



Postgres Enterprise Manager - Planning Guide

リリース 8.5.0

EDB

2022 年 07 月 15 日

目次

第 1 章 Understanding Postgres Enterprise Manager components and architecture	3
1.1 PEM サーバー	4
1.2 PEM エージェント	6
1.3 PEM Web クライアント	7
1.4 SQL プロファイラプラグイン	8
第 2 章 Deployment considerations	9
2.1 Web サーバーのインストールオプション	9
2.1.1 別のホスト上の PEM サーバーと Apache Web サーバー	9
2.1.2 同じホスト上の PEM サーバーと Apache Web サーバー	10
2.1.2.1 Using Failover Manager for high availability	10
2.1.2.1.1 製品の初期インストールと構成	11
2.1.2.1.2 ストリーミングレプリケーション用のプライマリノードを設定する	11
2.1.2.1.3 ストリーミングレプリケーションのスタンバイノードを設定する	13
2.1.2.1.4 すべてのホストでフェイルオーバーを管理するように EFM を設定する	15
2.1.2.1.5 現在の制限	20
2.1.2.2 Connection pooling using pgBouncer	20
2.1.2.2.1 PEM server and agent connection management mechanism	21
2.1.2.2.2 Preparing the PEM database server	22
2.1.2.2.3 Configuring PgBouncer	23
2.1.2.2.4 Configuring the PEM agent	26
2.1.2.3 Authentication options	28
2.1.2.3.1 Configuring the PEM server to use Kerberos authentication	28
2.1.2.3.2 Configuring the PEM server to use Windows Active Directory Domain Services for Kerberos authentication (SSPI)	33
2.1.2.3.3 Configuring the PEM server to use two-factor authentication	45
2.1.2.4 Securing your deployment	47
2.1.2.4.1 Apache HTTPD security configurations	47
2.1.2.4.2 PEM application security configurations	53
2.1.2.5 Web server installation options	57
2.1.2.5.1 別のホスト上の PEM サーバーと Apache Web サーバー	57

Postgres Enterprise Manager (PEM) 計画ガイドへようこそ。このガイドでは、PEM の使用を計画する際のアーキテクチャと考慮事項について説明します。PEM は、EDB Postgres Advanced Server または PostgreSQL データベースの管理および分析機能を提供するコンポーネントで構成されています。PEM は、オープンソースの pgAdmin 4 プロジェクトに基づいています。PEM は、包括的なデータベース設計および管理システムです。PEM は、初心者と経験豊富な Postgres ユーザーの両方のニーズを満たすように設計されており、データベースオブジェクトの作成、メンテナンス、使用を簡素化する強力なグラフィカルインターフェイスを提供します。

第1章 Understanding Postgres Enterprise Manager components and architecture

Postgres Enterprise Manager (PEM) は、単一の GUI インターフェイスを介して複数の Postgres サーバーを監視および管理するように設計されたツールです。PEM は、インフラストラクチャの次の領域を監視できます。

- **ホスト** - 1 つ以上のサーバー（物理または仮想）とそのオペレーティングシステム。
- **データベース** - 1 つ以上のデータベースとその中のスキーマオブジェクト（テーブル、インデックスなど）。

注釈:

- Postgres という用語は、PostgreSQL または EDB Postgres Advanced Server または EDB Postgres Extended Server を指します。
-

PEM は、多数の個々のソフトウェアコンポーネントで構成されています。個々のコンポーネントについて以下に説明します。

- **PEM サーバー** - PEM サーバーは、データを監視するためのデータリポジトリとして、およびエージェントとクライアントの両方が接続するサーバーとして使用されます。PEM サーバーは、PostgreSQL のインスタンスと、監視データを保存するための関連データベース、および Web サービスを提供するサーバーで構成されています。
- **PEM エージェント** - PEM エージェントは、タスクを実行し、エージェントホストおよび監視対象の Postgres インスタンスから PEM サーバーに統計を報告します。単一の PEM エージェントは、1 つ以上のホストに存在する Postgres の複数のインスタンスを監視できます。
- **PEM Web クライアント** - PEM Web インターフェイスを使用すると、Postgres サーバーを管理および監視し、PEM 拡張機能を利用できます。Web インターフェイスソフトウェアは PEM サーバーと共にインストールされ、サポートされている Web ブラウザーを介してアクセスされます。
- **SQL Profiler** - SQL Profiler は、PEM の SQL Profiler ツールで分析される監視データとクエリプランを記録するための Postgres サーバープラグインです。これは PEM のオプションコンポーネントですが、プラグインは、SQL プロファイラーツールを使用する Postgres の各インスタンスにインストールする必要があります。SQL プロファイラーは、PostgreSQL サーバーまたは Advanced Server（PEM サーバーで管理されているものだけでなく）の EDB ディストリビューションのサポートされているバージョンで使用できます。詳細とサポートされているバージョンについては、

SQL Profiler Configuration を参照してください。

PEM アーキテクチャ

次のアーキテクチャ図は、PEM サーバー、クライアント、管理対象および非管理対象の Postgres サーバー間の関係を示しています。

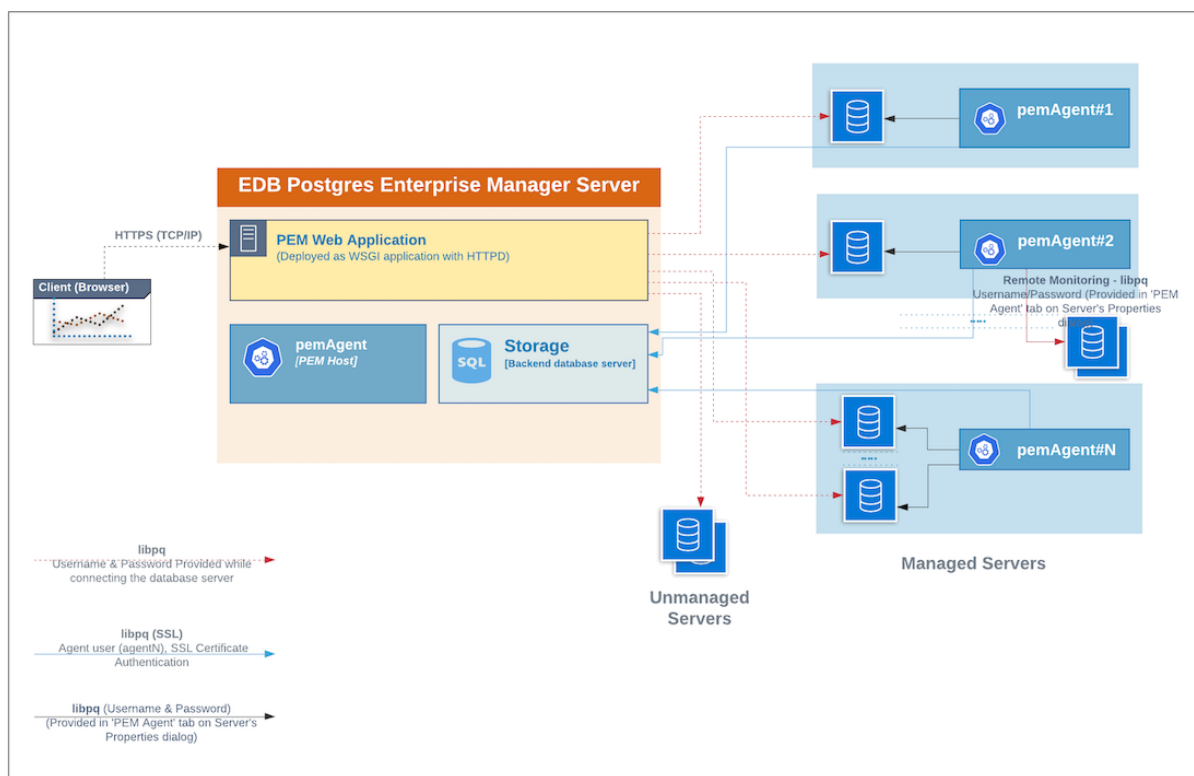


図 1: PEM Architecture

1.1 PEM サーバー

PEM サーバー

PEM サーバーは、Postgres のインスタンス、クライアントに Web サービスを提供する Apache Web サーバーのインスタンス、および PEM エージェントで構成されています。PEM は、サーバー側の暗号化プラグインを利用して認証証明書を生成します。

Postgres のインスタンス（データベースサーバー）と Apache ウェブサーバー（HTTPD）のインスタンスは、同じホスト上または別のホスト上に置くことができます。

- **Postgres インスタンス（データベースサーバー）** - これはバックエンドデータベースサーバーです。PEM サーバーのリポジトリとして機能する****pem****という名前のデータベースをホストします。****pem****データベースには、監視対象の各ホスト、サーバー、データベースから収集されたメトリックデータを保存するいくつかのスキーマが含まれています。.. **Note::** 選択したバックエンドデータベースサーバー PostgreSQL または EDB Postgres Advanced Server に関係なく、すべての PEM 機能を使用できます。
- **pem** - このスキーマは、PEM アプリケーションのコアです。アプリケーションに必要な構成関数、テーブル、またはビューの定義が含まれています。- **pemdata** - このスキーマは、監視対象データの現在のスナップショットを保存します。- **pemhistory** - このスキーマには、監視対象の履歴データが保存されます。

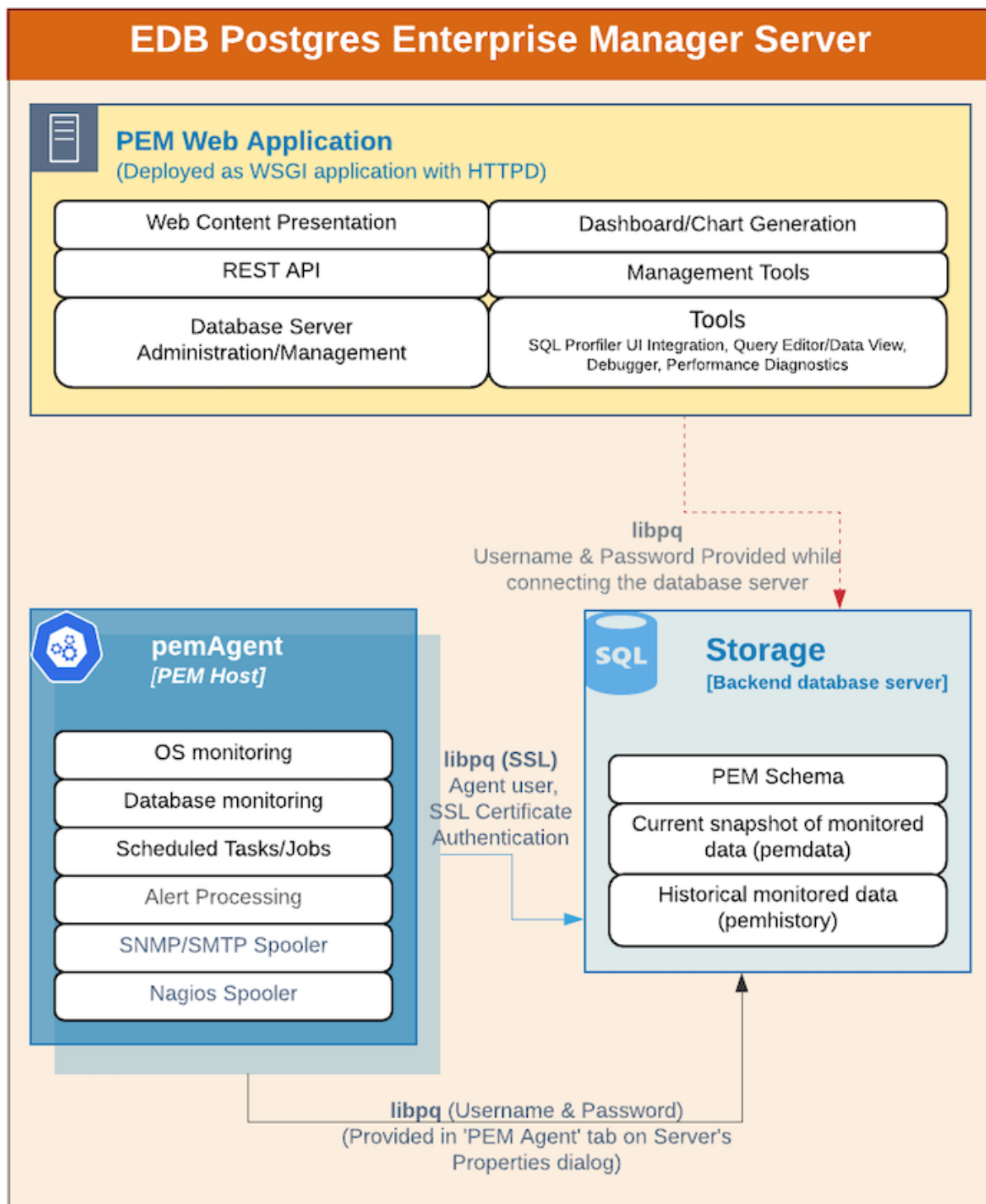


図 2: PEM Server

- **ApacheWeb サーバー (HTTPD)** -PEM Web アプリケーションは、HTTPD を備えた WSGI アプリケーションとして展開され、クライアントに Web サービスを提供します。これは以下で構成されています。- **Web コンテンツプレゼンテーション** - プレゼンテーションレイヤーは Web アプリケーション（たとえば、ブラウザ、ログインページなど）によって作成されます。- **RestAPI** - REST API を使用すると、他のアプリやサービスとの統合が可能になります。- **データベースサーバーの管理/管理** - CREATE、ALTER、DROP などのデータベースサーバーの管理および管理アクティビティは、管理対象サーバーと非管理対象サーバーで実行できます。- **ダッシュボード/チャート生成** -内部的に、Web アプリケーションにはダッシュボードとチャートを生成する機能が含まれています。- **管理ツール** - Audit Manager、Capacity Manager、Log Manager、Postgres Expert、Postgres Log Analysis Expert、およびチューニングウィザードが Web アプリケーションで利用できるようになりました。- **他のツール**は、管理対象または非管理 SQL サーバーで機能を提供し SQL。- **クエリエディター/データビュー** - クエリエディターを使用すると、データを照会、編集、表示できます。- **デバッガー** - デバッガーは、クエリのデバッグに役立ちます。- **パフォーマンス診断** -パフォーマンス診断は、Postgres インスタンスのパフォーマンスの分析に役立ちます。

専用のマシンを使用して、PEM バックエンドデータベースの本番インスタンスをホストすることをお勧めします。監視対象のデータベースサーバーの数とサーバーが処理しているワークロードによっては、ホストのデータスループットが高くなる場合があります。

1.2 PEM エージェント

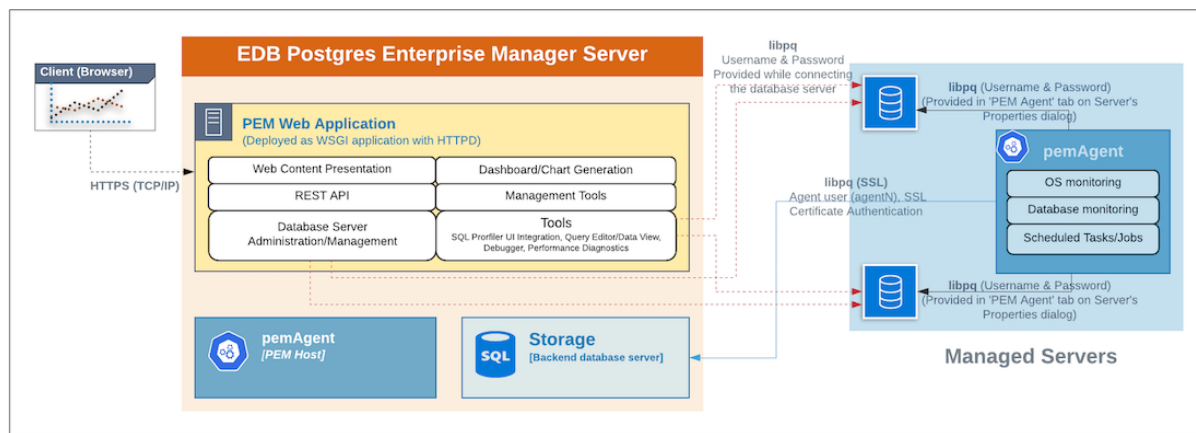


図 3: PEM Agent

PEM エージェント

PEM エージェントは、マシンとオペレーティングシステム、およびバインドされている各 Postgres インスタンスからの監視データの収集を担当します。各 PEM エージェントは 1 つの物理マシンまたは仮想マシンを監視でき、同じシステムにインストールされた、またはリモートでインストールされた複数のデータベースサーバーを監視できます。また、ユーザーがスケジュールできる他のタスク（サーバーのシャットダウン、SQL プロファイラーのトレース、ユーザー定義のジョブなど）を実行します。

PEM エージェントは、PEM サーバーのインストールとともに PEM サーバーにデフォルトでインストールされます。通常、PEM ホスト上の PEM エージェントと呼ばれます。別に、PEM を使用して監視する Postgres インスタンスをホストする他のサーバーに PEM エージェントをインストールすることもできます。

ローカルで監視するかリモートで監視するかにかかわらず、PEM エージェントは、SSL 証明書ベースの認証を使用して、PostgreSQL の libpq を使用して PEM サーバーに接続します。Windows の PEM エージェントインストーラーと Linux の pemworker CLI は、各エージェントを PEM サーバーに登録し、必要な証明書を生成してインストールします。

