2023 年 12 月 1 日

HeartCore Robo 管理ポータルセットアップおよび API ガイド

作成：HeartCore Robo サポートチーム

ハートコア株式会社

目次

管理ポータルのセットアップ

HeartCore Robo 管理ポータルソリューションは、サーバ側とクライアント側の両方のコンポーネントで構成されています。**サーバ側のコンポーネントは次のとおりです。**

- **API サーバ**
- **MySQL データベース**
- **コンソール/ダッシュボード UI**
- **ワークフローエンジン**

これらのコンポーネントは、最初は Docker コンテナとして配信されます。このインストール手順については「Docker の構成手順」ページを参照してください。

クライアント側のコンポーネントは次のとおりです。

- **HeartCore Robo**
- **管理ポータルエージェント**

これらのコンポーネントはどちらもクロスプラットフォーム zip パッケージとして提供される Java アプリであり、Robot の場合は Windows および Mac OS のネイティブ インストーラーでもあります。

HeartCore Robo の PC 導入手順については、別途「HeartCore Robo のシステム要件とインストール手順」を参照してください。

エージェントのセットアップ手順については、「管理ポータル エージェントのセットアップ」ページを参照してください。

コンテナ化された管理ポータルサーバの構成 - ステップバイステップ

導入

この項では、安全でパブリックにアクセス可能な 管理ポータルインスタンスをセットアップするために必要なすべての手順を説明します。

管理ポータル サーバは、次の 4 つのコア コンポーネントで構成されます。

- **MySQL データベース**
- **API サーバ**
- **管理ポータル ポータルアプリケーション**
- **ワークフローエンジンデーモンの 1 個以上のインスタンス**

さらに、API サーバは、適切な製品ライセンスがインストールされているライセンス サーバにアクセスする必要があります。

最後に、インストールがまったく簡単で単一のデバイスにローカライズされていない限り、通常は、URL 経由でサービスへのアクセスを管理するために何らかの形式のプロキシ サーバが必要になります。

始める前に

管理ポータルサーバをインストールする前に、次の 2 つの重要な決定を行う必要があります。

① **MySQL データベースサーバの場所** - 企業内で利用可能な既存の MySQL サーバはありますか? それとも、管理ポータルのコアコンポーネント内にあるコンテナ化された DB を使用しますか?

② **Robo ライセンスサーバの場所** - すでに構築された Robo のライセンスサーバにアクセスできますか? それともライセンスサーバのインスタンスが新たに必要ですか?

これらの重要なポイントへの回答によって、インストール ワークフローでどのパスを設定・実行するかが決定されます。

管理ポータルコンテナをホストするにはサーバが必要です。このドキュメントでは主に、AWS や VPS 環境等の仮想化ホストへ管理ポータルをインストールする場合に関する手順が構成されています。

管理ポータルのインストールと実行に関して、このテキストには次の想定でのセッティング方法が記載されています。

- ホストサーバは Linux で実行され、ここでは Ubuntu Server 20.04.4 LTS を元に解説しています。コマンド等の扱いに注意すれば、おそらく他の Linux ディストリビューションでも動作するでしょう。
- 管理ポータルサービスは、ドメイン名 myrapserver.myenterprise.com の CA 署名付き証明書を使用して、HTTPS/SSL のみを使用してアクセスされます。
- サービスはホスト myrapserver 上で公開され、リクエストは URL myrapserver.myenterprise.com 経由でサーバホストにルーティングされます。
- 管理ポータルサーバの URL は、内部ネットワーク上の企業内からだけでなく外部からも有効になっています（このルーティングがどのように行われるかについてのセマンティクス（解説）は、このドキュメントの対応範囲外です)。

ホストサーバ

ホストサーバのサイジング（構成容量）は正確に確定できない - システムにどれだけのデータをロードするかは利用者ごとに変化するので確定はできません。最初は 120 GB のディスク領域と最小 8 GB の RAM を備えた VM から始めるのが適切です。別のサーバでホストされているデータベースをリンクする場合は、必要なメモリはもっと少なく済みます。一般的には、ここで提案されているセットアップより安全なため、そのような構成をお勧めしています。この項では説明を簡単にするため、シングルインスタンス（ローカル）内に設定される MySQL DB サーバを使っています。

この項では Ubuntu Server 20.04 LTS を実行している Linux VM を使用しています。これはデスクトップ GUI 環境のないヘッドレスサーバです。ホストサーバには、

- **インターネットアクセス**
- **サーバにリモートでログインできるよう構成された SSH サーバ**
- **ファイアウォールで開いているポート 443**

が必要です。

相互の依存関係

管理ポータルを Docker コンテナでホスティングするため、Docker Engine のセットアップが主な依存関係になります。**以下のインストールと設定が必要です：**

Docker Engine Version 20.10.20 以降 - それ以前のバージョンでは動作しない可能性が高いです。(詳細はこちら: https://docs.docker.com/engine/install/ubuntu)。

Docker Compose - **最近の Docker Engine に同梱されている docker-compose プラグインは使用しないでください。**すでにインストールされている場合は問題ありませんが、Docker スクリプトを動作させるには、別途 docker-compose スクリプトをインストールする必要があります。

※ Docker 関連を便利に利用できるよう、管理ポータルの構成に使用するユーザーアカウントが docker グループのメンバーになるよう設定しておくと、sudo コマンドなしで docker コンテナを実行できるようになります。

クライアントがインターネット（場合によってはイントラネット）から管理ポータルにアクセスするには、DNS で解決可能なドメインが必要で、また、ホストサーバにルーティングする必要があります。これを設定する方法はたくさんありますが、ルーティングについての詳細はこのドキュメントの範囲外です。ご自身で情報を検索するなどして設定作業を行ってください。

