスト名のみによって表されるドメインに対してアクティブです。
- ※ **すべてのドメイン**: ロールは、顧客が所有するすべてのドメインに対してアクティブです。

- ※ **すべての顧客:** ロールは、現在のリセラーが管理するすべての顧客が所有するすべてのドメインに対してアクティブです。

7.5.2. アナリティクス (analytics)

- ※ プラットフォームのアナリティクスセクションにアクセスできる。
- ※ プロジェクトの使用率を監視できる。

チームに割り当てられる以下のパーミッションにより、アナリティクスへの同等のアクセスが許可されます。

- ※ ドメイン
 - アナリティクス (表示)
 - プロジェクト (表示)
 - 接続 (アクセスなし)
 - クラウドリソース (アクセスなし)

7.5.3. フォームエディター (formseditor)

- ※ フォームやテーマを作成できる。
- ※ フォームプロジェクトを作成できる。
- ※ フォーム、テーマ、および自分のグループに関連付けられているプロジェクトを編集できる。
- ※ フォームとテーマを該当するグループのプロジェクトに関連付けることができる。

チームに割り当てられる以下のパーミッションにより、フォームへの同等のアクセスが許可されます。

- ※ ドメイン
 - プロジェクト (表示 & 編集)
 - Drag & Drop App (表示 & 編集)
- ※ フォーム
 - 送信 (アクセスなし)



注記

グループの概念は Drag & Drop App から削除され、**Form** ビジネスオブジェクトに置き換えられました。

7.5.4. フォーム管理者 (formsadmin)

- ※ フォームやテーマを作成できる。
- ※ フォームプロジェクトを作成できる。
- ※ グループを作成できる。

- ※ ユーザーをグループに割り当てることができる。
- ※ 特定のドメインで作成されたすべてのフォーム & テーマを表示、アクセス、管理できる。
- ※ ドメイン上の全プロジェクトからの提出を表示できる。
- ※ ドメイン上のプロジェクトからの提出を編集できる。

チームに割り当てられる以下のパーミッションにより、フォームへの同等のアクセスが許可されます。

- ※ ドメイン
 - プロジェクト (表示 & 編集)
 - Drag & Drop App (表示 & 編集)



注記

グループの概念は Drag & Drop App から削除され、**Form** ビジネスオブジェクトに置き換えられました。

7.5.5. 提出ビューワ (submissionsviewer)

- ※ グループ内のプロジェクトからの提出を見ることができる。

チームに割り当てられる以下のパーミッションにより、フォームへの同等のアクセスが許可されます。

- ※ プロジェクト (表示)
 - クラウドリソース (アクセスなし)
 - 接続 (アクセスなし)
 - アナリティクス (アクセスなし)
 - クライアントアプリ (アクセスなし)
 - クラウドアプリ (アクセスなし)
- ※ Drag & Drop App (表示)
 - テーマ (アクセスなし)

7.5.6. 提出エディター (submissionseditor)

- ※ グループ内のプロジェクトからの提出を見ることができる。
- ※ グループ内のプロジェクトからの提出を編集することができる。

チームに割り当てられる以下のパーミッションにより、フォームへの同等のアクセスが許可されます。

- ※ プロジェクト (表示)
 - クラウドリソース (アクセスなし)

- 接続 (アクセスなし)
- アナリティクス (アクセスなし)
- クライアントアプリ (アクセスなし)
- クラウドアプリ (アクセスなし)
- ※ Drag & Drop App (表示 & 編集)
 - フォーム (表示)
- ※ 送信 (表示 & 編集)
 - テーマ (アクセスなし)

7.5.7. 開発管理者 (devadmin)