PEM エージェントと PEM サーバー間のトラフィックは一方通行であることに注意してください。PEM エージェントは常に PEM サーバーに接続します。

PEM エージェントは、監視する各データベースサーバーに接続できる必要があります。この接続は TCP/IP 接続（またはオプションで Unix ホストの Unix ドメインソケット）を介して行われ、オプションで SSL を使用できます。ユーザーは、監視対象サーバーへの接続と認証を構成する必要があります。

構成が完了すると、各エージェントは、ホストと、監視対象の各データベースサーバーとデータベースに関する統計情報やその他の情報を収集します。各情報は**メトリック**と呼ばれ、**プローブ**によって収集されます。ほとんどのプローブは、効率のために一度に複数のメトリックを収集します。収集されるメトリックの例は次のとおりです。

- ディスク I/O 統計
- ネットワーク統計
- データベースサーバーのバージョン文字列
- データベースサーバー構成オプション（GUC）の値
- テーブルアクセス統計
- テーブルとインデックスのサイズ

PEM プローブのリストは [here](#) です。

デフォルトでは、データベースサーバーにバインドされた PEM エージェントは OS /データベースモニタリング統計を収集し、その特定のデータベースサーバーのスケジュールされたタスク/ジョブを実行し、PEM サーバーの pem データベースにデータを保存します。

アラート処理、SNMP / SMTP スプーラー、および Nagios スプーラーデータは、PEM サーバーの pem データベースに保存され、デフォルトで PEM ホスト上の PEM エージェントによって処理されます。ただし、PEM エージェントの SNMP / SMTP および Nagios パラメーターを調整することにより、他の PEM エージェントによる処理を有効にできます。

これらのパラメーターの詳細については、[メールサーバー構成](#)を参照してください。

1.3 PEM Web クライアント

PEM クライアントは、サポートされているブラウザで実行される Web ベースのアプリケーションです。クライアントの Web インターフェイスは PEM サーバーに接続し、管理対象または非管理対象サーバー、およびそれらにあるデータベースとスキーマを直接管理できます。

クライアントでは、ダッシュボード、Postgres Log Analysis Expert、Capacity Manager などの機能を介してサーバーに記録されたデータを利用する PEM 機能を使用できます。

1.4 SQL プロファイラープラグイン

すべてのサーバーに SQL Profiler プラグインをインストールする必要はありませんが、SQL Profiler を使用する各サーバーにプラグインをインストールして構成する必要があります。監視されていない開発サーバーに SQL プロファイラーをインストールして構成することもできます。アドホック使用のために、SQL Profiler プラグインを一時的にインストールすることもできます。

プラグインは EDB Postgres Advanced Server ディストリビューションとともにインストールされますが、PostgreSQL で使用するには別途インストールする必要があります。SQL プロファイラーインストーラーは、[EDB website](#) から入手できます。

SQL Profiler は、PEM で管理されていないサーバーで使用できますが、スケジュールされたトレースを実行するには、サーバーにプラグインをインストールし、インストールおよび構成済みの PEM エージェントで管理する必要があります。

SQL プロファイラーの使用の詳細については、[SQL プロファイラープラグイン](#)を参照してください。

第2章 Deployment considerations

Postgres Enterprise Manager を展開する前に考慮すべきことがたくさんあります。

Considerations	Implementation instructions
スタンドアロンサーバーで十分ですか、それとも高可用性アーキテクチャが必要ですか？	Installing the server or Deploying high availability
接続プーリングを実装する必要がありますか？	Deploying connection pooling
使用する認証の種類	<i>Authentication options</i>
セキュリティの脆弱性を回避するには、どのようなアクションをとるべきですか？	<i>Securing your deployment</i>
Apache Web サーバーをホストする場所は？	<i>Web サーバーのインストールオプション</i>

2.1 Web サーバーのインストールオプション

PEM サーバーのインストール中に、Apache Web サーバーのホスティング設定を指定できます。

2.1.1 別のホスト上の PEM サーバーと Apache Web サーバー

1. 両方のホストに PEM サーバーをインストールします。手順については、[Installing the PEM server on Linux](#) を参照してください。
2. 最初のホストで Database オプションを選択して、PEM サーバーホストを構成します。
3. 2 番目のホストで Web Services オプションを選択して、Apache Web サーバーを構成します。

PEM サーバーの構成の詳細については、[Configuring the PEM server on Linux platforms](#) を参照してください。

注釈: 実稼働環境では、PEM サーバーと Apache Web サーバーを別のホストに配置することをお勧めします。

2.1.2 同じホスト上の PEM サーバーと Apache Web サーバー

1. PEM サーバーをインストールします。手順については、Installing the PEM server on Linux を参照してください。

2. 構成スクリプトを実行し、Web Services and Database オプションを選択して PEM サーバーと Apache Web サーバーを同じホストにインストールします。手順については

Configuring the PEM server on Linux platforms を参照してください。

注釈: 非実稼働環境の場合、PEM サーバーと Apache Web サーバーを同じホストに配置することをお勧めします。

2.1.2.1 Using Failover Manager for high availability

Postgres Enterprise Manager (PEM) は、データベース管理者、システムアーキテクト、およびパフォーマンスアナリストが Postgres データベースサーバーを管理、監視、およびチューニングするのを支援します。

Failover Manager は、プライマリでソフトウェアまたはハードウェアに障害が発生した場合に Postgres プライマリノードがスタンバイノードに自動的にフェールオーバーできるようにする EDB の高可用性ツールです。

このドキュメントでは、Failover Manager を使用して PEM の高可用性 (HA) 環境をセットアップするための詳細情報を提供します。

次のセクションの例では、次の IP アドレスを使用します。

- 172.16.161.200-PEM プライマリ
- 172.16.161.201-PEM スタンバイ 1
- 172.16.161.202-PEM スタンバイ 2
- 172.16.161.203-EFM ウィットネスノード
- 172.16.161.245-PEM VIP (エージェントとユーザーが接続に使用)

以下は VIP アドレスを使用する必要があります。

- 監視対象データベースサーバーの PEM エージェントバインディング
- PEM Web クライアントへのアクセス
- Web サーバーサービスへのアクセス

2.1.2.1.1 製品の初期インストールと構成

1. プライマリと 1 つ以上のスタンバイに次をインストールします。

- EDB Postgres Advanced Server (PEM サーバーのバックエンドデータベース) - [PEM サーバー](#) - EDB Failover Manager 4.1

上記のリンクを使用して、製品ドキュメントのインストール手順を参照するか、

[EDB repos website](#) に記載されている手順を参照してください。

EDB リポジトリにアクセスする手順の USERNAME:PASSWORD をユーザー名とパスワードに置き換える必要があります。

PEM サーバー構成スクリプトは trust 認証では機能しないため、データベースサーバーが scram-sha-256 認証方法を使用するように構成されていることを確認します。

EFM をインストールするには、java-1.8.0-openjdk パッケージをインストールする必要があります。

2. タイプ 1 (つまり、Web サービスとデータベース) の初期構成でプライマリサーバーで PEM サーバーを構成します。

edb_notranlate_0 構成タイプの詳細については、[Configuring the PEM server to use Kerberos authentication](#) を参照してください。

3. プライマリサーバーとすべてのスタンバイサーバーのファイアウォールに次のポートを追加して、アクセスを許可します。

- 8443 - PEM サーバー用 (https) - 5444 - EPAS 13 用 - 7800 - EFM 用 - 7908 - EFM 管理者用

例：

```
$ sudo firewall-cmd --zone=public --add-port=5444/tcp --permanent
success
$ sudo firewall-cmd --zone=public --add-port=8443/tcp --permanent
success
$ sudo firewall-cmd --zone=public --add-port=7800/tcp --permanent
success
$ sudo firewall-cmd --zone=public --add-port=7809/tcp --permanent
success
$ sudo firewall-cmd --reload
success
```

2.1.2.1.2 ストリーミングレプリケーション用のプライマリノードを設定する

1. 次のコマンドを使用して、レプリケーションロールを作成します。

```
$ /usr/edb/as13/bin/psql -h 172.16.161.200 -p 5444 -U enterprisedb edb -c "CREATE
→ROLE repl REPLICATION LOGIN PASSWORD password";
CREATE ROLE
```

選択したパスワードを入力します。

2. postgresql.conf ファイルで次を構成します。

```
wal_level = replica
max_wal_senders = 10
wal_keep_size = 500
max_replication_slots = 10
```

ストリーミングレプリケーションのパラメーターの構成の詳細については、

[PostgreSQL documentation](#) を参照してください。

注釈: 構成パラメーターは、データベースサーバーのバージョンによって異なる場合があります。これらのパラメーターの設定については、techsupport@enterprisedb.com のサポートに電子メールでお問い合わせください。

3. ホストベースの認証 (/var/lib/edb/as13/data/pg_hba.conf) ファイルに次のエントリを追加して、レプリケーションユーザーがすべてのスタンバイから接続できるようにします。

```
hostssl replication repl 172.16.161.201/24 scram-sha-256
```

注釈: 必要に応じて、IP アドレスの cidr 範囲を変更できます。

4. pem_user ロールのホストベース認証 (/var/lib/edb/as13/data/pg_hba.conf) ファイルを変更して、以下の scram-sha-256 認証方法を使用してすべてのデータベースに接続します。

```
# Allow local PEM agents and admins to connect to PEM server
hostssl all +pem_user 172.16.161.201/24 scram-sha-256
hostssl pem +pem_user 127.0.0.1/32 scram-sha-256
hostssl pem +pem_agent 127.0.0.1/32 cert
# Allow remote PEM agents and users to connect to PEM server
hostssl pem +pem_user 0.0.0.0/0 scram-sha-256
hostssl pem +pem_agent 0.0.0.0/0 cert
```

5. EPAS 13 サーバーを再起動します。

```
systemctl restart edb-as-13.service
```

2.1.2.1.3 ストリーミングレプリケーションのスタンバイノードを設定する

1. すべてのスタンバイノードで EPAS 13 のサービスを停止します。

```
$ systemctl stop edb-as-13.service
```

注釈: この例では、`pg_basebackup` ユーティリティを使用して、スタンバイサーバーに PEM バックエンドデータベースサーバーのレプリカを作成します。`pg_basebackup` を使用する場合、既存のデータベースサーバーを停止し、既存のデータディレクトリを削除する必要があります。

2. すべてのスタンバイノードでデータベースサーバーのデータディレクトリを削除します。

```
$ sudo su - enterprisedb

$ rm -rf /var/lib/edb/as13/data/*
```

3. すべてのスタンバイノードで `enterprisedb` ユーザーのホームディレクトリに `.pgpass` ファイルを作成します。

```
$ sudo su - enterprisedb

$ cat > ~/.pgpass << _EOF_
172.16.161.200:5444:replication:repl:CHANGE_ME
172.16.161.201:5444:replication:repl:CHANGE_ME
172.16.161.202:5444:replication:repl:CHANGE_ME
_EOF_

$ chmod 600 ~/.pgpass
```

4. `pg_basebackup` を使用して、各スタンバイノードでプライマリノードのバックアップを取得します。

```
$ sudo su - enterprisedb /usr/edb/as13/bin/pg_basebackup -h 172.16.161.200 \
-D /var/lib/edb/as13/data -U repl -v -P -Fp -R -p 5444
```

backup コマンドは、スタンバイノードに `postgresql.auto.conf` および `standby.signal` ファイルを作成します。`postgresql.auto.conf` ファイルの内容は次のとおりです。

```
sudo su - enterprisedb cat /var/lib/edb/as13/data/postgresql.auto.conf
# Do not edit this file manually
# It will be overwritten by the ALTER SYSTEM command.
primary_conninfo = 'user=repl passfile=' /var/lib/edb/.pgpass' channel_
→binding=prefer host=172.16.161.200 port=5444 sslmode=prefer sslcompression=0 ssl_
→min_protocol_version=TLSv1.2 gssencmode=prefer krbsvname=postgres target_session_
→attrs=any'
```

5. 各スタンバイノードの `postgresql.conf` ファイルの次のパラメーターを編集します。

```
hot_standby = on
```

6. 各スタンバイノードで EPAS 13 データベースサーバーを起動します。

```
$ systemctl enable edb-as-13

$ systemctl start edb-as-13
```

7. 次のファイルをプライマリノードからスタンバイノードの同じ場所にコピーして、既存のファイルを上書きし、ファイルに次の権限を設定します。

- /etc/httpd/conf.d/edb-pem.conf - /etc/httpd/conf.d/edb-ssl-pem.conf - /root/.pem/agent1.crt - /root/.pem/agent1.key - /usr/edb/pem/agent/etc/agent.cfg - /usr/edb/pem/share/.install-config - /usr/edb/pem/web/pem.wsgi - /usr/edb/pem/web/config_setup.py

たとえば、

```
$ mkdir -p /root/.pem
$ chown root:root /root/.pem
$ chmod 0755 /root/.pem
$ mkdir -p /var/lib/pemhome/.pem
$ chown pem:pem /var/lib/pemhome/.pem
$ chmod 0700 /var/lib/pemhome/.pem
$ mkdir -p /usr/edb/pem/logs
$ chown root:root /usr/edb/pem/logs
$ chmod 0755 /usr/edb/pem/logs
$ for file in /etc/httpd/conf.d/edb-pem.conf \
              /etc/httpd/conf.d/edb-ssl-pem.conf \
              /root/.pem/agent1.crt \
              /usr/edb/pem/agent/etc/agent.cfg \
              /usr/edb/pem/share/.install-config \
              /usr/edb/pem/web/pem.wsgi \
              /usr/edb/pem/web/config_setup.py; do \
    chown root:root ${file}; \
    chmod 0644 ${file}; \
done;
$ chmod 0600 /root/.pem/agent1.key
$ chown root:root /root/.pem/agent1.key
```

これにより、Web サーバーがスタンバイで構成され、デフォルトで無効になります。EFM によるスイッチオーバー時に自動的に有効になります。

注釈: 証明書が更新されるたびに、マスターとスタンバイで証明書を手動で同期する必要があります。

8. configure-selinux.sh スクリプトを実行して、次のように PEM の SELinux ポリシーを構成します。

```
$ /usr/edb/pem/bin/configure-selinux.sh
getenforce found, now executing getenforce command
Configure the httpd to work with the SELinux
Allow the httpd to connect the database (httpd_can_network_connect_db = on)
Allow the httpd to connect the network (httpd_can_network_connect = on)
Allow the httpd to work with cgi (httpd_enable_cgi = on)
Allow to read & write permission on the pem user home directory
SELinux policy is configured for PEM

$ sudo chmod 640 /root/.pem/agent1.crt
```

9. すべてのレプリカノードで実行されている場合は、Httpd および PEM エージェントサービスを無効にします。

```
systemctl stop pemagent
systemctl stop httpd
systemctl disable pemagent
systemctl disable httpd
```

注釈: この時点で、必要に応じてプライマリから引き継ぐ準備ができていない PEM プライマリサーバーと 2 つのスタンバイが必要です。

2.1.2.1.4 すべてのホストでフェイルオーバーを管理するように EFM を設定する

1. EFM をサポートするようにプライマリノードを準備します。

- データベースサーバーに接続するデータベースユーザー `efm` を作成します。・ユーザに WAL ログ関連機能の実行権限および監視権限を付与します。- `pg_hba.conf` にエントリを追加して、`efm` データベースユーザーがすべてのホスト上のすべてのノードからデータベースサーバーに接続できるようにします。- すべてのデータベースサーバーで構成をリロードします。たとえば、

```
$ cat > /tmp/efm-role.sql << _EOF_
-- Create a role for EFM
CREATE ROLE efm LOGIN PASSWORD password;

-- Give privilege to efm user to connect to a database
GRANT CONNECT ON DATABASE edb TO efm;

-- Give privilege to efm user to do backup operations
GRANT EXECUTE ON FUNCTION pg_current_wal_lsn() TO efm;
GRANT EXECUTE ON FUNCTION pg_last_wal_replay_lsn() TO efm;
GRANT EXECUTE ON FUNCTION pg_wal_replay_resume() TO efm;
GRANT EXECUTE ON FUNCTION pg_wal_replay_pause() TO efm;
GRANT EXECUTE ON FUNCTION pg_reload_conf() TO efm;
```

(次のページに続く)

(前のページからの続き)

```
-- Grant monitoring privilege to the efm user
GRANT pg_monitor TO efm;
_EOF_

$ /usr/edb/as13/bin/psql -h 172.16.161.200 -p 5444 -U enterprisedb edb -f /tmp/efm-
→role.sql
CREATE ROLE
GRANT
GRANT
GRANT
GRANT
GRANT
GRANT
GRANT ROLE

$ rm -f /tmp/efm-role.sql

$ cat > /var/lib/edb/as13/data/pg_hba.conf <<< _EOF_
hostssl      edb      efm      172.16.161.200/32      scram-sha-256
hostssl      edb      efm      172.16.161.201/32      scram-sha-256
hostssl      edb      efm      172.16.161.202/32      scram-sha-256
hostssl      edb      efm      172.16.161.203/32      scram-sha-256
_EOF_

$ /usr/edb/as13/bin/psql -h 172.16.161.200 -p 5444 -U enterprisedb edb -c "SELECT pg_
→reload_conf();"