上記要件は、ネットワーク内部および外部の両方から Web ブラウザから行われたリクエストをホストサーバにルーティングする必要があることを示しています。

設定を開始する

※ OS 初期設定や SSH 接続の詳細等についてはお客様側でご確認ください（本テキスト解説の対象外）。

まず、使用するユーザーアカウントで SSH を使用してホストサーバにログインし、作業に適したフォルダを作成します。下記コマンドを入力してフォルダを作成し、作成したフォルダへ移動してください。

```
rapadmin@myrapserver:~$ mkdir rap
rapadmin@myrapserver:~$ cd rap
rapadmin@myrapserver:~/rap$
```

※「rapadmin@myrapserver:~$」などの先頭から $ までの部分はシェルに記載されたユーザー・（カレント）ディレクトリ情報なので $ より後ろが入力する内容になります。

ここで、使用する管理ポータル導入パックをダウンロードする必要があります。RAP 3.1 が現在のリリースなので、それを使用します。

```
rapadmin@myrapserver:~/rap$ wget https://downloads.t-plan.com/releases/rap/rap3/hc-rap-docker-3.1.0.6028.tar.gz
```

ファイルを解凍して内容を確認します。

```
rapadmin@myrapserver:~/rap$ tar -xzf hc-rap-docker-3.1.0.6027.tar.gz
rapadmin@myrapserver:~/rap$ ls
docker  proxy  README.md  startdemo  startinstance  hc-rap-docker-3.1.0.6027.tar.gz
rapadmin@myrapserver:~/rap$
```

上記の状態になっていれば、管理ポータルを設定する準備が整ったことになります。この先の設定作業を進めてください。

インスタンス化

管理ポータルサーバは、いくつかのコンテナ化されたサービスから構成されています。これらセットまたはスタックそれぞれを「インスタンス」と呼び、管理ポータルホストサーバでインスタンスをセットアップするには、

- **シングルインスタンス**
- **マルチインスタンス**

2 つの方法で設定できます。

シングルインスタンス

これは、管理ポータルサーバをセットアップする最も簡単な方法です。単一のインスタンスを実行する場合、インスタンス内の様々なサービスへのルーティングを処理するための内部リバースプロキシを含む、必要なすべてのコンテナが単一の Docker 構成内にカプセル化されています。このシナリオでは、それぞれのアプリケーションにアクセスするため HTTPS アクセスできるユーザーURL とそのホームディレクトリ下に 2 つのエンドポイント「/」（ポータル UI 用）と「/rap」（管理ポータル API 用）が必要となります。

例： https://rap.hcrobo.co.jp/および https://rap.hcrobo.co.jp/rap など

マルチインスタンス

単一ホスト上で複数の管理ポータルインスタンスを実行することは完全に可能です。このような構造から恩恵を受けるユースケースは数多くあります。たとえば、ビジネスグループを分離したり、異なるバージョンの管理ポータルサーバを並列実行できます。

この場合、各インスタンスはそれぞれ独自の docker.env ファイルを持ち、startinstance スクリプトを個別に呼び出して起動します。

シングルインスタンス管理ポータルホストを実行する場合との主な違いは、受信リクエストのプロキシ処理です。各インスタンスは同じ URL 上で個別の URL セグメントに存在し、プロキシサーバは受信リクエストを正しいインスタンスにルーティングできる必要があります。このため、マルチインスタンス化する場合は各インスタンスの Docker 構成の外部に存在する外部プロキシサーバを使用する必要があります。

マルチインスタンス化は高度なトピックです。

このチュートリアルではシングルインスタンスサーバセットアップを解説しています。

管理ポータルを構成する

「docker.env」の編集

管理ポータルサーバを構成する最初の段階は、/rap/docker ディレクトリ内の環境ファイル「docker.env」を編集することです。

```
rapadmin@myrapserver:~/rap$ cd docker
rapadmin@myrapserver:~/rap/docker$ nano docker.env
```

これで、nano エディタで「docker.env」が開きます。別のエディタを使用したい場合でも問題ありません。このチュートリアルでは nano を使用します。

データベース接続

コンテナ化された DB（シングルインスタンス）

DB をコンテナ・クラスタの一部として実行する場合、管理ポータルやライセンスサーバの接続文字列を設定する必要はなく、インストールされた DB はすぐに初期化されます。

（外部またはローカル内の）リモート DB

リモートデータベースを使用している場合は、以下の必要があります。

- 「docker.env」に接続文字列の一方または両方を設定
- 少なくとも、接続先のデータベースに対して有効になるように RAPServerDB 接続文字列を変更
- MySQL で DB スキーマを手動作成し接続文字列で使用される権限を MySQL ユーザーアカウントに割当

設定するとアプリケーションが初回実行時にデータベースを初期化します。

コンテナ化されたライセンスサーバを使用しており、それを外部 DB で使用したい場合は、データベースに対して有効になるように LICServerDB 接続文字列を設定する必要もあります。必要に応じて、管理ポータルデータベースとライセンスサーバデータベースの両方で単一の MySQL スキーマを使用することもできます。

RAP URL

エディタで開いた docker.env ファイルを”RAPAPIURL=”で始まる行までスクロールし、これを編集して正しい URL を設定します。ここではサンプルとして myrapserver.myenterprise.com を使用していますが、ホストとドメインに合わせ適宜変換して記入してください。

次に、RAPSERVERPATH を「/rap」に変更します。これにより、API サーバが URL：https://myrapserver.myenterprise.com/rap でホストされるようになります。このセクションの他のすべてはそのままにしておきます。

```
RAPAPIURL='https://myrapserver.myenterprise.com/rap/api/v3/rap'
RAPSERVERPATH='/rap'
# RAPBASE forms base href in UI; DO NOT PROVIDE LEADING / BUT ALWAYS PROVIDE
TRAILING /. To leave default '/', just keep this blank.
RAPBASE=''
RAPSERVERPORT=5000
RAPUIPORT=8181
```

***これらの設定を変更する必要があるのはどのような場合ですか?**

標準では、公開される 2 つの管理ポータルサービス（メインのポータル UI と API サーバ）は、それぞれ URL セグメント「/」と「/rap」で公開されます。これらを別の URL セグメントで公開する必要がある場合は、ここで変更します。たとえば、「apps」セグメント内で管理ポータルを公開したい場合は次のように変更します。

```
RAPAPIURL='https://myrapserver.myenterprise.com/apps/rap/api/v3/rap'
RAPSERVERPATH='/apps/rap'
RAPBASE='apps/'
```

