



Postgres Enterprise Manager Planning Guide

リリース 8.6.0

EnterpriseDB Corporation

2022 年 10 月 28 日

目次

第 1 章	Understanding Postgres Enterprise Manager components and architecture	3
1.1	PEM アーキテクチャ	4
1.1.1	PEM サーバー	4
1.1.2	PEM エージェント	6
1.1.3	PEM Web クライアント	8
1.1.4	SQL プロファイラープラグイン	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
第 3 章	Deployment considerations	11
3.1	Using Failover Manager for high availability	11
3.1.1	製品の初期インストールと構成	12
3.1.2	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	13
3.1.3	ストリーミングレプリケーション用のスタンバイノードを設定する	14
3.1.4	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	16
3.1.5	現在の制限	21
3.2	Connection pooling using pgBouncer	22
3.2.1	PEM server and agent connection management mechanism	22
3.2.2	Preparing the PEM database server	23
3.2.3	Configuring PgBouncer	25
3.2.4	Configuring the PEM agent	27
3.3	Authentication options	29
3.3.1	Configuring the PEM server to use Windows Active Directory domain services for Kerberos authentication (SSPI)	29
3.3.2	Configuring the PEM server to use Kerberos authentication	38
3.3.3	Configuring the PEM server to use two-factor authentication	43
3.4	Securing your deployment	44
3.4.1	Apache HTTPD security configurations	45
3.4.2	PEM application security configurations	51
3.5	Web server installation options	55
3.5.1	別のホスト上の PEM サーバーと Apache Web サーバー	55

Postgres Enterprise Manager (PEM) 計画ガイドへようこそ。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、単一のグラフィカル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します。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。
- **データベースサーバー** ホストで実行されている PostgreSQL または EDB Postgres Advanced Server または EDB Postgres Extended Server（旧称 2ndQPostgres）の 1 つ以上のインスタンス。
- **データベース** 1 つ以上のデータベースとそのスキーマオブジェクト（テーブルやインデックスなど）。

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

PEM は、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。

- **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であり、エージェントとクライアントが接続するサーバーです。PEM サーバーは、PostgreSQL のインスタンス、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と監視対象の 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 の複数のインストール済み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 拡張機能を使用できます。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 ブラウザーを使用してアクセスします。
- **SQL プロファイラー** SQL Profiler は、PEM で分析する SQL Profiler ツールの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、PEM サーバーで管理されているものだけでなく、サポートされているバージョンの PostgreSQL サーバーまたは EDB Postgres Advanced Server 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。

1.1 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。

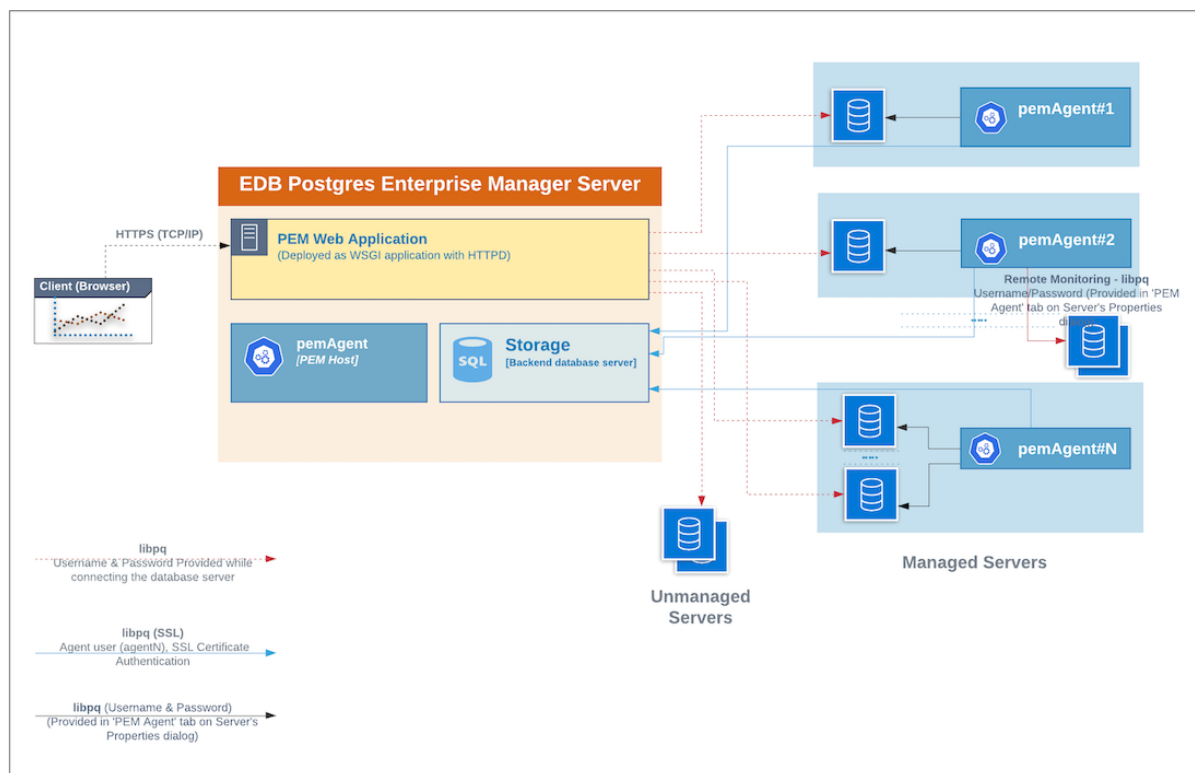


図 1: PEM Architecture

1.1.1 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 はサーバーサイドの暗号化プラグインを使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あってもかまいません。

注釈: すべての PEM 機能は、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

- **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、およびデータベースから収集されたメトリックデータを格納するいくつかのスキーマが含まれています。

