JP-LINK Adapter Server インストールガイド

バージョン 1.6.2

一般社団法人コンパクトスマートシティプラットフォーム協議会

コンテンツ

はじめに

このドキュメントでは、JP-LINK Adapter Server のインストール、更新、および構成について説明します。

※Adapter Serverインストールの際、センター側でIPアドレス制限をかけておりますので、本作業を行う前に、CSPFCサポートデスクポータルの問合せ作成画面にて、
「AS Global IP登録申請」テンプレートを使って以下の様式で申請してください。。

インストール プロセスは、次の手順で行います。

1. 依存関係をインストールする (git と docker)
2. git からソースコードを複製する
3. SSL 証明書を設定する
4. Docker イメージのビルド

環境要件

- オペレーションシステム： Ubuntu 20.04 or 22.04 LTS (64 ビット サーバー)または RHEL 9.1
- ハードディスク容量：50GB 以上
- CPU：2 Core 以上
- メモリ： 4GB RAM 以上

アプリケーションのインストール

sudo ユーザーとして実行する必要があります。 sudo モードを有効にするには、 Ubuntu ターミナルで次のコマンドを実行します。

```
sudo -i
```

Ubuntu と RHEL でのパッケージのインストールは異なりますが、その後の手順は同じです。

パッケージのインストール

Ubuntu

最新のパッケージに更新します。

```
apt update
apt upgrade
```

docker / docker-compose / git をインストールします。

```
apt install docker docker-compose git
```

RHEL

CentOS 公式リポジトリと Docker リポジトリを追加し、RHEL パッケージ マネージャーを更新します。

```
wget https://www.centos.org/keys/RPM-GPG-KEY-CentOS-Official
rpm --import RPM-GPG-KEY-CentOS-Official
dnf config-manager --add-repo=https://mirror.stream.centos.org/9-stream/AppStream/x86_64/os/
dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
dnf upgrade --refresh
```

docker のインストールとのコンフリクトを避けるために、 podman の依存関係を削除します。次に、docker をインストールし、docker-compose 実行可能ファイルをセットアップして、docker デーモンを開始します。

```
dnf erase podman buildah
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin git
echo 'docker compose --compatibility "$@"' > /bin/docker-compose
chmod +x /bin/docker-compose
systemctl start docker
```

RHEL ガイドの上部にある注釈が CentOS パッケージに言及しているため、 Cent OS Docker インストール ガイドの指示に従ってインストールされます。

タイムゾーンの設定

Web アプリケーションとログに正確な時刻を表示するには、タイム ゾーンが必要です。

利用可能なタイム ゾーンを一覧表示します。

```
timedatectl list-timezones
```

リストをフィルタリングするには、grep を使用してください。

例）アジア圏のタイムゾーンのみ表示

```
timedatectl list-timezones | grep Asia
```

タイムゾーンをリストアップされたものから選択して、設定します。
例）以下はタイムゾーンを日本標準時に設定する場合のコマンドになります。

```
timedatectl set-timezone Asia/Tokyo
```

タイムゾーンが正しく設定されているかどうかを確認します。

```
timedatectl
```

「 Time zone: 」と表示されている行には、設定されたばかりのタイム ゾーンが表示されます (いくつかの追加情報が表示されます)。

アプリケーションのビルドと実行

ソース コード、SSL 証明書、および DB バックアップを保持するディレクトリを作成します。

```
mkdir -p /var/lib/uxa/{git,certs,backup}
```

UXA リポジトリのクローンを作成します。

```
git clone https://UXA:MxVzwHG4Ud1imzsLyk7x@git.aktors.ee/root/UXA.git
/var/lib/uxa/git
```

*この git URL は、UXA (Universal X-Road Adapter) git リポジトリへの Read アクセスを許可するもので、本書に記載する用途以外では利用しないでください。

