

Trusted Postgres Architect

リリース *23.44.0*

EnterpriseDB Corporation

2026 年 06 月 15 日

目 次

第 1 章 Introduction **3**
1.1 TPA では何ができますか? . . . 3
1.2 どのように使用すればよいですか？ . . . 3
1.2.1 構成 . . . 4
1.2.2 プロビジョニング . . . 4
1.2.3 導入 . . . 5
1.2.4 テスト . . . 5
1.2.5 インクリメンタルな変更 . . . 5
1.2.6 クラスター管理 . . . 5
1.2.7 Ansible を介して拡張可能 . . . 6
1.3 それはただの Postgres です . . . 6
1.4 TPA でのバージョニング . . . 6
1.4.1 下位互換性 . . . 6
1.5 はじめに . . . 7

第 2 章 Trusted Postgres Architect release notes **9**
2.1 Trusted Postgres Architect 23.44.0 release notes . . . 10
2.1.1 ハイライト . . . 10
2.1.2 機能強化 . . . 10
2.1.3 の変更 . . . 12
2.1.4 バグ修正 . . . 14
2.1.5 非推奨 . . . 20
2.2 Trusted Postgres Architect 23.43.0 release notes . . . 20
2.2.1 ハイライト . . . 20
2.2.2 機能強化 . . . 21
2.2.3 の変更 . . . 21
2.2.4 バグ修正 . . . 22
2.3 Trusted Postgres Architect 23.42.0 release notes . . . 23
2.3.1 ハイライト . . . 24
2.3.2 機能強化 . . . 24
2.3.3 の変更 . . . 26
2.3.4 バグ修正 . . . 27
2.4 Trusted Postgres Architect 23.41.0 release notes . . . 31
2.4.1 ハイライト . . . 32
2.4.2 機能強化 . . . 32
2.4.3 の変更 . . . 34
2.4.4 バグ修正 . . . 36
2.5 Trusted Postgres Architect 23.40.1 release notes . . . 38
2.5.1 バグ修正 . . . 38
2.6 Trusted Postgres Architect 23.40.0 release notes . . . 39

2.6.1 機能強化 . . . 39
2.6.2 の変更 . . . 39
2.6.3 バグ修正 . . . 40
2.7 Trusted Postgres Architect 23.39.0 release notes . . . 40
2.7.1 ハイライト . . . 40
2.7.2 機能強化 . . . 40
2.7.3 の変更 . . . 41
2.7.4 バグ修正 . . . 42
2.8 Trusted Postgres Architect 23.38.1 release notes . . . 44
2.8.1 バグ修正 . . . 44
2.9 Trusted Postgres Architect 23.38.0 release notes . . . 45
2.9.1 ハイライト . . . 45
2.9.2 機能強化 . . . 46
2.9.3 の変更 . . . 46
2.9.4 バグ修正 . . . 48
2.10 Trusted Postgres Architect 23.37.0 release notes . . . 50
2.10.1 ハイライト . . . 50
2.10.2 機能強化 . . . 50
2.10.3 の変更 . . . 51
2.10.4 バグ修正 . . . 51
2.11 Trusted Postgres Architect 23.36.0 release notes . . . 52
2.11.1 ハイライト . . . 52
2.11.2 機能強化 . . . 52
2.11.3 の変更 . . . 53
2.11.4 バグ修正 . . . 54
2.12 Trusted Postgres Architect 23.35.0 release notes . . . 55
2.12.1 ハイライト . . . 56
2.12.2 機能強化 . . . 56
2.12.3 の変更 . . . 58
2.12.4 バグ修正 . . . 59
2.13 Trusted Postgres Architect 23.34.1 release notes . . . 60
2.14 Trusted Postgres Architect 23.34 release notes . . . 60
2.15 Trusted Postgres Architect 23.33 release notes . . . 62
2.16 Trusted Postgres Architect 23.32 release notes . . . 64
2.17 Trusted Postgres Architect 23.31 release notes . . . 65
2.18 Trusted Postgres Architect 23.30 release notes . . . 66
2.19 Trusted Postgres Architect 23.29 release notes . . . 67
2.20 Trusted Postgres Architect 23.28 release notes . . . 67
2.21 Trusted Postgres Architect 23.27 release notes . . . 69
2.22 Trusted Postgres Architect 23.26 release notes . . . 69
2.23 Trusted Postgres Architect 23.25 release notes . . . 70
2.24 Trusted Postgres Architect 23.24 release notes . . . 71
2.25 Trusted Postgres Architect 23.23 release notes . . . 72
2.26 Trusted Postgres Architect 23.22 release notes . . . 73
2.27 Trusted Postgres Architect 23.21 release notes . . . 73
2.28 Trusted Postgres Architect 23.20 release notes . . . 73
2.29 Trusted Postgres Architect 23.19 release notes . . . 74

2.30 Trusted Postgres Architect 23.18 release notes . . . 75
2.31 Trusted Postgres Architect 23.17 release notes . . . 76
2.32 Trusted Postgres Architect 23.16 release notes . . . 76
2.33 Trusted Postgres Architect 23.15 release notes . . . 77
2.34 Trusted Postgres Architect 23.14 release notes . . . 77
2.35 Trusted Postgres Architect 23.13 release notes . . . 78
2.36 Trusted Postgres Architect 23.12 release notes . . . 78
2.37 Trusted Postgres Architect 23.1 to 23.11 release notes . . . 79
2.37.1 TPA 23.11 . . . 79
2.37.2 TPA 23.10 . . . 80
2.37.3 TPA 23.9 . . . 80
2.37.4 TPA 23.8 . . . 81
2.37.5 TPA 23.7 . . . 81
2.37.6 TPA 23.6 . . . 82
2.37.7 TPA 23.5 . . . 83
2.37.8 TPA 23.4 . . . 83
2.37.9 TPA 23.3 . . . 83
2.37.10 TPA 23.2 . . . 85
2.37.11 TPA 23.1 . . . 85

第 3 章 TPA installation **89**
3.1 クイックスタート . . . 89
3.1.1 TPA のインストール . . . 89
3.1.2 追加の依存関係をインストールする . . . 89
3.1.3 インストールの検証一般のユーザーとして実行 . . . 89
3.2 TPA をインストールする場所 . . . 89
3.3 TPA パッケージのインストール . . . 90
3.3.1 Debian または Ubuntu にインストールする . . . 90
3.3.2 RHEL、Rocky、AlmaLinux、または Oracle Linux にインストールする . . . 90
3.3.3 SLES . . . 90
3.4 TPA Python 環境のセットアップ . . . 90
3.5 インターネットまたはネットワークアクセスエアギャップなしで TPA をインストールする 91
3.5.1 TPA パッケージのダウンロード . . . 91
3.5.2 Python パッケージインデックスへのアクセスを使用しないインストール . . . 91
3.6 TPA インストールの検証 . . . 92
3.7 TPA のアップグレード . . . 92
3.8 Ansible バージョン . . . 92

第 4 章 Open source TPA **93**
4.1 Trusted Postgres ArchitectTPA とは何ですか? . . . 93
4.2 次のステップ . . . 93
4.3 TPA オープンソースのよくある質問 . . . 93
4.3.1 EDB の顧客でない場合、これを使用できますか? . . . 93
4.3.2 問題を報告できますか? . . . 93
4.3.3 貢献できますか? . . . 93

第 5 章 Known issues **95**

5.1 複数のロケーションを持つ EFM クラスターにロケーション間の HBA ルールがありません TPA-1247 . . . 95
5.1.1 詳細 . . . 95
5.1.2 回避策 . . . 95
5.2 TPA 23.34 以前で展開された PGD クラスターには primary_slot_name 設定があり、bdr_init_physical が動作しない TPA-1229 . . . 96
5.2.1 詳細 . . . 96
5.2.2 回避策 . . . 96
5.3 forward_ssh_agent: true を使用したクラスターでの SSH 構成テンプレートエラー . . . 96
5.3.1 詳細 . . . 96
5.3.2 回避策 . . . 96
5.4 pemagent_registration_opts が提供する PEM エージェント登録オプションは agent.cfg に反映されないため、有効になりません TPA-1376 . . . 96
5.4.1 詳細 . . . 96
5.4.2 回避策 . . . 97
5.5 TPA を使用して展開すると、PEM 10.4.x Web アプリケーションに「内部サーバーエラー」が表示される TPA-1415 . . . 97
5.5.1 詳細 . . . 97
5.5.2 回避策 . . . 97

第 6 章 Installing TPA from source **99**
6.1 前提条件 . . . 99
6.2 クローンとセットアップ . . . 99
6.3 依存関係 . . . 100
6.3.1 Python 3.12+ . . . 100
6.3.2 仮想環境オプション . . . 101

第 7 章 CREATING A CLUSTER **103**
7.1 A First Cluster Deployment . . . 103
7.1.1 TPA のインストール . . . 103
7.1.2 Docker のインストール . . . 103
7.1.3 TPA を使用した構成の作成 . . . 104
7.2 Cluster configuration . . . 107
7.2.1 クイックスタート . . . 107
7.2.2 構成オプション . . . 107
7.2.3 ソフトウェアの選択 . . . 110
7.2.4 オーバーライド . . . 113
7.2.5 Ansible Tower . . . 114
7.2.6 PGAI エージェント . . . 114
7.2.7 Git リポジトリ . . . 114
7.2.8 ボールトパスワードのキーリングバックエンド . . . 114
7.2.9 例 . . . 115
7.3 tpaexec provision . . . 117
7.3.1 前提条件 . . . 118
7.3.2 クイックスタート . . . 118
7.3.3 オプション . . . 119
7.3.4 インスタンスへのアクセス . . . 119

7.3.5 生成されたファイル . . . 120
7.4 tpaexec deploy . . . 122
7.4.1 前提条件 . . . 122
7.4.2 クイックスタート . . . 122
7.4.3 選択的な展開 . . . 123
7.4.4 deploy.yml . . . 123
7.5 tpaexec test . . . 123
7.5.1 クイックスタート . . . 123

第 8 章 ARCHITECTURES 125
8.1 PGD-S . . . 125
8.1.1 クラスター構成 . . . 125
8.2 PGD-X . . . 127
8.2.1 クラスター構成 . . . 127
8.3 PGD-Always-ON . . . 129
8.3.1 クラスター構成 . . . 129
8.4 PGD Lightweight . . . 132
8.4.1 クラスター構成 . . . 132
8.5 BDR-Always-ON . . . 133
8.5.1 クラスター構成 . . . 134
8.6 M1 . . . 135
8.6.1 フェイルオーバー管理 . . . 136
8.6.2 クラスター構成 . . . 137

第 9 章 PLATFORMS 139
9.1 aws . . . 139
9.1.1 API アクセス設定 . . . 139
9.1.2 はじめに . . . 141
9.1.3 構成 . . . 142
9.2 bare(-metal servers) . . . 146
9.2.1 SSH アクセス . . . 147
9.2.2 ディストリビューションのサポート . . . 147
9.2.3 IP アドレス . . . 147
9.2.4 新しく始める . . . 148
9.3 Docker . . . 148
9.3.1 概要 . . . 148
9.3.2 オペレーティングシステムの選択 . . . 148
9.3.3 Docker のインストール . . . 149
9.3.4 Docker コンテナ管理 . . . 152

第 10 章 CUSTOMIZATIONS 153
10.1 Cluster configuration . . . 153
10.1.1 config.yml . . . 153
10.1.2 変数 . . . 154
10.1.3 クラスター変数 . . . 154
10.1.4 インスタンス変数 . . . 155
10.1.5 instance_defaults . . . 156
10.1.6 ロケーション . . . 157

10.2 Instance configuration . . . 157
10.2.1 システムレベルの構成 . . . 158
10.2.2 Postgres . . . 159
10.2.3 その他のコンポーネント . . . 160
10.2.4 Postgres を起動した後 . . . 160
10.3 Building from source . . . 161
10.3.1 クイックスタート . . . 161
10.3.2 構成 . . . 162
10.3.3 ローカルソースディレクトリ . . . 162
10.3.4 リビルド . . . 164
10.4 TPA hooks . . . 164
10.4.1 概要 . . . 164
10.4.2 汎用フック . . . 164
10.4.3 PGD フック . . . 166
10.4.4 その他のフック . . . 167
10.4.5 新しいフック . . . 167

第 11 章 CLUSTER MANAGEMENT **169**
11.1 Upgrading your cluster . . . 169
11.1.1 アップグレードする前に . . . 170
11.1.2 アップグレード中 . . . 172
11.1.3 アップグレード後 . . . 173
11.1.4 失敗したアップグレードからの復旧 . . . 174
11.1.5 よくある落とし穴 . . . 175
11.1.6 コンポーネントの選択 . . . 175
11.1.7 パッケージバージョンの選択 . . . 176
11.1.8 構成 . . . 177
11.1.9 BDR-Always-ON から PGD-Always-ON へのアップグレード . . . 177
11.1.10 PGD-Always-ON から PGD-X へのアップグレード . . . 178
11.1.11 PGD-S または PGD-X . . . 181
11.1.12 PGD-Always-ON . . . 182
11.1.13 BDR-Always-ON . . . 182
11.1.14 M1 . . . 183
11.1.15 アップグレードプロセスの制御 . . . 183
11.1.16 ノードのサブセットのアップグレード . . . 184
11.2 tpaexec switchover . . . 184
11.2.1 例 . . . 185
11.2.2 アーキテクチャオプション . . . 185
11.2.3 Repmgr リダイレクト pgbouncer . . . 185
11.3 BDR/HAProxy server pool management . . . 185
11.3.1 例 . . . 186
11.4 tpaexec rehydrate . . . 186
11.4.1 前提条件 . . . 186
11.4.2 例 . . . 187
11.4.3 段階的に rehydration する . . . 188
11.4.4 付録 . . . 188