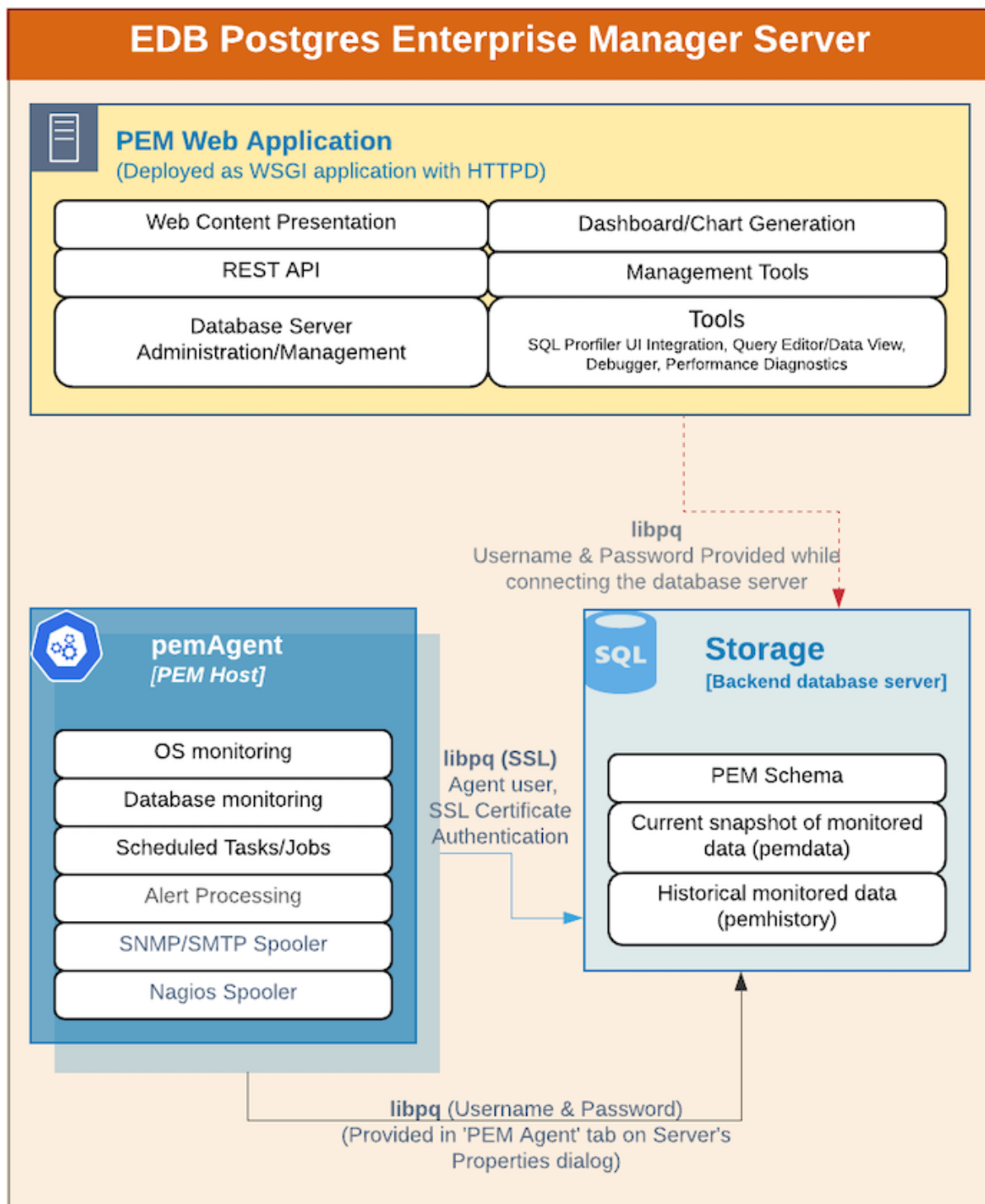


図 2: PEM Server

PEM サーバーをインストールすると、PEM エージェントがデフォルトで PEM サーバーに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および 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 サーバーの間は一方通行です。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。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、監視する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 プローブのリストについては、[Probes](#) を参照してください。

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。

1.1.3 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と、それらに存在す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。

1.1.4 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。

PEM で管理されていないサーバーで SQL プロファイラーを使用できます。ただし、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 エージェントによって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、これらの問題を検討してください。

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
どの種類の認証を使用しますか?	<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実行する必要がありますか?	<i>Securing your deployment</i>
Apache Web サーバーはどこでホストする必要がありますか?	<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 **データベース** 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 **WebServices** 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: 実稼働環境の場合、ベストプラクティスは 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です。### 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ます。 **WebServicesandDatabase** 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、ベストプラクティスは 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です。

第3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実行する必要がありますか?	<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>

3.1 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、および調整するのに役立ちます。

Failover Manager は、EDB の高可用性ツールであり、プライマリでのソフトウェアまたはハードウェアの障害中に Postgres プライマリノードをスタンバイノードにフェールオーバーできます。

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
- ウェブサーバーサービスへのアクセス

3.1.1 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

これら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](#) の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` を自分のユーザー名とパスワードに置き換えます。

PEM サーバー構成スクリプトは信頼認証で動作しないため、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てください。

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 (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 on Linux](#) 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。

- PEM サーバー用の 8443 (https) - EPAS 13 用の 5444 - EFM 用の 7800 - EFM Admin 用の 7908

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
```

3.1.2 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. 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、EDB サポート（techsupport@enterprisedb.com）に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、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
```

3.1.3 ストリーミングレプリケーション用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 を使用する場合は、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 ファイルには次の内容があります。

```
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 krbsvrname=postgres target_session_
→attrs=any'
```

5. 各スタンバイノードの postgresql.conf ファイルで、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. 以下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、既存のファイルを上書きします。ファイルにアクセス許可を設定します。

- /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

例：

```
$ 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
```

このコードにより、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により、ウェブサーバーが有効になります。

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します。

8. configure-selinux.sh スクリプトを実行して、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. HTTPD および PEM エージェントサービスがすべてのレプリカノードで実行されている場合は、無効にして停止します。

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

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

3.1.4 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。例：

```
$ 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;
```

(次のページに続く)

(前のページからの続き)

```

GRANT EXECUTE ON FUNCTION pg_reload_conf() TO efm;

-- 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 PEM エージェントを起動/停止するスクリプトを作成します。