アプリケーションの構成と実行

SSL サーバー証明書

SSL 証明書を設定するには、次の 2 つのオプションがあります。

1. 証明書から PKCS12 キーストアを作成する
2. 既存の自己署名証明書の使用する

証明書からの PKCS12 キーストアの作成

CA が提供する証明書ファイル、または自己署名証明書を使用しますので事前に準備してください。

以下のパラメータを置き換えてコマンドを実行すると PKCS12 keystore ファイルが作成されます。

- <CERTIFICATE> - サーバー証明書 ファイル.（例：server.crt）

- <CERTIFICATE_KEY> -証明書 キーファイル.（例：server.key）
- <ALIAS> - キーストア内の証明書の名前

以下コマンドを実行してください。パスワードの入力を求められたら、パスワードを入力します。このパスワードは後続の Docker の設定にて <SSL_KEYSTORE_PASSWORD> で使用します。

```
openssl pkcs12 -export \
 -in <CERTIFICATE> -inkey <CERTIFICATE_KEY> \
 -out keystore.p12 -name <ALIAS> \
 -CAfile ca.crt -caname root
```

コマンドを実行すると、カレントディレクトリに keystore.p 12 ファイルが作成されます。

以下コマンドで PKCS12 ファイルを UXA 証明書フォルダーに移動します。

```
mv keystore.p12 /var/lib/uxa/certs/keystore.p12
```

以下コマンドですべてのユーザーに読み取り権限を与えます。

```
chmod a+r /var/lib/uxa/certs/keystore.p12
```

新しく作成された PKCS12 キーストアを使用するには、Docker 設定時に次のプロパティ値を指定します。

- SSL_KEYSTORE – /var/lib/uxa/certs/keystore.p12
- SSL_KEYSTORE_ALIAS – certificate alias name used for creating keystore
- SSL_KEYSTORE_PASSWORD – password used for creating keystore

既存の自己署名証明書の使用

既存の自己署名 PKCS12 キーストアを使用するには、 Docker を設定するときに次のプロパティ値を使用します。

- SSL_KEYSTORE – /var/lib/uxa/git/backend/src/main/resources/uxa_ssl.p12
- SSL_KEYSTORE_ALIAS – uxa_oz1
- SSL_KEYSTORE_PASSWORD – uxauxauxa

Docker コンテナーの設定

プレースホルダーを実際の値に置き換えて、次のコマンドを実行します。

初回インストールは 30 分程度の時間がかかります。

- < APPLICATION_NAME > – アプリケーションの表示名。アダプタサーバの UI アクセス時にメニューバーとブラウザタブに表示されます。**”JP-LINK Adapter Server” と設定ください。**
- <SERVER_CODE> – 特定のインストールのサーバーコードを設定します。任意の名称をつけてください。サーバーコードは、アプリケーション名の横のメニューバーの中央に表示されます。
例：開発検証環境の場合、Dev-AS01　本番環境の場合：Prod-AS01
- <POSTGRES_PASSWORD> – データベースのパスワードを指定します (コマンドを初めて実行すると、指定したパスワードで新しいデータベース ユーザーが作成されます)
- <SSL_KEYSTORE> – SSL キーストアの絶対パス: /var/lib/ uxa /certs/keystore.p12
- <SSL_KEYSTORE_ALIAS> – SSL に使用されるキーストア内のキーストア名
- <SSL_KEYSTORE_PASSWORD> – SSL キーストアのパスワード

```
APPLICATION_NAME=< APPLICATION_NAME > \
 SERVER_CODE=<SERVER_CODE> \
 POSTGRES_PASSWORD=<POSTGRES_PASSWORD> \
 SSL_KEYSTORE=<SSL_KEYSTORE> \
 SSL_KEYSTORE_ALIAS=<SSL_KEYSTORE_ALIAS> \
 SSL_KEYSTORE_PASSWORD=<SSL_KEYSTORE_PASSWORD> \
 docker-compose \
 -f /var/lib/uxa/git/docker-compose.yml \
 --env-file /var/lib/uxa/git/.env \
 up -d --build webapp
```