この変更により、ポータル UI は "https://myrapserver.myenterprise.com/apps "でアクセスされ、API サーバは "https://myrapserver.myenterprise.com/apps/rap "で公開されるようになります。

ライセンスサーバーの設定

管理ポータルと HeartCore Robo を使用するには、2 つの製品ライセンスが必要です。これらのライセンスは、ライセンスサーバでホストする必要があります。この例では、Docker コンテナクラスタの内部ライセンスサーバを使用します。

docker.env ファイルを'LICSERVERPATH='まで下にスクロールして「/licserver」に変更します。これにより、内部ライセンスサーバが URL： https://myrapserver.myenterprise.com/licserver でホストされるようになります。

次に、'LICSERVERURL='の設定を編集します。これは API サーバが使用する内部アドレスであり、コンテナの外部に公開されることはないためここでベース URL を変更する必要はありません。ただし、管理ポータルサーバライセンスの UID を、T-Plan Ltd.から提供されたものに変更する必要があります。

UID は、URL の?lid= 部分の後にあるすべてです。

※ シングルインスタンスでこれを設定する場合は、インスタンスを起動しライセンスサーバを提供された XML ファイルで有効にした上で、RAP ライセンスを設定後その UID を Docker.env に設定してから startinstance スクリプトを再実行する必要があります。

ソフトウェアのバージョン

管理ポータルの公式リリース (現在 3.1) を使用している場合は、Docker タグセクションで参照されている Docker イメージを変更する必要はありません。(2023 年 10 月 27 日現在)

※最新版にリプレースする場合は適宜変更する必要があります。

プロキシサーバーの設定

SSL リクエストをサポートし同じ URL で両方の管理ポータルサービスを公開するために、nginx リバースプロキシを使用します。シングルインスタンスサーバを構築しているため、独自の内部プロキシサーバを使用してインスタンスを実行できます。

構成

プロキシサーバを構成するには、まず docker フォルダに移動し proxy.conf ファイルを編集します。

```
rapadmin@myrapserver:~/rap$ cd docker
rapadmin@myrapserver:~/rap/docker$ nano proxy.conf
```

サーバ名を「example.com」から自分のドメインに変更します。

```
server {
    server_name myenterprise.com;

    listen 80;
    listen [::]:80;
    listen 443 ssl;
    listen [::]:443 ssl;

    ssl_certificate /etc/ssl/private/fullchain.pem;
    ssl_certificate_key /etc/ssl/private/privkey.pem;

    include /etc/nginx/conf.d/locations/*.conf;

}
```

これを保存し、エディタを終了します。

次に、ライセンスサーバルートを location フォルダにコピーする必要があります。内部ライセンスサーバを使用していない場合は、これを location ディレクトリ（デフォルト）から除外する必要がありますが、内部ライセンスサーバを使用している場合は、そこにコピーしておく必要があります。

```
rapadmin@myrapserver:~/rap/docker$ cp licserver.conf ./locations/.
rapadmin@myrapserver:~/rap/docker$
```

次に、SSL 証明書を certs ディレクトリにコピーします。ここでサーバには 2 つのファイルが必要です。

privkey.pem は証明書の秘密キーであり、fullchain.pem は完全な CA チェーンを含む SSL 証明書である必要があります。

```
rapadmin@myrapserver:~/rap/docker$ ls certs
fullchain.pem   privkey.pem
rapadmin@myrapserver:~/rap/docker$
```

これでプロキシサーバの構成が完了し、サーバを起動できる状態になります。

最初の実行と初期化

各サーバの起動

サーバを起動するには、次のように startinstance スクリプトを実行します。

```
rapadmin@myrapserver:~/rap$ ./startinstance -p -f docker/docker.env -n myrap
```

これにより、前に編集した docker.env ファイルを使用したサーバインスタンスが起動され、そのインスタンスは「myrap」と名付けられています。インスタンスには好きな名前を付けることができますが、これが意味を持つのは、マルチインスタンスでサーバを実行していてコンテナを区別する必要がある場合のみです。

これで、T-Plan Dockerhub から必要なイメージがすべて取り込まれ各サービスが起動します。

```
Creating myrap_dbserver_1      ... done
Creating myrap_wfe_1           ... done
Creating myrap_licserver_1     ... done
Creating myrap_portal_1        ... done
Creating myrap_apiserver_1     ... done
Creating myrap_ngproxy_1       ... done
```