第 12 章 MISCELLANEOUS **189**
12.1 TPA and Ansible Tower/Ansible Automation Platform . . . 189
12.1.1 AAP の初期セットアップ . . . 189
12.1.2 クラスターのセットアップ . . . 190
12.1.3 カスタムプレイブックの実行 . . . 192
12.1.4 複数のインベントリに 1 つのプロジェクトを使用する . . . 192
12.1.5 AAP で TPA を更新 . . . 193
12.1.6 TPA の EE をビルドする . . . 194
12.2 TPA, Ansible, and sudo . . . 195
12.2.1 Ansible sudo 呼び出し . . . 195
12.2.2 おすすめ . . . 196
12.2.3 SSH および sudo パスワード . . . 196
12.2.4 sudo オプション . . . 197
12.2.5 特権エスカレーション構成の管理 . . . 197
12.2.6 ロギング . . . 199
12.2.7 ローカル権限 . . . 199
12.3 TPA - PuTTY Configuration guide . . . 199
12.3.1 キー変換 . . . 200
12.3.2 **PuTTY の構成** . . . 200
12.4 Troubleshooting . . . 201
12.4.1 Python 仮想環境を再作成します . . . 201
12.4.2 資格情報に関する奇妙な AWS エラー . . . 201
12.4.3 ロギング . . . 202
12.4.4 クラスターテスト . . . 202
12.4.5 TPA サーバーテスト . . . 202
12.4.6 特定のタスクを含めるまたは除外する . . . 202
12.5 Selective task execution . . . 203
12.5.1 タスクセレクターを使用する . . . 203
12.5.2 例 . . . 203
12.5.3 `tpaexec deploy` でサポートされているセレクター . . . 203
12.5.4 `tpaexec test` でサポートされているセレクター . . . 208
12.6 Reference . . . 209
12.6.1 Running TPA in a Docker container . . . 209
12.6.2 Managing clusters in a disconnected or air-gapped environment . . . 210
12.6.3 Distribution support . . . 212
12.6.4 TPA capabilities and supported software . . . 213
12.6.5 Reconciling changes made outside of TPA . . . 214
12.6.6 EDB Postgres Distributed configuration . . . 219
12.6.7 Barman . . . 224
12.6.8 Configuring EFM . . . 230
12.6.9 Configuring haproxy . . . 233
12.6.10 Configuring HARP . . . 235
12.6.11 Configuring Postgres Enterprise Manager (PEM) . . . 239
12.6.12 PgBouncer . . . 245
12.6.13 pgd-proxy . . . 247
12.6.14 pglogical . . . 251
12.6.15 repmgr . . . 256

12.6.16 Configuring EDB Repos 2.0 repositories . . . 258
12.6.17 Configuring APT repositories . . . 258
12.6.18 Configuring YUM repositories . . . 259
12.6.19 Creating and using a local repository . . . 260
12.6.20 Installing from source . . . 263
12.6.21 Git credentials . . . 264
12.6.22 Environment . . . 265
12.6.23 Python environment . . . 266
12.6.24 Configuring /etc/hosts . . . 268
12.6.25 Filesystem configuration . . . 268
12.6.26 Uploading artifacts . . . 274
12.6.27 ssh_key_file . . . 275
12.6.28 Managing SSH host keys . . . 276
12.6.29 Postgres source installation . . . 276
12.6.30 Installing packages . . . 278
12.6.31 Running initdb . . . 279
12.6.32 Installing Postgres-related packages . . . 280
12.6.33 PG Backup API . . . 280
12.6.34 Setting sysctl values . . . 282
12.6.35 Creating Postgres databases . . . 283
12.6.36 Creating Postgres tablespaces . . . 284
12.6.37 Configuring .pgpass . . . 285
12.6.38 The postgres Unix user . . . 286
12.6.39 Creating Postgres users . . . 287
12.6.40 tpaexec archive-logs . . . 288
12.6.41 tpaexec download-packages . . . 290
12.6.42 TPA custom commands . . . 292
12.6.43 TPA custom tests . . . 293
12.6.44 Configuring the PGAI agent . . . 295
12.6.45 Compliance . . . 297
12.6.46 etcd . . . 299
12.6.47 Locale . . . 302
12.6.48 Patroni cluster management commands . . . 302
12.6.49 pg_hba.conf . . . 309
12.6.50 pg_ident.conf . . . 310
12.6.51 pgd-cli . . . 310
12.6.52 Adding Postgres extensions . . . 311
12.6.53 postgresql.conf . . . 313
12.6.54 tpaexec deprovision . . . 318
12.6.55 tpaexec info . . . 318
12.6.56 tpaexec reconfigure . . . 319
12.7 tpaexec eval . . . 320
12.7.1 使用法 . . . 320
12.7.2 出力例 . . . 321

© Copyright EnterpriseDB UK Limited 2015-2026 - 無断複写・転載を禁じます。

第1章　Introduction

TPA は、Ansible を使用して EDB の推奨事項に従って Postgres クラスターを展開するオーケストレーションツールです。

TPA は、Postgres の展開とサポートに関する長年の苦労して得た経験に基づいて、EDB が追従するベストプラクティスを具体化しています。これらの推奨事項は、運用環境と同様に簡単なテストベッドのセットアップに適用されます。

1.1 TPA では何ができますか?

TPA は、トポロジから構成の最小の詳細まで Postgres クラスターを記述するために使用できる宣言的構成メカニズムを中心に構築されています。

`tpaexec configure` を実行して、いくつかの高レベルの選択たとえば、インストールする Postgres のバージョンなどに基づいて、初期クラスター構成を生成することから始めます。デフォルト構成はそのまま使用できますが、ニーズに合わせて編集できます。生成される構成は単なるテキストファイル config.yml です。

この構成を使用して、TPA は次のことを行うことができます。

1. サーバー AWS EC2 インスタンスまたは Docker コンテナなど、クラスターをホストするために必要なその他のリソースをプロビジョニングしますまたは、接続の詳細を指定するだけで既存のサーバーまたは VM に展開できます。
2. オペレーティングシステムを構成しますカーネル設定の調整、ユーザーと SSH キーの作成、パッケージのインストール、systemd サービスの定義、ログローテーションの設定など。
3. Postgres および関連コンポーネントたとえば、 PGD、Barman、pgbouncer、repmgr、およびさまざまな Postgres 拡張機能などをインストールおよび構成します。
4. 展開後にクラスターで自動テストを実行します。
5. 構成に将来の変更を展開します Postgres 設定の変更、パッケージのインストールとアップグレード、新しいサーバーの追加など。

1.2 どのように使用すればよいですか？

TPA を使用するには、TPA をインストールし、`tpaexec setup` コマンドを実行する必要があります。ご使用のプラットフォームの *TPA installation* に従ってください。

TPA は 4 つの異なるステージで動作して、Postgres クラスターを起動します。

- クラスターの生成 **クラスター構成**
- クラスターをホストする **デプロイメントのプロビジョニング** サーバー VM、コンテナ

- プロビジョニングされたインスタンスへの *A First Cluster Deployment* ソフトウェア
- テスト デプロイされたクラスター

```
#  1. Configuration: decide what kind of cluster you want

tpaexec configure clustername --architecture M1 --platform aws \
        --postgresql 14 \
        --failover-manager repmgr

#  2. Provisioning: create the servers needed to host the cluster

tpaexec provision clustername

#  3. Deployment: install and configure the necessary software

tpaexec deploy clustername

#  4. Testing: make sure everything is working as expected

tpaexec test clustername
```

TPA は、ラップトップ、EC2 インスタンス、またはネットワーク経由でクラスターのサーバーに到達できるマシンから実行できます。

これは *TPA capabilities and supported software* です。

1.2.1 構成

クラスター構成

コマンドは、選択したオプションに基づいて、クラスターを記述する単純な YAML 構成ファイルを生成します。この構成はすぐに使用できますが、ニーズに合わせて変更できます。構成ファイルの編集は、**クラスター構成** の作成前と作成後の両方での通常の方法です。

この段階で、クラスターのアーキテクチャとプラットフォームを選択する必要があります。**アーキテクチャ**は、特定の目的に Postgres をセットアップするためのサーバーとソフトウェアの推奨レイアウトです。例には、「M1」プライマリおよびストリーミングレプリカを備えた Postgres および「PGD-Always-ON」Always On 構成の EDB Postgres 分散 5 が含まれます。 **プラットフォーム**は、AWS、Docker、ベアメタルサーバーなどのアーキテクチャを展開するサーバーをホストする手段です。

1.2.2 プロビジョニング

tpaexec provision

コマンドは、クラスターに必要なインスタンスとその他のリソースを作成します。プロセスの詳細は、クラスターの構成中に選択したアーキテクチャ M1 などとプラットフォーム AWS などによって異なります。

たとえば、必要な特権で AWS アクセスが与えられると、TPA は EC2 インスタンス、VPC、サブネット、ルーティングテーブル、インターネットゲートウェイ、セキュリティグループ、EBS ボリューム、Elastic IP などをプロビジョニングします。

「ベア」プラットフォームを選択し、接続の詳細を提供することにより、既存のサーバーを「プロビジョニング」することもできます。これらはベアメタルサーバーであるか、クラウドプラットフォームで別途プロビジョニングされたサーバーであるかにかかわらず、TPA によって作成されたかのように使用できます。

単一のプラットフォームに制限されません。必要に応じて、クラスターをいくつかの AWS インスタンスマルチプルのリージョンと一部のオンプレミスサーバー、または他のデータセンターのサーバーに分散できます。

プロビジョニングステージの最後に、基本的なオペレーティングシステムがインストールされた必要な数のインスタンスができます。TPA は、SSH 経由でルートに sudo を使用してアクセスできます。

1.2.3 導入

tpaexec deploy

コマンドは、プロビジョニングされたサーバーに Postgres およびその他のソフトウェアをインストールおよび構成します TPA によって作成された場合とされていない場合があります。ただし、SSH と sudo アクセスが利用できる限り誰が作成したかは関係ありません。これには、レプリケーション、バックアップなどの設定が含まれます。

展開段階の最後に、Postgres が起動して実行されます。

1.2.4 テスト

tpaexec テストを実行する コマンドは、展開されたクラスターに対してさまざまなアーキテクチャとプラットフォーム固有のテストを実行して、予想通りに動作していることを確認します。

テスト段階の最後には、完全に機能するクラスターが完成します。

1.2.5 インクリメンタルな変更

TPA は、プロビジョニング、展開、およびテストがべき等であるように注意深く設計されています。それらを実行し、config.yml を変更し、プロセスを再度実行して変更を展開できます。構成またはインスタンスで何も変更されない場合、プロセス全体を再実行しても何も変更されません。

1.2.6 クラスター管理

クラスターが起動して実行されると、TPA は、構成変更、スイッチオーバー、ゼロダウンタイムのマイナーバージョンのアップグレードなどの便利なクラスター管理機能を提供します。これらの機能により、手動で変更を行うよりもクラスターの管理が簡単かつ安全になります。

1.2.7 Ansible を介して拡張可能

TPA は *Instance configuration* をサポートしているため、config.yml を編集し、プロビジョニング/展開/テストを再実行するだけで、多くのことができます。TPA が既にサポートしているものを超える必要がある場合は、次のように書くことができます

- *TPA custom commands* 、クラスターで実行するプレイブックの作成を簡単にします。クラスターディレクトリに `commands/xyz.yml` を作成し、 `tpaexec xyz /path/to/cluster` を使用して呼び出すだけです。自動化する必要がある管理タスクまたはプロセスに最適です。
- *TPA custom tests* 、環境とアプリケーションに固有の詳細な検証でビルトインテストを強化します。`tpaexec test` を使用してすべてのテストを均一で反復可能な方法で実行すると、危機に対処するときも、日常的なクラスター管理中にも、重要な要素を見逃すことがありません。
- *TPA hooks* 、展開のさまざまな段階で呼び出されます。たとえば、`hooks/pre-deploy.yml` のタスクは、メイン展開の前に実行されます。`post-deploy` を含む他にも多くのフックがあります。これにより、Ansible の全機能を自由に利用できます。

1.3 それはただの Postgres です

TPA は、多くの機能が構成された複雑なクラスターを作成できますが、結果は単なる Postgres です。インストールは、生活を簡素化するように設計されたいくつかの規則に従いますが、あなたとデータベースの間には、隠された魔法も何も存在しません。他の Postgres インストールで実行できることはすべて TPA クラスターで実行できます。

1.4 TPA でのバージョニング

TPA は、以前にメジャーバージョンが年から派生する日付ベースのバージョン管理スキームを使用していました。バージョン 23 から TPA はセマンティックバージョニングに移行し、最初は 2 部の `major-minor` パターンを使用して、バージョン 23.34.1 で完全な 3 部セマンティックバージョニングを採用しました。このスキームでは、メジャーバージョンは、以下の下位互換性の原則に準拠するために必要な場合にのみインクリメントされます。

1.4.1 下位互換性

TPA の重要な開発原則は、下位互換性を維持することであるため、ユーザーが TPA の最新バージョン以外のものが必要になる理由はありません。下位互換性を次のように定義します。

- TPA X.a で作成された config.yml は、TPA X.b で有効です。ここで、 b>=a
- その config.yml から作成されたクラスターは、TPA X.b で保守および再展開が可能です。

したがって、新しいメジャーバージョンは、下位互換性の中断を意味します。そのため、私たちはメジャーバージョンのリリースを回避することを目指しており、例外的な状況でのみそうします。

1.5 はじめに

システムの *TPA installation* に従って、次に **クラスター構成** に従います。

第2章　Trusted Postgres Architect release notes

Trusted Postgres Architect ドキュメントでは、Trused Postgres Architect 23 の最新バージョンについて説明しています。

Trusted Postgres Architect version	Release Date
Trusted Postgres Architect 23.44.0 release notes	10 Jun 2026
Trusted Postgres Architect 23.43.0 release notes	18 Mar 2026
Trusted Postgres Architect 23.42.0 release notes	25 Feb 2026
Trusted Postgres Architect 23.41.0 release notes	26 Nov 2025
Trusted Postgres Architect 23.40.1 release notes	28 Oct 2025
Ansible Tower	02 Oct 2025
Trusted Postgres Architect 23.39.0 release notes	22 Aug 2025
Trusted Postgres Architect 23.38.1 release notes	25 Jun 2025
Trusted Postgres Architect 23.38.0 release notes	09 Jun 2025
Trusted Postgres Architect 23.37.0 release notes	24 Mar 2025
Trusted Postgres Architect 23.36.0 release notes	19 Feb 2025
Trusted Postgres Architect 23.35.0 release notes	25 Nov 2024
Trusted Postgres Architect 23.34.1 release notes	09 Sep 2024
Trusted Postgres Architect 23.34 release notes	22 Aug 2024
Trusted Postgres Architect 23.33 release notes	24 Jun 2024
Trusted Postgres Architect 23.32 release notes	15 May 2024
Trusted Postgres Architect 23.31 release notes	19 Mar 2024
Trusted Postgres Architect 23.30 release notes	19 Mar 2024
Trusted Postgres Architect 23.29 release notes	15 Feb 2024
Trusted Postgres Architect 23.28 release notes	23 Jan 2024
Trusted Postgres Architect 23.27 release notes	19 Dec 2023
Trusted Postgres Architect 23.26 release notes	30 Nov 2023
Trusted Postgres Architect 23.25 release notes	14 Nov 2023
Trusted Postgres Architect 23.24 release notes	17 Oct 2023
Trusted Postgres Architect 23.23 release notes	21 Sep 2023
Trusted Postgres Architect 23.22 release notes	06 Sep 2023
Trusted Postgres Architect 23.21 release notes	05 Sep 2023
Trusted Postgres Architect 23.20 release notes	01 Aug 2023
Trusted Postgres Architect 23.19 release notes	12 Jul 2023
Trusted Postgres Architect 23.18 release notes	23 May 2023
Trusted Postgres Architect 23.17 release notes	10 May 2023
Trusted Postgres Architect 23.16 release notes	21 Mar 2023
Trusted Postgres Architect 23.15 release notes	15 Mar 2023