```

$ 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

```

(次のページに続く)

(前のページからの続き)

```

_EOF_
$ 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 アドレスと `efm` ポートを追加します。

```

$ 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` ファイルを作成します。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

```

(次のページに続く)

(前のページからの続き)

```
db.port=5444
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
```

(次のページに続く)

(前のページからの続き)

```
pgpool.enable=false
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

(次のページに続く)

(前のページからの続き)

Allowed node host list:

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示しています。

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、PEM エージェントが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に置き換えることができます。

3.1.5 現在の制限

現在の制限には次のものがあります。

- スイッチオーバー中にユーザーの Web コンソールセッションが失われます。
- [環境設定] ダイアログボックスから設定し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。
- [バックアップ]、[復元]、および [メンテナンス] ダイアログボックスで開始されたバックグラウンドプロセスと、それらのログはシステム間で共有されません。これらはスイッチオーバー中に失われます。

3.2 Connection pooling using pgBouncer

PEM エージェントから非 Windows マシン上の 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 pgBouncer を使用できます。

- PEM データベースサーバーの準備は、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 の構成は、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は、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。

3.2.1 PEM server and agent connection management mechanism

各 PEM エージェントは、ユーザーごとに 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。

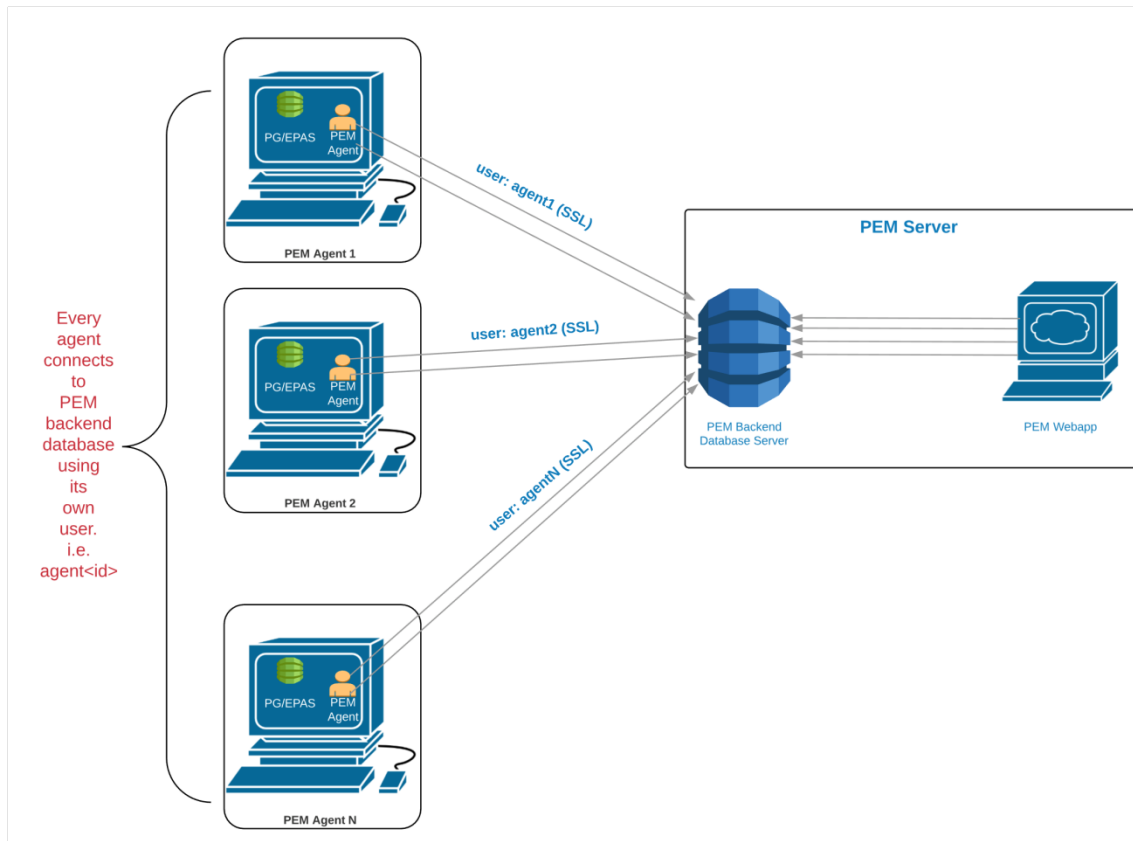


図 1: Connecting to the PEM database without pgBouncer

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ー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の代わりに）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す。

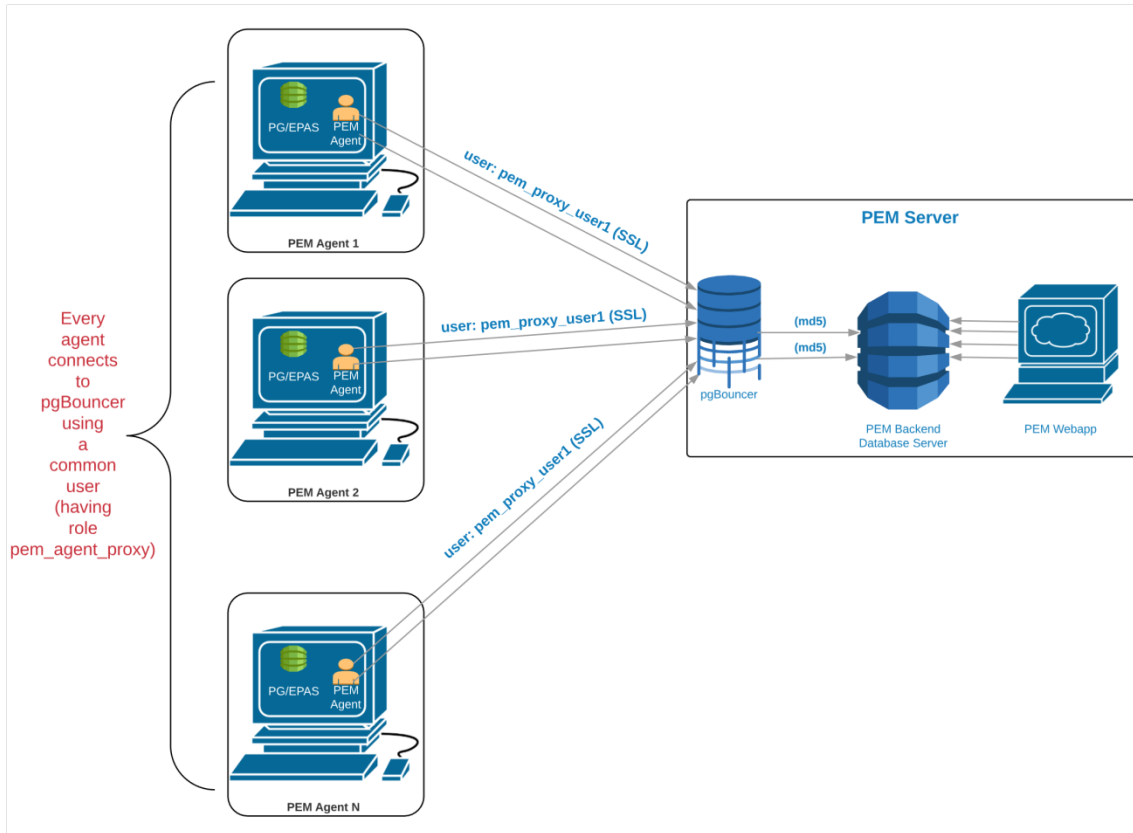


図 2: Connecting to pgBouncer.

接続プラーとして PgBouncer バージョン 1.9.0 以降を使用することをお勧めします。バージョン 1.9.0 以降は証明書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。

3.2.2 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。この例は、PEM データベースサーバーを構成する方法を示しています。

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。

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

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