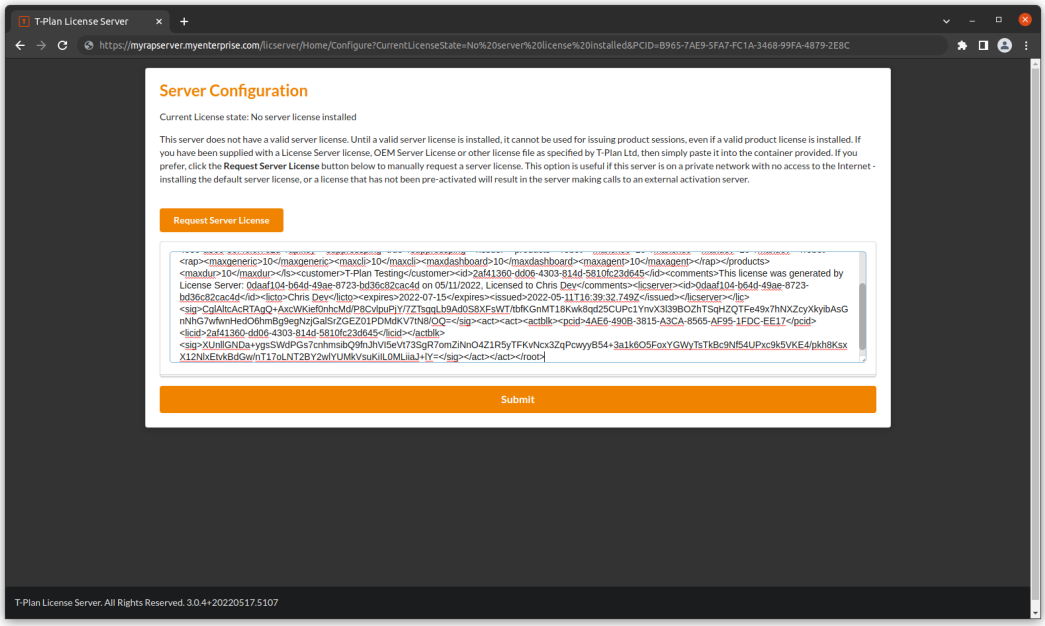
＊上記は順調に起動した場合のメッセージ

ライセンスサーバの初期化

サーバインスタンスが起動したので、ライセンスサーバから始めてアプリを初期化できます。

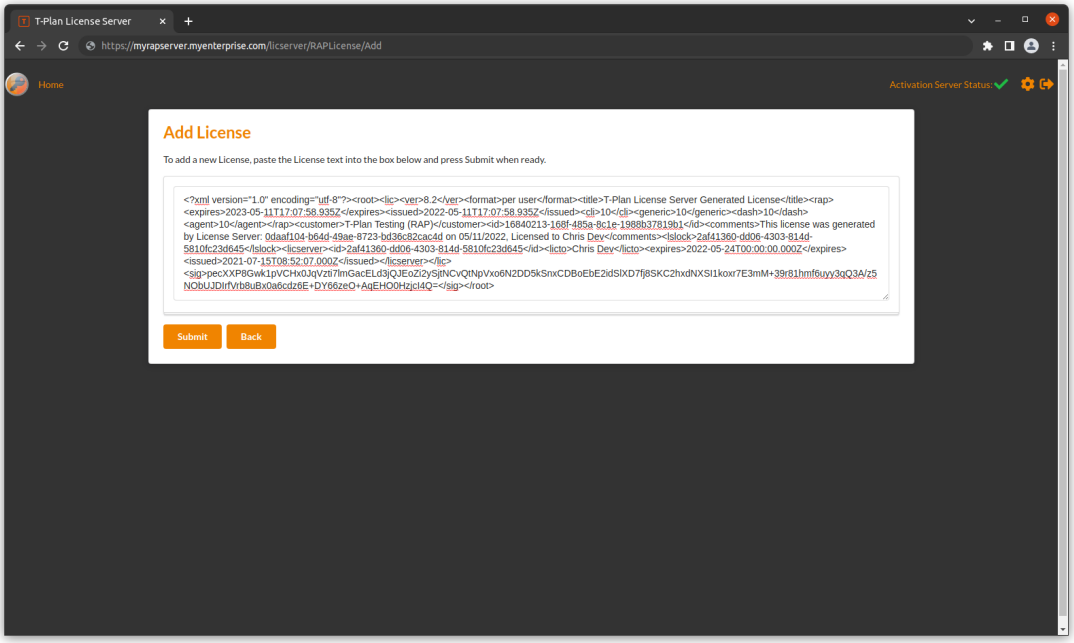
Web ブラウザを開き、ライセンスサーバ URL に移動します。この例では、https://myrapserver.myenterprise.com/licserver にあります。まずはシステムを初期化する必要があります。この手順を完了するには、「Initialize」ボタンを押すだけです。

次に、サーバライセンス（提供された XML ファイル内テキスト全文）を指定されたスペースに貼り付けます。

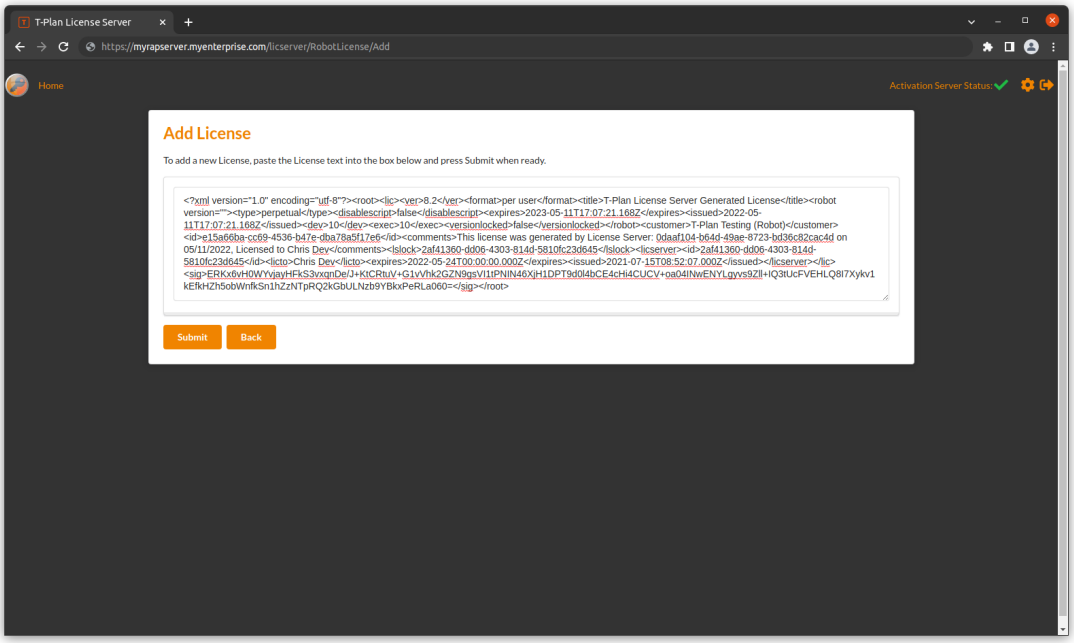


サーバへのログインに使用する資格情報が表示されます。（**資格情報は必ずメモしてください。ログインページへの遷移後は以後表示されません**）ここでページを遷移してログインし、[RAP Licenses] タブを開きます。