- ※ すべてのタイプのプロジェクトを作成、管理できる。
- ※ 利用可能なプラットフォーム用にアプリバイナリーを構築できる。
- ※ プライベートキーや証明書などの開発リソースをアップロードできる。
- ※ デプロイメントターゲットの作成、編集、およびメンテナンスができる。
- ※ 特定ドメイン上のプロジェクト & アプリすべてを表示、それらにアクセスできる。
- ※ 特定ドメイン上のデプロイメントターゲットすべてを表示、編集できる。
- ※ サービス (表示)
 - ソースコード (アクセスなし)
 - アナリティクス (アクセスなし)
 - 環境変数 (アクセスなし)
 - データブラウザー (アクセスなし)
 - クラウドデプロイメント (アクセスなし)
 - 統計 (アクセスなし)
 - 通知 (アクセスなし)
 - ログ (アクセスなし)
 - エンドポイント (アクセスなし)
- ※ プロジェクト (表示 & 編集)
- ※ Drag & Drop App (表示)
 - フォーム (表示)
- ※ 送信 (アクセスなし)

7.5.8. 開発者 (dev)

- ※ すべてのタイプのプロジェクトを作成、管理できる。

- ※ 利用可能なプラットフォーム用にアプリバイナリーを構築できる。
- ※ プライベートキーや証明書などの開発リソースをアップロードできる。
- ※ デプロイメントターゲットの作成、編集、およびメンテナンスができる。
- ※ パブリックサービスをプロジェクトに追加できる。

チームに割り当てられる以下のパーミッションにより、同等のアクセスが許可されます。

- ※ サービス (表示)
 - ソースコード (アクセスなし)
 - アナリティクス (アクセスなし)
 - 環境変数 (アクセスなし)
 - データブラウザー (アクセスなし)
 - クラウドデプロイメント (アクセスなし)
 - 統計 (アクセスなし)
 - 通知 (アクセスなし)
 - ログ (アクセスなし)
 - エンドポイント (アクセスなし)
- ※ プロジェクト (表示 & 編集)
- ※ Drag & Drop App (表示)
 - フォーム (表示)
- ※ 送信 (アクセスなし)

7.5.9. サービス管理者 (serviceadmin)

- ※ mBaaS サービスのプロビジョニングができる。
- ※ mBaaS サービスの管理ができる。

チームに割り当てられる以下のパーミッションにより、同等のアクセスが許可されます。

- ※ ドメイン
 - サービス (表示 & 編集)

7.5.10. ドメイン管理者 (portaladmin)

- ※ 認証ポリシーを作成および編集できる。
- ※ ドメイン向けのアプリストアの更新ができる。
- ※ アプリストアのアプリを作成および管理できる。
- ※ アプリストアのグループを作成および管理できる。
- ※ アプリストアのデバイスを管理できる。

- ※ アプリストアのログを表示できる。
- ※ 管理者ページの「モバイルアプリの管理」セクションにアクセスできる。
- ※ ユーザーの追加と削除ができる。
- ※ ユーザーにロールを割り当てることができる。

チームに割り当てられる以下のパーミッションにより、同等のアクセスが許可されます。

- ※ ドメイン
 - 認証ポリシー (表示 & 編集)
 - App Store (表示 & 編集)
 - 管理者 (表示 & 編集)

7.5.11. 顧客管理者 (customeradmin)

- ※ この顧客に属するドメイン内のユーザーを管理できる。

チームに割り当てられる以下のパーミッションにより、同等のアクセスが許可されます。

- ※ 顧客
 - 管理者 (表示 & 編集)

7.5.12. リセラー管理者 (reselleradmin)

- ※ 顧客に属するドメイン内のユーザーを管理できる。
- ※ ユーザーに顧客管理者のロールを割り当てることができる。

チームに割り当てられる以下のパーミッションにより、同等のアクセスが許可されます。

- ※ リセラー
 - 管理者 (表示 & 編集)

7.5.13. 管理者 (admin)

- ※ プラットフォーム内ですべてのアクションを実行できる。
- ※ ユーザーにリセラー管理者と顧客管理者のロールを割り当てることができる。

チームに割り当てられる以下のパーミッションにより、同等のアクセスが許可されます。

- ※ リセラー
 - 管理者 (表示 & 編集)