```

2. 各ノードでスクリプトを作成して、PEM エージェントを起動/停止します。

```
$ sudo cat > /usr/local/bin/start-httpd-pemagent.sh << _EOF_
#!/bin/sh
/bin/sudo /bin/systemctl enable httpd
/bin/sudo /bin/systemctl start httpd
/bin/sudo /bin/systemctl enable pemagent
/bin/sudo /bin/systemctl start pemagent
_EOF_
$ sudo cat > /usr/local/bin/stop-httpd-pemagent.sh << _EOF_
#!/bin/sh

/bin/sudo /bin/systemctl stop pemagent
/bin/sudo /bin/systemctl disable pemagent
/bin/sudo /bin/systemctl stop httpd
/bin/sudo /bin/systemctl disable httpd
_EOF_

```

(次のページに続く)

(前のページからの続き)

```
$ sudo chmod 770 /usr/local/bin/start-pemagent.sh
$ sudo chmod 770 /usr/local/bin/stop-pemagent.sh
```

3. 各ノードで `sudoers` ファイル (`/etc/sudoers.d/efm-pem`) を作成して、`efm` ユーザーが `pemagent` を起動/停止できるようにします。

```
$ sudo cat > /etc/sudoers.d/efm-pem << _EOF_
efm    ALL=(ALL)          NOPASSWD:  /bin/systemctl enable pemagent
efm    ALL=(ALL)          NOPASSWD:  /bin/systemctl disable pemagent
efm    ALL=(ALL)          NOPASSWD:  /bin/systemctl stop pemagent
efm    ALL=(ALL)          NOPASSWD:  /bin/systemctl start pemagent
efm    ALL=(ALL)          NOPASSWD:  /bin/systemctl status pemagent
_EOF_
```

4. サンプルファイル (`/etc/edb/efm-4.1/efm.nodes.in`) を使用して、すべてのノードで `efm.nodes` ファイルを作成し、`efm` OS ユーザーに読み取り/書き込みアクセスを付与します。

```
$ sudo cp /etc/edb/efm-4.1/efm.nodes.in /etc/edb/efm-4.1/efm.nodes
$ sudo chown efm:efm /etc/edb/efm-4.1/efm.nodes
$ sudo chmod 600 /etc/edb/efm-4.1/efm.nodes
```

5. スタンバイノードの `/etc/edb/efm-4.1/efm.nodes` ファイルにプライマリノードの `ip address` と `EFM PORT` を追加します。

```
$ sudo cat > /etc/edb/efm-4.1/efm.nodes <<< _EOF_
172.16.161.200:7800
_EOF_
```

6. サンプルファイル (`/etc/edb/efm-4.1/efm.properties.in`) を使用して、すべてのノードで `efm.properties` ファイルを作成し、すべてのユーザーに読み取りアクセスを許可します。

```
$ sudo cp /etc/edb/efm-4.1/efm.properties.in /etc/edb/efm-4.1/efm.properties
$ sudo chown efm:efm /etc/edb/efm-4.1/efm.properties
$ sudo chmod a+r /etc/edb/efm-4.1/efm.properties
```

7. `efm` ユーティリティを使用して `efm` ユーザーのパスワードを暗号化します。

```
$ export EFMPASS=password
$ /usr/edb/efm-4.1/bin/efm encrypt efm --from-env
096666746b05b081d1a98e43d94c9dad
```

8. プロパティファイルで次のパラメーターを編集します。

```
db.user=efm
db.password.encrypted=096666746b05b081d1a98e43d94c9dad
db.port=5444
```

(次のページに続く)

(前のページからの続き)

```
db.database=edb
db.service.owner=enterprisedb
db.service.name=edb-as-13
db.bin=/usr/edb/as13/bin
db.data.dir=/var/lib/edb/as13/data
jdbc.sslmode=require
user.email=username@example.com
from.email=node1@efm-pem
notification.level=INFO
notification.text.prefix=[PEM/EFM]
bind.address=172.16.161.200:7800
admin.port=7809
is.witness=false
local.period=10
local.timeout=60
local.timeout.final=10
remote.timeout=10
node.timeout=50
encrypt.agent.messages=true
stop.isolated.primary=true
stop.failed.primary=true
primary.shutdown.as.failure=false
update.physical.slots.period=0
ping.server.ip=8.8.8.8
ping.server.command=/bin/ping -q -c3 -w5
auto.allow.hosts=false
stable.nodes.file=false
db.reuse.connection.count=0
auto.failover=true
auto.reconfigure=true
promotable=true
use.replay.tiebreaker=true
standby.restart.delay=0
reconfigure.num.sync=false
reconfigure.sync.primary=false
minimum.standbys=0
recovery.check.period=1
restart.connection.timeout=60
auto.resume.period=0
virtual.ip=172.16.161.245
virtual.ip.interface=ens33
virtual.ip.prefix=24
virtual.ip.single=true
check.vip.before.promotion=true
pgpool.enable=false
```

(次のページに続く)

(前のページからの続き)

```

sudo.command=sudo
sudo.user.command=sudo -u %u
syslog.host=localhost
syslog.port=514
syslog.protocol=UDP
syslog.facility=LOCAL1
file.log.enabled=true
syslog.enabled=false
jgroups.loglevel=INFO
efm.loglevel=INFO
jvm.options=-Xmx128m
script.remote.post.promotion=/usr/local/bin/stop-pemagent.sh
script.post.promotion=/usr/local/bin/start-pemagent.sh

```

9. 監視ノードの `is.witness` 構成パラメーターの値を `true` に設定します。

```
is.witness=true
```

10. プライマリノードで EFM サービスを有効にして開始します。

```

$ systemctl enable edb-efm-4.1
$ systemctl start edb-efm-4.1

```

11. プライマリノードで起動されたクラスターにスタンバイが参加できるようにします。

```

/usr/edb/efm-4.1/bin/efm allow-node efm 172.16.161.201
/usr/edb/efm-4.1/bin/efm allow-node efm 172.16.161.202
/usr/edb/efm-4.1/bin/efm allow-node efm 172.16.161.203

```

12. スタンバイノードと EFM 監視ノードで EFM サービスを有効にして開始します。

```

$ systemctl enable edb-efm-4.1
$ systemctl start edb-efm-4.1

```

13. 任意のノードから EFM クラスターのステータスを確認します。

```

$ sudo /usr/edb/efm-4.1/bin/efm cluster-status efm
Cluster Status: efm

```

Agent Type	Address	DB	VIP
Primary	172.16.161.200	UP	172.16.161.245*
Standby	172.16.161.201	UP	172.16.161.245
Standby	172.16.161.202	UP	172.16.161.245
Witness	172.16.161.203	N/A	172.16.161.245

```

Allowed node host list:

```

(次のページに続く)

```
172.16.161.200 172.16.161.201 172.16.161.202 172.16.161.203
```

```
Membership coordinator: 172.16.161.200
```

```
Standby priority host list:
```

```
172.16.161.201 172.16.161.202
```

```
Promote Status:
```

DB Type	Address	WAL Received LSN	WAL Replayed LSN	Info
Primary	172.16.161.200		0/F7A3808	
Standby	172.16.161.201	0/F7A3808	0/F7A3808	
Standby	172.16.161.202	0/F7A3808	0/F7A3808	

```
Standby database(s) in sync with primary. It is safe to promote.
```

このステータスは、EFM が正常に設定され、PEM サーバーのフェールオーバーを管理していることを確認します。

フェールオーバーの場合、スタンバイのいずれかがプライマリノードとして昇格します。PEM エージェントは自動的に新しいプライマリノードに接続します。この手順を使用して、障害が発生したプライマリノードを新しいスタンバイノードに置き換えることができます。

2.1.2.1.5 現在の制限

現在の制限には次のものがあります。

- 切り替え中にユーザーの Web コンソールセッションが失われます。
- **Preferences** ダイアログから行われたユーザーごとの設定は、ファイルシステム上のローカル構成ファイルに保存されるため失われます。
- **Backup**、**Restore**、および **Maintenance** ダイアログを介して開始されるバックグラウンドプロセス、およびそれらのログはシステム間で共有されず、スイッチオーバー中に失われます。

2.1.2.2 Connection pooling using pgBouncer

このドキュメントでは、非 Windows マシン上の PEM エージェントから Postgres Enterprise Manager (PEM) サーバーへの接続数を制限するための接続プーラーとしての pgBouncer の使用に関する詳細情報を提供します。

- PEM データベースサーバーの準備 – pgBouncer で使用する PEM データベースサーバーの準備に関する情報を提供します。
- pgBouncer の構成 – PEM データベースサーバーで動作するように pgBouncer を構成することに関する詳細情報を提供します。

- PEM エージェントの構成 – pgBouncer に接続するための PEM エージェントの構成に関する詳細情報を提供します。

PEM Web インタフェースの使用の詳細については、Accessing the web interface を参照してください。

このドキュメントでは、Postgres という用語を使用して、PostgreSQL または Advanced Server データベースを意味します。

2.1.2.2.1 PEM server and agent connection management mechanism

各 PEM エージェントは、個々のユーザーの SSL 証明書を使用して PEM データベースサーバーに接続します。たとえば、ID#1 のエージェントは、agent1 ユーザーを使用して PEM データベースサーバーに接続します。

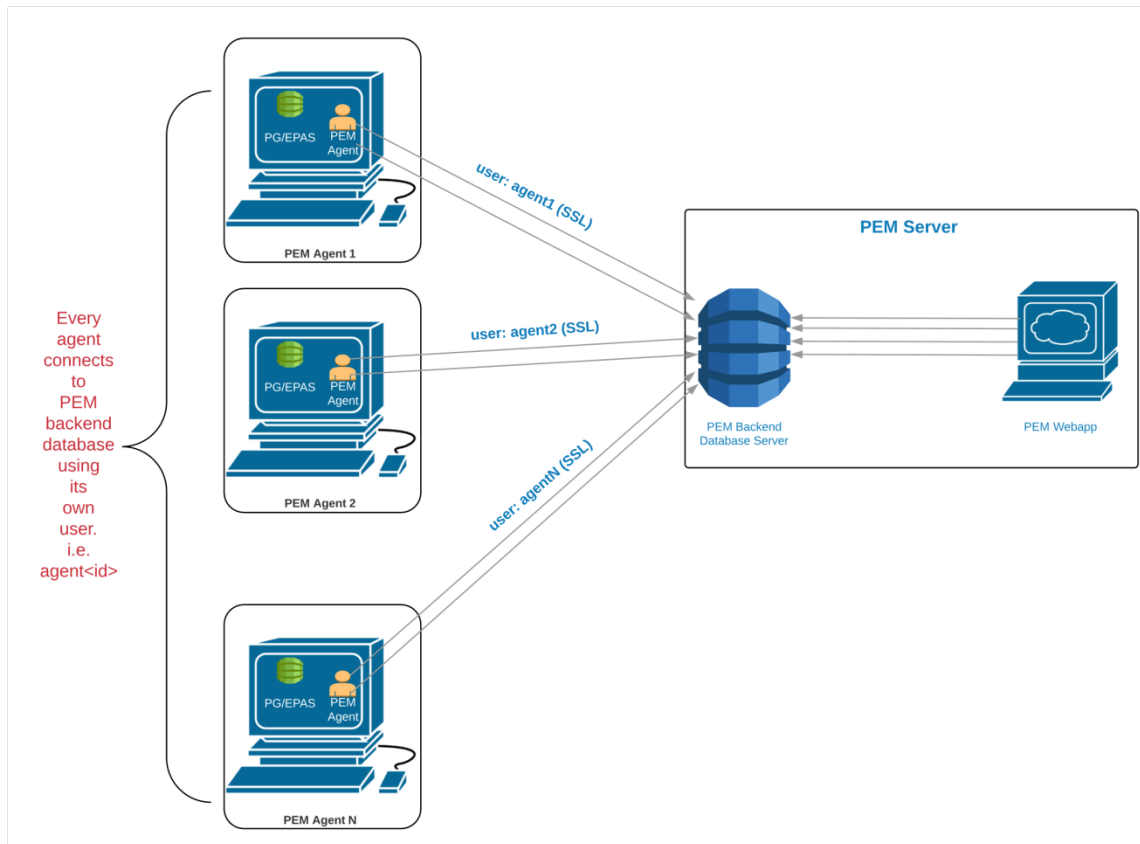


図 1: Connecting to the PEM database without pgBouncer

PEM バージョン 7.5 より前は、次の制限により、PEM サーバーと PEM エージェント間の接続プラーの使用が許可されませんでした。

- PEM エージェントは SSL 証明書を使用して PEM データベースサーバーに接続します。
- PEM データベースサーバーに接続するときに個々のユーザー識別子を使用します。

EDB は PEM エージェントを変更して、エージェントが（専用エージェントユーザーではなく）共通データベースユーザーを使用して PEM データベースサーバーに接続できるようにしました。

接続プラーとしてバージョン 1.9.0 以降の PgBouncer バージョンを使用することをお勧めします。バージョ

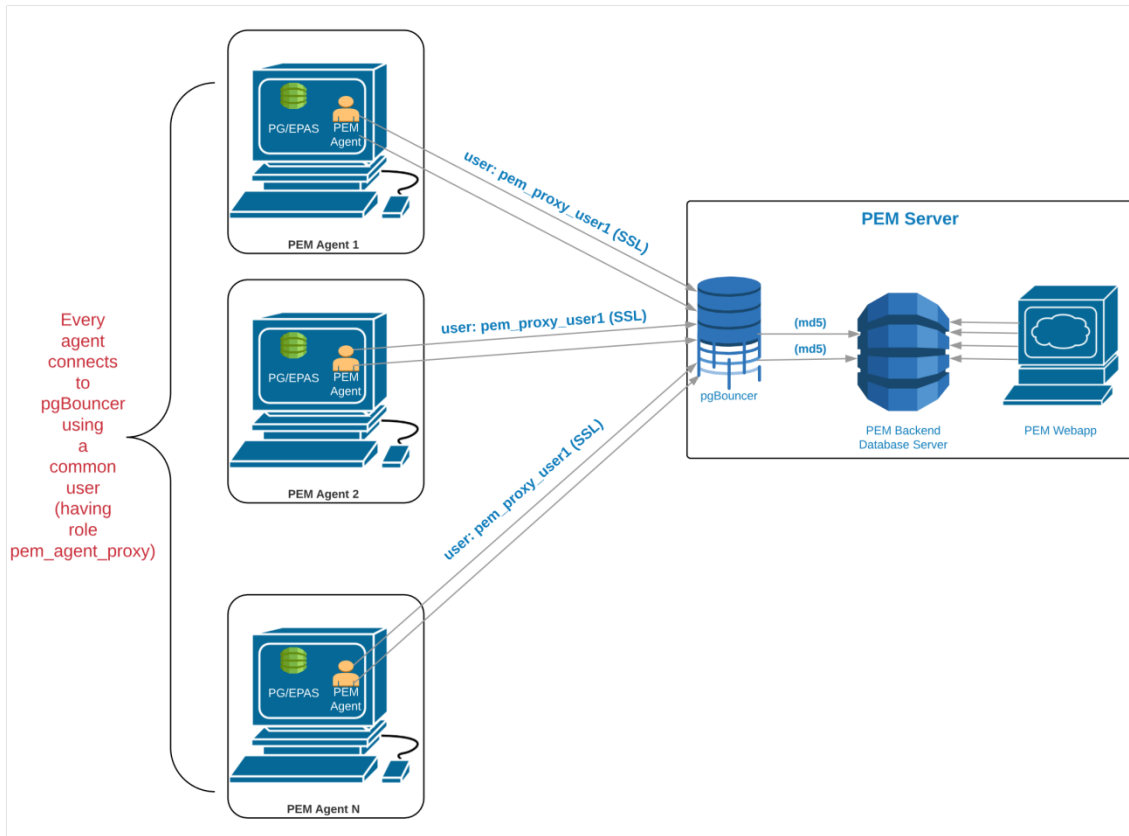


図 2: Connecting to pgBouncer.