```
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 権限を付与します。

```
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 USAGE 権限を pgbouncer ユーザーに付与します。

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

5. pem データベースの pem.get_agent_pool_auth(text) ファンクションに対する EXECUTE 権限を pgbouncer ユーザーに付与します。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。

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

このファンクションは、ランダムなパスワードで同じ名前のユーザーを作成し、pem_agent および pem_agent_pool ロールをユーザーに付与します。このアプローチにより、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。これらのエントリにより、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。

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

local pem pgbouncer,pem_admin1 md5
```

3.2.3 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、pgBouncer バージョン 1.9 以降を構成するプロセスの概要を説明します。

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。

```
$ 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
```

(次のページに続く)

(前のページからの続き)

```

;; 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
;; 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. 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（例では `enterprisedb`）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、ディレクトリの権限を `0600` に変更します。

```
$ 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。

```
$ 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
```

3.2.4 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。

pgBouncer で SNMP 通知を送信する PEM エージェントを構成しないでください。たとえば、PEM サーバーと共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 通知に使用されている場合、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!
```

このコマンドでは、`--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
```

PEM エージェントはキーを使用して、pem_agent_user1 として PEM データベースサーバーに接続します。また、/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。

使用するエージェントユーザーを記載した行が agent.cfg 構成ファイルに表示されます。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は、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て、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、既存の agent.cfg ファイル内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。

エージェントとして agent_user を使用する行を追加します。

```
agent_user=pem_agent_user1
```

ポートを更新して pgBouncer ポートを指定します。

```
pem_port=6432
```

証明書とキーのパスの場所を更新します。

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

または、エージェントの自己登録スクリプトを実行できます。ただし、そのプロセスは新しいエージェント ID を作成します。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。

3.3 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 の場合：

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

Linux および Windows の場合：

- *Configuring the PEM server to use two-factor authentication*

3.3.1 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、IP アドレスではなくホスト名で動作します。Active Directory ドメインサービスを使用して 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します。

- Windows サーバー（ドメインコントローラー）
- PEM サーバー（PEM Web サーバーと 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 のレルムが `edbpem.internal` の場合、Windows サーバーのホスト名を `Krb5server.edbpem.internal`、PEM サーバーのホスト名を `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。

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

次のインストールを実行します。

- ・ 認証サーバとして機能する Windows サーバ (ドメインコントローラー) 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してから開始します。

- ・ 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 on Linux](#) を参照してください。
- ・ PEM サーバーと同じ Linux マシンまたは別の 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ピングするユーザーを Active Directory に作成します

Windows サーバーの Active Directory 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。

1. **ActiveDirectory ユーザーとコンピューター>>DOMAIN_NAME>>ユーザー**を開きます。右クリックして、**New>User** を選択します。
 1. ユーザーの詳細を入力します。
 1. パスワードを入力し、[**ユーザーは次のログオン時にパスワードの変更が必要]** をクリアします。また、**ユーザーはパスワードを変更できない**と**パスワードは無期限**を選択します。
 1. ユーザーの詳細を確認します。
 2. PEMServer Web Properties ダイアログボックスで、ユーザーを Domain Users グループのメンバーとして追加します。

Windows サーバーの Active Directory でユーザー (たとえば、`pemserverdb`) 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。