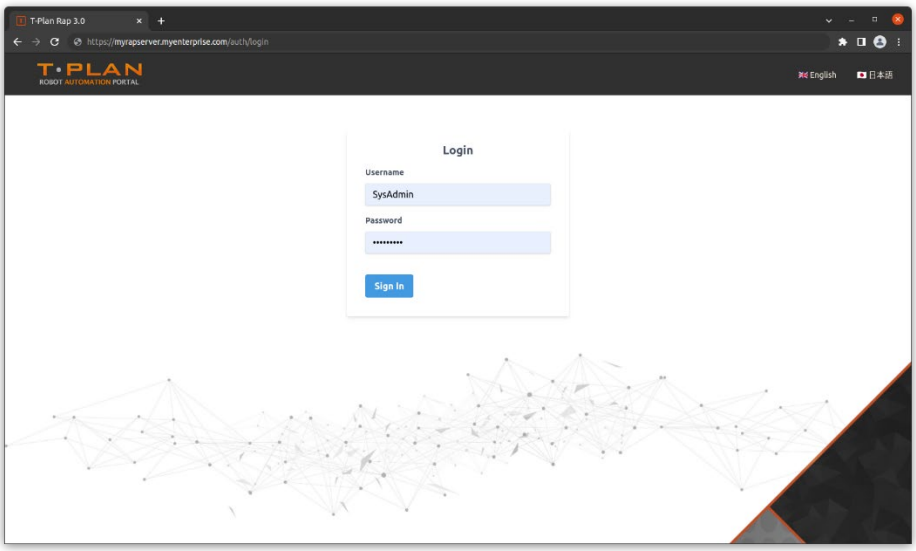
[Add License]をクリックし、管理ポータルライセンスを指定されたスペースに貼り付けます。



これを送信してから、[Robot Licenses] タブを開き、[**Add License**]をクリックして、ロボットライセンスを所定のスペースに貼り付けます。



これでライセンスサーバの設定は完了です。これで管理ポータルにログインできるようになります。ポータル URL に移動します。この例では、 https://myrapserver.myenterprise.com/にあります。



デフォルトの管理者資格情報（ユーザー: SysAdmin、パスワード: sysadmin1）を入力すると、ログインして準備完了です。

Docker に関する解説事項

導入設定

管理ポータル用の RAP 3 Docker サーバは、デプロイ（展開）オプションに応じて、2〜4 個のコンテナで構成される Docker compose スタックです。これにより、RAP 3 サーバ側のコンポーネントがデプロイされます。

サーバコンポーネントは、以下のように構成されています：

- **管理ポータル UI**
- **API サーバー**
- **MySQL データベース**
- **ワークフローエンジン**

さらに、組み込みのリバース プロキシを使用してスタックを構成し、単一の URL を介してリクエストを API サーバおよび管理ポータル UI コンテナにルーティングできます。

MySQL データベースをスタックに含めることはオプションで、スタック外部にある既存データベースに接続できます。

ライセンスサーバについても同様で、内部（スタック）のインスタンスまたは管理ポータル docker スタックの外部にある別のライセンスサーバインスタンスを使用できます。

システム要件

Windows での Docker 用に RAP 3（管理ポータル）を構成することも可能ですが、導入は Linux 環境で行われる可能性が高く、このドキュメントはその想定で記載されています。スタックはインストールの前提条件である次のコンポーネントを使用してテストされました。

- **Docker: 20.10.2+**
- **Docker-compose: 1.26.0+**

※「docker-compose-plugin」を使用せず、代わりに「sudo apt install docker-compose」を使用して docker compose を個別にインストールしてください。

インストールされたインスタンスは新しいバージョンでは機能するはずですが記載より古いバージョンでは機能しない可能性があります。

コンテンツ

提供された tar ファイルのルートディレクトリには、docker スタックを実行するためのスタートスクリプトが含まれています。 また、パッケージには 2 つのディレクトリが含まれています：

docker - さまざまな構成オプションの .yml スクリプトが含まれています。このディレクトリ内のキーファイルは「docker.env」です。これには、すべての構成設定が含まれています。

proxy - 実行中のサーバにリクエストをルーティングするためのリバースプロキシを実行するための docker ファイルと構成が含まれています。これは、外部ネットワーク（インターネットを含む）からサーバにリクエストをルーティングするとき、またはマルチテナントサーバをセットアップするときに最も頻繁に必要になります。

起動する

管理ポータルサーバを起動するには 2 つの方法があります。

デモモード

Linux ターミナルでスクリプト "**startdemo** "を実行すると、デモ目的に役立つ完全な自己完結型サーバが起動します。管理ポータルのこのインスタンスは "localhost "からしかアクセスできないため、リバースプロキシの設定などの設定なしにヘッドレスサーバで実行するには適していません。このモードは、主にデモ用のスクラッチ環境を迅速に作成・実行することを目的としており、ローカルのデスクトップセッションからアクセスします。

インスタンスモード

Linux ターミナルでスクリプト「**startinstance**」を実行すると、管理ポータルサーバがインスタンスモードで起動します。これは、プライマリ構成ファイル（docker.env）の設定とコマンドラインオプションの設定に従い動作します。これらの設定は、主に実行時の 2 つの側面に影響します。

データベースの場所

管理ポータルインスタンスは MySQL データベースを使用します。管理ポータルの docker stack 内で完全にカプセル化されたデータベースを使用するか、ネットワーク上の別の場所にあるデータベースに接続するかを選択できます。

ローカルデータベースサーバでデータベースをホストする場合は、管理ポータルサービスを開始する前にデータベースを作成し、いくつかの権限を割り当てる必要があります。通常は、MySQL コマンドラインクライアント、または MySQL Workbench などのデスクトップクライアントを使用して MySQL セッションにログインし、次のコマンドを実行します。

```
create database MyRAPDatabase;
create user MyRAPUser identified by 'SomePassword';
grant all on MyRAPDatabase.* to 'MyRAPUser'@'networklocation';
flush privileges;
```