ン 1.9.0 以降は `cert` 認証をサポートしています。PEM エージェントは、SSL 証明書を使用して pgBouncer に接続できます。

2.1.2.2.2 Preparing the PEM database server

PgBouncer と連携するように PEM データベースサーバーを構成する必要があります。次の例は、PEM データベースサーバーの構成に必要な手順を示しています。

1. PEM データベースサーバーで `pgbouncer` という名前の専用ユーザーを作成します。例：

```
pem=# CREATE USER pgbouncer PASSWORD ANY_PASSWORD LOGIN;
CREATE ROLE
```

2. PEM データベースサーバーに `pem_admin` および `pem_agent_pool role` メンバーシップを持つ `pem_admin1`（非スーパーユーザー）という名前のユーザーを作成します。例：

```
pem=# CREATE USER pem_admin1 PASSWORD ANY_PASSWORD LOGIN
CREATEROLE;
CREATE ROLE
pem=# GRANT pem_admin, pem_agent_pool TO pem_admin1;
GRANT ROLE
```

3. `pem` データベースで `pgbouncer` ユーザーに `CONNECT` 特権を付与します。例：

```
pem=# GRANT CONNECT ON DATABASE pem TO pgbouncer ;GRANT USAGE ON
SCHEMA pem TO pgbouncer;
GRANT
```

4. pem データベースの pem スキーマの pgbouncer ユーザーに USAGE 特権を付与します。例：

```
pem=# GRANT USAGE ON SCHEMA pem TO pgbouncer;
GRANT
```

5. pem データベースの pem.get_agent_pool_auth(text) ファンクションで pgbouncer ユーザーに EXECUTE 権限を付与します。例：

```
pem=# GRANT EXECUTE ON FUNCTION pem.get_agent_pool_auth(text) TO
pgbouncer;
GRANT
```

6. pem.create_proxy_agent_user(varchar) ファンクションを使用して、PEM データベースサーバーに pem_agent_user1 という名前のユーザーを作成します。例：

```
pem=# SELECT pem.create_proxy_agent_user(pem_agent_user1);
__OUTPUT__
create_proxy_agent_user
-----
(1 row)
```

この機能は、ランダムなパスワードで同じ名前のユーザーを作成し、ユーザーに pem_agent および pem_agent_pool ロールを付与します。これにより、pgBouncer はエージェントに代わってプロキシユーザーを使用できます。

7. PEM データベースサーバーの pg_hba.conf ファイルの先頭に次のエントリを追加します。これにより、pgBouncer ユーザーは md5 認証方法を使用して pem データベースに接続できます。例：

```
# Allow the PEM agent proxy user (used by
# pgbouncer) to connect the to PEM server using
# md5

local pem pgbouncer,pem_admin1 md5
```

2.1.2.2.3 Configuring PgBouncer

PEM データベースサーバーと連携するように PgBouncer を構成する必要があります。この例では、enterprisedb システムユーザーとして PgBouncer を実行します。次の手順は、pgBouncer (バージョン >= 1.9) を構成するプロセスの概要を示しています。

1. ターミナルウィンドウを開き、pgBouncer ディレクトリに移動します。
2. pgBouncer (pgbouncer.ini が存在する場所) の etc ディレクトリの所有者を enterprisedb に変更し、ディレクトリのアクセス許可を 0700 に変更します。例：

```
$ chown enterprisedb:enterprisedb /etc/edb/pgbouncer1.9
$ chmod 0700 /etc/edb/pgbouncer1.9
```

3. pgbouncer.ini または edb-pgbouncer.ini ファイルの内容を次のように変更します。

```
[databases]
;; Change the pool_size according to maximum connections allowed
;; to the PEM database server as required.
;; auth_user will be used for authenticate the db user (proxy
;; agent user in our case)

pem = port=5444 host=/tmp dbname=pem auth_user=pgbouncer
pool_size=80 pool_mode=transaction
* = port=5444 host=/tmp dbname=pem auth_user=pgbouncer
pool_size=10

[pgbouncer]
logfile = /var/log/edb/pgbouncer1.9/edb-pgbouncer-1.9.log
pidfile = /var/run/edb/pgbouncer1.9/edb-pgbouncer-1.9.pid
listen_addr = *
;; Agent needs to use this port to connect the pem database now
listen_port = 6432
;; Require to support for the SSL Certificate authentications
;; for PEM Agents
client_tls_sslmode = require
;; These are the root.crt, server.key, server.crt files present
;; in the present under the data directory of the PEM database
;; server, used by the PEM Agents for connections.
client_tls_ca_file = /var/lib/edb/as11/data/root.crt
client_tls_key_file = /var/lib/edb/as11/data/server.key
client_tls_cert_file = /var/lib/edb/as11/data/server.crt
;; Use hba file for client connections
auth_type = hba
;; Authentication file, Reference:
;; https://pgbouncer.github.io/config.html#auth_file
auth_file = /etc/edb/pgbouncer1.9/userlist.txt
;; HBA file
auth_hba_file = /etc/edb/pgbouncer1.9/hba_file
;; Use pem.get_agent_pool_auth(TEXT) function to authenticate
;; the db user (used as a proxy agent user).
auth_query = SELECT * FROM pem.get_agent_pool_auth($1)
;; DB User for administration of the pgbouncer
admin_users = pem_admin1
;; DB User for collecting the statistics of pgbouncer
stats_users = pem_admin1
server_reset_query = DISCARD ALL
```

(次のページに続く)

(前のページからの続き)

```
;; Change based on the number of agents installed/required
max_client_conn = 500
;; Close server connection if its not been used in this time.
;; Allows to clean unnecessary connections from pool after peak.
server_idle_timeout = 60
```

4. 次のコマンドを使用して、PgBouncer の/etc/edb/pgbouncer1.9/userlist.txt 認証ファイルを作成および更新します。

```
pem=# COPY (
SELECT pgbouncer::TEXT, pgbouncer_password
UNION ALL
SELECT pem_admin1::TEXT, pem_admin1_password
TO /etc/edb/pgbouncer1.9/userlist.txt
WITH (FORMAT CSV, DELIMITER , FORCE_QUOTE *);

COPY 2
```

注釈: スーパーユーザーは PEM 認証クエリ機能 `pem.get\_proxy\_auth(text)` を呼び出すことができません。`pem\_admin` ユーザーがスーパーユーザーの場合、パスワードを認証ファイル (*enterprisedb in the above example*) に追加する必要があります。

5. 次のコンテンツを含む PgBouncer 用の HBA ファイル (/etc/edb/pgbouncer1.9/hba_file) を作成します。

```
# Use authentication method md5 for the local connections to
# connect pem database & pgbouncer (virtual) database.
local pgbouncer all md5
# Use authentication method md5 for the remote connections to
# connect to pgbouncer (virtual database) using enterprisedb
# user.

host pgbouncer,pem pem_admin1 0.0.0.0/0 md5
```

```
# Use authentication method cert for the TCP/IP connections to
# connect the pem database using pem_agent_user1

hostssl pem pem_agent_user1 0.0.0.0/0 cert
```

6. HBA ファイル (/etc/edb/pgbouncer1.9/hba_file) の所有者を enterprisedb に変更し、ディレクタリのパーミッションを 0600 に変更します。例：

```
$ chown enterprisedb:enterprisedb /etc/edb/pgbouncer1.9/hba_file
$ chmod 0600 /etc/edb/pgbouncer1.9/hba_file
```

7. PgBouncer サービスを有効にし、サービスを開始します。例：

```
$ systemctl enable edb-pgbouncer-1.9

Created symlink from
/etc/systemd/system/multi-user.target.wants/edb-pgbouncer-1.9.service
to /usr/lib/systemd/system/edb-pgbouncer-1.9.service.

$ systemctl start edb-pgbouncer-1.9
```

2.1.2.2.4 Configuring the PEM agent

RPM パッケージを使用して PEM エージェントをインストールできます。詳細なインストール情報については、

Installing agent on Linux を参照してください。

SNMP 通知の送信を担当する PEM エージェントは pgBouncer で構成しないでください。たとえば、PEM サーバーとともにインストールされるデフォルトの PEM エージェントが SNMP 通知に使用される場合、pgBouncer で構成しないでください。

新しい PEM エージェントの構成（RPM 経由でインストール）

RPM パッケージを使用して PEM エージェントをインストールした後、特定の PEM データベースサーバーに対して動作するように構成する必要があります。次のコマンドを使用します。

```
$ PGSSLMODE=require PEM_SERVER_PASSWORD=pem_admin1_password
/usr/edb/pem/agent/bin/pemworker --register-agent --pem-server
pem_agent_user1 --display-name "Agent Name"
```

```
Postgres Enterprise Manager Agent registered successfully!
```

上記のコマンドでは、`--pem-agent-user` 引数は、`/root/.pem` ディレクトリに `pem_agent_user1` データベースユーザーの SSL 証明書とキーペアを作成するようにエージェントに指示します。

例：

```
/root/.pem/pem_agent_user1.crt

/root/.pem/pem_agent_user1.key
```

キーは、PEM エージェントが `pem_agent_user1` として PEM データベースサーバーに接続するために使用されます。`/usr/edb/pem/agent/etc/agent.cfg` という名前のエージェント構成ファイルも作成します `agent.cfg` 構成ファイルで使われる `agent-user` を記載した行があります。

例：

```
$ cat /usr/edb/pem/agent/etc/agent.cfg
[PEM/agent]
pem_host=172.16.254.22
pem_port=6432
agent_id=12
agent_user=pem_agent_user1
agent_ssl_key=/root/.pem/pem_agent_user1.key
agent_ssl_cert=/root/.pem/pem_agent_user1.crt
log_level=warning
log_location=/var/log/pem/worker.log
agent_log_location=/var/log/pem/agent.log
long_wait=30
short_wait=10
alert_threads=0
enable_smtp=false
enable_snmp=false
enable_webhook=false
max_webhook_retries=3
allow_server_restart=true
allow_package_management=false
allow_streaming_replication=false
max_connections=0
connect_timeout=-1
connection_lifetime=0
allow_batch_probes=false
heartbeat_connection=false
```

既存の PEM エージェントの構成（RPM 経由でインストール）

既存の PEM エージェントを使用している場合、SSL 証明書とキーファイルをターゲットマシンにコピーし、ファイルを再利用できます。ファイルを変更し、新しいパラメーターを追加して、既存の `agent.cfg` ファイルのいくつかのパラメーターを置き換える必要があります。

エージェントで使用する `agent_user` の行を追加します。例：

```
agent_user=pem_agent_user1
```

ポートを更新して pgBouncer ポートを指定します。例：

```
pem_port=6432
```

証明書とキーのパスの場所を更新します。例：

```
agent_ssl_key=/root/.pem/pem_agent_user1.key
agent_ssl_cert=/root/.pem/pem_agent_user1.crt
```

注：別の方法として、エージェントの自己登録スクリプトを実行できますが、新しいエージェント ID が作成されます。エージェントの自己登録スクリプトを実行する場合は、新しいエージェント ID を既存の ID に置き換え、`pem.agent` テーブルの新しいエージェント ID のエントリを無効にする必要があります。例：

```
pem=# UPDATE pem.agent SET active = false WHERE id = <new_agent_id>;
```

```
__OUTPUT__
```

```
UPDATE 1
```

注釈： 既存の SSL 証明書、キーファイル、およびエージェント構成ファイルのバックアップを保持します。

2.1.2.3 Authentication options

PEM は、Kerberos および 2FA 認証もサポートしています。実装手順については、以下を参照してください。

Linux では、

- *Configuring the PEM server to use Kerberos authentication*
- Configuring the PEM server to use Windows Kerberos server

Linux および Windows、

- Configuring 2FA authentication

2.1.2.3.1 Configuring the PEM server to use Kerberos authentication

- Kerberos サーバーは、ip addresses ではなくホスト名で動作します。Kerberos Authentication を使用して PEM サーバーでシングルサインオンを使用するには、次のマシンを DNS (REALM) を使用してホスト名で構成する必要があります。
- Kerberos サーバー - PEM サーバー (PEM Web サーバーおよび PEM バックエンドデータベースサーバー) - クライアントマシン
- たとえば、Kerberos サーバーの REALM が `edbpem.org` である場合、Kerberos サーバーのホスト名を `Krb5server.edbpem.org`、PEM サーバーのホスト名を `pem.edbpem.org`、クライアントのホスト名を `pg12.edbpem.org` に設定できます。
- レルムの名前として DNS ドメイン名を使用するのは慣習です。

手順に従って、PEM サーバーの Kerberos 認証を構成します。

1. Kerberos、PEM サーバー、PEM バックエンドデータベースをインストールする

- ・ 認証サーバーとして機能するマシンに Kerberos をインストールします。
- ・ 別のマシンに PEM サーバーをインストールします。詳細については、[Installing the PEM Server](#) を参照してください。
- ・ PEM サーバーと同じまたは別のマシンに PEM バックエンドデータベース（Postgres / EDB Postgres Advanced Server）をインストールします。詳細については、[EDB Docs Website](#) のインストール手順を参照してください。

2. Kerberos サーバーにプリンシパルを追加する

- ・ Apache Web サーバー（HTTPD / Apache2）および PEM バックエンドデータベースサーバー（PostgreSQL / EDB Postgres Advanced Server）の下にデプロイされた PEM Web アプリケーションのプリンシパルを追加します。

```
$ sudo kadmin.local -q "addprinc -randkey HTTP/<HOSTNAME_OF_PEM_SERVER>"
$ sudo kadmin.local -q "addprinc -randkey postgres/<HOSTNAME_OF_PEM_SERVER>"
```

HOSTNAME_OF_PEM_SERVER には、Kerberos Server の REALM が含まれている必要があります。例- REALM として edbpem.org を使用して、PEM サーバーのホスト名として pemdb.edbpem.org を指定できます。

注釈:

- ・ PEM Web アプリケーションと PEM バックエンドデータベースサーバーが異なるマシンにある場合、両方のホスト名は異なります。

3. Kerberos サーバーからキーテーブルを抽出する

- PEM Web アプリケーションと PEM バックエンドデータベースサーバーの Kerberos からキーテーブルを抽出します。

```
sudo kadmin.local "ktadd -k <NAME_OF_PEM_WEB_FILE>.keytab HTTP/<HOSTNAME_OF_PEM_
→SERVER>"
sudo kadmin.local "ktadd -k <NAME_OF_PEM_DB_FILE>.keytab postgres/<HOSTNAME_OF_PEM_
→SERVER>"
```

- Kerberos サーバーから PEM サーバーにキーテーブルをコピーします。

```
scp <NAME_OF_PEM_WEB_FILE>.keytab <OS_USERNAME_ON_PEM_SERVER>@<HOSTNAME_OF_PEM_SERVER>
→:/tmp
scp <NAME_OF_PEM_DB_FILE>.keytab <OS_USERNAME_ON_PEM_SERVER>@<HOSTNAME_OF_PEM_SERVER>
→:/tmp
```

- PEM サーバーで、キーテーブルを必要な場所に移動し、次のように所有権を変更します。

```
mv /tmp/<NAME_OF_PEM_WEB_FILE>.keytab <PEM_INSTALLATION_DIRECTORY>/share
chown pem <PEM_INSTALLATION_DIRECTORY>/share/<NAME_OF_PEM_WEB_FILE>.keytab
```

```
mv /tmp/<NAME_OF_PEM_DB_FILE>.keytab <DATA_DIRECTORY_OF_POSTGRES>/
chown enterprisedb <DATA_DIRECTORY_OF_POSTGRES>/<NAME_OF_PEM_DB_FILE>.keytab
```

どこ、

- NAME_OF_PEM_WEB_FILE は、PEM Web アプリケーションの Key Table に指定した名前です。
- NAME_OF_PEM_DB_FILE は、PEM バックエンドデータベースサーバーのキーテーブルに指定された名前です。
- OS_USERNAME_ON_PEM_SERVER は、PEM サーバーのオペレーティングシステムユーザーの名前です。
- DATA_DIRECTORY_OF_POSTGRES は、インストールされた postgres (PostgreSQL/ EDB Postgres Advanced Server) のデータディレクトリのパスです。

4. PEM バックエンドデータベースサーバーを構成する

- postgresql.conf ファイルにキーテーブルの場所を追加

```
krb_server_keyfile=FILE: /<DATA_DIRECTORY_OF_POSTGRES>/<NAME_OF_PEM_DB_FILE>.keytab
```

どこ、

- NAME_OF_PEM_DB_FILE は、PEM バックエンドデータベースサーバーのキーテーブルに指定された名前です。
- DATA_DIRECTORY_OF_POSTGRES は、インストールされた postgres のデータディレクトリのパスです (PostgreSQL / EDB Postgres Advanced Server).
- krb5.conf ファイルを編集します

```
$ sudo vim /etc/krb5.conf
[libdefaults]
default_realm = EDBPEM.ORG
Forwardable = True

[domain_realm]
.edbpem.org = EDBPEM.ORG
edbpem.org = EDBPEM.ORG

[realms]
EDBPEM.ORG = {
    kdc = krb5server.edbpem.org
    admin_server = krb5server.edbpem.org
}
```

- データベースサーバを再起動して変更を反映します

```
systemctl restart <POSTGRES_SERVICE_NAME>
```

どこ、

- `POSTGRES_SERVICE_NAME` は Postgres (PostgreSQL/ EDB Postgres Advanced Server) データベースのサービス名です。例: CentOS、RHEL または Rocky Linux プラットフォーム上の PostgreSQL 13 データベースの場合は `postgresql-13`。

5. 初期チケットを取得して表示する

- `kinit` ユーティリティは、Kerberos チケットを取得してキャッシュします。このユーティリティは通常、ユーザーが入力したパスワードを使用して、キー配布センター (KDC) から資格情報を復号化し、チケット保証チケットを取得するために使用されます。チケット保証チケットは、ユーザーの資格情報キャッシュに保存されます。
- `klist` ユーティリティを使用してチケットの詳細を表示できます。

注釈: `kinit` および `klist` を使用するには、PEM サーバーとクライアントマシンに `Kerberos Client` をインストールする必要があります。

```
$ kinit <USERNAME@REALM>
$ klist
```

プリンシパルと Kerberos チケットが表示されます。

注釈: ここで指定する `USERNAME@REALM` は、`pem` データベースに対する `pem_admin` ロールと `CONNECT` 特権を持つデータベースユーザーである必要があります。