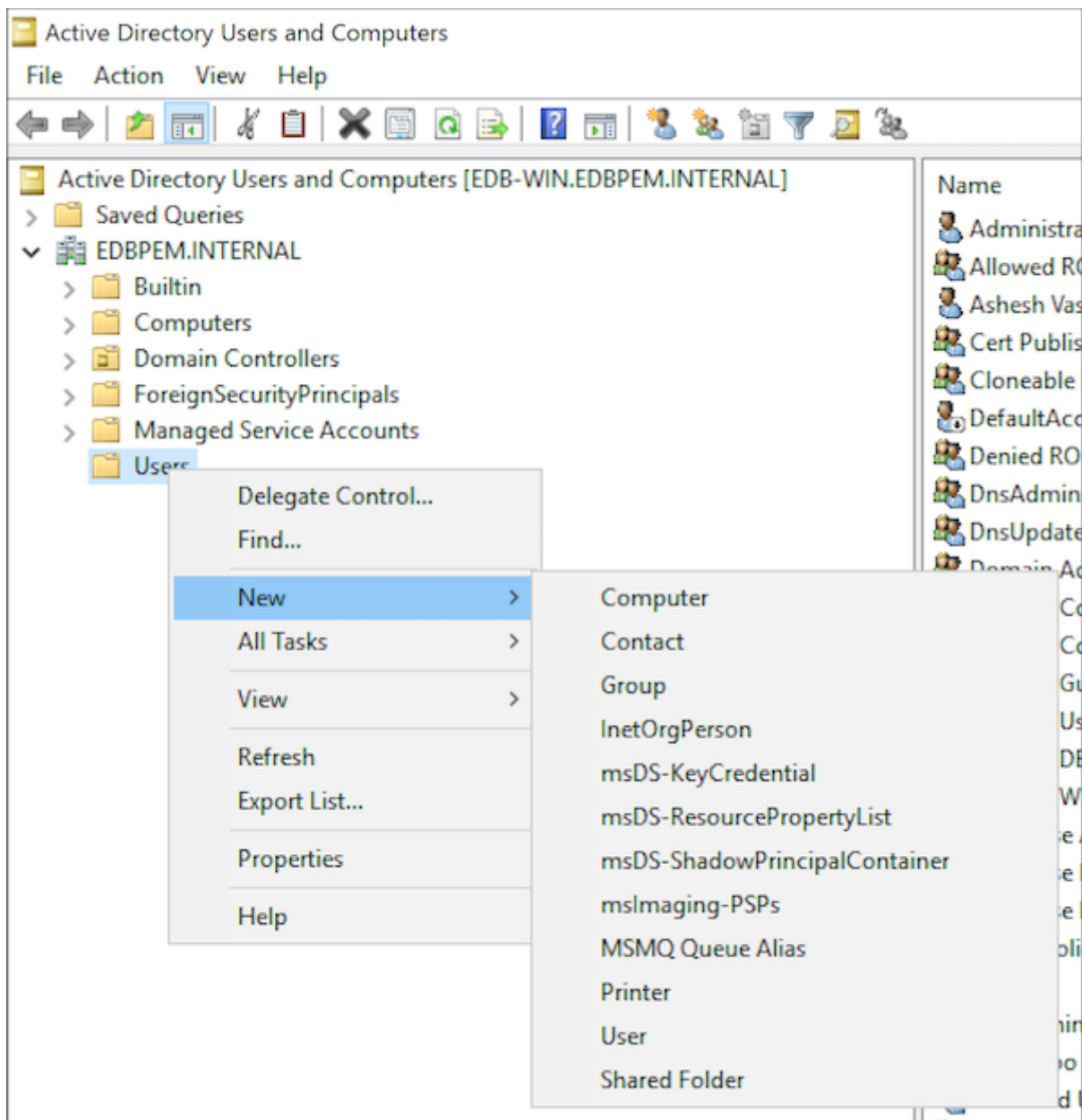


図 3: Active Directory Users and Computers

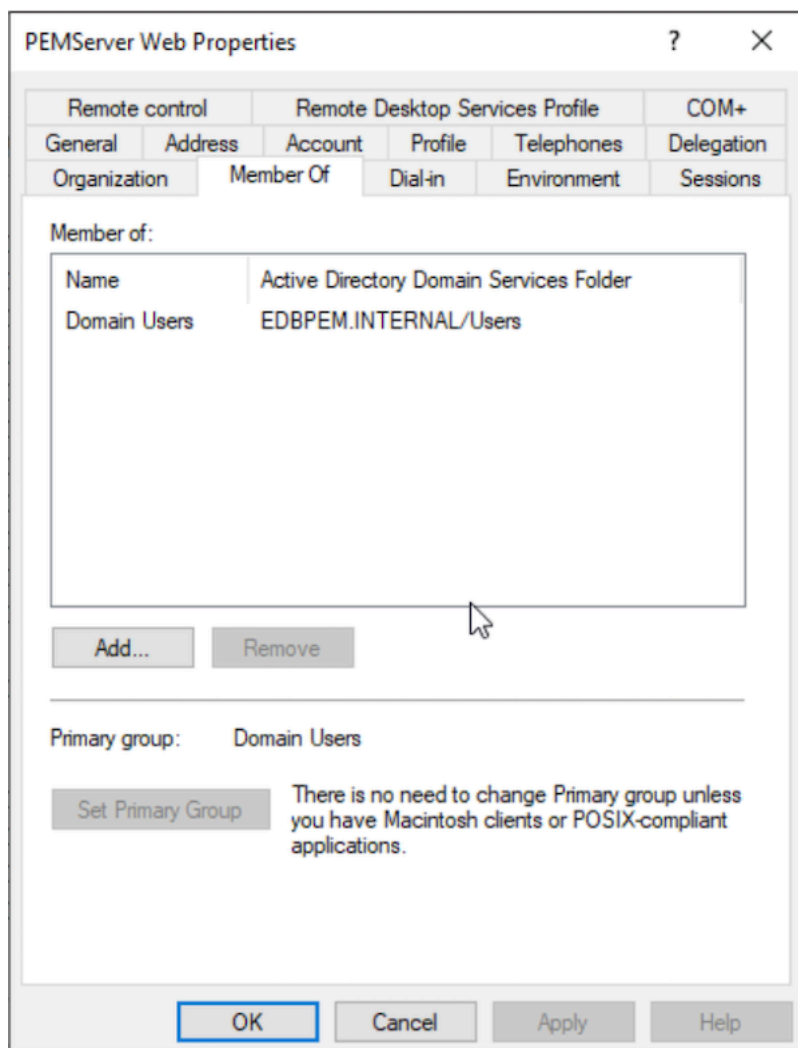


図 4: PEM Server Web Properties

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、作成したそれぞれのドメインユーザーにマップします。

1. Windows PowerShell を管理者として開きます。ユーザ 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 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
```

そこで：

- 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。

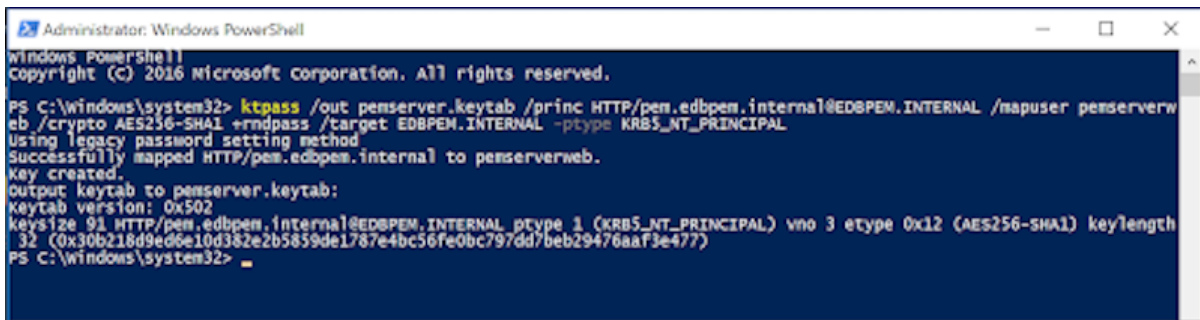


図 5: Windows PowerShell ktpass

注釈: コマンドライン引数 `+rndpass` は、ドメインユーザ 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` オプションはオプションです。

1. [アカウント] タブで、ユーザーアカウントの Kerberos サポートを追加します。

注釈: [アカウント] タブで、ユーザーのログオン名に `**HTTP/pem.edbpem.internal@EDBPEM.INTERNAL**` と表示されます。 pemserverweb ユーザの `**委任**` タブが有効になっています。

1. 委任 タブで、作成したユーザーのサービスへの委任についてこのユーザーを信頼する（Kerberos のみ）を選択します。

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 6: PEM Server Web Properties

1. 両方のキーテーブルを PEM サーバーホスト、または 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（別のホストにインストールされている場合）にコピーします。

1. PEM サーバーで、キーテーブルを必要な場所に移動し、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
```

そこで：

- 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 データベース（PostgreSQL/ EDB Postgres Advanced Server）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します。

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

DATA_DIRECTORY_OF_POSTGRES は、インストールされた Postgres データベース（PostgreSQL/ EDB Postgres Advanced Server）のパスです。

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]
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>
```

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。通常、このユーティリティを使用して、キー配布センター (KDC) からの資格情報を復号化するために入力したパスワードを使用して、チケット保証チケットを取得します。チケット保証チケットは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 と klist を使用する前に、PEM サーバーとクライアントマシンに Kerberos クライアント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 権限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、`config_local.py` ファイルで、パラメーター `ALLOW_DATABASE_CONNECTION_WITHOUT_KERBEROS` を追加し、それを `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、`config_local.py` ファイルで、パラメーター `PEM_USER_KRB_INCLUDE_REALM` を追加し、それを `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` 認証方法を使用します。次に、データベースサーバをリロードします。

```
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>
```

`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: 接続タイプを `hostgssenc` に指定することはできません。Windows は `gss` 暗号化接続をサポートしていません。

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ーを構成します。

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。
- 検索ボックスに、「`network.negotiate-auth.trusted-uris`」と入力します。
- `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。
- 検索ボックスに、「`network.negotiate-auth.delegation-uris`」と入力します。
- `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** を選択します。

Linux または MacOS での Google Chrome の場合：

- `google-chrome` コマンドに `--auth-server-whitelist` パラメーターを追加します。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。次の場所へ移動します。

```