次のページに続く

表 1 – 前のページからの続き

Trusted Postgres Architect version	Release Date
Trusted Postgres Architect 23.14 release notes	23 Feb 2023
Trusted Postgres Architect 23.13 release notes	22 Feb 2023
Trusted Postgres Architect 23.12 release notes	21 Feb 2023
Trusted Postgres Architect 23.1 to 23.11 release notes	21 Jun 2023 to 31 Jan 2023

2.1 Trusted Postgres Architect 23.44.0 release notes

リリース日 2026 年 6 月 10 日

Trusted Postgres Architect 23.44.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.1.1 ハイライト

- PGD-X クラスター内のコンポーネントのアップグレードのサポート
- PGD アップグレードの堅牢性の向上
- EFM ユーザーはデフォルトでスーパーユーザーでなくなりました

2.1.2 機能強化

Description	Addresses
TPA は、 PGD-X アーキテクチャのコンポーネントのアップグレードをサポートするようになりました。 PGD-X のアップグレードプレイブックには、tpaexec `tpaexec upgrade (...) --components=<component>`を介して選択したコンポーネント PgBouncer、pg-backup-api、 Barman、PEM サーバーを確実にアップグレードする追加手順が含まれているため、個々のコンポーネントは必要なくアップグレードできます。クラスター全体をアップグレードします。このアップグレードプロセスは、コンポーネントのアップグレード、マイナー PGD-X アップグレード、および PGD-X アーキテクチャへのメジャーアップグレードのスタンドアロン機能として含まれています。	
TPA は、必要な最小限の権限のみで EFM データベースユーザーを展開できるようになりました。通常の EFM 展開では、TPA は EFM ユーザーに SUPERUSER 特権を付与しました。 `efm_user_is_superuser`と呼ばれる新しい変数が導入され、TPA は Postgres SUPERUSER として、または特定のカタログ機能と事前定義されたロール pg_monitor、 EFM が必要とする pg_read_all_settings、pg_read_all_stats 変数はスイッチとして機能し、通常の展開で切り替えることができ、既に展開されたクラスターで使用して EFM ユーザーの特権を切り替えることができます。	

次のページに続く

表 2 – 前のページからの続き

Description	Addresses
TPA が Ansible Automation Platform 2.6 をサポートするようになりました。 TPA は AAP 2.6 でテストされています。既存の実行環境は、AAP 2.4 および AAP 2.6 の両方と互換性があります。	
PGD 5 から PGD 6 へのアップグレード中にプロキシモニタリングのサポートを追加しました。 `tpaexec upgrade`への `enable_proxy_monitoring=yes`オプションは、 PGD-Always-ON PGD 5 から PGD-X PGD 6 へのアップグレード中に使用できるようになりました。以前は、モニタが接続マネージャーを実行しない監視ノードの接続マネージャーポートに接続しようとしたため、アップグレードがすぐに失敗していました。 Witness ノードが、監視のターゲットであるエンドポイントのリストから除外されます。	
TPA はデフォルト以外の巨大ページサイズをサポートするようになりました。 TPA の hugepages 設定は、以前にアーキテクチャのデフォルトのページサイズ x86_64 で 2MB を想定していたため、Postgres で 1GB huge page を使用する構成では、ユーザーがカーネルコマンドライン、*vm.nr_hugepages* sysctl、および `huge_page_size` GUC、および値を設定する必要がありました生成された TPA はそれらに対して作用しました。 TPA は、 ***huge_page_size*`変数 `*1GB*`などの *Postgres* スタイルのメモリ文字列をサポートするようになりました。設定されると、*TPA* は `*hugepagesz*=`を使用してカーネルコマンドラインでそのサイズのページを予約し、**/etc/sysctl.conf から `vm.nr_hugepages`を省略し、アーキテクチャのデフォルトサイズのプールにのみ適用され、Postgres が選択されたプールから描画するように `huge_page_size`を `postgresql.conf`に設定します。ページ数は、新しい `nr_hugepages`変数でオーバーライドできます。下位互換性のため、 `nr_hugepages`が明示的に設定されていない場合でも、既存の `sysctl_values['vm.nr_hugepages']`が優先されます。すべての予約はブート時に通常のメモリから取得されるため、 `huge_page_size`は、通常、クラスター全体ではなく個々の Postgres インスタンスで設定するのが最適です。	
repmgr サービスユニットファイルに環境変数オプションを追加しました。 `repmgr_service_environment`と呼ばれる新しい変数が repmgr の repmgr サービスユニットテンプレートファイルに追加されました。この変数を使用すると、ユーザーは repmgr サービスに設定されるカスタム環境変数を指定できます。 `repmgr_service_environment`変数は、クラスター構成ファイルに、キーが環境変数の名前であり、値がこれらの変数に対応する値である辞書として定義できます。これは、repmgr プロセスがカスタムライブラリパスや認証資格情報など、特定のランタイム値へのアクセスが必要な環境に役立ちます。	56200
TPA の選択した Python インタープリターとパッケージに対して、サポートされている cluster_vars オーバーライドを追加しました。 2 つの新しいまたは新しく有効な cluster_vars が、TPA がクラスターノードで Python を使用する方法を制御するようになりました。詳細については、 `docs/src/python.md`を参照してください。複数の Python インタープリターがインストールされているホストで 2 つの変数をカップリングすることの推奨を含みます。	

次のページに続く

表 2 – 前のページからの続き

Description	Addresses
TPA は、PGD 6 マイナーアップグレード中に enable_proxy_monitoring をサポートするようになりました。 `tpaexec upgrade`への `enable_proxy_monitoring=yes`オプションは、既にサポートされていたメジャーアップグレードパスに加えて、PGD 6 から PGD 6 のマイナーアップグレードでも機能するようになりました。TPA は有効にすると、ローリングフェンス/再起動サイクル中に各データノードで接続マネージャーのダウンタイムを記録します。	
再起動とサービスの除外タスクのカバレッジが向上しました。 TPA は、Barman、beacon-agent、EFM、etcd、harp、Patroni、PEM エージェントとサーバー、pgbackupapi、PgBouncer、PGD プロキシを含むサービスの再起動または systemd サービス管理を実行するすべてのロールにわたって `excluded_tasks`の `restart`および `service`値を尊重するようになりました、Postgres、repmgr 再起動、スイッチオーバー、レプリカファイナル、rsyslog、および OpenVPN 以前は、 `restart`または `service`タスクを除外するユーザーは、展開中にこれらのエリアで意図しない再起動が発生する可能性がありました。この変更により、すべての再起動およびダイレクトサービス操作が一貫してガードされるため、excluded_tasks はそれらを確実に抑制します。	
`tpaexec reconfigure`が--bdr-package-version を受け入れられるようになりました。tpaexec reconfigure は、BDR-Always-ON、PGD-Always-ON、PGD-X、および PGD-S クラスターで--bdr-package-version を受け入れ、config.yml に値を記録し、バージョンゲートされた構成オプションを適用するようになりました選択したバージョンには以下が必要です。現在、このようなオプションは PGD 5.5+の default_pgd_proxy_options の read_listen_port であり、PGD 5.5 クラスターの `tpaexec configure`の動作と一致します。新しいロジックは、 BDR-Always-ON → PGD-Always-ON および PGD-Always-ON → PGD-X アーキテクチャ変更の require() としても実行されるため、これらのアップグレードパスはバージョンゲートされたオプションを選択します。 -bdr-package-version が提供されます。	
TPA は、PGD 5 マイナーアップグレードの enable_proxy_monitoring をサポートするようになりました。 Enable_proxy_monitoring オプションは、PGD-Always-ON クラスターでの PGD 5 マイナーバージョンのアップグレード中に機能するようになりました。以前は、プロキシモニタリングは、 BDR 4 から PGD 5 メジャーアップグレード中にのみサポートされていました。プロキシモニタは、有効にすると、アップグレードプロセス全体で pgd-proxy エンドポイントを介した接続の可用性を追跡し、中断が報告されます。	

2.1.3 の変更

Description	Addresses
upgrade_legacy.yml を専用の upgrade_major_4to5.yml プレイブックに置き換えました。 BDR 4→5 メジャーアップグレードロジックは、モノリシック upgrade_legacy.yml から専用のプレイブックに抽出されました。これにより、すべての条件分岐、フェールオーバーマネージャーとしての harp のハードコーディング、およびバージョン 4 としての upgrade_from を削除することにより、アップグレードパスが簡素化されます。現在、孤立した upgrade_legacy.yml ファイルは、PGD-Always-ON および Lightweight アーキテクチャから削除されています。	
BDR 5.x マイナーアップグレード用の専用の upgrade_minor_5.yml プレイブックを追加しました。この新しいプレイブックは、すべての BDR 5.x マイナーアップグレードを処理します。これは、モノリシック upgrade_legacy.yml に以前に埋め込まれた関連ロジックを使用し、アップグレードプロセスのメンテナンスが簡単になります。	
TPA は、空の--team および--owner を pemworker に渡さなくなりました。Postgres サーバーを PEM に登録する際、 *monitoring_team* `または `*monitoring_server_owner* `変数が設定されていない場合、*TPA* は以前に--team ""および--owner ""`を`pemworker --register-server`に渡していました。以前の PEM バージョンは、これらの空の文字列をオプションを省略するのと同等のものとして扱いましたが、PEM 10.5 では CLI 検証を強化し、空の文字列を拒否するため、登録タスクが失敗します。 TPA は、対応する変数が設定されていない場合にこれらのフラグを完全に省略するようになったため、構成の変更を必要とせずに PEM 10.5 に対するサーバー登録が成功します。	
AAP でカスタムプレイブックを実行する方法を文書化しました。 AAP ユーザードキュメントでは、カスタムプレイブックをクラスターに追加し、`commands/`サブディレクトリ規約と AAP の Duplicate Template アクションを使用して、AAP を介して実行する方法を説明するようになりました。	
不足しているフラグを tpaexec ヘルプ出力に追加しました。 'tpaexec help' には、実際にサポートされているいくつかのフラグがありませんでした。この変更により、これらのフラグがヘルプ出力に追加されます。	
TPA は、展開の開始時に EPAS クラスターで pgaudit を拒否するようになりました。EPAS クラスターの `extra_postgres_extensions`に `pgaudit`を含めると、Postgres のスタートアップに遅くに失敗する展開が発生していました。これは、EPAS で pgaudit がロード可能ではないためです。 TPA は、展開の開始時にこの組み合わせを検出し、失敗して、EPAS のビルトイン監査機能を推奨するメッセージを表示します。	
{stop,start,list}コンテナを tpaexec --help 出力に追加しました。 tpaexec で Docker コンテナを管理するための既に実装されたオプションは、ドキュメントでは紹介されていますが、`tpaexec --help`の出力ではありません。この変更により、`tpaexec --help`コマンドにこれらのコマンドの動作の簡単な概要が含まれるようになりました。	

2.1.4 バグ修正

Description	Addresses
BDR 4 から BDR 5 へのアップグレード中に、 tpaexec reconfigure が postgres_distributed を追加しない問題を修正しました。 `tpaexec reconfigure`コマンドは、 BDR 4 から BDR 5 にアップグレードする際に、 `postgres_distributed`リポジトリを追加しませんでした。これにより、新しい BDR 5 パッケージをダウンロードしようとすると失敗します。この修正により、`postgres_distributed`リポジトリが構成に含まれるようになります。	
スタンバイノードが BDR プライマリ候補に昇格できる問題を修正しました。展開中にスタンバイノードが BDR プライマリ候補として選択される場合がある問題が見つかり、一部のクラスターシナリオたとえば、スタンバイノードを介して BDR ノードグループに参加する場合など、意図しない結果を引き起こす可能性があります。この修正により、スタンバイノードが潜在的な BDR プライマリ候補のリストから除外されます。	
EDBPGE および EPAS でパトローニをフィーチャーすると SLES 15 の展開が失敗する問題を修正しました。 EDBPGE / EPAS PostgreSQL フレーバーでフェールオーバーマネージャーとして patroni を選択した場合、etcd 関連パッケージのインストールに必要なモジュールが欠落しており、展開が正しく機能するようにいくつかの追加の調整が必要でした。この修正により、パトロニをフェールオーバーマネージャーとして SLES 15 に展開するときに、必要なパッケージがインストールされるようになり、いくつかの軽微な調整が行われます。	
PEM エージェント<9.6 を要求すると展開が失敗する問題を修正しました。 TPA は`pemworker --enable-probe`オプションを使用して、必要に応じて EFM プローブを有効にします。ただし、このオプションは PEM 9.6 より前には存在しないため、呼び出そうとするとエラーが発生します。この修正では、このタスクにバージョンチェックを追加します。エージェントバージョンが 9.6 未満の場合、スキップされることを意味します。	
pgd-cli がアップグレードされていない場合に pgd-cli タスクが実行される問題を修正しました。 postgres のみのアップグレードデフォルト中に、pgd-cli 診断出力をログに記録し、書き込みリーダー選択を待機するタスクは、pgd-cli がその実行の一部としてアップグレードされていない場合でも、常に最初の BDR プライマリで実行されました。これにより、古い pgd-cli バイナリがそのサブコマンドをサポートしていないため、「unknown コマンド show-groups」でアップグレードが失敗しました。 pgdcli またはすべてのコンポーネントがアップグレードに含まれない限り、両方のタスクがスキップされるようになりました。	
デフォルトの AWS AMI を現在利用可能なイメージに更新しました。公式にサポートされているプラットフォーム用に`tpaexec configure`が選択するデフォルトの AWS AMI が、現在利用可能なイメージに更新されました。以前は、いくつかの AMI 特に Ubuntu 22.04、Ubuntu 24.04、RHEL 8.10、および SLES 15 SP7 がベンダーによって登録解除されており、これらのプラットフォームの新しいクラスターで`tpaexec provision`が失敗する原因となっていました。 9.5 は現在のマイナーでなくなり、更新された 9.5 AMI は公開されないため、RHEL 9 および Rocky 9 もマイナーバージョン 9.5 から 9.7 に移行します。	