6. PEM サーバーを構成する

- PEM サーバーで PEM 構成スクリプトを実行して、Kerberos 認証を使用します

```
$ sudo PEM_APP_HOST=<HOSTNAME_OF_PEM_SERVER> PEM_KRB_KTNAME=<PEM_INSTALLATION_
→DIRECTORY/share/<NAME_OF_PEM_WEB_FILE>.keytab <PEM_INSTALLATION_DIRECTORY>/bin/
→configure-pem-server.sh
```

- `config_setup.py` ファイルで `PEM_DB_HOST` を構成し、`PEM_AUTH_METHOD` の値が `'kerberos'` に設定されていることを確認します。 `edb_notranlate_10`
- `.install-config` ファイルで `HOST` を構成します

```
$ sudo vim <PEM_INSTALLATION_DIRECTORY>/share/.install-config
HOST=`<HOSTNAME_OF_PEM_SERVER>`
```

- PEM サーバーが Kerberos 認証を使用する場合、
- すべての監視対象サーバーはデフォルトで同じ認証を使用します。デフォルトをオーバーライドするには、パラメーター `ALLOW_DATABASE_CONNECTION_WITHOUT_KERBEROS` を追加し、`config_local.py` ファイルで `True` に設定します。
- すべての認証されたユーザープリンシパルにはレルム (`USERNAME@REALM`) が追加され、デフォルトでデータベースユーザー名として渡されます。デフォルトをオーバーライドするには、パラメーター `PEM_USER_KRB_INCLUDE_REALM` を追加し、`config_local.py` ファイルで `False` に設定します。
- Apache サーバーを再起動します

```
sudo systemctl restart <SERVICE_NAME>
```

- `pg_hba.conf` の上部のエントリを編集して `gss` 認証方法を使用し、データベースサーバーをリロードします。

```
host    pem          +pem_user    <ip_of_pem_server>/32    gss
host    postgres     +pem_user    <ip_of_pem_server>/32    gss
```

```
systemctl reload <POSTGRES_SERVICE_NAME>
```

どこ、

- `POSTGRES_SERVICE_NAME` は Postgres (PostgreSQL/ EDB Postgres Advanced Server) データベースのサービス名です。例：CentOS、RHEL または Rocky Linux プラットフォーム上の PostgreSQL 13 データベースの場合は `postgresql-13`。

注釈: PostgreSQL または EDB Postgres Advanced Server 12 以降を使用している場合は、接続タイプを `'hostgssenc'` として指定して、`gss` 暗号化接続のみを許可できます。

7. ブラウザ設定

Spnego/Kerberos を使用するように PEM Web クライアントにアクセスするようにクライアントマシンでブラウザを構成します。

- Mozilla Firefox の場合
- `about:config` ページをロードして、低レベルの Firefox 構成ページを開きます。 - [検索] テキストボックスに、次を入力します。 `network.negotiate-auth.trusted-uris - network.negotiate-auth.trusted-uris` 設定をダブルクリックし、Kerberos HTTP SPNEGO で保護されている Web サーバーのホスト名またはドメインを入力します。複数のドメインとホスト名はコンマで区切ります。 - [検索] テキストボックスに、次を入力します。 `network.negotiate-auth.delegation-uris - network.negotiate-auth.delegation-uris` 設定をダブルクリックし、Kerberos HTTP SPNEGO で保護されている Web サーバーのホスト名またはドメインを入力します。複数のドメインとホスト名はコンマで区切ります。 - [OK] をクリックします。
- Linux または MacOS の Google Chrome の場合：

- `--auth-server-whitelist` パラメーターを `google-chrome` コマンドに追加します。たとえば、Linux プロンプトから Chrome を実行するには、次のように `google-chrome` コマンドを実行します。

```
google-chrome --auth-server-whitelist = "hostname/domain"
```

- PEM サーバーを構成した後、ブラウザで PEM Web インターフェースにアクセスできます。次の場所に移動します。

```
https://<ip_address_of_PEM_server>:8443/pem
```

注釈: Postgres クラスターへの接続中に次のエラーが発生した場合。

```
psql -h hostname template1 psql: GSSAPI continuation error: Unspecified GSS failure.
Minor code may provide more information GSSAPI continuation error: Key version is not
available
```

解決策は、`ktutil` を使用してキータブに追加の暗号化タイプを追加するか、AD のすべての暗号化システムで Postgres キータブを再作成することです。

2.1.2.3.2 Configuring the PEM server to use Windows Active Directory Domain Services for Kerberos authentication (SSPI)

- Windows Active Directory ドメインサービスは、ホスト名で動作し、`ip addresses` では動作しません。Active Directory Domain Services を使用して PEM サーバーでシングルサインオンを使用するには、DNS を使用してホスト名で次のマシンを構成する必要があります。
- Windows Server (ドメインコントローラ) - PEM サーバー (PEM Web サーバーおよび PEM バックエンドデータベースサーバー) - クライアントマシン
- たとえば、Windows Active Directory の REALM が `edbpem.internal` である場合、Windows Server ホスト名を `Krb5server.edbpem.internal`、PEM サーバーホスト名を `pem.edbpem.internal`、クライアントのホスト名を `pg12.edbpem.internal` に設定できます。

手順に従って、PEM サーバーの Kerberos 認証を構成します。

1.Active Directory、PEM サーバー、および PEM バックエンドデータベースサーバーをインストールする

・ 認証サーバとして機能する Windows Server(ドメインコントローラ)に Active Directory をインストールします。また、Kerberos 認証を使用するように Active Directory ドメインサービスを構成すると、稼働しているはずですが。

- 別の Linux マシンに PEM サーバーをインストールします。詳細については、[Installing the PEM Server](#) を参照してください。
- PEM サーバーと同じまたは別の Linux マシンに PEM バックエンドデータベース (Postgres / EDB Postgres Advanced Server) をインストールします。詳細については、[EDB Docs Website](#) のインストー

ル手順を参照してください。

2. サービスプリンシパルにマップするユーザーを **Active Directory** に作成します

Windows Server の Active ディレクトリにユーザーを作成して、PEM Web アプリケーションの HTTP サービスプリンシパルにマッピングします。

- a. Active Directory Users and Computers を開く -> <DOMAIN_NAME> -> Users -> 右クリック -> New -> User

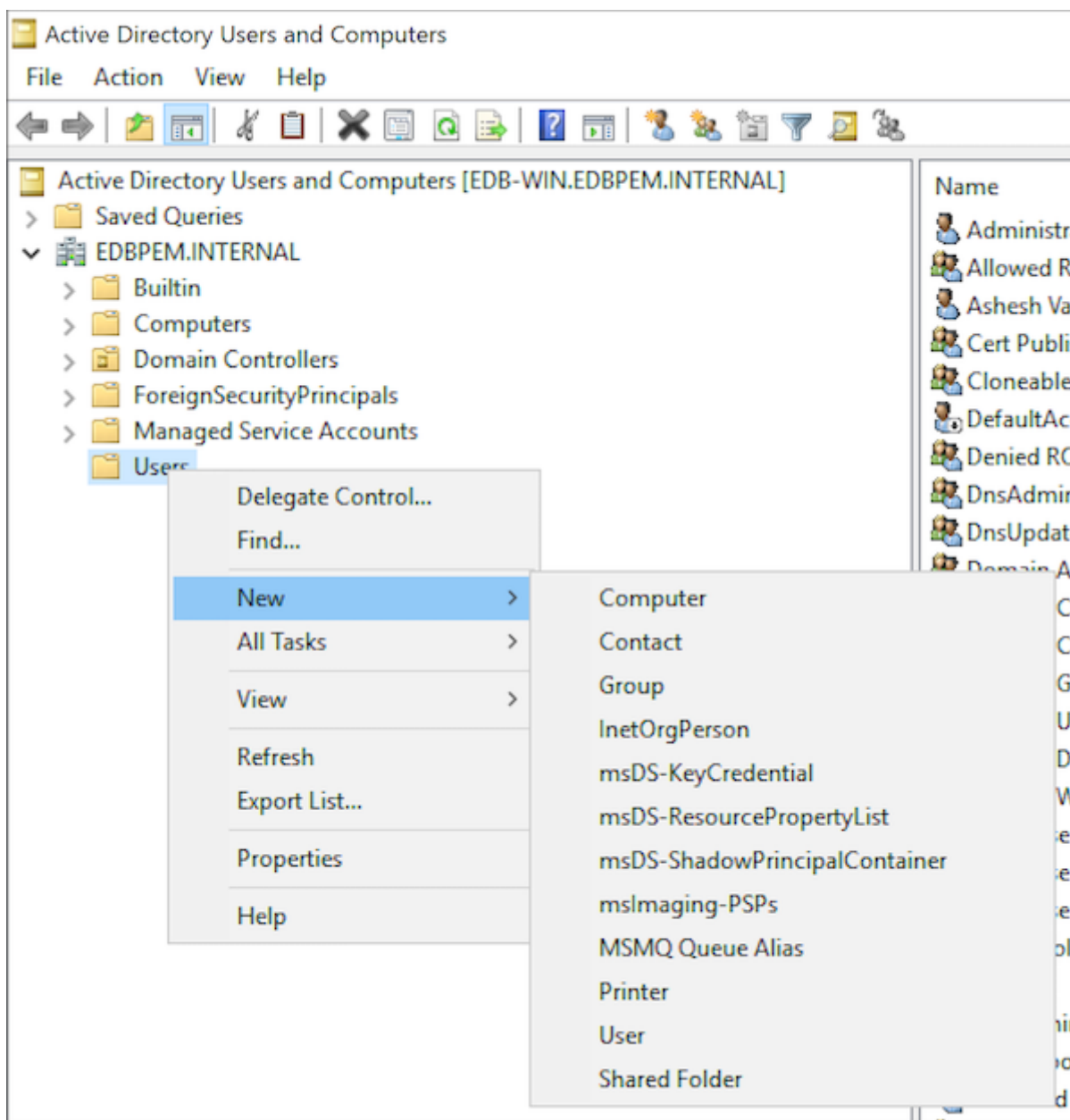
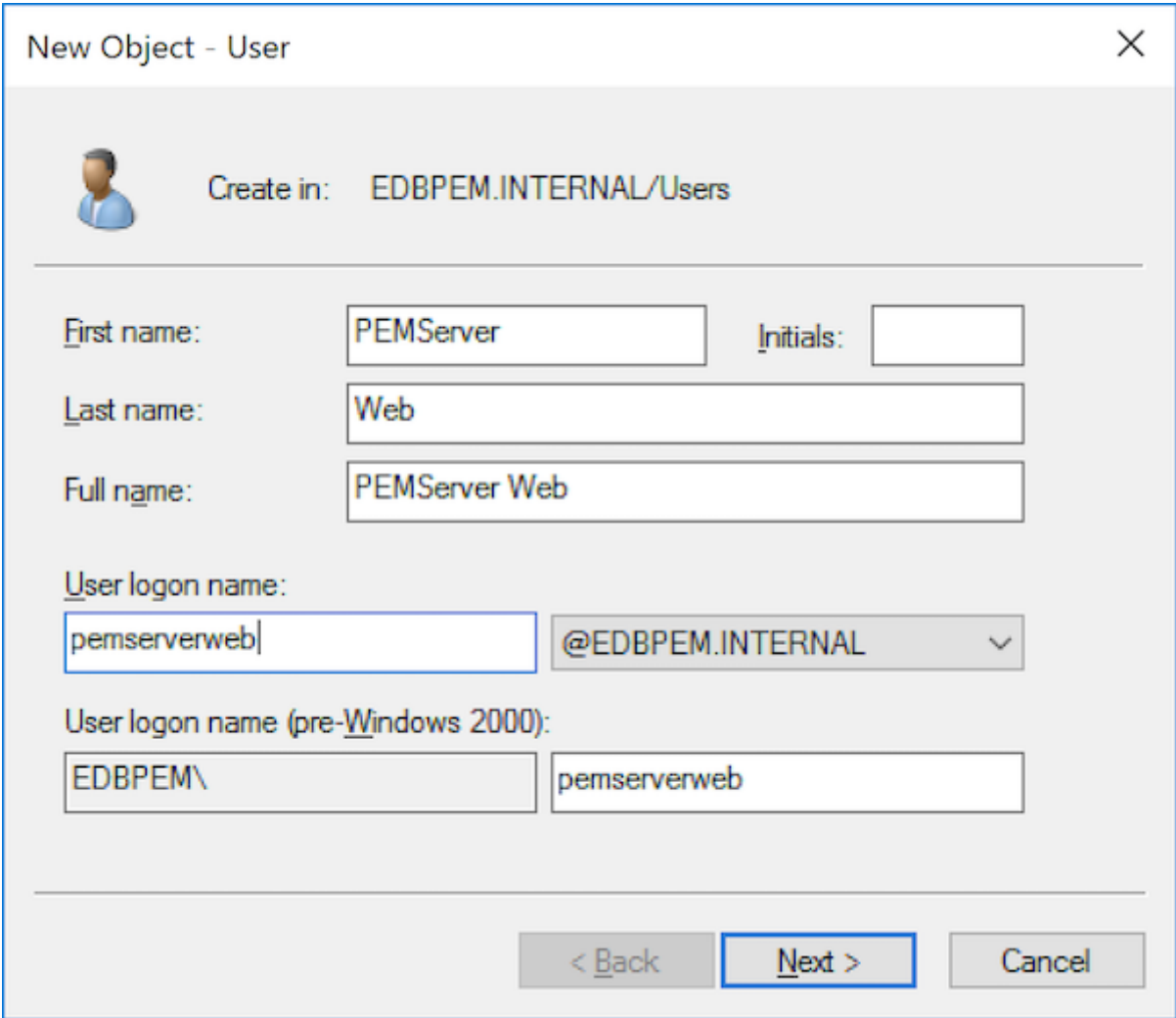


図 3: Active Directory Users and Computers

- b. ユーザーの詳細を入力します
- c. パスワードを入力し、必ず **User must change password at next login** の選択を解除し、**User cannot**



New Object - User

Create in: EDBPEM.INTERNAL/Users

First name: PEMServer Initials:

Last name: Web

Full name: PEMServer Web

User logon name:
pemserverweb @EDBPEM.INTERNAL

User logon name (pre-Windows 2000):
EDBPEM\ pemserverweb

< Back Next > Cancel

図 4: Active Directory Users and Computers Create User

change password と Password never expires を選択します。

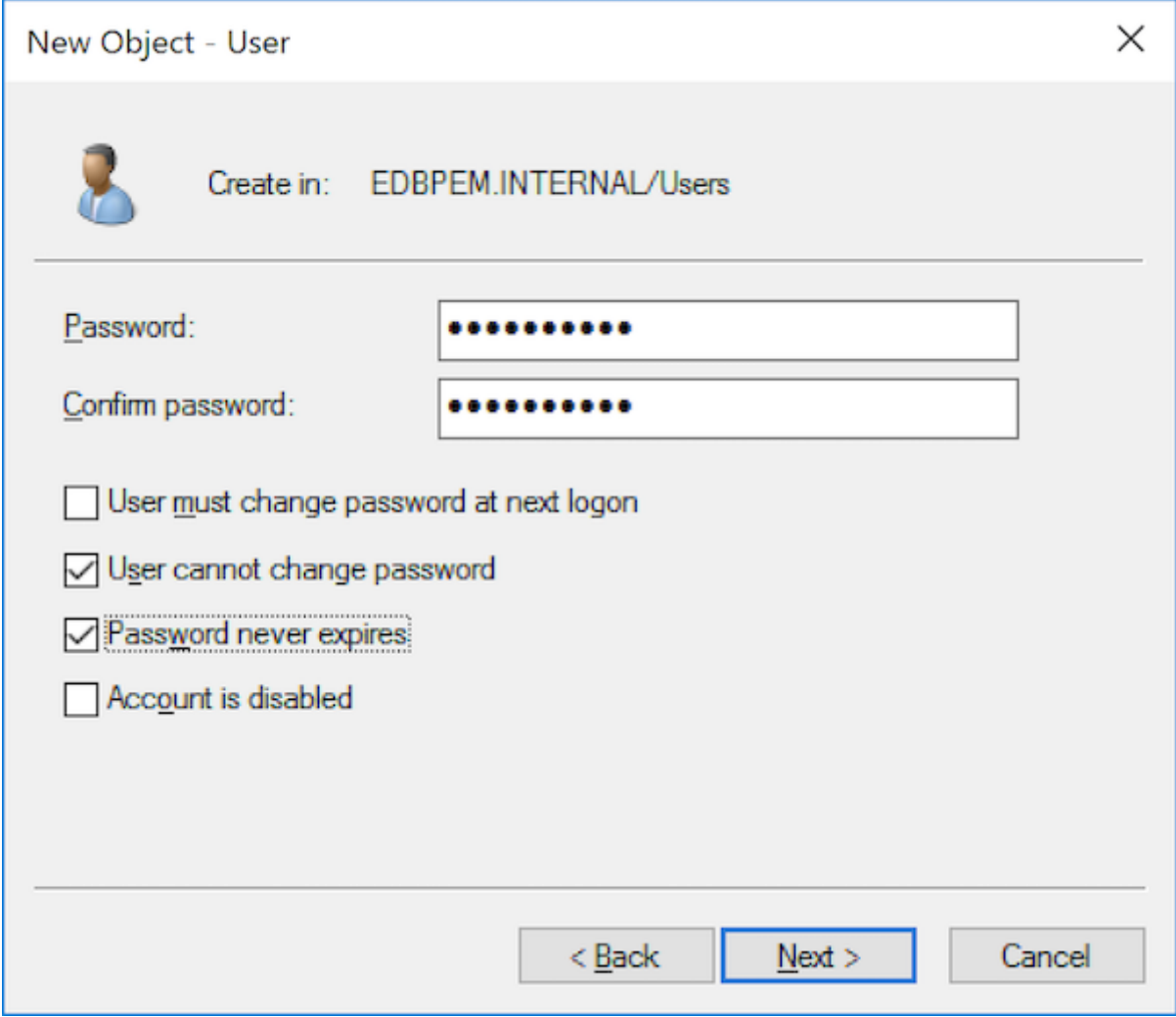


図 5: Active Directory Users and Computers set password

d. ユーザーの詳細を確認する

e. PEMServer Web Properties で、Users を Domain Users グループのメンバーとして追加します。

Windows Server の Active ディレクトリにユーザー（例：pemservedb）を作成して、PEM バックエンドデータベースの Postgres サービスプリンシパルにマッピングします。