※インストールコマンドに「--build webapp」は必須ではありませんが追加されているのでアップデートコマンドと同じです。

以下コマンドの例、プレースホルダーの値が置き換えられています。

```
APPLICATION_NAME=“JP-LINK Adapter Server”  \
 SERVER_CODE=“Dev-AS01“ \
 POSTGRES_PASSWORD=uxa \
 SSL_KEYSTORE=/var/lib/uxa/certs/keystore.p12 \
 SSL_KEYSTORE_ALIAS=uxa_oz1 \
 SSL_KEYSTORE_PASSWORD=uxauxauxa \
```

```
docker-compose \
-f /var/lib/uxa/git/docker-compose.yml \
--env-file /var/lib/uxa/git/.env \
up -d --build webapp
```

ファーストユーザーの追加

アプリケーション サーバーが実行されている場合 (場合によっては 1 分かかることがあります)、ターミナルで次のコマンドを sudo として実行します。

最初のユーザーは端末コマンドで追加する必要があります。 <USERNAME>と<PASSWORD> を実際の値に置き換えて、次のコマンドを実行します (インストール後にサーバーの準備が整うまでに数分かかる場合があります)。

```
sudo docker exec -it webapp curl -k -X PUT -w '\n' \
 https://localhost:8443/api/local/user/save --header \
 'Content-Type:application/json' --data-raw \
 '{"username": "<USERNAME>", "password": "<PASSWORD>"}'
```

ユーザーの追加に成功すると、「 Success!」というメッセージが表示されます。

アプリケーションへのアクセス

アプリケーションは、URL「https:// <IPADDRESS>/」からアクセスできるようになりました。 <IPADDRESS> は、アプリケーションが実行されているサーバーの IP アドレスです。

最初のユーザーパラグラフの追加で作成したユーザーで認証する

アプリケーションが実行されているかどうかをテストするには、次のコマンドを使用できます。

```
curl -v -X GET -k https://localhost/login
```

HTML が表示されれば成功です。何らかのエラーが表示された場合、エラーメッセージの内容を確認の上、ここまで手順を見直してみてください。

以上でインストール作業が終了です。

アプリケーションの管理

アプリケーションを更新しています

バックアップディレクトリが存在しない場合ディレクトリ作成：

```
mkdir -p /var/lib/uxa/backup
```

git から更新情報を取得：

```
git -C /var/lib/uxa/git pull
```

プレースホルダーを実際の値に置き換え、コマンドで Docker コンテナを更新します（sudo で実行する必要があります）：

- <APPLICATION_NAME> – “JP-LINK Adapter Server”
- <SERVER_CODE> – 特定のインストールのサーバー コードを表示します。サーバー コードは、アプリケーション名の横のメニュー バーの中央に表示されます。
- <POSTGRES_PASSWORD> –インストール中に設定されたデータベース パスワード
- <SSL_KEYSTORE> – SSL キーストアの絶対パス、つまり: /var/lib/ uxa /certs/keystore.p12
- <SSL_KEYSTORE_ALIAS> – SSL に使用されるキーストア内のキーストア名
- <SSL_KEYSTORE_PASSWORD> – SSL キーストアのパスワード

```
APPLICATION_NAME=“JP-LINK Adapter Server” \
 SERVER_CODE=<SERVER_CODE> \
 POSTGRES_PASSWORD=<POSTGRES_PASSWORD> \
 SSL_KEYSTORE=<SSL_KEYSTORE> \
 SSL_KEYSTORE_ALIAS=<SSL_KEYSTORE_ALIAS> \
 SSL_KEYSTORE_PASSWORD=<SSL_KEYSTORE_PASSWORD> \
 docker-compose \
 -f /var/lib/uxa/git/docker-compose.yml \
 --env-file /var/lib/uxa/git/.env \
 up -d --build webapp
```

```
APPLICATION_NAME=“JP-LINK Adapter Server” \
 SERVER_CODE=“Dev-AS01“ \
 POSTGRES_PASSWORD=uxa \
 SSL_KEYSTORE=/var/lib/uxa/certs/keystore.p12 \
 SSL_KEYSTORE_ALIAS=uxa_oz1 \
 SSL_KEYSTORE_PASSWORD=uxauxauxa \
 docker-compose \
```

```
-f /var/lib/uxa/git/docker-compose.yml \
--env-file /var/lib/uxa/git/.env \
up -d --build webapp
```