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表示される場合があります。

```
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
```

ktutil を使用するか、AD のすべての暗号化システムで Postgres キータブを再作成して、暗号化タイプをキータブに追加します。

3.3.2 Configuring the PEM server to use Kerberos authentication

PEM サーバーの Kerberos 認証を構成できます。Kerberos サーバーは、IP アドレスではなくホスト名で動作します。Kerberos 認証を使用して PEM サーバーでシングルサインオンを使用するには、DNS（レルム）を使用してホスト名で次のマシンを構成します。

- 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 サーバーのレルムが `edbpem.org` の場合、Kerberos サーバーのホスト名を `Krb5server.edbpem.org` に、PEM サーバーのホスト名を `pem.edbpem.org` に、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。

1.Kerberos、PEM サーバー、および 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。PEM サーバーを別のマシンに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 サーバーと同じマシン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) と PEM バックエンドデータベースサーバー (PostgreSQL / EDB Postgres Advanced Server) の下に展開された 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 サーバーのレルムが含まれている必要があります。たとえば、レルムとして 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、ホスト名はそれぞれ異なります。

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、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
```

そこで：

- NAME_OF_PEM_WEB_FILE は、PEM Web アプリケーションのキーテーブル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のキーテーブルに指定します。

- 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 データベース（PostgreSQL/ EDB Postgres Advanced Server）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します。

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

そこで：

- NAME_OF_PEM_DB_FILE は、PEM バックエンドデータベースサーバのキーテーブルに指定します。
- DATA_DIRECTORY_OF_POSTGRES は、インストールされた Postgres データベース（PostgreSQL/ EDB Postgres Advanced Server）のパスです。

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>
```

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。通常、このユーティリティを使用してチケット保証チケットを取得し、パスワードを入力してキー配布センター（KDC）から資格情報を復号化します。チケット保証チケットは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 と klist を使用する前に、PEM サーバーとクライアントマシンに Kerberos クライアントをインストールします。

```
$ 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 権限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 を構成します。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=`<HOSTNAME_OF_PEM_SERVER>`
```

.install-config ファイルでホストを構成します。

```
$ 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、config_local.py ファイルで、パラメーター ALLOW_DATABASE_CONNECTION_WITHOUT_KERBEROS を追加し、それを 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、`config_local.py` ファイルで、パラメーター `PEM_USER_KRB_INCLUDE_REALM` を追加し、それを `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>
```

`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、接続タイプを `host-gssenc` として指定して、gss で暗号化された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：

1. `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。
1. 検索ボックスに、「`network.negotiate-auth.trusted-uris`」と入力します。
 1. `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。
1. 検索ボックスに、「`network.negotiate-auth.delegation-uris`」と入力します。
 1. `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。
2. **OK** を選択します。