次のページに続く

表 4 – 前のページからの続き

Description	Addresses
TPA は、インストールされた BDR/PGD パッケージから bdr_version_num を自動的に取得するようになりました。 TPA は bdr_version_num を使用して、 CAMO 構成に関する正確な決定を下します。以前は、この値は実行中のデータベースを照会することによってのみ取得されていたため、初期展開 Postgres がまだ実行されていない場合、 CAMO 構成の一部を正しくレンダリングできず、ユーザーは回避策として config.yml で bdr_version_num を手動で設定する必要がありました。 TPA は、各ノードにインストールされている BDR または PGD パッケージのバージョンから bdr_version_num を取得するようになったため、手動設定は必要ありません。 config.yml で設定された値は引き続き優先され、データベースが実行されると、 bdr.bdr_version_num() によって報告される正確な値が権限のあるままです。	
PGD 6 マイナーアップグレードのアップグレード後のヘルスチェックを安定させました。 PGD 6 マイナーアップグレードの最後のローリングリスタートの後、BDR レプリケーションスロットは、更新されたばかりのノードと再ハンドシェイクしている間、一時的に非アクティブなままになる場合があります。アップグレード後の `pgd cluster show --health`チェックは、その期間に起動され、偽の「レプリケーションスロットクリティカル」障害を報告する可能性があります。 TPA は、厳密なヘルスチェックを実行する前に、すべてのスロットがアクティブになるのを待機する最大 2 分間 `bdr.group_replslots_details`をポーリングするようになりました。ウィンドウが使い果たされた場合でも、ヘルスチェックは実行されるため、本物の問題は引き続き報告されます。	
カスタム rc-local サービスの作成に関する問題を修正しました。 TPA はカスタム systemd サービスファイルを作成して、デフォルトでサポートしていないディストリビューションで起動時に rc-local スクリプトが実行されるようにします。この修正では、サービスファイルの場所を変更して規約に従い、以前に選択したパスで発生する可能性のある親フォルダーの欠落による障害を回避します。この修正により、このサービスファイル作成タスクは実際に必要な場合にのみ適用されます。	
TPA は、アーキテクチャに基づいて Postgres 拡張を `pgextended`および `edbpge`に正規化するようになりました。アーキテクチャに基づいて Postgres 拡張フレーバーを正規化するための修正が導入されました。 *tpaexec configure* --pgextended、--edbpge、--edb-postgres-extended、または --postgres-flavour pgextended/edbpge を介して Postgres 拡張バリアントを選択すると、BDR-Always-ON または古いアーキテクチャが `pgextended`に正規化されますが、新しいアーキテクチャは `edbpge`に正規化されます。互換性のないアーキテクチャに `pgextended`フレーバークラスターを展開しようとするとエラーが発生し、構成ミスを防ぎます。	
read_listen_port が 5.5 より前の BDR バージョンの構成に追加された問題を修正しました。 tpaexec manage は、古い BDR バージョンの場合でも read_listen_port を default_pgd_proxy_options に追加しました。 PGD 5.5 以降にアップグレードするときに欠落している read_listen_port をキャッチするために、展開時アサーションも追加されました。	

次のページに続く

表 4 – 前のページからの続き

Description	Addresses
TPA は、構成時に無効なクラスター名を拒否するようになりました。以前は、`tpaexec configure`は、ピリオドなどの文字を含むクラスター名を受け入れましたたとえば、クラスターディレクトリ名に`v1.1.0-rc.1`のようなバージョン文字列が埋め込まれている場合。構成は正常に書き込まれましたが、クラスター名が必要なパターン`^[_a-zA-Z0-9-]+$`と一致しなかったため、後の`tpaexec provision`は致命的なアサーションで失敗しました。`tpaexec configure`は、クラスターディレクトリが作成される前に同じチェックを前もって適用し、無効な名前を拒否するようになりました。そのため、問題は数ステップ後ではなく、すぐに報告されます。	
Playbook をアップグレードすると、pgrep の代わりに systemctl が使用されるようになりました。メジャーおよびマイナー PGD アップグレードプレイブックは`pgrep`を使用して、ノードで harp-proxy または harp-manager がまだ実行されているかどうかを判断します。 procps がインストールされていないホストでは、これらのチェックのうち 2 つは、「バイナリがありません」を「プロセスが実行されていません」と静かに誤読しました。 BDR 4→5 アップグレードは、harp-manager を停止する前に`harpctl unmanage cluster`をスキップし、PGD 5 マイナーアップグレードは間違ったプロキシを実行しました健康診断。チェックでは、`systemctl is-active --quiet`を使用するようになりました。これは、すべてのディストリビューション TPA で動作し、権限のあるサービス状態をサポートおよび反映します。	
TPA は、EPAS クラスターでの CIS コンプライアンスから pgaudit を省略するようになりました。以前は、`tpaexec configure --compliance cis`は無条件に`pgaudit`を`extra_postgres_extensions`に追加しました。これにより、監査ログが組み込まれている EPAS での起動障害が発生したため、EPAS クラスターでは`pgaudit`が省略されます。	
pgbouncer と pgd-proxy がリッスンポートを共有する場合、TPA は展開に失敗するようになりました。インスタンスのロールリストに`pgbouncer`と`pgd-proxy`の両方があり、両方のサービスが同じポートデフォルトの 6432 でリッスンするように構成されている場合、一方のみがバインドできます。通常、pgBouncer がレースに勝ち、pgd-proxy は障害が発生した状態のままになりましたが、TPA の展開は成功を返しました。構成ミスは、最初のアップグレード中に、pgd-proxy の代わりに pgBouncer に接続する TLS 保護テストからの`FATAL: SSL required`エラーとしてのみ可視化されました。 TPA は、影響を受けるホストと関連する 2 つの変数`pgbouncer_port`および`pgd_proxy_options.listen_port`を名前付けた明確なメッセージで、展開時にこの構成を拒否するようになりました。 pgbouncer と pgd-proxy を共同ホストするユーザーは、2 つのリスナーが衝突しないように、いずれか 1 つを`config.yml`でデフォルト以外の値に設定する必要があります。	

次のページに続く

表 4 – 前のページからの続き

Description	Addresses
TPA は、4to5 アップグレード中に pgd-proxy ノードに pgd-cli をインストールするようになりました。 BDR-Always-ON から PGD-Always-ON へのアップグレードは、pgd-proxy ロールを以前の harp-プロキシノードに追加しますが、アップグレード中に pgd-cli はこれらのノードにインストールされません。アップグレード後診断は、「そのようなファイルまたはディレクトリはありません」で失敗しました。 pgdcli-upgrade 再生は pgd-プロキシホストもターゲットとするようになったため、診断の実行時に pgd-cli が配置されます。	
自動監視ノード追加が、偶数データノードを含む PGD-X クラスターで機能しない問題を修正しました。偶数の `--data-nodes-per-location`で PGD-X クラスターを構成する場合、*tpaexec configure* `は各場所に監視ノードを正しく追加するようになりました。これにより、ユーザーが`--add-witness-node-per-location`を明示的に渡すことなく Raft コンセンサスを確立できます。この動作は、PGD-Always-ON アーキテクチャおよび PGD-X の文書化された動作と一致するようになりました。 *tpaexec configure* `は、*2* より小さい`--data-nodes-per-location`値も拒否するようになりました。これにより、以前はデータノードのない無効なクラスターが生成されました。	59694
TPA には、PEM サーバーバージョン 10.4.0 以降の正しい mod_wsgi モジュールが含まれるようになりました。この変更により、以前のバージョンの既存の動作を保持しながら、正しい mod_wsgi モジュールが PEM サーバーバージョン 10.4.0 以降に含まれることが保証されます。	
TPA には、SLES 15 に EPAS を展開するための追加モジュールが含まれるようになりました。 SLES 15 クラスターを展開する場合、EPAS と PEM インストールの両方の初期システム登録の一部として PackageHub 登録が実行されるようになりました。さらに、libclang13 などの依存関係をインストールするために、EPAS 展開で 2 つの新しいモジュール sle-module-desktop-applications および sle-module-development-tools が有効になりました。	
TPA は、古いクラスターディレクトリに switch2cm.yml リンクを自動作成するようになりました。 `tpaexec switch2cm`は、クラスターディレクトリに `architectures/PGD-Always-ON/commands/switch2cm.yml`へのリンクがなかったため、 `switch2cm`コマンドより前の TPA リリースで構成されたクラスターで「the playbook:command/switch2cm.yml が見つかりませんでした」で失敗しました。 `tpaexec switch2cm`は、リンクが見つからない場合に `tpaexec relink`を自動的に呼び出すようになりました。これは、 `tpaexec upgrade`が独自のコマンドリンクに対して既に持っていた動作と一致します。	

次のページに続く

表 4 – 前のページからの続き

Description	Addresses
アップグレード中にノード 2+で HARP マネージャーのシンボリックリンクの作成が失敗する問題を修正しました。 edbpge へのローリングアップグレード中に、 HARP マネージャーがデータベースに接続するための /var/run/postgresql/.s.PGSQL.<port>にシンボリックリンクを作成するタスクは、最初の「変換を拒否」の後、すべてのノードで失敗しますファイルからシンボリックリンクへ」。宛先パスは、アップグレード前の postgres フレーバー移行またはクリーンでない停止から残された実際のソケットファイルを保持する可能性があります。影響を受けるすべてのアップグレード Playbook のシンボリックリンクタスクに force:true を追加すると、そのパスの以前の状態に関係なく、シンボリックリンクが正しく作成されます。	
アップグレード時の BDR レプリケーションスロット間の再試行ロジックを導入しました。 BDR クラスターで postgresql パッケージバージョンをアップグレードする場合、レプリケーションスロットの再接続に数秒かかる場合があり、リカバリプロセスに十分な時間が割り当てられていないため、アップグレードプロセスで終了エラーが発生する場合があります。この修正では、アップグレードプロセスのヘルスチェックに再試行ロジックを追加して、スロットが回復する時間を確保し、タイミング関連の障害を回避します。	
--location-names がアーキテクチャの要件と一致しない場合、TPA は明確なエラーを報告するようになりました。以前は、 *--location-names* **`に間違った数の名前たとえば、*2* つ必要な *`bronze`* レイアウトを使用する *`BDR-Always-ON`* クラスターの単一のロケーションを提供すると、*`tpaexec configure`* が未処理の *Python* トレースバックでアボートしていました。 *`tpaexec configure`* は、指定されたロケーション名と、選択されたアーキテクチャとレイアウトが実際に必要なロケーションの数と比較し、それらが異なる場合は明確なエラーをレポートするようになりました。この変更は、*`M1`*、*BDR-Always-ON*、および `PGD-Always-ON` アーキテクチャに適用されます。**	
PGD-X が--hostnames-from で多すぎるホスト名を要求する問題を修正しました。 --hostnames-from を使用して PGD-X クラスターを構成すると、指定されたファイルが 16 未満の名前を保持するときは常に、クラスターがほんの一握りのみを必要とする場合でも、「…と一致する N/16 の名前のみが見つかりました」で失敗していました。基になる PGD-X アーキテクチャは、ビルドするインスタンスの実数ロケーションごとのデータ+監視+バーマンノード、および要求に応じてオプションの監視専用ロケーションと pem サーバーを計算するようになったため、 --hostnames-from は正しく受け入れます。ファイルのサイズが大きくなり、16 エントリにパディングする必要はなくなりました。	
CAMO 構成が誤って config.yml に追加される問題を修正しました。 --enable-camo で CAMO を明示的に有効にせずに--data-nodes-per-location 2 で PGD-X クラスターを構成すると、生成された config.yml には、CAMO コミットスコープとパートナーの割り当てが誤って含まれます。これにより、予期しない CAMO 構成が要求しなかったクラスターに適用されます。 CAMO 構成は、 --enable-camo が明示的に tpaexecconfigure に渡された場合にのみ生成されるようになりました。	59694

次のページに続く

表 4 – 前のページからの続き

Description	Addresses
PGD 5/6 マイナーおよび 5-6 メジャーアップグレードが時折ノードをフェンスされたままにする問題を修正しました。ループの以前にフェンスされたノードがアンフェンスされないため、ノードごとのアップグレード中に、「書き込みリーダー選択の完了を待機する」タスクがタイムアウトになる場合があります。 unfence ステップは、常にノードの更新された状態を参照するとはせず、誤ってスキップされ、ノードがフェンスされたままになります。 TPA は、ノードが Raft コンセンサスに再参加するのを待機し、アンフェンシングする前にその状態を更新するため、ノードは確実にアンフェンスされ、アップグレードを続行できます。	
tpaexec configure `の--primary-location`のエラー出力を改善しました。プライマリロケーションがロケーションのリストで見つからない場合、*--primary-location* `および--location-names`引数間の引数解析ロジックのエラーにより、ビルトイン Python エラーがスローされました。これは、TPA ではないため、ユーザーの誤解を招く可能性があります関連のエラー。この修正では、提供されたプライマリ場所と既知の場所のリスト間の不一致を適切に処理し、混乱を招くエラーメッセージを回避します。	
--overrides-from が指定されたときに tpaexecconfigure がクラッシュする問題を修正しました。 `--overrides-from <file>`とともに `tpaexec configure`を実行すると、「tpaexec configure の実行中にエラーが発生しました。名前'reduce' は定義されていません」で中止され、クラスターディレクトリは生成されません。 `tpaexec configure --overrides-from`は正常に完了し、指定された YAML ファイルの値は、文書化されているように、生成されたクラスター構成にマージされます。	
`tpaexec switch2cm`は、 ansible_user が root でない場合に動作するようになりました。 PGD-Always-ON の 3 つのプレイ switch2cm コマンドはプレイレベルになることを宣言していないため、コマンドは ansible_user が root の場合にのみ機能します。非ルートの ansible_user では、/pgdata/data/conf.d で「Permission denied」で失敗しました。これらのプレイは明示的にルートになり、PGD-Always-ON の他の場所で使用されるパターンと一致します。タスクごとは、postgres ユーザーとして実行される SQL クエリのオーバーライドになります。影響を受けません。	
TPA は、PostgreSQL 18+ PGD クラスターの max_active_replication_origins を構成するようになりました。 TPA は、「3 *number_of_nodes + 3」を使用して、PostgreSQL 18 以降で PGD クラスターの max_active_replication_origins パラメーターを設定するようになりました。これは、EDB PGD ドキュメントで推奨されている最小「ピアノードごとに 3 つのオリジン」を超える安全マージン式です。これにより、マルチノードクラスターに対してデフォルト値が低すぎることによって発生する PGD ノード参加の失敗が防止されます。	