特定のバージョンのアプリケーションをインストールする

特定のバージョンのアプリケーションを使用するには、そのバージョンのソース コードを git から取得する必要があります。

アプリケーションのビルドと実行で説明されているように、最初にリポジトリのクローンを作成します。

利用可能なバージョンを表示するには、次のコマンドを使用します。

```
git tag -l -n
```

次に、git リポジトリをそのタグに切り替えます。

- <TAG_NAME> -最後のコマンドから受け取ったタグの名前。すなわち。「v1.1」

```
git checkout tags/<TAG_NAME>
```

正しいバージョンがアクティブであることを確認するには:

```
git status
```

* 次のメッセージが表示されます:「HEAD detached at <TAG_NAME> 」

アプリケーションのインストールまたはアプリケーションの更新の章の指示に従いますが、git 関連のコマンドは既に実行されているためスキップしてください。

最新バージョンに戻す

git から master ブランチをチェックアウトする

```
git checkout master
```

正しいバージョンがアクティブであることを確認するには:

```
git status
```

*次のメッセージが表示されます。

“On branch master. Your branch is up to date with 'origin/master'.”

その後、アプリケーションの更新のセクションから再開してください。ただし、git 関連の操作は既に完了しているため、実行せず、スキップしてください。

ログの表示

root に切り替えます。

```
sudo -i
```

ログは次のディレクトリに保存されます。

```
cd /var/lib/docker/containers/$(docker inspect --format="{{.Id}}"
webapp)
```

最新のログ ファイルは<container-id >.json.log です。<container-id> はランダムな英数字で、意味はありません。

古いログ ファイルは圧縮され、 <container-id >.json.log .<order-nr> という名前が付けられます。gz 最小の <order-nr> は、最新のアーカイブ ログを示します。 < orded -nr>が大きいほど、古いログです。

ログは次のコマンドでも表示できます。

```
docker logs webapp
```

データベースパスワードの変更

データベースのパスワードを変更する必要がある場合:

database コンテナのデータベースに接続します。

```
docker exec -it database psql -U uxa uxa
```

デフォルトユーザー”uxa “のパスワードを変更します。

<POSTGRES_PASSWORD>をデータベースの新しいパスワードに置き換えます。

```
ALTER USER uxa WITH PASSWORD '<POSTGRES_PASSWORD>';
```

ディスク容量のクリア

アプリケーションの新しいイメージをビルドした後、古い Docker イメージがディスクに残り、スペースを占有します。使用されていない未使用のイメージは、次のコマンドで削除できます。

```
docker image prune -a
```

タイムゾーンの変更

インストール後にタイム ゾーンを変更するには、以前の Docker コンテナーを削除する必要があります。
これにより、保存されたデータが削除されることはありません。

```
docker stop database webapp liquibase
docker rm database webapp liquibase
```

次に、インストール コマンドを実行してコンテナーを再作成します。

次に、データベースのタイム ゾーンを変更します。

データベースに接続:

```
docker exec -it database psql -U uxa uxa
```

タイムゾーンを変更します (以前と同じオプション):

```
SET TIMEZONE='Asia/Tokyo';
```

タイム ゾーンが正しく設定されていることを確認します。

```
SHOW TIMEZONE;
```

Docker コンテナ コマンド

便利な Docker コマンド (docker コマンドには sudo が必要です):