Linux または MacOS での Google Chrome の場合：

- `google-chrome` コマンドに `--auth-server-whitelist` パラメーターを追加します。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表示される場合があります。

```
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
```

ktutil を使用するか、AD のすべての暗号化システムで Postgres キータブを再作成して、暗号化タイプをキータブに追加します。

3.3.3 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
- 認証アプリ（Google Authenticator など）

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	<code>true`</code> に設定して、二要素認証を有効にします。デフォルト値は <code>false`</code> です。
MFA_FORCE_REGISTRATION	<code>true`</code> に設定して、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ます。デフォルト値は <code>false`</code> です。
MFA_SUPPORTED_METHODS	使用する認証方法 (ワンタイムコードをメールで送信) を使用する場合は <code>email`</code> 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 <code>authenticator`</code> に設定します。
MFA_EMAIL_SUBJECT	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 <code><APP_NAME>- Verification Code`</code> です。

メールサーバー構成

メール認証方法を利用するには、メールサーバーの設定が必要です。

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。

3.4 Securing your deployment

PEM を構成するときは、考慮すべきセキュリティ慣行があります。これらのセキュリティ対策はオプションです。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、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プログラムを適用してセキュリティの脆弱性を回避します。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を減らす
- 生産性の低下を回避

- ランサムウェアなどのマルウェアからデータを保護
- 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する

3.4.1 Apache HTTPD security configurations

Windows、Apache HTTPD の名前は PEM HTTPD です。Apache HTTPD 構成ファイルは `pme.conf` で、SSL 構成ファイルは `httpd-ssl-pem.conf` です。両方の構成ファイルは `<Apache_Installation_Path>/conf/addons` ディレクトリにあります。

Linux では、Apache HTTPD 構成ファイルは `edb-pem.conf` であり、SSL 構成ファイルは `edb-ssl-pem.conf` です。両方の構成ファイルは `<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 を無効にします。これらのバージョンは最も脆弱で、暗号化の問題があります。

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ます。

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

Web サーバーを再起動して、構成ファイルに変更を適用します。

PEM は次の行を SSL 構成ファイルに追加して、セキュリティとして TLS 1.2 を使用できるようにします。

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

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

SSL 証明書で HTTPD を保護

Web アプリケーションには追加の SSL セキュリティレイヤーをお勧めします。

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 では、テキストエディターで Apache HTTPD 構成ファイル (`edb-ssl-pem.conf`) を開き、ファイルに対する書き込み権限を持つユーザーである必要があります。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。

PEM VirtualHost セクションで次の 2 つの SSL ディレクティブを更新します。

- `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、次の更新を行います。

```

SSLEngine on
SSLCertificateFile /path/to/your_domain_name.crt
SSLCertificateKeyFile /path/to/your_private.key

```

PEM 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、この

[Postgres tutorial](#) を参照してください。

Web サーバー情報の公開を無効にする

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 サーバーの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。PEM は、Apache HTTPD 構成ファイルにパラメーターを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。この情報の公開を制限するには、Apache HTTPD 構成ファイルでディレクティブを無効にすることをお勧めします。

Web サーバーを再起動して、Apache HTTPD 構成ファイルに変更を適用します。

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、攻撃者はディレクトリの完全な内容を表示できます。デフォルトでは、Web サーバーはこのリストを有効にしており、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。

このリスクを回避するには、Apache HTTPD 構成ファイルで **Options** 指示子を設定して、ディレクトリのリストを無効にします。PEM は、Web サーバー構成ファイルで次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>
```

Web サーバーを再起動して、構成ファイルに加えた変更を適用します。

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 では、アプリケーションにアクセスできるクライアントのホスト名または IP アドレスを提供できます。IP アドレスまたはネットワークへのアクセスを許可するには、Allow 指示子にネットワークアドレスを入力します。

```
<Directory /application/hostname>
Options None
AllowOverride None
Order deny,allow
Deny from all
Allow from 192.168.0.0/24
</Directory>
```

アプリケーション構成ファイルの ALLOWED_HOSTS 構成パラメーターは、許可されるホストを IP アドレスで設定します。アプリケーション構成ファイル (config_local.py) は <PEM_INSTALLATION_PATH>/web にあります。

PEM により、すべてのホストがアプリケーションに接続できます。

例：

```
# You can set the range of IP addresses in the configuration file:

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

# You can set the IP addresses to allow a host on a subnet level in the configuration
→file:
ALLOWED_HOSTS = [192.0.2.0/28, ::192.0.2.0/124]

# You can set a specific individual host address (based on the IP address) in the
→configuration file:
ALLOWED_HOSTS = [127.0.0.1, 192.168.0.1]
```

アプリケーション構成ファイルの変更を適用するには、Web サーバーを再起動します。

クロスサイトトレース

TRACE と TRACK の 2 つの HTTP メソッドを使用して、Web サーバー接続をデバッグできます。HTTP TRACE 要求がサポートされている Web サーバーに送信されると、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 構成ファイルに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 サービスを実行すると、セキュリティの問題が発生する可能性があります。Web サーバーは、特権のない一意のユーザーとして実行することをお勧めします。そうすることで、セキュリティ侵害中に実行されている他のサービスを保護できます。

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。一方、PEM は pem ユーザーとしてデーモンプロセスを実行します。Windows では、WSGIDaemonProcess 指示子と機能は使用できません。インストール中に、PEM HTTPD はサービスとしてインストールされます。このサービスを正常にインストールするには、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別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。config_local.py ファイルは一部のシステムにのみ存在します。アプリケーションレベルの構成をオーバーライドするには、これを作成する必要があります。PEM のアップグレード中に、config_local.py を上書きすることはできませんが、config.py および config_distro.py の変更を上書きすることはできます。PEM をアンインストールした後、config_local.py を削除します。

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 です。

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 を生成できます。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 は、ブラウザが<iframe>タグでページをレンダリングできるかどうかを示します。特にクリックジャッキングから保護します。PEM には、これら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 標準の一部です。置き換えられる X-Frame-Options ヘッダーよりも幅広い保護を提供します。Web サイトの作成者が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。作成者は、ホワイトリストに登録されたドメインから、またページを埋め込むことができるドメインからリソース（スクリプト、スタイルシート、フォントなど）をロードできます。

PEM には、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 (HSTS) 応答ヘッダーは、中間者攻撃を防ぐことができます。このオプションを有効にすると、Web サイトまたは Web アプリケーションはブラウザに HTTP ではなく HTTPS のみを受け入れることを通知します。デフォルトは次のとおりです。

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

注釈: このパラメーターを追加すると、構成が変更された場合に問題が発生する可能性があります。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追加することをお勧めします。

X-Content-Type-Options

X-Content-Type-Options 応答の HTTP ヘッダーはマーカーです。サーバーはマーカーを使用して、Content-Type ヘッダーでアドバタイズされた MIME タイプを変更したり追跡したりできないことを示します。以下は、MIME タイプのスニффイング、つまり MIME タイプが意図的に構成されていることをオプトアウトする方法です。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表示する Web ページにクライアントサイドの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、Internet Explorer、Chrome、および Safari の機能です。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、ページの読み込みを停止します。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不要です。ただし、これらの保護は、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します。