次のページに続く

表 4 – 前のページからの続き

Description	Addresses
TPA は、バーマン構成のバックアップサーバーごとに path_prefix を設定するようになりました。 Barman の path_prefix は、以前は barman ホスト自分自身の postgres バイナリディレクトリを使用して、barman.conf のグローバル値としてのみ設定されていました。これにより、混合バージョンのシナリオローリングアップグレード中、またはバーマンホストがバックアップするノードとは異なる PostgreSQL メジャーバージョンを実行している場合に、不正な動作が発生しました。 path_prefix も、バックアップサーバーごとに/etc/barman.d/<backup>.conf に書き込まれ、バックアップされたノード自分自身の postgres_bin_dir から解決されます。これにより、barman は、バックアップする各ノードに正しいクライアントバイナリを使用することが保証されます。 barman.conf のグローバル path_prefix は変更されず、同じバージョンのシナリオのデフォルトとして機能し続けます。バックアップノードの PostgreSQL バージョンが barman ホストのバージョンと異なる場合、一致するクライアントパッケージが barman ホストに自動的にインストールされるように、サーバーごとの path_prefix が実際のバイナリに解決されます。 barman ホストには、追加の PostgreSQL バージョンのリポジトリアクセスが必要です通常は EDB エンタープライズリポジトリの場合。インスタンスごとの明示的なオーバーライドは、config.yml のノードの vars で barman_path_prefix を設定することにより使用できます。	

2.1.5 非推奨

Description	Addresses
PGD-X の非推奨の `--cohost-proxies`および `--add-proxy-nodes-per-location`構成フラグを削除しました。 PGD-X の--cohost-proxies および--add-proxy-nodes-per-location フラグは削除され、PGD-X アーキテクチャの `tpaexec configure`の有効なオプションではなくなります。 PGD 6 では、すべてのプロキシがコホストされるため、両方のオプションが時代遅れになります。	

2.2 Trusted Postgres Architect 23.43.0 release notes

リリース日 2026 年 3 月 18 日

Trusted Postgres Architect 23.43.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.2.1 ハイライト

- SLES 15 SP7 のサポート
- SSH 構成オプションの改善

2.2.2 機能強化

Description	Addresses
すべてのプラットフォームで SLES 15 SP7 のサポートを追加しました。 TPA は SLES 15 SP7 をサポートするようになりました、SP6 からのアップグレード。このサポートは、ベアメタル、AWS、Docker を含むすべてのプラットフォームに拡張されます。 TPA は SLES 15 SP7 で展開ノードとして使用できるようになり、この Service Pack を実行しているターゲットノードをプロビジョニングおよび管理することもできます。更新には、新しい AMI イメージ、Docker ベースイメージ、および検証済みパッケージビルドが含まれています。	
SSH 構成を改善し、ジャンプホストをサポート。 TPA は、生成された `ssh_config`に、ansible に必要なすべてのオプションを完全に設定するようになりました。 Ansible は、コマンドラインにオプションを追加する代わりに、構成を使用します。構成にジャンプホストを含めることができるようになりました。したがって、展開時に自動的に使用されます。	
enabled_keys でのクラスター SSH キーのインストールをスキップするタスクセレクターを追加しました。新しいタスクセレクター `ssh-cluster-key-config`を追加しました。これにより、展開中に'Ensure individually_keys allow admin access' タスクをスキップできます。これは、 `authorized_keys`ファイルが外部で管理されている環境または読み取り専用である環境 Oracle Cloud Infrastructure などに役立ちます。	

2.2.3 の変更

Description	Addresses
barman-wal-restore `で--keep-compression`をサポートしました。 Barman 3.12+を使用したインストールの場合、 Barman *restore_command* `で非推奨の-z`フラグを `--keep-compression`に置き換えました。 TPA は、古いバージョンの非推奨フラグへの条件付きフォールバックを実装し、下位互換性を維持しながら、検出または指定された Barman バージョンに基づいて最新の CLI 標準を採用します。	
PEM 10.4 の `pemworker`引数の順序を標準化します。 `pemworker`コマンドラインの使用法をリファクタリングして、サブコマンドたとえば `--enable-probe`を最初の引数として配置します。この更新は、任意の引数の配置をサポートしなくなった PEM 10.4 CLI の変更に対処し、TPA が PEM エージェントを正常に構成および管理し続けます。	

次のページに続く

表 7 – 前のページからの続き

Description	Addresses
Docker プラットフォームの最近の Docker バージョンとの互換性を確認します。Docker の最近のバージョンは、コンテナを検査するときにトップレベルの `NetworkSettings.IPAddress` フィールドを設定しなくなりました。IP は `NetworkSettings.Networks[<name>].IPAddress` 内でのみ使用できるようになりました。これにより、TPA がコンテナの IP を決定しようとしたときに、コンテナのプロビジョニングが失敗しました。 TPA は、古い Docker バージョンとの互換性についてトップレベルフィールドをチェックし、構成済みの `docker_networks` エントリーが定義されている場合はフォールバックし、それ以外の場合は、最初に利用可能なネットワーク IP を使用するようになりました。	
EFM SLES 15 展開で使用される OpenJDK バージョンの更新。 SLES 15 展開の EFM で使用される OpenJDK バージョンは、新しいバージョン バージョン 11 から 21 に更新されました。背後にある理由は、Docker イメージで使用される SLES 15 リポジトリでの OpenJDK バージョンの可用性を保証することです。	
RHEL、Debian、Ubuntu で EFM 依存関係の OpenJDK を更新しました。この変更により、EFM は、サポートされているディストリビューションで OpenJDK の最新バージョンを使用します。-RHEL 8 および 9、バージョン 11 から 21.-Debian 11、バージョン 11 から 17.-Ubuntu 22.04 および 24.04、バージョン 11 から 21 。	
TPA は、rc-local のないシステムで rc-local サービスを作成するようになりました。デフォルトで rc-local を持たないディストリビューションをサポートするために変更が導入されました。この変更には、カスタム TPA rc-local サービスの作成と、システムのブート時にそれが有効になっていることの確認が含まれています。これにより、TPA は起動時に `/etc/tpa/rc.local` を実行でき、異なるディストリビューション間で TPA 起動タスクを実行する一貫した方法を提供します。	
TPA は、PGD 4 以降での pgextended フレーバーの使用をブロックするようになりました。 *bdr_version* 4 以降で `postgres_flavour` が `pgextended` に設定されている場合、展開がクリアエラーで失敗するようになりました。 `pgextended` フレーバーは、 BDR 3 でのみサポートされています。 PGD 4 以降では、代わりに `edbpge` を使用します。	51418

2.2.4 バグ修正

Description	Addresses
TPA は、アップグレード後に EFM のヘルスチェックを再試行するようになりました。ビジーなシステムでは、クラスターの復旧に少し時間がかかり、1 つのヘルスチェックだけでは不十分になる場合があります。この変更では、アップグレード後に efm のヘルスチェックの再試行メカニズムを導入して、アップグレードが失敗したと判断するまでにクラスターに多くの時間を与えます。	

次のページに続く

表 8 – 前のページからの続き

Description	Addresses
Debian / Ubuntu での PEM バージョンの固定の問題を修正しました。 PEM バージョンを固定するときに発生した Debian / Ubuntu システムでの依存関係の競合を解決しました。 TPA は、`edb-pem-cli`と`edb-pem-agent`の両方の依存関係を同じバージョンに正しく固定するようになり、バージョン要件の不一致のためにパッケージマネージャーがインストールを拒否するのを防ぎます。	
スタンバイラグのあるビジーシステムでの EFM スイッチオーバーの失敗を修正しました。スタンバイサーバーがプライマリサーバーから遅延している場合、ビジーシステムのアップグレードまたはメンテナンス操作中に EFM スイッチオーバー操作が失敗する可能性がある問題を修正しました。以前は、スタンバイノードがプライマリと完全に同期していない場合、TPA のスイッチオーバー前ヘルスチェックがすぐに失敗し、ノードが数秒以内に追いつく場合でも、アップグレード手順が中断されていました。 TPA は、クラスターステータスを監視し、スイッチオーバーを続行する前にすべてのノードが同期するのを待機する EFM ヘルスチェックに待機および再試行メカニズムを実装するようになりました。これにより、クラスターがプロモーションにとって安全な状態にある場合にのみスイッチオーバー操作が開始されます。これにより、アクティブなワークロードを持つ実稼働システムでのアップグレードワークフロー中に一時的なレプリケーションラグが存在する分散システムでのスプリアス障害が解決されます。	
AWS プロビジョニングは、指定されたセキュリティグループが存在することを検証するようになりました。 `config.yml`で構成された`ec2_groups`を使用して AWS クラスターをプロビジョニングする場合、TPA は、指定されたセキュリティグループが実際に存在することを明示的に検証するようになりました。以前は、存在しないセキュリティグループが指定された場合たとえば、グループ名のタイプミスのため、プロビジョニングはサイレントに続行され、代わりに VPC のデフォルトのセキュリティグループを使用して EC2 インスタンスが作成されました。これにより、インスタンスに、ユーザーが気づかないうちに、誤ったまたは予期しないネットワークアクセスルールが設定される可能性があります。プロビジョニングはすぐに失敗し、問題があるリージョン、使用されたフィルタ、および問題を修正する方法に関するガイダンスを示す明確なエラーメッセージが表示されます。これにより、セキュリティの誤った構成が防止され、トラブルシューティングが簡単になります。さらに、TPA は、インターネットゲートウェイ、ルートテーブル、Elastic IP アソシエーションなどのオプションの AWS リソースを作成するときに、アクション可能なエラーメッセージを含む明示的な検証を提供し、プロビジョニング中のエラーの可視性を向上させます。	

2.3 Trusted Postgres Architect 23.42.0 release notes

リリース日 2026 年 2 月 25 日

Trusted Postgres Architect 23.42.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.3.1 ハイライト

- RHEL 10 の実験的サポート EDB パッケージが利用可能になるまで、オープンソースクラスターのみ
- Postgres ログ設定の制御の向上
- より幅広いユーザー環境に対応するための新しい `privilege_escalation_command` オプション

2.3.2 機能強化

Description	Addresses
ログの宛先とログディレクトリの処理を改善および文書化しました。デフォルトでは、TPA は`syslog`にログを記録するように Postgres を構成します。`rsyslog`のログの場所は、`postgres_log_file`で設定できます。ユーザーが`syslog`以外の宛先を選択した場合、TPA は Postgres ロギングコレクターが`postgres_log_file`にログを記録するように構成します。以前は、この動作はユーザーが`stderr`を選択した場合にのみ実装されていました。`jsonlog`や`csvlog`など、syslog 以外の設定でも動作するようになりました。この変更により、ディレクトリ権限の設定に使用できる新しいクラスター変数`postgres_log_directory_mode`が追加されます。また、すべてのロギング関連のクラスター変数に関する新しいドキュメントを追加します。 Logrotate は、ログが収集される方法に関係なく、ログファイルをターゲットとするように構成されます。	50711
RHEL 10 での TPA の実行をサポート。TPA コントローラは、RHEL 10 または同等の OS Rocky Linux、AlmaLinux、または Oracle Linux 10 を実行しているシステムになりました。これらのシステムでは、システム Python パッケージ Python 3.12 を使用できます。	
RHEL 10 ノードへの展開の実験的なサポート。TPA は、RHEL 10 または RHEL 10 ベースのディストリビューション Rocky Linux、AlmaLinux、または Oracle Linux を実行しているノードに展開できるようになりました。 docker または aws プラットフォームを使用している場合、適切な docker イメージまたは ec2 AMI が選択されます。この機能は実験的と考えられています。展開は RHEL 10 のさまざまなパッケージの可用性に依存しているため、パッケージが上流リポジトリにアップロードされるまで、すべてのアーキテクチャまたはソフトウェアオプションが動作するとは限りません。	

次のページに続く

表 9 – 前のページからの続き

Description	Addresses
代替の特権エスカレーションコマンド用の `privilege_escalation_command`構成変数を追加しました。このリリースでは、新しい cluster_vars 変数 `privilege_escalation_command`が導入され、ユーザーは管理対象アプリケーションが実行時にクラスター操作に使用する特権エスカレーションコマンドを構成できます。デフォルト値は`sudo`で、これは、TPA が sudo パッケージが存在しない場合は自動的にインストールし、さまざまなコンポーネント Postgres ユーザー、EFM ユーザー、フェールオーバーマネージャーの sudoers ファイルを構成する現在の動作を維持します。ユーザーは `privilege_escalation_command`を設定することにより、代替の特権エスカレーションコマンドを指定できるようになりました。代替の特権エスカレーションコマンド`sudo`以外の場合、TPA は次のことを行います。 sudo パッケージのインストールをスキップする Postgres ユーザーの sudoers 構成をスキップする EFM ユーザーの sudoers 構成をスキップする すべてのフェールオーバーマネージャー操作 repmgr、 HARP、EFM に configured コマンドを使用する）この機能は、次のような環境に役立ちます。代替の特権エスカレーションコマンドを使用する sudo 構成の手動管理が必要 特権エスカレーションコマンドに関する厳密なポリシーがある 重要 代替の特権エスカレーションコマンドを使用する場合、ユーザーは以下を確認する必要があります。選択したコマンドがプリインストールされ、構成されているすべてのターゲットシステム このコマンドにより、postgres ユーザーはサービス管理のための systemctl コマンドを実行できます このコマンドにより、EFM ユーザーは `efm_db_functions`を実行できます EFM を使用する場合 Ansible の make メカニズムが適切に構成されている `ansible_become_method`経由 コンポーネント固有の変更 Postgres ユーザー sudoers 代替コマンドを使用して、TPA は sudoers の構成をスキップします e postgres システムユーザー。ユーザーはサービス管理コマンドの特権エスカレーションを構成する必要があります。 EFM sudoers 代替コマンドを使用する場合、TPA は EFM ユーザーの sudoers の構成をスキップします。ユーザーは特権エスカレーションを構成して、EFM ユーザーが postgres ユーザーとして `efm_db_functions`を実行できるようにする必要があります。フェールオーバーマネージャー repmgr と EFM は、ハードコーディング sudo の代わりに、サービス管理コマンドに設定済みの特権エスカレーションコマンドを使用するようになりました。 Barman SELinux Barman SELinux 構成タスクは、構成された特権エスカレーションコマンドを尊重して、明示的な sudo コマンドの代わりに Ansible の become メカニズムを使用するようになりました。詳細な使用例と構成ガイドラインについては、 ansible-and-sudo.md および configure-cluster.md のドキュメントを参照してください。	