ここで「networklocation」は、そのユーザーが MySQL にアクセスできる場所を制限するパターンです。管理ポータルの場合、ローカルネットワークのみにアクセスを制限するには、一般的に「192.168.1.%」のようなサブネットパターンを使用することをお勧めしています。

ネットワークへのアクセシビリティ

管理ポータル用の RAP Docker Suite は API サーバとクライアント UI (「Portal」)の 2 サービスを公開しています。Web ブラウザから「localhost」経由でこれらに直接アクセスすることも可能ですが、その場合スイートを実行している同じホストからのみのサービス接続となります。ネットワーク上の他の場所からスイートにアクセスする必要がある場合は、スイートをプロキシモードで実行することをお勧めします。この場合、スイートにはプロキシサーバが含まれているので、単一の URL エンドポイントを介して、リモートのブラウザからのリクエストをサーバ内の関連サービスに転送します。

構成・設定について

テキストエディタで/docker/docker.env ファイルを開き、env ファイルに含まれるヒントに従って、アプリケーションの実行方法に応じて変数を設定します。docker.env には次の設定が含まれています。

- ファイル/データストレージ
- プロキシ構成
- データベース接続
- アクセス URL とポート設定
- 管理ポータルライセンスとライセンスサーバの詳細
- 電子メール通知の SMTP 設定
- Docker コンテナリファレンス

データの永続性について

Docker コンテナの外部にあるファイルの保存場所を決定する環境変数が **LOCALFILES** と **SITEFILES** の 2 つあります。標準ではどちらも同じ値 ~/.rap に設定されています。

LOCALFILES：これは、内部 DB サーバによって MySQL データベース制御ファイルの場所として使用されます。これにより、Docker コンテナが再起動された場合でも、データベースに保存されたデータが確実に保持されます。この変数は、プロキシサーバが必要に応じて SSL 証明書を見つけることができると期待される場所でもあります。

SITEFILES：これは、API サーバによって管理ポータルワークアイテムのペイロードと実行レポートを保存するために使用されます。

SSL の設定について

SSL をサポートする管理ポータルサーバをセットアップするには、SSL エンドポイントをホストするリバースプロキシが含まれた管理ポータルサーバ構成を使用する必要があります。

証明書は正しいドメインに対して設定する必要があり、この URL へのリクエストは RAP 3 Docker パッケージを実行しているサーバに正しくルーティングされる必要があります。自己署名証明書も使用できますが、ローカル署名機関を信頼するように設定されていないブラウザではエラーが発生する可能性があります。

docker パッケージには、SSL エンドポイントをサポートする nginx 用のサンプル proxy.conf ファイルが付属しています。SSL 証明書とキーは、変数 LOCALFILES が指す場所にあるフォルダー「nginx/certs」に配置する必要があります（これはインスタンスのメインの docker.env ファイルで設定します - 上記と同様)。

proxy.conf ファイルのサーバ名が、証明書が有効なサーバ名と一致していることを確認する必要があります。

サービスを起動する

必要に応じ構成を設定したら、「startinstance」スクリプトを使用してスイートを開始します。

「startdemo」スクリプトは、構成ファイルに設定されている多くのオプションをバイパスしてローカルホストからのみアクセスでき、カプセル化されたデータベースを使用するサンドボックス環境を開始します（上記を参照)。

インスタンスの開始

startinstance スクリプトのコマンドラインは以下の通りです。

startinstance [-p] [-r] -f <environmentfile> -p <instancename>

-p :

内部プロキシサーバを使用してインスタンスを作成する場合は、-p 引数を含めます。この引数を省略した場合のデフォルトでは、プロキシなしでインスタンスが開始されます。

-r :

この Docker スタック内のインスタンスではなく、リモート MySQL サーバに接続するインスタンスを起動する場合は-r 引数を含めます。

-f <environmentfile> :

使用する環境ファイルのパスとファイル名。各インスタンスには独自の環境ファイルが個別に必要です。デフォルトでは、docker ディレクトリ内の docker.env が使用されます。

-p <instancename> :

管理ポータルサーバの各インスタンスは、自己完結型環境で 2～5 個のコンテナ（構成に応じて）間で実行されます。サーバでの複数インスタンスの管理を容易にするために、各インスタンスに名前が必要です。

各サービスへのアクセスについて

スクリプトを実行すると、ポータル UI と API の両方に設定した URL 経由でアクセスできるようになります。

URL 内の構成は、スイートが実行されているモードと接続元によって異なります。次に例を示します。

Mode	Access URLs
localhost	UI: http://localhost:8181 , API: http://localhost:5000
local with proxy	UI: http://yourhost , API: http://yourhost/rap
external with proxy	UI: https://yourhost.example.com , API: https://yourhost.example.com/rap

サーバ構成

このセクションでは、サーバまたはサーバにデプロイされたファイルに対して行う必要がある変更について説明します。オリジナルのコンテンツは、 https://angular.io/guide/deployment#server-configuration でご覧いただけます。

ルーティングされたアプリは index.html にフォールバックしなければいけない

Angular アプリは、シンプルな静的 HTML サーバでサービスを提供するのに最適な候補です。Angular はクライアント側でアプリケーションページを動的に作成するため、サーバ側のエンジンは必要ありません。

アプリが Angular ルータを使用する場合、サーバに存在しないファイルを要求された際にアプリケーションのホストページ（index.html）を返すようにサーバを設定する必要があります。

ルーティングされたアプリケーションは「ディープリンク」をサポートする必要があります。ディープリンクとは、アプリ内のコンポーネントへのパスを指定する URL のことです。

例えば、http://www.mysite.com/heroes/42 は、id: 42 のヒーローを表示するヒーロー詳細ページへのディープリンクです。

実行中のクライアントからユーザーがこの URL に移動しても問題はありません。Angular ルータが URL を解釈し、そのページとヒーローにルーティングします。しかし、電子メール内のリンクをクリックしたり、ブラウザのアドレスバーにリンクを入力したり、ヒーローの詳細ページを表示しているときに単にブラウザを更新したりするなど、これらのアクションはすべて、実行中のアプリケーションの外部で、ブラウザ自体によって処理されます。ブラウザは、ルータをバイパスしてサーバに対してその URL を直接リクエストします。