```
<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 サーバーを再起動します。

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。

変更を適用するには、Apache サービスを再起動します。

config.py ファイルの詳細については、Managing configuration settings を参照してください。

クッキーのセキュリティ

Cookie は、サーバーが構成データを保存するためにブラウザに送信するデータの小さなパケットです。ブラウザはそれらと他のすべての要求を同じサーバーに送信するため、Cookie を保護する方法を知ることが重要です。config.py の複数の構成オプションにより、Cookie を安全にできます。これらは 3 つの最も重要なオプションです。

- SESSION_COOKIE_SECURE —このフラグは、Cookie が暗号化されていない接続を介して送信されるのを防ぎます。ブラウザは、暗号化されたチャネルなしではサーバーへの要求に Cookie を追加できません。ブラウザは、HTTPS などの接続にのみ Cookie を追加できます。デフォルトは次のとおりです。

```
SESSION_COOKIE_SECURE = True
```

- SESSION_COOKIE_HTTPONLY —デフォルトでは、JavaScript は Cookie のコンテンツを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。代わりに、ブラウザは HTTP または HTTPS 要求でのみ Cookie を使用します。ハッカーは XSS の脆弱性を悪用して Cookie の内容を学習することはできません。たとえば、sessionId Cookie は、クライアントサイドのスクリプトで読み取る必要はありません。したがって、sessionId Cookie に 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
```

- **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。後続のすべての要求で、トークンは再計算して、最初の要求で計算されたものと比較します。セッション Cookie が盗まれ、攻撃者が別の場所から使用した場合、生成されるトークンは異なります。その場合、拡張機能はセッションをクリアし、要求をブロックします。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または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` に設定します。

変更を適用するには、Apache サービスを再起動します。

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

3.4.2 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な場合、他のセッションベースの攻撃の露出が増加します。攻撃者は、有効なセッション ID を再利用し、関連付けられたセッションをハイジャックするためのより多くの時間を持ちます。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。このセキュリティの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。

PEM では、ユーザセッションのタイムアウト値を設定できます。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 Web コンソールからユーザーをログアウトします。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、ユーザーごとではなく、アプリケーション全体を対象としています。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。

変更を適用するには、Apache サービスを再起動します。

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

RestAPI ヘッダーのカスタマイズ

要件に合わせて 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` ファイルで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、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 `X-Subject-Token` ヘッダーが含まれます。このヘッダーには、新しく生成されたトークンの値が含まれています。ヘッダー名をカスタマイズする場合は、`config_local.py` ファイルを更新できます。

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

このコマンドは次の出力を生成します。

```
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、HTTP 要求ヘッダー名を変更できます。このヘッダー名を使用して、トークンを PEM サーバーに送信できます。デフォルトでは、トークンを生成する要求を送信するときのトークンヘッダー名は `X-Auth-Token` です。 RestAPI リクエストヘッダー名をカスタマイズする場合は、`config_local.py` ファイルを更新できます。

```
PEM_HEADER_TOKEN_KEY = Pem-Token
```

この設定により、トークンを送信できます。

```
$ curl -Lk -X GET -H "Pem-Token: gw5rzaloxyp91tttd1c97w24b5sv60clic24sxy9" 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、`config_local.py` ファイルを更新できます。

```
PEM_TOKEN_EXPIRY = 600
```

変更を適用するには、Apache サービスを再起動します。

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、アプリケーションに対するユーザーのアクセスレベルを参照します。ユーザーは、自分の仕事を行うために必要な情報のみにアクセスできます。PEM のロールは、添え字ではなく継承可能で追加的です。言い換えれば、PEM 管理者は、ユーザーに最低レベルのロールを付与し、ユーザーがジョブ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 プロファイラーのみに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

多くの場合、SQL インジェクション攻撃の防止はアプリケーション開発者の責任ですが、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、標準のデータベースセキュリティポリシーに加えて、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調べます。

攻撃者は、いくつかの異なる手法を使用して SQL インジェクション攻撃を実行できます。特定の署名が各技術の特徴付けます。SQL/Protect は、許可されていないリレーション、ユーティリティコマンド、SQL トートロジー、および無制限の DML ステートメントのクエリを調べます。SQL/Protect は、管理者に潜在的に危険なクエリを警告し、それらのクエリ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 EDB Postgres Advanced Server サーバーでのみ機能するため、PEM データベースが EDB Postgres Advanced Server 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、[SQL プロファイラープラグイン](#)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新しいものに変更することです。パスワードの変更:

- 複数のアカウントの侵害を防ぎます
- 常時アクセスを防止
- 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ぎます
- キーストロークロガーによるアクセスを制限します

非 root ユーザーで 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 権限を持つ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、スクリプトを実行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、

Managing certificates を参照してください。

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 と ca_key.key の場所を変更しないでください。

3.5 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。

3.5.1 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。

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

2. 最初のホストで データベース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 WebServices 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: 実稼働環境の場合、ベストプラクティスは 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です。 ##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。 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ます。 **WebServicesandDatabase** 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。

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、ベストプラクティスは 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です。