次のページに続く

表 9 – 前のページからの続き

Description	Addresses
download-packages `コマンドに「*refresh repository metadata*」オプションが追加されました。新しいオプションが `*tpaexec download-packages* `コマンド--refresh-repository`、動作するには Docker とそれに対応する Docker イメージが必要です。この変更を行う前は、クラスターと OS に応じて、ローカルリポジトリを手動で再作成するために必要なコマンドをユーザーに依存していました。この修正では、パッケージが配置され、構成ファイルで上記パッケージの新しいバージョンが選択されると、`tpaexec upgrade`を使用してリポジトリをアップグレードする自動方法を提供します。また、さまざまなエアギャップ条件を考慮して実行する必要がある手順をカバーするドキュメント、およびエアギャップシナリオの DNF リポジトリに関連して発見されたマイナー修正も含まれています。	52313

2.3.3 の変更

Description	Addresses
複数の EFM バージョンの同時インストールを防止します。TPA は、複数の EFM バージョンのインストールを防止し、サービススタートアップの競合を解決します。バージョンチェックは`upgrade_in_progress`状態を正しく検出するようになり、`tpaexec upgrade --components=efm`は既存のバージョンを置き換えることができ、通常の展開操作中に誤ってマルチバージョンのインストールをブロックし続けます。	48997
TPA は、PGD 6 クラスターでサポートされていない Postgres 13 を拒否するようになりました。 Postgres 13 と PGD 6 を組み合わせた構成を明示的に拒否するように`tpaexec configure`を更新しました。 Postgres 13 は PGD 6 でサポートされていないため、このチェックは、ユーザーが後で失敗する互換性のないセットアップを続行することを防ぎます。導入フェーズ。	
デフォルト“enterprise”PGD-S アーキテクチャのリポジトリ。 PGD-S アーキテクチャを使用するように構成されたクラスターは、デフォルトで“enterprise”になります。 "postgres_distributed"ではないリポジトリリポジトリ。したがって、デフォルトのリポジトリが必要なサブスクリプションレベルと一致するようになりました。	
ローカルのみの展開の必須 EDB トークンチェックを削除しました。`--use-local-repo-only`オプションが有効になっている場合でも、`tpaexec deploy`が誤って EDB_SUBSCRIPTION_TOKEN を必要とする問題を解決しました。必要なパッケージがすべてローカルに利用できる場合、TPA はローカルリポジトリから展開するときにこの検証をバイパスするようになりました。最初にパッケージをダウンロードし、ローカルリポジトリにデータを入力するには、引き続きトークンが必要であることに注意してください。	

次のページに続く

表 10 – 前のページからの続き

Description	Addresses
未使用のモニタリングサーバーロールを削除しました。モニタリングサーバーロールは、以前にクラスターの集中監視サーバーをセットアップするために使用されていましたが、この機能は使用またはサポートされません。この変更により、コードベースからモニタリングサーバーロールへのすべての参照が削除されます。roles/init でのグループ作成からモニタリングサーバーロールの削除モニタリング_server 変数を設定するタスクの削除 テストフィクスチャからモニタリングサーバーの削除 このクリーンアップでは、デッドコードを削除し、TPA 展開でサポートされているロールに関する混乱を防ぎます。	
RH8 用の個別の要件ファイル.TPA は、RH8 で実行されているかどうかに応じて、Python パッケージのインストール時に個別の要件ファイルを使用するようになりました。さらに、クラスター内のインスタンスが RH8 を実行している場合、使用中の ansible バージョンが 2.16 以降であることを展開時にチェックします。これにより、TPA は他のディストリビューションで ansible-core の最新バージョンを使用しながら、RH8 への展開を引き続きサポートできます。	
パッチへの文書化された依存関係を削除しました。TPA はパッチユーティリティを必要としないため、インストール手順からそれに関する記載を削除し、それを呼び出した到達不能コードを削除しました。	

2.3.4 バグ修正

Description	Addresses
mTLS を使用するクラスターのアップグレードの失敗を修正しました。mTLS が有効になっているクラスターで `tpaexec upgrade`が失敗する問題を解決しました。以前は、アップグレードフェーズ中に必要な TLS 変数が欠落していたため、「certificate require」というメッセージが発生していました。 etcd のエラーと Patroni の未定義変数エラー。この修正により、アップグレード中にすべての mTLS 構成が正しく伝播され、安全な接続が維持されます。	
Barman の依存関係として'file' を削除しました。 Barman はパッケージの依存関係として「file」を必要とするため、パッケージマネージャーによってインストールされます。別の依存関係として追跡する必要はありません。	
PGD-X 構成で `bdr_node_groups`が重複する問題を修正しました。 PGD-X クラスターの `tpaexec configure`中に接続マネージャーポートオプションを指定すると、`config.yml`に `bdr_node_groups`エントリが重複してしまう問題を解決しました。構成ロジックはこれらのオプションを正しく処理し、クリーンで有効なクラスター定義を保証します。	

次のページに続く

表 11 – 前のページからの続き

Description	Addresses
PGDG リポジトリを正しく処理するように `tpaexec configure`コマンドの `--edb-repositories`を修正しました。このバージョンの TPA よりも前では、*tpaexec configure* で--edb-repositories`オプションを使用すると、`(apt,yum,suse)_repository_list`変数を定義するコードが誤ってスキップされていました。これにより、EDB と PGDG リポジトリの両方が定義された誤った構成が生成され、パッケージ解決の競合が発生しました。あらゆる状況で正しい構成が生成されるように、修正が導入されました。	
TPA は、`max_replication_slots`計算でパラレル適用を考慮するようになりました。 PGD 5 パラレル適用をサポートするように`max_replication_slots`の式を更新しました。 Parallel apply ではライタごとに追加のレプリケーションオリジンが必要であるため、TPA は `bdr.writers_per_subscription`設定を自動計算に組み込み、PGD 5 クラスターでのスロットの枯渇を防ぎます。重要 この変更は、TPA が更新された計算結果を適用するため、PGD バージョン 5 以降を使用する場合、次回の展開で postgres クラスターが再起動します。ダウンタイムを回避するには、 `cluster_vars`の `max_replication_slots`および `max_wal_senders`の現在値を確認および定義します。	51548, 57005
TPA は、ansible_user が root でない場合のテストの実行をサポートするようになりました。 *tpaexec tasks* SSL 証明書を読み取ることができるように、ルートまたは正しいユーザーとして実行する必要があるデータベースクエリーを実行します。これらのテストは、以前はユーザーを指定せずに実行されたため、ユーザーが root でない場合に失敗しました。これらは`become_user`で実行されるようになりました。属性を作成し、ノードタイプに応じて postgres または pgd_proxy ユーザーとして実行します。	96876
TPA では、`--hostnames-from`を使用して Docker インスタンスの IP アドレスを設定できるようになりました。TPA の以前のバージョンは、ホスト名ファイルで指定された場合でも、`ip_address`を自動的に生成しました。この修正は、TPA が構成時に IP アドレスがまだ指定されていない場合にのみ生成することを意味します。 TPA は、提供された IP アドレスを含むクラスターネットワークを自動的に選択しないことに注意してください。そのため、 `--network`オプションを使用して、一致する CIDR を指定する必要があります。	
アップグレードおよびテストフェーズ中の未定義の Patroni SSL ファクトに関する問題を修正しました。展開中に設定された SSL 関連のファクトは、プレイブックの実行にわたって持続されず、SSL が有効になっている Patroni クラスターに対して`upgrade`または`test`を実行すると、`'patroni_ssl_ca_file' is undefined`などのエラーが発生しました。これらのファクトは、アップグレードプレイブックとテストタスクの関連ポイントに含まれる専用の `patroni/facts`ロールによって設定され、どのフェーズが実行されているかに関係なく、常に利用できるようになります。	

次のページに続く

表 11 – 前のページからの続き

Description	Addresses
DCS 解析エラーを防ぐために Patroni `ignore_slots`データ構造を修正しました。以前は、`ignore_slots`パラメーターは、必要なリスト形式の代わりに辞書として生成されました。これにより、DCS 解析中に AttributeError が発生し、`ignore_slots`構成が有効になることを妨げるだけでなく、後続の動的および静的構成の更新の適用もブロックされました。この変更により、`ignore_slots`が辞書のリストとして正しくフォーマットされ、解析エラーが解決され、すべての構成変更が正常に適用されます。	
TPA は、インスタンスのロールリストに空のエントリが含まれていないことを検証するようになりました。`config.yml`でインスタンスのロールリストを手動で編集するときに空のエントリを追加するか、ダッシュを削除するのを忘れると、デプロイメントが失敗するというバグを修正します。インスタンスロールリストの空のエントリは無効であるため、展開プロセスの早い段階でキャッチする必要があります。これにより、展開中にできるだけ早く各ノードのロールリストの内容を検証するタスクが追加されます。	46041
tpaexec configure `の *PGD-X* の欠落していた--enable-camo`オプションを追加しました。 *tpaexec configure* `内の *PGD-X* アーキテクチャの--enable-camo`フラグを導入しました。この更新では、自動互換性チェックや展開の成功に必要な構成調整を含む、クラスターの Commit At Most OnceCAMO が有効になります。	
欠落している etcd ロールをキャッチするための検証を追加しました。`harp_consensus_protocol: etcd`を使用して HARP を使用して BDR 4 クラスターを展開する場合、etcd ロールが定義されたインスタンスがない場合、展開は遅れて失敗し、欠落している etcd エンドポイントに関する不明瞭なエラーが発生しました。このような問題のデバッグは費用がかかりますが、この検証は問題を早期にキャッチし、何がうまくいかなかったのかについてユーザーにいくつかの手がかりを提供するようになりました。	46041
TPA は現在、`PEM サーバーの資格情報の記録`をバイパスするようになりました。共有 PEM 展開中のタスク。以前は、共有 PEM サーバーを展開するときに、タスク「PEM サーバーの資格情報を記録する」'postgres_password' 属性が必要なため失敗します。この属性は、'postgres' ロールを持つノードが共有 PEM ノードとともに展開される場合にのみ定義されます。これにより、'postgres' ロールが存在しない共有 PEM を使用して独立した Barman ノードを展開するなど、特定のシナリオで障害が発生し、属性欠落エラーが発生しました。 'postgres_password' は代わりに後続のステップで'EDB_PEM_CREDENTIALS_FILE' から派生するため、共有 PEM が使用されている場合、修正はこのタスクをスキップします。	
BDR クラスターの `barman_role`特権付与を修正しました。 BDR クラスターでは、 Barman が必要な特権が、 Barman が実際に接続する `postgres`データベースではなく BDR データベースに誤って付与されました。これにより、「バックアップ機能へのアクセスがありません。FAILED」が発生しました。 TPA 23.40 にアップグレードした後のエラー。この修正を展開した後、付与は `postgres`データベースに正しく適用されます。影響を受ける既存のクラスターは、`tpaexec deploy`を再実行することにより修正できます。	54698

次のページに続く

表 11 – 前のページからの続き

Description	Addresses
`bdr_package_version`のみが設定されている場合に、pgd-proxy および pgdcli パッケージバージョンが`bdr_package_version`と一致するようにダウンローダーを修正しました。エアギャップ環境で'bdr_package_version' を選択すると、BDR の選択したバージョンがダウンロードされますが、PGD CLI および PGD プロキシのバージョンはダウンロードされません。パッケージの不一致があるため、'tpaexecdeploy' のエラー。これは、非エアギャップ環境の場合ではなく、'bdr_package_version' の特定のバージョンを選択すると、PGD CLI および PGD プロキシの対応するバージョンも選択されます。この修正により、'tpaexec download-packages' を実行するときに、最初に'pgd_proxy_package_version' または'pgdcli_package_version' が定義されているかどうかを確認し、定義されていない場合は、 BDR パッケージが不一致であるため、'bdr_package_version' 変数を再利用して指定された同じバージョンをダウンロードします運用環境で重大な問題を引き起こす可能性があります。	
tpaexec configure **`の** ***PGD-X*** **ポートフラグを標準化しました。** *`tpaexec configure* **`を更新して、** ***PGD-X*** **の有効なオプションとして**--proxy-listen-port`および`--proxy-read-only-port`を削除しました。ドキュメントが調整されて、これらの削除を反映し、サポートされている`--read-write-port`および`--read-only-port`フラグのカバレッジを追加して、CLI とマニュアル間の調整を保証します。	57408
TPA は、スロットのデフォルト計算で PEM サーバーインスタンスを無視するようになりました。PEM サーバーバックエンド専用のインスタンスを無視するように`num_postgres_instances`のロジックを変更しました。この変数は`max_replication_slots`のデフォルト値を決定するため、PEM ノードを除外することで過剰割り当てを防止します。重要 TPA は更新された計算結果を適用するため、`config.yml`で`max_replication_slots`が明示的に定義されていない場合、PEM サーバーを使用したクラスターへの展開は、次回の展開時に再起動がトリガーされます。ダウンタイムを回避するには、`cluster_vars`の現在の値を確認および定義します。	
外部変数ファイルで tpaexec apply を実行したときに発生したクラッシュを修正しました。外部変数ファイル `tpaexec deploy --extra-vars "@./vars.yml"`などで tpaexecdeploy を実行したときに発生したクラッシュを修正しました。以前は、deploy コマンドによってトリガーされる内部プロビジョニングステップがこれらの追加変数を継承せず、実行が失敗していました。この修正により、展開が開始される前に、指定されたすべての変数が tpaexec プロビジョニングに正しく渡されるようになります。	56864
コミットスコープが選択されたときに PGD-X の展開が失敗する問題を修正しました。デフォルトで PGD-X は、一連の不変のコミットスコープを作成します。この変更により、TPA は、構成ファイル CAMO などで以前に定義されていないコミットスコープを追加するときに、PGD-X のこれらのコミットスコープを削除しようとします。	