静的サーバは、http://www.mysite.com/のリクエストを受けると、index.html を返すのが普通です。しかし、代わりに index.html を返すように設定されていない限り、http://www.mysite.com/heroes/42 を拒否し、404 - Not Found エラーを返します。

フォールバック構成の例

すべてのサーバに有効な唯一の設定というものはありません。以下のセクションでは、最も一般的なサーバの設定について説明します。このリストは決して網羅的なものではありませんが、良い出発点となるはずです。

Apache

Apache：.htaccess ファイルに、図のようにリライトルールを追加します（https://ngmilk.rocks/2015/03/09/angularjs-html5-mode-or-pretty-urls-on-apache-using-htaccess/）：

```
RewriteEngine On
# If an existing asset or directory is requested go to it as it is
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -f [OR]
RewriteCond %{DOCUMENT_ROOT}%{REQUEST_URI} -d
RewriteRule ^ - [L]

# If the requested resource doesn't exist, use index.html
RewriteRule ^ /index.html
```

Nginx

Nginx: Front Controller Pattern Web Apps で説明されている try_files を使用し、index.html を提供するように修正します：

```
try_files $uri $uri/ /index.html;
```

Ruby

Ruby: (sinatra)を使って、サーバ server.rb を設定する基本的な Ruby ファイルとともに Ruby サーバーを作成する：

```
require 'sinatra'

# Folder structure
# .
# -- server.rb
# -- public
#       |-- dist
#             |-- index.html

get '/' do
      folderDir = settings.public_folder + '/dist'    # ng build output folder
      send_file File.join(folderDir, 'index.html')
end
```

IIS

IIS: ここに示すものと同様の web.config にリライトルールを追加する。

*サーバに IIS URL Rewrite をインストールすることを忘れないでください。

```
<system.webServer>
  <rewrite>
    <rules>
      <rule name="Angular Routes" stopProcessing="true">
        <match url=".*" />
        <conditions logicalGrouping="MatchAll">
          <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" />
          <add input="{REQUEST_FILENAME}" matchType="IsDirectory" negate="true" />
        </conditions>
        <action type="Rewrite" url="/index.html" />
      </rule>
    </rules>
  </rewrite>
</system.webServer>
```

web.config の例:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <defaultDocument>
            <files>
                <clear />
                <add value="index.html" />
                <add value="Default.htm" />
                <add value="Default.asp" />
                <add value="index.htm" />
                <add value="iisstart.htm" />
                <add value="default.aspx" />
            </files>
        </defaultDocument>
    <rewrite>
        <rules>
          <rule name="Angular Routes" stopProcessing="true">
            <match url=".*" />
            <conditions logicalGrouping="MatchAll">
              <add input="{REQUEST_FILENAME}" matchType="IsFile" negate="true" />
              <add input="{REQUEST_FILENAME}" matchType="IsDirectory" negate="true" />
            </conditions>
            <action type="Rewrite" url="/index.html" />
          </rule>
        </rules>
    </rewrite>
    </system.webServer>
</configuration>
```

管理ポータルエージェントのセットアップ

管理ポータルエージェントは、自動化タスクの主要なドライバです。これはワーカーとして機能し、アクション可能なジョブの管理ポータル実行キューを継続的に監視します。ジョブが選択されると、その実行が完了し、その結果が管理ポータルに報告されるまで、そのジョブはビジーであると報告されます。

詳細とインストール手順については、別紙「管理ポータルユーザーガイド」の「管理ポータルエージェントについて」の項を参照してください。

API ガイド

API 使用方法

ログインする

管理ポータル API は 2 つの URL セグメント上に存在します。1 つはセキュリティで保護されていないルートセグメント「/」、もう 1 つはトークンを使用してのみアクセスできるセキュリティで保護されたセグメント「/api/v3/rap」です。

トークンを取得するには、/Tokens エンドポイント(非セキュア)に GET リクエストを送信します。このリクエストには、使用するユーザーアカウントのユーザー名とパスワードを含む基本認証（Basic Auth）ヘッダが含まれている必要があります（Basic Auth の場合、「Username:Password」の形式で文字列を作成し、Base64 エンコードします)。また、クエリパラメータ「lictype」を指定して、このユーザーが要求するライセンスの種類を指定する必要があります。「lictype」パラメータには、使用されている管理ポータルライセンスのライセンスタイプに対応する 4 つの値（generic、Agent、cli 、または dumpboard）のいずれかを指定します。

例：

```
curl --location --request GET 'https://myserver:5000/Tokens?lictype=generic' --header
'Authorization: Basic BASE64ENCODEDAUTH'
```

ここで、BASE64ENCODEDAUTH は文字列 Username:Password を Base64 エンコードしたものです。

成功した応答には、セキュアな「/api/v3/rap」セグメント上のすべてのクエリで使用するトークンを含む、セッションが必要とするすべてのものが含まれています。

```
{
    "status": "success",
    "data": {
        "serverVersion": "3.1.1.5692",
        "serverVersionInfo": "3.1.1+20220627.5692",
        "databaseVersion": "rapdb-3.1",
        "engineVersion": "0.0.1",
        "session": {
            "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmIxdWVfbmFtZSI6IlN5c0FkbWluIiwibmFtZWlkIjoi
NDkxN2ZmNGYtNDJjZC0xMWU3LWE0ZDYtMDAxNTVkMDk5NDFkIiwiaHR0cDovL3NjaGVtYXMue
G1sc29hcC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvc2lkIjoiZjQzNGY0YTgtMTRjNy
0xMWVkLTkzMTQtMDAxNTVkMDE2NzBhIiwibHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93
cy8yMDA4LzA2L2lkZW50aXR5L2NsYWltcy9leHBpcmF0aW9uIjoiNjM3OTUzMDg4NTQwMDAwMD
AwIiwibmJmIjoxNjU5NzA4NDU1LCJleHAiOjE2NTk3MTIwNTUsImlhdCI6MTY1OTcwODQ1NSwiaX
NzIjoiaHR0cDovL2xvY2FsaG9zdDo1MTAwIiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zdDo1MTAwIn0.-
cgOBLVO7Te-T4Qrql1LwZ580LbBeQdNHMEbSmGBtJY",
            "expires": 1659708455,
            "user": {
                "id": 1,
                "name": "SysAdmin",
                "lictype": "generic",
                "displayname": "Administrator",
                "uid": "4917ff4f-42cd-11e7-a4d6-00155d09941d",
                "email": "admin@mycompany.com",
                "uisettings": "{}",
                "memberships": [
                    {
                        "id": 1,
                        "groupName": "Administrators"
                    },
                    {
                        "id": 2,
                        "groupName": "Users"
                    }
                ]
            }
        },
        "rapConfig": {
            "defaultRobotLicense": "",
            "defaultMaxUploadSize": 209715200
        }
    }