上記の a から e までの手順を繰り返します。

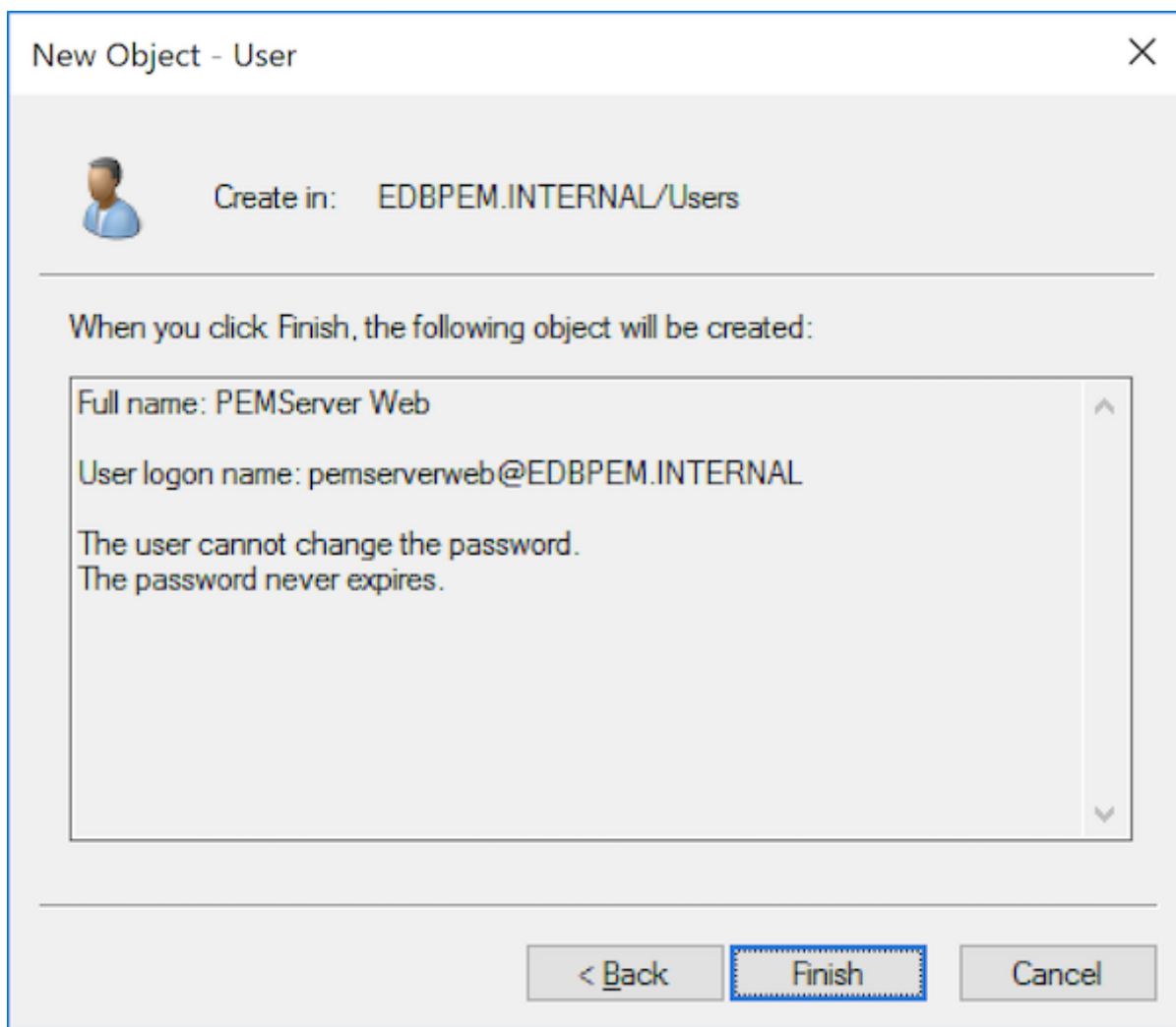


図 6: Active Directory Users and Computers User Details

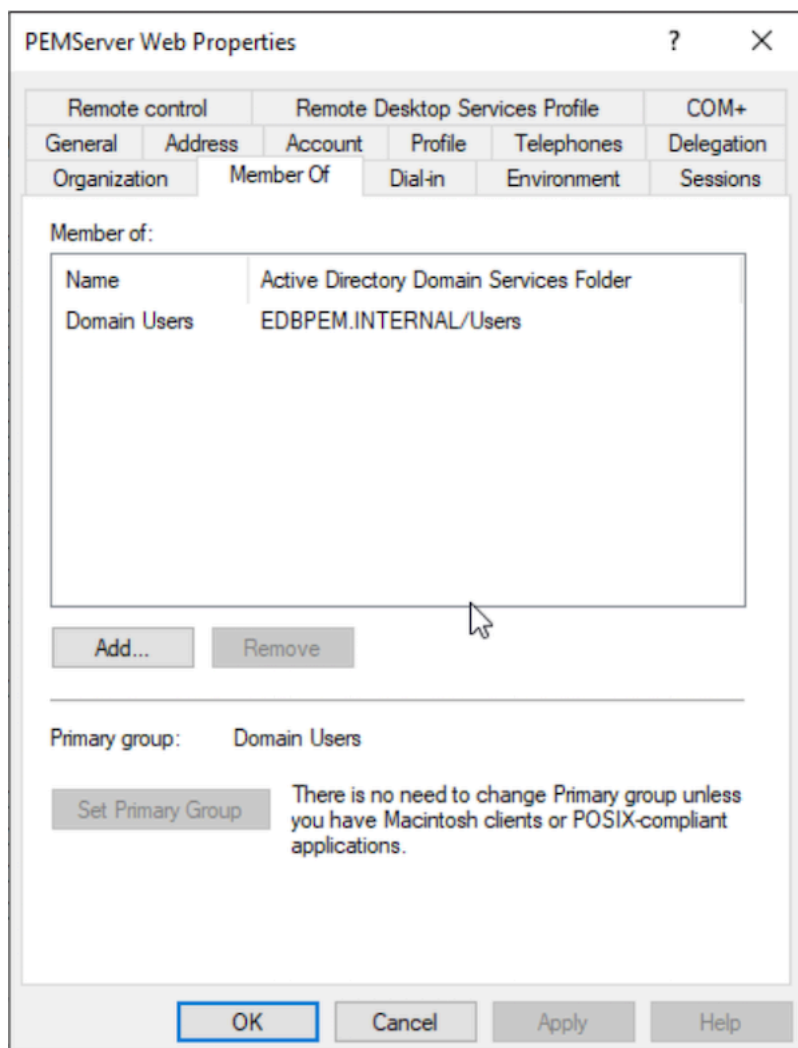


図 7: PEM Server Web Properties

3.Active Directory からキーテーブルを抽出する

- サービスプリンシパルのキーテーブルを抽出し、上記の手順で作成したそれぞれのドメインユーザーにマッピングします。
- a. Windows PowerShell を Administrator ユーザーとして開き、ユーザー pemserverweb で HTTP サービスプリンシパルマッピングのキーテーブルを作成し、ユーザー pemserverdb で Postgres サービスプリンシパルマッピングのキーテーブルを作成します。

```
ktpass /out pemserver.keytab /princ HTTP/pem.edbpem.internal@EDBPEM.INTERNAL /mapuser_
→pemserverweb /crypto AES256-SHA1 +rndpass /target EDBPEM.INTERNAL -ptype KRB5_NT_
→PRINCIPAL -kvno 0
ktpass /out pemdb.keytab /princ postgres/pem.edbpem.internal@EDBPEM.INTERNAL /mapuser_
→pemserverdb /crypto AES256-SHA1 +rndpass /target EDBPEM.INTERNAL -ptype KRB5_NT_
→PRINCIPAL -kvno 0
```

どこ、

- pemserver.keytab は PEM Web アプリケーションのキーテーブルの名前です
- pemdb.keytab は、PEM バックエンドデータベースサーバーのキーテーブルの名前です
- pem.edbpem.internal@EDBPEM.INTERNAL は PEM サーバーのホスト名です（ここで、@EDBPEM.INTERNAL は@REALM を意味します）
- pemserverweb は PEM Web アプリケーションのユーザーです
- pemserverdb は PEM バックエンドデータベースサーバーのユーザーです
- EDBPEM.INTERNAL はターゲットのドメインです

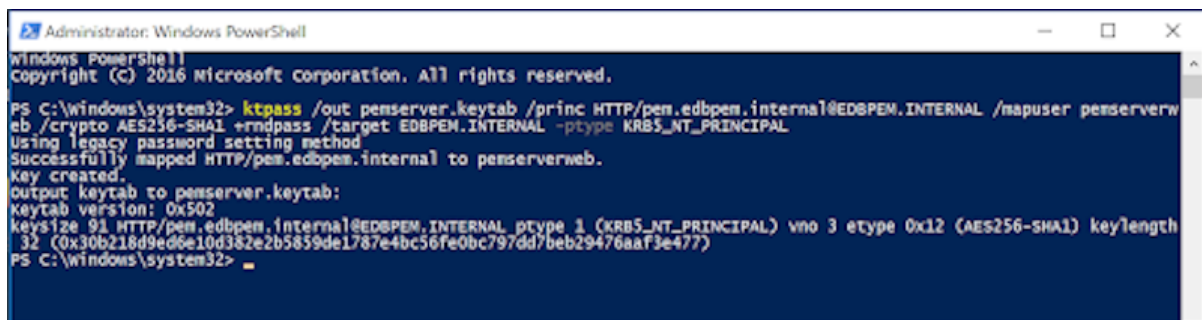


図 8: Windows PowerShell ktpass

注釈:

- コマンドライン引数 `+rndpass` は、ドメインユーザー `pemserverweb` のパスワードをランダムなパスワードにリセットします。`-/target` オプションは optional です。

- b. Accounts タブでユーザーアカウントの Kerberos サポートを追加します

PEMServer Web Properties

Organization Member Of Dial-in Environment Sessions
Remote control Remote Desktop Services Profile COM+
General Address Account Profile Telephones Delegation

User logon name:
HTTP/pem.edbpem.intenal @EDBPEM.INTERNAL

User logon name (pre-Windows 2000):
EDBPEM\ pemserverweb

Logon Hours... Log On To...

☐ Unlock account

Account options:

- ☐ Use only Kerberos DES encryption types for this account
- ☒ This account supports Kerberos AES 128 bit encryption.
- ☒ This account supports Kerberos AES 256 bit encryption.
- ☐ Do not require Kerberos preauthentication

Account expires

☒ Never

☐ End of: Wednesday, July 14, 2021

OK Cancel Apply Help

図 9: PEM Server Web Properties

注釈: [アカウント] タブで、*User logon name* に `HTTP/pem.edbpem.internal@EDBPEM.INTERNAL` が表示され、*Delegation* タブが `pemserverweb` ユーザーに対して有効になっています。

- c. 作成したユーザーの *Delegation* タブで *Trust this user for delegation to any service (Kerberos only)* オプションを選択します。
- d. 両方のキーテーブルを PEM サーバーホストにコピーします（または、別のホストにインストールされている場合はそれぞれ PEM Web サーバーと PEM バックエンドデータベースサーバーホスト）。
- e. PEM サーバーで、キーテーブルを必要な場所に移動し、次のように所有権を変更します。

```
mv /tmp/pemserver.keytab <PEM_INSTALLATION_DIRECTORY>/share
chown pem <PEM_INSTALLATION_DIRECTORY>/share/pemserver.keytab
```

```
mv /tmp/pemdb.keytab <DATA_DIRECTORY_OF_POSTGRES>/
chown enterprisedb <DATA_DIRECTORY_OF_POSTGRES>/pemdb.keytab
```

どこ、

- `OS_USERNAME_ON_PEM_SERVER` は、PEM サーバーのオペレーティングシステムユーザーの名前です。
- `DATA_DIRECTORY_OF_POSTGRES` は、インストールされた postgres (PostgreSQL/ EDB Postgres Advanced Server) のデータディレクトリのパスです。

4.PEM バックエンドデータベースサーバーを構成する

- `postgresql.conf` ファイルにキーテーブルの場所を追加

```
krb_server_keyfile=FILE: /<DATA_DIRECTORY_OF_POSTGRES>/pemdb.keytab
```

どこ、

- `DATA_DIRECTORY_OF_POSTGRES` は、インストールされた postgres のデータディレクトリのパスです (PostgreSQL / EDB Postgres Advanced Server).
- `krb5.conf` ファイルを編集します。

```
$ sudo vim /etc/krb5.conf
[libdefaults]
default_realm = EDBPEM.INTERNAL
Forwardable = True

[domain_realm]
.edbpem.org = EDBPEM.INTERNAL
edbpem.org = EDBPEM.INTERNAL

[realms]
```

(次のページに続く)

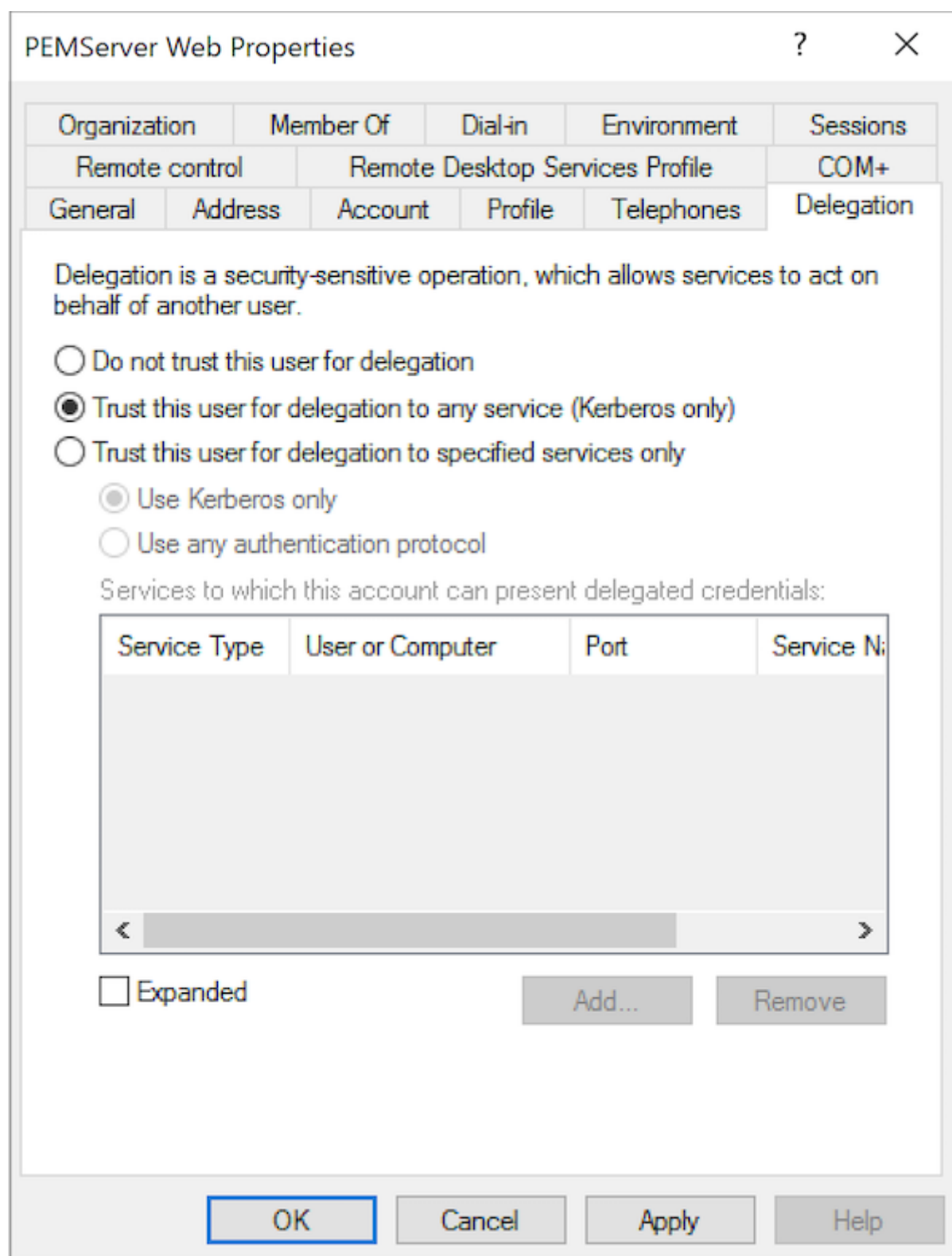


図 10: PEM Server Web Properties Delegation

(前のページからの続き)

```
EDBPem.INTERNAL = {  
    kdc = krb5server.edbpem.internal  
    admin_server = krb5server.edbpem.internal  
}
```