- docker ps – 実行中のコンテナを表示
- docker start <container name> - コンテナーを開始します。コンテナー名: データベース、webapp 、および Liquibase
- docker stop <コンテナ名> - コンテナを停止します
- docker logs <コンテナ名> - コンテナのログを表示します
- docker exec -it <コンテナ名> bash – コンテナターミナルに接続

データベースのバックアップ

database-backup docker コンテナーによって作成されます。このコンテナーは、 webapp コンテナーで自動的に開始されます。バックアップは、ホスト ファイル システムの/var/lib/ uxa /backup ディレクトリに保存されます。デフォルトでは、バックアップは毎日 00:00 に 1 回作成され、5 日間保持されます。また、バックアップ スクリプトは、実行中に古いバックアップ ファイルを削除します。

バックアップ時に、バックアップディレクトリがまだ存在しない場合は、コンテナを実行する前に手動でバックアップディレクトリを作成する必要があります。

```
mkdir -p /var/lib/uxa/backup
```

コンテナログは、次の方法で表示できます。

```
docker logs database-backup
```

database-backup コンテナが正常に開始された場合、ログには以下が表示されます。
“Starting CRON in the foreground.”

CRON ジョブがバックアップを作成するたびに、次の文が追加されます:
“DB backup successful.”

バックアップは database-backup Docker コンテナ内の CRON ジョブによって自動的に作成されます。 CRON ジョブは、コンテナの起動プロセス中に/var/lib/uxa/git/database/backup/setup-cron-db-backup.sh スクリプトによってセットアップされます。このプロセスでは、バックアップスクリプトもコンテナ内の/root/backup.sh ファイルに保存されます。

バックアップ スクリプトは、次のコマンドで手動実行できます。

```
docker exec database-backup /root/backup.sh
```

データベースは、/var/lib/xa/backup ディレクトリにある任意のバックアップファイルからリストアすることができます。

既存のバックアップを確認するには:

```
ls /var/lib/uxa/backup/
```

database-backup Docker コンテナからアクセスできるリストアスクリプトがあります。第一引数としてバックアップファイル名を受け取ります。たとえば、 /var/lib/ uxa /backup/ディレクトリにあるバックアップファイル uxa --10.04.2023_00-00-00.sql から UXA データベースを復元するには、次のようにします。

```
docker exec -it database-backup /db-backup-script/restore.sh 'uxa--
10.04.2023_00-00-00.sql'
```

バックアップ ファイルが存在する場合、スクリプトは続行するかどうかを確認するプロンプトを表示します。
Restoring from 'uxa--10.04.2023_00-00-00.sql' erases existing 'uxa' database. Are you sure you want to continue? (y/n)
続行するには、 y を入力してから return キーを押下します。その後、スクリプトは UXA データベー

スをクリアし、バックアップから DB を復元します。復元が成功した場合、スクリプトは
次のように出力します。

Restore finished successfully!

データベースバックアップ (およびデータベース) コンテナを手動で再起動するには以下:

```
docker-compose up -d --no-deps --build --remove-orphans --force-
recreate database-backup
```

コンテナが実行されているかどうかを確認するには以下:

```
docker exec database-backup cat /root/backup.sh | head -n 1
```

#!/ bin/bash で始まる bash スクリプトが返されます。

docker-compose コマンドに追加された次の環境変数を使用して、バックアップの実行時間とバックアップが保持される日数を設定できます。

```
DB_BACKUP_CRON_TRIGGER='0 2 * * *' \
DB_BACKUP_DAYS_KEPT=10 \
```

上記の環境変数値は、 DB バックアップが毎日午前 2 時に作成され、結果のバックアップ ファイルが 10 日間保持されるように設定します。

DB_BACKUP_CRON_TRIGGER 変数の値は、いつ、どのくらいの頻度でバックアップを作成するかを決定する CRON 式です。 Bash による * 拡張を避けるために、変数値を引用符で囲みます。

DB_BACKUP_DAYS_KEPT 変数の値は、バックアップが保持される日数です。指定された日数よりも古いファイルは、 backup.sh スクリプトの次回の実行時に削除されます。