次のページに続く

表 11 – 前のページからの続き

Description	Addresses
Patroni クラスターでの RHEL 8 の展開の失敗を修正しました。Python 3.6 以降の互換性のために`patroni_compare_config.py`スクリプトを更新することにより、RHEL 8 での展開の失敗を解決しました。この修正により、スクリプトのコアロジックを維持しながら、Patroni 構成の検証が最新の Python 環境で正しく実行されることが保証されます。	
Patroni のアップグレード中に etcd_protocol の KeyError を修正しました。これは、最初の etcd ノードが pyformat_hostvars を使用してすべての etcd ノードのエンドポイントをビルドしようとしたときの競合状態です。各ホストの hostvars で etcd_protocol を探しました。ただし、現在のホストのみがインストール中にこのファクトを設定していましたアップグレード。これにより、他のホストにアクセスするときに KeyError が発生しました。	
'etcd_authentication_mode' の'basic' 認証に関する問題を修正しました。'etcd_authentication_mode' で'basic' 認証を選択すると、呼び出しを担当する etcd の'patroni' ロールに権限を付与するタスクで展開の実行が失敗します。etcd クラスターに。これは、ファクト変数がタスクのスコープで適切に定義されていないため、タスクの条件が失敗し、したがって権限が付与されないためでした。修正には、ファクト変数が正しく設定されてアクセスできることを確認し、'basic' 認証が選択されたときに権限付与タスクが意図したとおりに実行できるようにすることが含まれます。	
RHEL ベースのシステムでの PGD アップグレード中のパッケージの競合を修正しました。RHEL ベースのオペレーティングシステムで PGD 5 から PGD 6 にアップグレードするときに発生した依存関係解決 depsolve の問題を解決しました。アップグレードプロセスは条件付きで`allowerasing`オプションを利用するようになりました。これにより、アップグレードの前の予行チェック中に競合する PGD 5 パッケージの必要な置換が許可され、PGD 6 への正常な移行が保証されます。	
バージョン固定を使用する際の Barman 展開の失敗を修正しました。`barman_package_version`を介して Barman バージョンを固定するときに発生した Debian / Ubuntu システムでの依存関係の競合を解決しました。 TPA は、`barman-cli`とその`python3-barman`依存関係の両方を同じバージョンに正しく固定するようになり、パッケージマネージャーがバージョン要件の不一致を理由にインストールを拒否するのを防ぎます。	

2.4 Trusted Postgres Architect 23.41.0 release notes

リリース日 2025 年 11 月 26 日

Trusted Postgres Architect 23.41.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.4.1 ハイライト

- Postgres と PGD だけでなく、すべてのクラスターコンポーネントの更新を実行します
- デフォルトの Patroni 構成の強化、セキュリティ対策を微調整するための多くの新しいオプション
- PGD 6 PGD-X および PGD-S のマイナーバージョンアップグレードのサポート
- Postgres 18 のサポート

2.4.2 機能強化

Description	Addresses
新しい`tpaexec configure`クラスターのセキュア etcd および Patroni のデフォルト。 `tpaexec configure`を使用して新しいクラスター構成を作成する場合、生成された`config.yml`ファイルは、デフォルトで`etcd`と`Patroni`の両方の SSL/TLS 暗号化と相互 TLSmTLS 認証を自動的に有効にします。これにより、新しい展開がすぐに使える安全な通信ベースラインで開始されます。 次の変数は、新しい構成のデフォルトで設定されるようになりました。	
コンポーネント M1 および PGD/ BDR クラスターのアップグレードをサポート。 すべてのクラスターソフトウェアコンポーネントフェールオーバーマネージャー、PEM、Barman などを、*config.yml*`でコンポーネントパッケージバージョンを指定することにより特定のバージョンに更新できるようになりました。または利用可能な最新のものが提供されていない場合。アップグレードのコンポーネントのリストは、コンマ区切りリストとして`--components`フラグに渡すことができます。このフラグが渡されない場合、`postgres`は更新されます。 `--components=all`を指定すると、該当するすべてのソフトウェアコンポーネントが更新されます。	
PGD5.9+から PGD6 へのアップグレードの自動化が向上しました。 PGD v5.9 から v6 の移行の場合、PGD プロキシの代わりに接続マネージャーが要件です。この移行を促進するために、新しいコマンド *switch2cm* `が導入されました。新しいコマンドでは、 `*tpaexec reconfigure <cluster> --enable-connection-manager*`を実行してクラスターの `*config.yml*`を準備する必要があります。 *:ref:`PGD-Always-ON から PGD-X へのアップグレード<PGD-Always-ON から PGD-X へのアップグレード>*	
`etcd`クラスターとの通信が、SSL/TLS を使用して保護できるようになりました。 新しい構成パラメーター`etcd_ssl_enabled`が導入され、すべての`etcd`通信の TLS 暗号化が有効になりました。これにより、クラスターの分散コントロールプレーンのセキュリティが大幅に強化されます。 *etcd_ssl_enabled* `が `*true*`に設定されている場合 • 展開プロセスは、各 `*etcd*`ノードに必要な *TLS* 証明書を自動的に生成します。これらの証明書には、堅牢な検証のための *DNS* と *IP* サブジェクト代替名 *SAN* の両方が含まれるようになりました。 • ピアツーピア接続とクライアントサーバーの両方の `*etcd*`接続は、*HTTPS* を使用するように構成されています。 • `*etcdctl*`を使用した内部ヘルスチェックなどのダウンストリームコンポーネントは、 *HTTPS* で安全に `*etcd*`に接続するように自動的に構成されます。既存の展開との下位互換性を維持するために、この機能はデフォルトで無効になっています `*etcd_ssl_enabled: false*	

次のページに続く

表 12 – 前のページからの続き

Description	Addresses
Patroni REST API の相互 TLSmTLS 認証。 相互 TLS のサポートが追加され、Patroni REST API のより安全な証明書ベースの認証方法を提供します。新しい構成変数 `patroni_authentication_mode`が利用できるようになりました。これを `mtls`に設定して、この機能を有効にできます。有効にすると `patroni_ssl_enabled: true`が必要 、展開自動化は次のことを行います。 • クライアント証明書を要求および検証するように Patroni REST API サーバーを構成します。 • `patronictl`および HAProxy ヘルスチェックを含むクライアントを、安全に接続するために必要なクライアント証明書とキーを使用して自動的に構成します。デフォルトの認証モードは `basic`ユーザー名とパスワードのままで、既存の展開との下位互換性を保証します。	
`allowlist`へのアクセスによる Patroni REST API のセキュリティの向上。 Patroni REST API のセキュリティは、安全でない API エンドポイント `POST`、*PUT*、*PATCH*、`DELETE`を受け入れるための `allowlist`を構成することにより強化されました。この対策により、無許可のノードがデータベースノードの再起動やクラスター構成の変更などの管理アクションを実行できなくなります。 `allowlist`には、Ansible インベントリーに基づいて Patroni クラスターメンバーのホスト名が動的に入力され、許可されたノードのみがこれらの機密 API エンドポイントへの要求を実行できます。	
TPA は、PGD-X のマイナーバージョンのアップグレードをサポートするようになりました。 `tpaexec upgrade`コマンドは、PGD-X アーキテクチャを実行しているクラスターで postgres および PGD のマイナーバージョンのアップグレードを実行できるようになりました。アップグレードプロセスは、クラスターの状態を確認し、ノードを一度に 1 つずつアップグレードします。このシナリオでは、 `update_hosts`変数をサポートし、ドキュメントに記載されているベストプラクティスの推奨事項に従って使用する場合、クラスターのサブセットをアップグレードします。	
TPA は、PGD-S のマイナーバージョンのアップグレードをサポートするようになりました。 `tpaexec upgrade`コマンドは、PGD-S アーキテクチャを実行しているクラスターで postgres および PGD のマイナーバージョンのアップグレードを実行できるようになりました。アップグレードプロセスは、クラスターの状態を確認し、ノードを一度に 1 つずつアップグレードします。このシナリオでは、 `update_hosts`変数をサポートし、ドキュメントに記載されているベストプラクティスの推奨事項に従って使用する場合、クラスターのサブセットをアップグレードします。	

次のページに続く

表 12 – 前のページからの続き

Description	Addresses
etcd 基本認証とロールベースのアクセス制御を有効にします。 このリリースでは、クライアント認証を使用して etcd 分散キーバリューストアを保護する新しい機能を導入します。基本認証を有効にし、きめ細かい最小の権限で Patroni の専用ユーザーを構成できるようになりました。機能を制御するために、新しい構成変数`etcd_authentication_mode`が追加されました。 *basic* **`に設定すると、展開自動化は次のことを行います。• 管理者** *`root`* **ユーザーを設定します。• 新しい** *`patroni_etcd_user`* **構成オプションで指定された** *Patroni* **の専用** *etcd* **ユーザーを作成し、クラスターのキープレフィックス**/tpa/cluster_name`などに制限した読み取り書き込み権限を付与します。• これらの資格情報を自動的に使用して、 etcd に安全に接続および認証するように Patroni を構成します。下位互換性を維持するため、この機能はデフォルトで無効になっています *etcd_authentication_mode: none* 。	
`etcd`の相互 TLSmTLS 認証。 相互 TLS のサポートが追加されて、すべての `etcd`通信が保護されました。新しい `mtls`オプションが `etcd_authentication_mode`変数で使用できるようになりました。 *etcd_authentication_mode* **`が** *`mtls`* `etcd_ssl_enabled: true`が必要に設定されている場合、展開自動化は次の証明書ベースの認証を強制します。• すべての `etcd`ノード間のピアツーピア通信。• Patroni のようなクライアントからのクライアントサーバー通信。 TPA は、必要な証明書とキーを使用して `etcd`サーバーと Patroni クライアントの両方を自動的に構成し、完全に安全で検証された接続を保証します。	
PEM でサーバーとエージェントを表示する方法を制御するための新しいインスタンスレベル変数を導入しました。 *pem-agent* **`ロールで使用する** *4* **つの新しいインスタンス変数を導入しました。** *`pem-server`* **`ロールと既存の変数** *`pem-server-group`*。全体的に、これらの変数は、PEM でサーバーとエージェントがどのグループとクラスターに割り当てられているかを決定します。完全な説明については、ドキュメントを参照してください。	
OpenSSL で生成された証明書の有効期間のサポートを追加しました。 以前は、TPA で使用される一部の SSL 証明書 OpenVPN、PEM、および TPA 自分自身に、有効期限までの日数を指定する方法が付属していませんでした。この変更により、 *openssl_certificate_validity* **`デフォルト** *3650* **日という変数が追加され、** *SSL* **証明書の有効期限が切れるまでの日数を決定するために使用できます。** *TPA* **に新しい有効期限の新しい証明書を強制的にリロードするには、適切なオプションを** *`tpaexec deploy -e pem_web_server_renew_tls_certificates=true"`* **または**-e tpa_tls_renew_tls_certificates=true")`に渡す必要があります。	

2.4.3 の変更

Description	Addresses
デフォルトの EFM パスワードの暗号化を scram-sha-256 に変更しました。 TPA は、フェールオーバーマネージャーとして EFM で構成されたすべての新しいクラスターでデフォルトの `efm_user_password_encryption: scram-sha-256`になりました。この変更は、 *--enable-efm* `または--failover-manager efm`とともに `tpaexec configure`を実行する場合に適用されます。これは、PostgreSQL 18 での MD5 パスワード暗号化の非推奨に対処し、すべての PostgreSQL バージョンのセキュリティを向上させます。既存のクラスターは影響を受けません。ユーザーは、必要に応じて `config.yml`で `efm_user_password_encryption: md5`を明示的に設定することにより、このデフォルトをオーバーライドできます。ただし、MD5 サポートは将来のリリースで TPA から削除されます。	
`tpaexec configure`コマンドは、`efm_version`を `config.yml`に追加するようになりました。 `tpaexec configure`を実行し、フェールオーバーマネージャーとして EFM を選択するときに導入される場合、`efm_version`の明示的な追加のサポート。これは、`tpaexec configure my-cluster-dir -a M1 --enable-efm --efm-version 5.2 (u2026)`のように `tpaexec`を呼び出したときに見られます。さらに、`efm_version`選択をカバーするドキュメントが追加されました。	
TPA は、PGD プロキシバージョン検出のために `journalctl`の代わりに `pgd-proxy`を実行するようになりました。 `journalctl`が予想されるバージョンを返さないエッジケースがいくつかあります。したがって、TPA は代わりに `pgd-proxy`を直接実行してバージョン情報を取得するようになりました。	46753, 46426
事前展開フックをリポジトリーロールの外部に移動しました。 以前のバージョンでは、このフックはリポジトリーロールの一部でした。 TPA が展開からこのロールを除外する可能性を導入したとき、このフックが意図せずにスキップされる可能性がある状況が発生しました。この状況を回避するために、フックをロールの外部に移動して、常に実行されるようにしました。	52412
EFM 5.2 以降の `auto.basebackup`プロパティを追加しました。 EFM 5.2 以降、新しい自動リビルド機能に追加される新しいプロパティ `auto.basebackup`があります。ユーザーは既存の pg_rewind 機能 pg_basebackup を使用するか、両方を使用するように EFM に指示できます。 pg_rewind が失敗した場合に pg_basebackup が実行されます。	
AWS リージョンの内部辞書を更新しました。 この変更により、AWS リージョンの内部辞書が更新され、サポートされているリージョンと対応する AZ のカバレッジが拡大されます。	
TPA ドキュメントの `postgres_port`を文書化しました。 TPA ドキュメントにパラメーター `postgres_port`の詳細を追加しました。 `postgres_port`は、Postgres のリスニングポートのカスタマイズを許可する変数です。この変数により、情報に依存するクラスターのすべてのコンポーネントで設定値の一貫した使用が保証されます。	
PEM HTTPD サーバーで無効な TRACK および TRACE メソッド。 PEM 10.3 には、HTTPD 構成の修正が含まれており、 TRACK / TRACE 要求を書き換える指示子を<VirtualHost> に移動します。 `pem_server_ssl_port`変数で定義されたポート上のデフォルトの仮想ホストのセクション。この変更は、バージョンに関係なく、PEM のすべての新しい展開に同じ修正を適用します。	54529

次のページに続く

表 13 – 前のページからの続き

Description	Addresses
`efm-post-config`フックを追加しました。 新しい`efm-post-config`フックが導入され、efm 構成が行われた後に新しいタスクを実行できるようになりました。その使用方法を説明する新しいドキュメントも'TPA Hooks' セクションで開発されました。	
PG18 での pgaudit および postgis プラグインのサポートを追加しました。 これらのプラグインの PG18 サポートを追加します。これらのバージョンは、サポートされている Postgres バージョンにも依存します。バージョン 11 および 12 は、広くサポートされなくなったため、プラグインリストから削除しました。	
ネットワークが利用できるようになった後、PGD プロキシを開始します。 このサービスは、開始する前にネットワークサービスをアップして実行している必要があります。これを保証するために、`Wants`および`After`指示子が PGD プロキシサービスユニットに追加されました。	54251, 52731
RHEL の PEM サーバー v10+に`libpq5`をインストールします。 PEM 10.1 は、RHEL 9 サーバーでの`kerberos`認証に関連する`psycopg`エラーを修正する依存関係として`libpq5`を追加しました。`libpq5`が確実にインストールされるように、PEM サーバーバージョンが 10 以上の場合、TPA は RHEL サーバーに明示的にインストールするようになりました。	
必要に応じて`libcurl-full`をインストールします。 `libcurl-minimal`は特定の RHEL 風のイメージがインストールされ、`edb-pem-agent`のパッケージ依存関係は満たしますが、必要な API の完全なセットつまり SMTP を提供しません。インストールされている場合、削除しますそれを作成し、`libcurl-full`メタパッケージをインストールして、PEM エージェントに完全な`libcurl`機能が存在することを確認します。`edb-pem`パッケージは`edb-pem-agent`パッケージに依存しているため、依存関係解決の競合を回避するには、PEM サーバーから`libcurl-minimal`パッケージも削除する必要があります。これは、`allowerase`パラメーターを使用してパッケージをインストールすることにより実行されます。	