- データベースサーバを再起動して変更を反映します

```
systemctl restart <POSTGRES_SERVICE_NAME>
```

どこ、

- POSTGRES_SERVICE_NAME は Postgres (PostgreSQL/ EDB Postgres Advanced Server) データベースのサービス名です。例：CentOS、RHEL または Rocky Linux プラットフォーム上の PostgreSQL 13 データベースの場合は postgresql-13。

5. 初期チケットを取得して表示する

- **kinit** ユーティリティは、Kerberos チケットを取得してキャッシュします。このユーティリティは通常、ユーザーが入力したパスワードを使用して、キー配布センター (KDC) から資格情報を復号化し、チケット保証チケットを取得するために使用されます。チケット保証チケットは、ユーザーの資格情報キャッシュに保存されます。
- **klist** ユーティリティを使用してチケットの詳細を表示できます。

注釈: *kinit* および *klist* を使用するには、PEM サーバーとクライアントマシンに *Kerberos Client* をインストールする必要があります。

```
$ kinit <USERNAME@REALM>  
$ klist
```

kerberos チケットとともにプリンシパルが表示されます。

注釈: ここで指定する *USERNAME@REALM* は、*pem* データベースに対する *pem_admin* ロールと *CONNECT* 特権を持つデータベースユーザーである必要があります。

6. PEM サーバーを構成する

- PEM サーバーで PEM 構成スクリプトを実行して、Kerberos 認証を使用します

```
$ sudo PEM_APP_HOST=pem.edbpem.internal PEM_KRB_KTNAME=<PEM_INSTALLATION_DIRECTORY/
→share/pemserver.keytab <PEM_INSTALLATION_DIRECTORY>/bin/configure-pem-server.sh
```

- config_setup.py ファイルで PEM_DB_HOST を構成し、PEM_AUTH_METHOD の値が 'kerberos' に設定されていることを確認します。

```
$ sudo vim <PEM_INSTALLATION_DIRECTORY>/share/web/config_setup.py
PEM_DB_HOST=`pem.edbpem.internal`
```

- .install-config ファイルで HOST を構成します

```
$ sudo vim <PEM_INSTALLATION_DIRECTORY>/share/.install-config
HOST=`pem.edbpem.internal`
```

- PEM サーバーが Kerberos 認証を使用する場合、
- すべての監視対象サーバーはデフォルトで同じ認証を使用します。デフォルトをオーバーライドするには、パラメーター ALLOW_DATABASE_CONNECTION_WITHOUT_KERBEROS を追加し、config_local.py ファイルで True に設定します。
- すべての認証されたユーザープリンシパルにはレルム (USERNAME@REALM) が追加され、デフォルトでデータベースユーザー名として渡されます。デフォルトをオーバーライドするには、パラメーター PEM_USER_KRB_INCLUDE_REALM を追加し、config_local.py ファイルで False に設定します。
- Apache サーバーを再起動します

```
sudo systemctl restart <SERVICE_NAME>
```

- pg_hba.conf の上部のエントリを編集して gss 認証方法を使用し、データベースサーバーをリロードします。

```
host pem      +pem_user    <ip_of_pem_server>/32  gss
host postgres +pem_user    <ip_of_pem_server>/32  gss
```

```
systemctl reload <POSTGRES_SERVICE_NAME>
```

どこ、

- POSTGRES_SERVICE_NAME は Postgres (PostgreSQL/ EDB Postgres Advanced Server) データベースのサービス名です。例：CentOS、RHEL または Rocky Linux プラットフォーム上の PostgreSQL 13 データベースの場合は postgresql-13。

注釈: Windows は gss 暗号化接続をサポートしていないため、接続タイプを `hostgssenc` として指定できません。

7. ブラウザ設定

Spnego/Kerberos を使用するように PEM Web クライアントにアクセスするようにクライアントマシンでブラウザを構成します。

- Mozilla Firefox の場合
- `about:config` ページをロードして、低レベルの Firefox 構成ページを開きます。 - [検索] テキストボックスに、次を入力します。 `network.negotiate-auth.trusted-uris - network.negotiate-auth.trusted-uris` 設定をダブルクリックし、Kerberos HTTP SPNEGO で保護されている Web サーバーのホスト名またはドメインを入力します。複数のドメインとホスト名はコンマで区切ります。 - [検索] テキストボックスに、次を入力します。 `network.negotiate-auth.delegation-uris - network.negotiate-auth.delegation-uris` 設定をダブルクリックし、Kerberos HTTP SPNEGO で保護されている Web サーバーのホスト名またはドメインを入力します。複数のドメインとホスト名はコンマで区切ります。 - [OK] をクリックします。
- Linux または MacOS の Google Chrome の場合：
- `--auth-server-whitelist` パラメーターを `google-chrome` コマンドに追加します。たとえば、Linux プロンプトから Chrome を実行するには、次のように `google-chrome` コマンドを実行します。

```
google-chrome --auth-server-whitelist = "hostname/domain"
```

- PEM サーバーを構成した後、ブラウザで PEM Web インターフェースにアクセスできます。次の場所に移動します。

```
https://<ip_address_of_PEM_server>:8443/pem
```

注釈: Postgres クラスターへの接続中に次のエラーが発生した場合。

```
psql -h hostname template1 psql: GSSAPI continuation error: Unspecified GSS failure.
Minor code may provide more information GSSAPI continuation error: Key version is not
available
```

解決策は、`ktutil` を使用してキータブに追加の暗号化タイプを追加するか、AD のすべての暗号化システムで Postgres キータブを再作成することです。

2.1.2.3.3 Configuring the PEM server to use two-factor authentication

- PEM は 2FA の 2 つの方法をサポートしています。
- メール認証 - Authenticator アプリ（例：Google Authenticator）

2FA を有効にするには、これらの設定を `config.py` ファイルから `config_local.py` ファイルにコピーし、次のパラメーターを変更します。

Parameter	Description
MFA_ENABLED	これを true に設定して、二要素認証を有効にします。デフォルト値は false です。
MFA_FORCE_REGISTRATION	ログイン時に 2 要素認証方法を強制的に登録するようにユーザーに求めるには、これを true に設定します。デフォルト値は false です。
MFA_SUPPORTED_METHODS	二要素認証方法(ワンタイムコードをメールで送信)を使用する場合は`email`に設定し、TOTP ベースのアプリケーション認証方法を使用する場合は`authenticator`に設定します。
MFA_EMAIL_SUBJECT	これをメール認証用のメールの件名に設定します。デフォルト値は`<APP_NAME> - Verification Code`です。

メールサーバー構成

メール認証方法を使用するには、メールサーバー設定を構成する必要があります。

PEM サーバーは、PEM 構成に保存された SMTP 構成または Flask-Mail を使用して電子メールを送信できます。

PEM 構成から内部 SMTP 構成を使用して電子メール確認コードを送信するには、パラメーター `MAIL_USE_PEM_INTERNAL` を True に設定します。False に設定すると、次のメール構成を使用して、ユーザーが指定したメールアドレスにコードが送信されます。

- `MAIL_SERVER` = 'localhost'
- `MAIL_PORT` = 25
- `MAIL_USE_TLS` = False
- `MAIL_USE_SSL` = False
- `MAIL_USERNAME` = なし
- `MAIL_PASSWORD` = なし
- `MAIL_DEFAULT_SENDER` = なし

これらの構成の詳細については、[Flask-Mail documentation](#) を参照してください。

注釈: PEM SMTP アラートはこの構成を使用しません。

2.1.2.4 Securing your deployment

このドキュメントでは、PEM を構成するときに考慮する必要があるセキュリティ対策に関する情報を提供します。PEM 機能では、これらのプラクティスを強制する必要はありません。ただし、EDB は、システム全体のセキュリティを強化するためにこれらのプラクティスをお勧めします。

PEM は、Apache Web サーバー、OpenSSL、snmp++、libcurl などを含む、ベンダーリポジトリのサードパーティコンポーネントに依存しています。これらのコンポーネントを最新の状態にするには、オペレーティングシステムを更新し、定期的にセキュリティ更新を適用して、セキュリティの脆弱性。最新のセキュリティパッチが適用されていない場合、システムはサイバー攻撃に対して潜在的に脆弱です。セキュリティパッチは、最新の更新を適用して最新の脅威から保護することにより、デバイスとそのデータを保護します。

セキュリティパッチを定期的に適用する利点には次のものがあります。

- サイバー攻撃への露出の削減
- 生産性の低下を回避
- マルウェアからのデータ保護（ランサムウェアなど）
- セキュリティの抜け穴を使用してネットワーク全体に広がるワーム感染を回避します

2.1.2.4.1 Apache HTTPD security configurations

Windows システムでは、Apache HTTPD の名前は PEM HTTPD であり、Apache httpd 構成ファイル (`pme.conf`) は `<Apache_Installation_Path>/conf/addons` ディレクトリにあります。SSL 構成ファイル (`httpd-ssl-pem.conf`) は、`<Apache_Installation_Path>/conf/addons` ディレクトリにあります。

Linux システムでは、apache httpd 構成ファイル (`edb-pem.conf`) は `<Apache_Installation_Path>/conf.d` ディレクトリにあり、SSL 構成ファイル (`edb-ssl-pem.conf`) は `<Apache_Installation_Path>/conf.d` ディレクトリにあります。

SSLv2 と SSLv3 を無効にする

SSL バージョン SSLv2、SSLv3、TLS 1、および TLS 1.1 は最も脆弱であり、暗号化の問題の影響を受けるため、無効にする必要があります。

バージョンを無効にするには、次のコマンドを Apache httpd 構成ファイルに追加してください。

```
SSLProtocol -ALL +TLSv1.2
```

構成ファイルに変更を適用するには、Web サーバーを再起動する必要があります。

デフォルトでは、PEM は次の行を SSL 構成ファイルに追加して、セキュリティに TLS 1.2 の使用を許可します。

```
SSLProtocol -All TLSv1.2
```

```
SSLProxyProtocol -All TLSv1.2
```

SSL 証明書で httpd を保護

EDB は、Web アプリケーションに SSL セキュリティのレイヤーを追加することをお勧めします。

デフォルトでは、PEM のインストール中に、PEM は Apache / HTTPD サーバーの自己署名証明書を生成して使用します。PEM に独自の SSL 証明書を使用する場合は、Apache 構成ファイルを更新する必要があります。

Linux システムでは、ファイルへの書き込み権限を持つユーザーとしてテキストエディターで Apache httpd 構成ファイル (`edb-ssl-pem.conf`) を開く必要があります。また、証明書ファイルと一致するように構成ファイル内のサーバー名とファイル名を変更する必要があります。

更新する必要がある PEM VirtualHost セクション内には 2 つの SSL ディレクティブがあります。

- `SSLCertificateFile` は、DigiCert 証明書ファイルです (例、`your_domain_name.crt`)。
- `SSLCertificateKeyFile` は、CSR の作成時に生成された `.key` ファイルです (例: `your_private.key`)。

たとえば、次を更新します。

`SSLEngine on`

`SSLCertificateFile /path/to/your_domain_name.crt`

`SSLCertificateKeyFile /path/to/your_private.key`

Postgres Enterprise Manager で httpd 自己署名 SSL 証明書を信頼できる CA 署名証明書に置き換えることもできます。手順については、以下に示すリンクに従ってください。

<https://www.enterprisedb.com/postgres-tutorials/how-replacing-httpd-self-signed-ssl-certificates-trusted-ca-signed-certificates>

Web サーバー情報の公開を無効にする

EDB は、Web サーバーセキュリティの一部としてすべての Web サーバー署名を無効にすることをお勧めします。Web サーバーはソフトウェア署名を公開します。署名を無効にするには、次のパラメーターを Apache httpd 構成ファイルに追加します。デフォルトでは、PEM は以下のパラメーターを Apache httpd 構成ファイルに追加することにより、情報の公開を無効にします。

`ServerTokens Prod`

`ServerSignature Off`

`ServerTokens` 指示子は、クライアントに返されるサーバー応答ヘッダーフィールドを制御します。Apache httpd 構成ファイルにこのパラメーターを追加して、Apache サーバーバージョンを非表示にすることをお勧めします。

`ServerSignature` 指示子には、サーバー生成ドキュメントのフッターが含まれます。フッターには、Apache やオペレーティングシステムのバージョンなど、Apache の構成に関する情報が含まれています。このような情報の表示を制限するには、Apache httpd 構成ファイルでこのディレクティブを無効にすることをお勧めします。

Apache httpd 構成ファイルに変更を適用するには、Web サーバーを再起動する必要があります。

ディレクトリリストを無効にする

ディレクトリのリストにより、攻撃者は完全なディレクトリの内容を表示できます。デフォルトでは、Web サーバーはこのオプションを有効にしており、攻撃者はファイルを発見して表示できます。このリストは、攻撃者がアプリケーションをリバースエンジニアリングしてソースコードを取得し、セキュリティ上の欠陥がないかどうかを分析し、アプリケーションに関する詳細を発見することにつながる可能性があります。

これを回避するには、Apache httpd 構成ファイルで `Options` 指示子を設定して、ディレクトリリストを無効にする必要があります。デフォルトでは、PEM は Web サーバー構成ファイルで以下のオプションを設定することにより、ディレクトリリストを無効にします。

```
<Directory /application/directory> Options -Indexes </Directory>
```

構成ファイルに加えた変更を適用するには、Web サーバーを再起動する必要があります。

ネットワークまたは IP アドレスへのアクセスを制限する

Apache は、クライアントのホスト名または IP アドレスに基づいたアクセス制御を提供します。特定の IP アドレスまたはネットワークでアプリケーションを表示するには、以下に示すように Apache 構成ファイルを変更して、`Allow` ディレクティブ内にネットワークアドレスを指定します。

```
<Directory /application/hostname>
```

```
Options None
```

```
AllowOverride None
```

```
Order deny,allow
```

```
Deny from all
```

```
Allow from 192.168.0.0/24
```

```
</Directory>
```

PEM は、アプリケーション構成ファイルの `ALLOWED_HOSTS` 構成パラメーターを使用して、許可されたホストの IP アドレスリストを提供します。アプリケーション構成 `config_local.py` ファイルは `<PEM_INSTALLATION_PATH>/web` にあります。

デフォルトでは、PEM はすべてのホストがアプリケーションに接続することを許可します。

例：

以下のように構成ファイルで IP アドレスの範囲を設定できます。

```
ALLOWED_HOSTS = ['225.0.0.0/8', '226.0.0.0/7', '228.0.0.0/6']
```

以下のように、構成ファイルでサブネットレベルでホストを許可するように IP アドレスを設定できます。

```
ALLOWED_HOSTS = ['192.0.2.0/28', '::192.0.2.0/124']
```

以下のように、構成ファイルで（IP アドレスに基づいて）特定の個々のホストアドレスを設定できます。

```
ALLOWED_HOSTS = ['127.0.0.1', '192.168.0.1']
```

アプリケーション構成ファイルに変更を適用するには、Web サーバーを再起動する必要があります。

クロスサイトトレース

Web サーバー接続をデバッグするには、TRACE と TRACK の 2 つの方法があります。HTTP TRACE 要求がそれをサポートする Web サーバーに送信されると、そのサーバーは応答し、HTTP ヘッダーを含めて渡されたデータをエコーします。Apache 構成内でこれらのメソッドを無効にすることをお勧めします。

すべての仮想ホストの TRACE メソッドを無効にするには、次の行を Apache httpd 構成ファイルに追加します。

```
TraceEnable off
```

特定の仮想ホストでこれらのメソッドを無効にするには、Apache 構成ファイルで仮想ホストごとに次の行を追加します。PEM は次の行を Apache httpd 構成ファイルに追加します。

```
RewriteEngine on
```

```
RewriteCond %{REQUEST_METHOD} ^(TRACE|TRACK|OPTIONS)
```

```
RewriteRule .\* - [F]
```

非特権ユーザーアカウントから Web サーバーを実行する

root ユーザーとして Apache Web サーバーを実行すると、セキュリティの問題が発生します。常に、一意の非特権ユーザーとして Web サーバーを実行することをお勧めします。これは、セキュリティ違反が発生した場合に実行されている他のサービスを保護するのに役立ちます。

PEM は WSGI アプリケーションとして実行されます。実行中の WSGI アプリケーションを委任するには、WSGIDaemonProcess 指示子を使用して個別のデーモンプロセスを作成します。

Linux では、Apache Web サーバーは root ユーザーとして起動しますが、デーモンは PEM アプリケーションを pem ユーザーとして実行します。Windows では、WSGIDaemonProcess 指示子とその機能は使用できません。PEM HTTPD はインストール中にサービスとしてインストールされます。サービスのインストールを成功させるには、ユーザーが Administrators グループのメンバーである必要があります。