}
```

Basic 認証はセキュアであると考えるべきではなく、セキュアな SSL 接続を介してのみこのリクエストを送信することを強く推奨します。SSL のために API サーバを保護する方法は、証明書と nginx のようなリバースプロキシで簡単に実現できます。

トークンは、この URL に対してのみ 60 分間有効です。このトークンは、その後のセキュアセグメントでのすべての API 呼び出しで「Bearer」ヘッダとともに使用する必要があります。たとえば、プロセスのリストを取得するには、次のようにします：

```
curl --location --request GET 'https://myserver:5000/api/v3/rap/Processes' --header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmlxdWVfbmFtZSI6IlN5c0FkbWluIiwibmFtZWlkIjoiNDkxN2ZmNGYtNDJjZC0xMWU3LWE0ZDYtMDAxNTVkMDk5NDFkIiwiaHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvd3MvMjAwNS8wNS9pZGVudGl0eS9jbGFpbXMvc2lkIjoiZjQzNGY0YTgtMTRjNy0xMWVkLTkzMTQtMDAxNTVkMDE2NzBhIiwibHR0cDovL3NjaGVtYXMubWljcm9zb2Z0LmNvbS93cy8yMDA4LzA2L2lkZW50aXR5L2NsYWltcy9leHBpcmF0aW9uIjoiNjM3OTUzMDg4NTQwMDAwMDAiLCJuYmYiOjE2NTk3MTIwNTUsImlhdCI6MTY1OTcwODQ1NSwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo1MTAwIiwiYXVkIjoiaHR0cDovL2xvY2FsaG9zdDo1MTAwIn0.-cgOBLVO7Te-T4Qrql1LwZ580LbBeQdNHMEbSmGBtJY'
```

プロセスとワークアイテム

HeartCore Robo 管理ポータルは、ロボットをリモート動作させるためにプロセスとワークアイテムという 2 つのエンティティを使用します。

プロセスとは、作業の高レベル定義であり、通常はある種の企業固有のワークフローが記述されます。このワークフローは、ロジックステップ、ワークアイテム、場合によってはサブプロセスを含む 1 つ以上のステップから構成されます。

ワークアイテムは低レベルの作業定義で、通常、エージェントによって実行または編成される作業自動化の 1 つまたは複数のステップから構成されます。

プロセスでの操作

プロセスのワークフロー定義は、「WorkflowML」と呼ばれる JSON マークアップ スキーマを使用して記述されます。(スキーマの詳細については、このドキュメントの範囲を超えているため記載しません)

管理ポータル UI アプリケーションは、プロセスの WorkflowML を操作するためのビジュアルエディタを提供します。

このため、プロセスの作成と編集はこのアプリケーションで実行するのが最適であり、API はプロセス実行の開始と監視に使用されることが大半です。プロセスの完全な型定義は次のようになります：


```
[
  {
    "Id": 0,
    "Name": "string",
    "Description": "string",
    "WorkflowML": "string",
    "Diagram": "string",
    "Priority": 0,
    "Uid": "string",
    "Version": 0,
    "PreviousVersionProcessId": 0,
    "State": "draft",
    "Archived": true,
    "audit": {
      "createdBy": "string",
      "createdOn": "2019-08-24T14:15:22Z",
      "lastModifiedBy": "string",
      "lastModifiedOn": "2019-08-24T14:15:22Z"
    },
    "Triggers": [
      {
        "id": 0,
        "name": "string"
      }
    ]
  }
]
```

プロセスのインスタンスを「Run（実行）」と呼びます。プロセスの Run は RunQueue で実行され、その結果は RunResults に出力されます。これらは Run の実行と結果に使用する 2 つのエンドポイントです。

Run（実行）キュー

プロセスを実行するには、POST リクエストを/RunQueue エンドポイントに送信して、プロセスのインスタンスを実行キューに追加する必要があります。リクエスト本文には、次のような Run Request オブジェクトの配列が含まれている必要があります。

```
[
    {
      "ProcessId": 0,
      "Trigger": string,
      "Parameters": {
        "input": [
          {
            "name": "string",
            "type": "string",
            "label": "string",
            "value": "string",
            "mandatory": true
          }
        ],
        "output": [
          {
            "name": "string",
            "type": "string",
            "label": "string",
            "value": "string",
            "mandatory": true
          }
        ]
      }
    }
]
```

注：「Parameters」セクションの出力パラメータブロックは、このコンテキストでは使用されないので、無視してよい。

同じリクエストで複数の Run をキューに入れるには、配列に複数の Run Request オブジェクトを追加するだけです。

エンドポイントに GET リクエストを送信することで、/RunQueue を照会することができます。プロセス実行を中断またはキャンセルする必要がある場合は、/RunQueue/{runId} エンドポイントに DELETE リクエストを送信します。各 Run の進行中のステータスは、実行中のワークフローエンジンによって自動的に更新されるため、実行状況に変更があれば自動的に表示されます。

プロセス実行が「完了」の状態遷移に達すると、ワークフローエンジンによって自動的に /RunQueue から /RunResults へ自動的に移動されます。