デフォルトでは、Apache サービスはシステムユーザー（LocalSystem アカウント）として実行されるように登録されています。

PEM Web サーバーのセキュリティ HTTP ヘッダーをカスタマイズする

PEM には、次のセキュリティ問題を修正するための独自の構成ファイルが含まれています。config.py ではなく、config_local.py のみの構成をオーバーライドすることをお勧めします。ほとんどの場合、config_local.py はシステムに存在しません。したがって、ユーザーはそれを作成してアプリケーションレベルの構成をオーバーライドする必要があります。PEM のアップグレード中に、config_local.py は上書きされませんが、config.py および config_distro.py の変更はオーバーライドされることに注意してください。ユーザーは PEM をアンインストールした後に config_local.py を削除する必要があります。

デフォルトでは、config_local.py は、Linux では /usr/edb/pem/web、WindowsC:\ProgramFiles\edb\pem\server\share\web にあります。

ホストヘッダーインジェクション攻撃

HTTP ホストヘッダー攻撃は、ホストヘッダー値を安全でない方法で処理する脆弱な Web サイトを悪用します。サーバーがホストヘッダーを暗黙的に信頼し、適切に検証またはエスケープできない場合、攻撃者はこの入力を使用して、サーバー側の動作を操作する有害なペイロードを挿入できる可能性があります。通常、Web アプリケーションは、セットアップ中に構成ファイルで指定されない限り、展開されているドメインを認識しません。たとえば、現在のドメインを知る必要がある場合、ホストヘッダーからドメインを取得して絶対 URL を生成する場合があります。ホストヘッダーは、他のさまざまな脆弱性、特に Web キャッシュポイズニングと SQL インジェクションを悪用する潜在的なベクトルです。

X-Frame-Options

X-Frame-Options を使用して、ブラウザが<iframe>タグでページをレンダリングできるようにするかどうかを示すことができます。クリックジャッキングから保護するように特別に設計されました。PEM には、このような種類の攻撃を防ぐためのホスト検証 `X_FRAME_OPTIONS` オプションがあり、`config_local.py` ファイルで構成できます。デフォルトは次のとおりです。

```
X_FRAME_OPTIONS = "SAMEORIGIN"
```

コンテンツセキュリティポリシー

Content-Security-Policy は HTML5 標準の一部であり、X-Frame-Options ヘッダー（置き換え）よりも幅広い保護を提供します。Web サイトの作成者がリソース（スクリプト、スタイルシート、フォントなど）をロードできる個々のドメインと、ページの埋め込みを許可するドメインをホワイトリストに登録できるように設計されています。

PEM には、このような種類の攻撃を防ぐためのホスト検証 `CONTENT_SECURITY_POLICY` オプションがあり、`config_local.py` ファイルで構成できます。デフォルトは次のとおりです。

```
CONTENT_SECURITY_POLICY = "default-src https: data: blob: 'unsafe-inline'
'unsafe-eval';"
```

厳密な輸送セキュリティ

Strict-Transport-Security 応答ヘッダー（多くの場合、HSTS と略されます）を使用すると、Web サイトまたは Web アプリケーションは、HTTP ではなく HTTPS を使用してのみアクセスする必要があることをブラウザに指示できます。このオプションを使用すると、中間者攻撃を防ぐことができます。デフォルトは次のとおりです。

```
STRICT_TRANSPORT_SECURITY = "max-age=31536000;includeSubDomains"
```

注釈: 構成が変更された場合、このパラメーターを追加すると問題が発生する場合があります。したがって、PEM のインストールが完了してテストされた後にのみこれを追加することをお勧めします。

X-Content-Type-Options

X-Content-Type-Options 応答 HTTP ヘッダーは、Content-Type ヘッダーでアドバタイズされた MIME タイプを変更および追跡しないことを示すためにサーバーによって使用されるマーカーです。これは、MIME タイプのスニффイングをオプトアウトする方法、つまり、MIME タイプが意図的に構成されていることを示します。デフォルトは次のとおりです。

```
X_CONTENT_TYPE_OPTIONS = "nosniff"
```

X-XSS-保護

クロスサイトスクリプティング (XSS) は、Web サーバーで最も一般的なアプリケーションレイヤーの脆弱性の 1 つです。XSS により、攻撃者は他のユーザが閲覧する Web ページにクライアントサイドスクリプトを挿入できます。ヘッダーへの HTTP X-XSS-Protection 応答は、リフレクトクロスサイトスクリプティング (XSS) 攻撃を検出するとページのロードを停止する Internet Explorer、Chrome、および Safari の機能です。サイトがインライン JavaScript の使用を無効にする強力なコンテンツセキュリティポリシー (「unsafe-inline」) を実装している場合、これらの保護は最新のブラウザではほとんど必要ありませんが、まだ CSP をサポートしていない古い Web ブラウザーのユーザーに保護を提供できます。デフォルトは次のとおりです。

```
X_XSS_PROTECTION = "1; mode=block"
```

これを回避するには、次のオプションを Apache 構成ファイルに追加する必要があります。

```
<IfModule mod_headers.c>
```

```
Header set X-XSS-Protection "1; mode=block"
```

```
</IfModule>
```

構成ファイルの変更を適用するには、Web サーバーの再起動が必要です。

デフォルトでは、PEM は `/usr/edb/pem/web/config.py` にあるアプリケーション構成ファイルに `X-XSS-Protection to "1; mode=block"` を設定します。

この変更を有効にするには、Apache サービスを再起動する必要があります

`config.py` ファイルの詳細については、PEM Online Help を参照してください。

クッキーのセキュリティ

Cookie は、サーバーがブラウザに送信して構成データを保存できるデータの小さなパケットです。ブラウザはすべての要求と共にそれらを同じサーバーに自動的に送信するため、Cookie を保護する方法を知ることが重要です。`config.py` で参照できる Cookie を安全にするために PEM が提供する構成オプションは複数ありますが、以下の 3 つが最も重要です。

1. **SESSION_COOKIE_SECURE** - セキュアフラグを設定すると、暗号化されていない接続で Cookie が送信されなくなります。基本的に、暗号化されたチャネルを使用しないサーバーへの要求に Cookie を追加しないようにブラウザに指示します。ブラウザは、HTTPS などの接続にのみ Cookie を追加します。デフォルトは次のとおりです。

`SESSION_COOKIE_SECURE = True`

2. **SESSION_COOKIE_HTTPONLY** - デフォルトでは、Cookie のコンテンツは JavaScript を介して読み取ることができます。HTTPOnly フラグは、スクリプトが Cookie を読み取らないようにします。HTTPOnly という名前が示すように、ブラウザーは HTTP(S) 要求でのみ Cookie を使用します。これにより、ハッカーが XSS の脆弱性を使用して Cookie の内容を学習するのを防ぎます。たとえば、sessionId Cookie の場合、クライアントサイドのスクリプトで Cookie を読み取る必要はないため、sessionId Cookie の場合、いつでも HTTPOnly フラグを設定できます。デフォルトは次のとおりです。

`SESSION_COOKIE_HTTPONLY = True`

3. **ENHANCED_COOKIE_PROTECTION**- このオプションを True に設定すると、IP アドレスとユーザーエージェントに従ってトークンが生成されます。後続のすべてのリクエストでは、トークンが再計算され、最初のリクエストで計算されたものと照合されます。セッション Cookie が盗まれ、攻撃者が別の場所から使用しようとした場合、生成されるトークンは異なり、その場合、拡張機能はセッションをクリアしてリクエストをブロックします。デフォルトは次のとおりです。

`ENHANCED_COOKIE_PROTECTION = True`

注釈: このオプションは、サーバーが Kubernetes などの動的 IP アドレスホスティング環境にデプロイされている場合や、ロードバランサーの背後にある場合に問題を引き起こす可能性があります。このような場合、このオプションを `False` に設定する必要があります。

この変更を有効にするには、Apache サービスを再起動する必要があります。

`config.py` ファイルの詳細については、PEM Online Help を参照してください。

2.1.2.4.2 PEM application security configurations

セッションタイムアウト

Web アプリケーションによるセッションの有効期限が不十分であると、攻撃者が有効なセッション ID を再利用して関連するセッションをハイジャックできるため、他のセッションベースの攻撃の露出が増加します。セッション間隔が短いほど、攻撃者が有効なセッション ID を使用する時間が短くなります。このセキュリティ問題を回避するには、Web アプリケーションの非アクティブタイムアウトを低い値に設定することをお勧めします。

Postgres Enterprise Manager は、ユーザーセッションのタイムアウト値を設定する方法を提供します。Web コンソールで指定された期間ユーザーアクティビティがない場合、PEM はユーザーを Web コンソールからログアウトします。PEM 管理者は、非アクティブの時間の長さを設定できます。この値は、個々のユーザー向けではなく、アプリケーションに関するものです。タイムアウト時間を構成するには、`<PEM_INSTALLATION_PATH>/web` ディレクトリにある `config_local.py` ファイルの `USER_INACTIVITY_TIMEOUT` パラメーターを変更します。デフォルトでは、この機能は無効になっています。

たとえば、アプリケーションが 15 分間操作しないとユーザーをログアウトするように指定するには、次のように設定します。

`USER_INACTIVITY_TIMEOUT = 900`

注釈: タイムアウト値は秒単位で指定します。

この変更を有効にするには、Apache サービスを再起動する必要があります。

`config.py` ファイルの詳細については、[PEM Online Help](#) を参照してください。

RestAPI ヘッダーのカスタマイズ

要件に応じて RestAPI トークンヘッダーをカスタマイズできます。デフォルト値は `config.py` ファイルでは公開されません。 `config_local.py` ファイルの次のヘッダーをカスタマイズします。

PEM_HEADER_SUBJECT_TOKEN_KEY

この構成オプションを使用すると、HTTP ヘッダー名を変更して、生成されたトークンを取得できます。デフォルトでは、ユーザーがトークンを作成する要求を送信すると、サーバーの応答には新しく生成されたトークンの値が含まれる「X-Subject-Token」ヘッダーが含まれます。ヘッダー名をカスタマイズする場合は、次のように `config_local.py` ファイルを更新できます。

```
PEM_HEADER_SUBJECT_TOKEN_KEY = 'Pem-RestAPI-Generate-Token'
```

次の出力が得られます。

```
curl -ik -X POST -d '{"username":"enterprisedb","password":"edb"}' -H "Content-Type: application/json" https://localhost:8443/pem/api/token/
```

```
HTTP/1.1 201 CREATED
```

```
Date: Thu, 29 Oct 2020 11:03:48 GMT
```

```
Server: Apache
```

```
Content-Length: 326
```

```
Pem-RestAPI-Generate-Token: 997aef95-d46d-4d84-932a-a80146eaf84f
```

PEM_HEADER_TOKEN_KEY

この構成オプションを使用すると、PEM サーバーにトークンを送信する HTTP 要求ヘッダー名を変更できます。デフォルトでは、ユーザーがトークンを生成する要求を送信すると、トークンヘッダーは `X-Auth-Token` になります。RestAPI リクエストヘッダー名をカスタマイズする場合は、次のように `config_local.py` ファイルを更新できます。

```
PEM_HEADER_TOKEN_KEY = 'Pem-Token'
```

トークンを送信できます。

```
$ curl -Lk -X GET -H "Pem-Token: gw5rzaloxyp91tttd1c97w24b5sv60cllc24sxy9" https://localhost:8443/pem/api/v4/agent
```

PEM_TOKEN_EXPIRY

この構成オプションを使用すると、生成後に PEM RestAPI トークンの有効期限を変更できます。デフォルトでは、トークンの有効期限は 20 分（1200 秒）に設定されています。トークンの有効期限を 10 分に変更する場合は、次のように `config_local.py` ファイルを更新できます。

```
PEM_TOKEN_EXPIRY = 600
```

この変更を有効にするには、Apache サービスを再起動する必要があります。

PEM のロールベースのアクセス制御

ロールベースのアクセス制御（RBAC）は、組織内のユーザーのロールに基づいてアプリケーションへのアクセスを制限し、アクセス制御の主要な方法の 1 つです。RBAC のロールは、ユーザーがアプリケーションに対して持つアクセスのレベルを参照します。ユーザーは、職務を効果的に実行するために必要な情報にのみアクセスできます。PEM のロールは、添え字ではなく継承可能で追加的です。簡単に言えば、PEM 管理者は、ユーザーに最低レベルのロールを付与し、ユーザーがそれぞれのタスクを実行するために必要なロールを付与する必要があります。たとえば、SQL プロファイラーにのみアクセスを許可するには：

```
CREATE ROLE user_sql_profiler WITH LOGIN NOSUPERUSER NOCREATEDB NOCREATEROLE INHERIT
NOREPLICATION CONNECTION LIMIT -1 PASSWORD 'xxxxxx';
```

```
GRANT pem_user, pem_comp_sqlprofiler TO user_sql_profiler;
```

ロールの詳細については、PEM Roles を参照してください。

SQL/Protect プラグイン

通常、SQL インジェクション攻撃を防ぐことはアプリケーション開発者の責任です。通常、データベース管理者は潜在的な脅威をほとんど、またはまったく制御できません。データベース管理者にとっての困難は、アプリケーションが適切に機能するためにデータにアクセスする必要があることです。

SQL/Protect は、データベース管理者が SQL インジェクション攻撃からデータベースを保護できるようにするモジュールです。SQL/Protect は、一般的な SQL インジェクションプロファイルの着信クエリを検査することにより、標準のデータベースセキュリティポリシーに加えてセキュリティレイヤーを提供します。

SQL インジェクション攻撃を行うには、いくつかの異なる手法が使用されます。特定の署名が各技術の特徴付けます。SQL/Protect は、不正なリレーション、ユーティリティコマンド、SQL トートロジー、無制限の DML ステートメントのクエリを調べます。SQL/Protect は、潜在的に危険なクエリを管理者に警告し、それらをブロックすることにより、データベース管理者に制御を戻します。

注釈: このプラグインは EPAS サーバーでのみ機能するため、PEM データベースが EPAS サーバーでホストされている場合にのみ役立ちます。

SQL プロファイラープラグインの詳細については、PEM Online Help - SQL Profiler を参照してください。

パスワード管理

PEM 管理ユーザーのセキュリティのヒントの 1 つは、PEM ログインパスワードを定期的に変更することです。パスワードを変更すると、次のような多くの危険を回避できます。

- 複数のアカウントの侵害
- 継続的なアクセスを防ぎます
- 物理的に保護されていないシステムで保存されたパスワードの使用を防止します
- キーストロークロガーによるアクセスを制限

root 以外の ser で pemAgent ジョブを実行する

ほとんどの場合、pemAgent は root ユーザーとしてインストールされ、root 権限でデーモンプロセスとして実行されます。デフォルトでは、PEM はスケジュールされたジョブ/タスクの実行を無効にします。PEM は、pemAgent 構成ファイルを変更することにより、非 root ユーザーとしてスケジュールされたジョブを実行するためのサポートを提供します。

スケジュールされたジョブを root 以外のユーザーとして実行するには、agent.cfg ファイルの batch_script_user パラメーターのエントリを変更し、スクリプトの実行に使用するユーザーを指定します。非 root ユーザーまたは root ユーザー ID を指定できます。ユーザーを指定しない場合、または指定したユーザーが存在しない場合、スクリプトは実行されません。ファイルを変更した後、エージェントを再起動します。非 root ユーザーが pemagent を実行している場合、batch_script_user の値は無視され、pemagent の実行に使用されたのと同じ非 root ユーザーがスクリプトを実行します。

Windows システムでスクリプトを呼び出すには、AllowBatchJobSteps のレジストリエントリを true に設定し、PEM エージェントを再起動します。PEM レジストリエントリは次の場所にあります。

HKEY_LOCAL_MACHINE\Software\Wow6432Node\EnterpriseDB\PEM\agent

pemAgent および PEM バックエンドデータベースサーバー証明書の変更

デフォルトでは、PEM をインストールすると、インストーラーは pemAgent と PEM データベースサーバーの自己署名証明書を生成して使用します。PemAgent は、PEM データベースサーバーに接続するときにこれらの証明書を使用します。pemAgent および PEM データベースサーバーに独自の SSL 証明書を使用するには、PEM Administrators's Guide の Managing Certificates セクションに記載されている手順に従います。

注釈: PEM は、カスタムの場所への SSL CA 証明書の配置をサポートしていません。ca_certificate.crt および `ca\_key.key` の場所は変更しないでください。

2.1.2.5 Web server installation options

PEM サーバーのインストール中に、Apache Web サーバーのホスティング設定を指定できます。

2.1.2.5.1 別のホスト上の PEM サーバーと Apache Web サーバー

1. 両方のホストに PEM サーバーをインストールします。手順については、

Installing the PEM server on Linux を参照してください。

2. 最初のホストで Database オプションを選択して、PEM サーバーホストを構成します。

3. 2 番目のホストで Web Services オプションを選択して、Apache Web サーバーを構成します。

PEM サーバーの構成の詳細については、Configuring the PEM server on Linux platforms を参照してください。

注釈: 実稼働環境では、PEM サーバーと Apache Web サーバーを別のホストに配置することをお勧めします。### 同じホスト上の PEM サーバーと Apache Web サーバー

1. PEM サーバーをインストールします。手順については、Installing the PEM server on Linux を参照してください。

2. 構成スクリプトを実行し、Web Services and Database オプションを選択して PEM サーバーと Apache Web サーバーを同じホストにインストールします。手順については

Configuring the PEM server on Linux platforms を参照してください。

注釈: 非実稼働環境の場合、PEM サーバーと Apache Web サーバーを同じホストに配置することをお勧めします。