2.4.4 バグ修正

Description	Addresses
Patroni REST API SSL/TLS 接続が正しく構成され、安全になりました。 以前は、Patroni REST API `patroni_ssl_enabled: true`の SSL を有効にすると、自己署名証明書を使用していました。これにより、`patronictl`のようなクライアントで`CERTIFICATE_VERIFY_FAILED`エラーが発生し、安全な構成で機能を使用できなくなりました。これは、適切な証明書生成プロセスを実装することにより修正されました。 Patroni REST API サーバーは、TPA クラスター用に TPA によって作成された認証局 CA によって署名されたサーバー証明書で構成されました。さらに、`patroni.yml`構成の`ctl`セクションが修正され、サーバー検証に CA を使用し、認証 mTLS 用に独自のクライアント証明書とキーを提示するように`patronictl`が適切に構成されています。その結果、Patroni REST API の SSL を有効にすることにより、すべてのクライアントに完全に安全で検証可能な通信チャネルを提供するようになりました。	
**Barman の同時更新中の競合条件を修正しました``.pgpass` file.** Previously, when multiple backed-up nodes attempted to update the Barman node's *.pgpass* file concurrently, a race condition could occur, resulting in partial or incomplete entries. TPA now serializes this process, ensuring that the Barman node's `.pgpass`ファイルは、損失を防ぐためにシークエンシャルに更新されます。	52460
すべてのディストリビューションに適用される'common' キーをサポートするように`unwanted_packages`を修正します。 以前は、`common`キーを使用して`unwanted_packages`を設定する場合はサポートされていませんでした。これは、`config.yml`に設定できる他のパッケージ辞書の動作とは異なります。この修正では、`common`キーのサポートが導入され、このキーの下にリストされているパッケージが、クラスター内で使用中のディストリビューションに存在する場合、削除されます。	
pgdcli のタスク選択を修正しました。 暗黙的に除外するタスクセレクターが使用された場合でも、`pgdcli`ロールが`bdr`ノードで実行されるバグを修正しました。	
Docker コンテナのプロビジョニング中の致命的なエラーを修正しました。 ユーザー定義のネットワークを使用して Docker の最新バージョンでコンテナを作成する場合、`provision`は場合によってクラッシュし、'dict object' has no property 'IPAddress' エラーが発生しました。これは、コンテナの IP アドレスを検出する古い方法が原因でした。ロジックが更新されて、最新の Docker ネットワーク構造と互換性があり、コンテナの作成がより堅牢になりました。	
SLES への`etcd`パッケージのインストールを修正しました。 `etcd`を必要とする SLES ベースのディストリビューション Patroni を使用するディストリビューションなどへの展開は、標準リポジトリで`etcd`パッケージが見つからないため、以前は失敗していました。これは、 PGDG `extras`リポジトリを有効にすることにより解決されました。これは、必要なパッケージを提供し、インストールを成功させます。	
PGD クラスターでのテスト後に`tpaexec test`がデータを削除しない問題を修正しました。 クラスターを展開し、そのクラスターで`tpaexec test`を実行した後、`bdr.replication_set_table`に追加のエントリーを作成します。これは後で削除されません。修正により、テストは`tpatest`と呼ばれる一時的なスキーマで実行されます。これは、テストが完了すると削除されます。	53224

次のページに続く

表 14 – 前のページからの続き

Description	Addresses
アップグレード 3.7 から 5 を実行するときにプロキシノードで PGD CLI が見つからない問題を修正しました。 通常の展開、または' BDR-Always-ON' セットアップから'PGD-Always-ON' へのアップグレードを実行する場合、プロキシノード以前に'harp-proxy' ロールで展開されたものは、アップグレードを実行するために必要な'pgd-proxy' ロールでインストールしようとしても、PGD CLI には同梱されません。この修正により、PGD CLI はプロキシノードと'bdr' ロールを持つノード、および'pgdcli' ロールのみを備えたノードにインストールされます。	

2.5 Trusted Postgres Architect 23.40.1 release notes

リリース日 2025 年 10 月 28 日

これはバグ修正リリースです。 Trusted Postgres Architect 23.40.01 で解決された問題には次のものが含まれます。

2.5.1 バグ修正

Description	Addresses
PGD `alter_node_option`タスクが物理レプリカに対して実行される問題を修正しました。 物理レプリカである BDR ノードを含む`PGD-AlwaysON`クラスターでは、BDR ノードが次のことを想定しているため、`alter_node_option`タスク中にアップグレードプロセスが失敗します。 RAFT は有効ですが、物理レプリカ BDR ノードは有効になっていません。 TPA は、`replica`ロールを持つノードでこのタスクをスキップし、アップグレードを正常に完了できるようになりました。	51997
pg_catalog の pgbouncer_auth_function の展開後チェックを削除することにより、展開後の不要な出力を修正しました。 TPA は最近、pgbouncer の使用中にユーザーの認証に使用される pgbouncer_auth_function を再配置しました。これは、ベストプラクティスに従い、pg_catalog ファンクションがデータベースのダンプの一部でないアップグレード中にの問題を回避するために行われました。TPA は、pg_catalog で定義された auth ファンクションにまだ依存しているユーザーが問題を特定し、システムから機能を削除する前にそれを修正できることを確認するのに役立つように、展開後チェックを導入しました。このチェックは移行メカニズムとして使用され、現在削除されています。これにより、そのチェックによって生成された不要な出力に関する問題が解決されます。	100065
`backup_name`を使用する場合の `.pgpass`ファイルの構成の誤りを修正しました。 以前は、`backup_name`変数を使用する場合、TPA は Barman サーバーの`pgpass`でこの名前を誤って使用していました。これは修正され、ドキュメントが更新されました。	

次のページに続く

表 15 – 前のページからの続き

Description	Addresses
TPA は、ビーコンエージェントがクラスターに含まれる場合に EDB リポジトリが使用されるようになりました。 TPA はビーコンエージェントを展開して、システムの状態をモニタできます。ただし、有効な EDB リポジトリへのアクセスが必要です。この変更により、ビーコンエージェントが展開されるときに標準リポジトリが有効になります。	
Barman が log_file パラメーターを正しく構成できなかった問題を修正しました。 barman ノードをセットアップするときに、変数'barman_log_file' はログファイルを配置するための代替ディレクトリをセットアップしていませんでした。代わりに、デフォルトの場所`/var/log/barman/barman.log`を使用しました。 voxnyx https://github.com/voxnyx によって報告されたバグと修正。	

2.6 Trusted Postgres Architect 23.40.0 release notes

リリース日 2025 年 10 月 2 日

Trusted Postgres Architect 23.40.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.6.1 機能強化

Description	Addresses
EDB Postgres 分散 PGD v5 クラスターを v6 にアップグレードするためのサポートを追加しました。	
PGD-S クラスターの構成の拡張サポート。	
`pgd-proxy-config`フックが導入されました。	49911
EFM 5.1 以降に 3 つの`release.vip.*`プロパティを追加しました。	
TPA では、証明書に RSA キーサイズを設定できるようになりました。	
TPA が Postgres サーバー TLS 証明書とキーを生成する方法と、これをオーバーライドする方法を文書化しました。	52049

2.6.2 の変更

Description	Addresses
TPA は、フェールオーバーマネージャーとして repmgr を使用した EPAS クラスターをサポートするようになりました。	

2.6.3 バグ修正

Description	Addresses
PGD 6 クラスターでバックアップが構成されていない問題を修正しました。	
PGD-Always-ON での `update_host`変数のサポートを修正しました。	52558, 45181
HAProxy ヘルスチェックは、SSL 対応の Patroni クラスターで HTTPS を正しく使用するようになりました。	
edbpge Postgres で使用した場合の `pgaudit`拡張機能の問題を修正しました。	
グローバルプロキシルーティングが PGD 5.9 で正しく設定されなかった問題を修正しました。	
TPA は、PEM エージェント Barman ノードで`postgres/pgpass`タスクを正しくスキップするようになりました	

2.7 Trusted Postgres Architect 23.39.0 release notes

リリース日 2025 年 8 月 22 日

Trusted Postgres Architect 23.39.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.7.1 ハイライト

- s390x アーキテクチャのサポート
- `edb_stat_monitor` 拡張機能のサポート
- `auto.rewind` 、`jdbc.properties` 、`jdbc.loglevel` 、および `check.vip.timeout` の EFM プロパティのサポート

2.7.2 機能強化

Description	Addresses
Ubuntu 24.04 での PEM のサポートを追加しました。このディストリビューションには、展開エラーを発生させた特定のパッケージが定義されていませんでした。さらに、これにより、mod_wsgi または edb の mod_wsgi モジュールが適用される場所でデフォルトで有効になります。	
認識される拡張機能に `edb_stat_monitor`を追加しました。ユーザーが `postgres_extensions`または `postgres_databases`の下で名前付けられた拡張機能のリストのエントリーとして `edb_stat_monitor`を指定すると、TPA は正しいパッケージのインストール、拡張機能の作成、および `shared_preload_libraries`に含めることを処理します。	

次のページに続く

表 19 – 前のページからの続き

Description	Addresses
s390x 別名 IBM Z および LinuxONE のサポートを追加します。 TPA が s390x CPU アーキテクチャをサポートするようになりました。サポートされている s390x オペレーティングシステムは、RHEL 8、9、および SLES 15 です。TPA はこれらのシステムで実行でき、クラスター展開のターゲットホストとして使用できます。 s390x バイナリはすべての PyPI パッケージで利用できるわけではないため、セットアップ中に PyPI からのインストールに依存するのではなく、EDB から `tpaexec-deps`パッケージをインストールすることを強くお勧めします。同様に、s390x の PGDG RPM リポジトリがないため、TPA の EDB Repos にアクセスして、パッケージから Postgres およびその他のクラスターコンポーネントをインストールする必要があります。	
EFM 5.1 以降に `auto.rewind`プロパティを追加しました。 EFM 5.1 以降、障害が発生したプライマリ DB サーバーを再構築する新機能の一部である新しいプロパティ `auto.rewind`があります。	
EFM `jdbc.properties`プロパティのサポートを追加します。これにより、EFM 5.0 以降での `jdbc.properties`プロパティのサポートが追加されます。	
EFM 5.1 以降に `jdbc.loglevel`プロパティを追加します。 EFM 5.1 から、JDBC ドライバーからログに記録される情報を増やすために使用される新しいプロパティ `jdbc.loglevel`があります。このプロパティは、接続に問題がある場合に詳細情報を取得するために使用できます。データベース接続に ssl を使用する場合。	
EFM 5.1 以降に `check.vip.timeout property`を追加します。 EFM 5.1 以降、スタンバイを昇格する前に VIP を使用する場合に到達可能かどうかを EFM がチェックし続ける時間を制御するために使用される新しいプロパティ `check.vip.timeout`があります。	

2.7.3 の変更

Description	Addresses
Ubuntu 20.04Focal は、古いディストリビューションになりました。 Ubuntu 20.04 はアップストリームのサポートが終了し、TPA でも完全にサポートされなくなりました。	
古い EDB / 2q リポジトリのサポートを削除しました。古い 2q リポジトリと EDB Repos 1.0 が利用できなくなったため、TPA はそれらを使用しようとせず、それらに関連する構成をチェックしません。これらのリポジトリで以前に利用できたすべてのパッケージは、EDB Repos 2.0 リポジトリで利用できるようになりました。	
TPA は、SLES で EDB リポジトリを使用する場合、PGDG リポジトリを抑制するようになりました。 SLES システムで `tpaexec configure`を実行する場合、他のオペレーティングシステムの動作と一致する EDB リポジトリまたは PGDG リポジトリの両方ではなく、いずれかを使用します。これにより、両方の場所で利用可能な barman などのパッケージが、他のパッケージと同じリポジトリから一貫してソースされることが保証されます。	

次のページに続く

表 20 – 前のページからの続き

Description	Addresses
TPA は、デフォルトの PGD プロキシバージョンとして `bdr_package_version` を使用するようになりました。 `pgd_proxy_package_version` および `pgd-cli_package_version` は、config.yml で明示的に定義されている場合があります。明示的に定義されていない場合、`bdr_package_version` の値がデフォルトで設定されている場合。それらが何も設定されていない場合、パッケージの最新の利用可能なバージョンがインストールされます。これらのパッケージバージョンを別の値に設定することはサポートされておらず、テストにのみ役立ちます。	49150
`vacuum_cost_delay` のデフォルトを 2 ミリ秒に削減します。最近の Postgres バージョンでは、`vacuum_cost_delay` の推奨デフォルトが 2 ミリ秒に削減されました。 TPA はこれに一致するようになりました。	
ノードのプロモーションロジックと EFM の `efm-not-promotable` ロールを文書化しました。 `efm.md` ドキュメントに新しいセクションを追加し、フェールオーバー中にノードがプロモーションの対象かどうかを TPA が判断する方法を説明します。この更新では、監視ノード、カスケードスタンバイ、`efm-not-promotable` ロールで明示的にマークされたノードのロールを含むプロモート性のルールを明確にします。この機能強化は、ユーザーがフェールオーバー動作を理解および制御するのに役立ち、EFM 管理クラスターでの意図しないプロモーションのリスクを削減します。	
テンプレート変数の `include_vars` 動作を文書化します。 `include_vars` モジュールは式をすぐに解析して評価するため、ネストされた変数は `config.yml` からロードされた時点では存在せず、したがって、テンプレート式で評価されるときに未定義です。これは例を使用して文書化されたため、ユーザーがわかりやすくなります。	45722
Rocky Linux 組織の Docker イメージを使用します。 TPA は、Docker Hub の「公式」イメージではなく、Rocky Linux 組織の docker イメージを使用するようになりました。これらはより頻繁に更新されるため、新しいパッケージで依存関係の問題が発生する可能性が低くなります。	
PGD 6 アーキテクチャのアーキテクチャ情報を更新しました。 PGD6 アーキテクチャに正しいメタデータがあるため、 `tpaexec info architectures` の出力に予想通りに表示されます。	

2.7.4 バグ修正

Description	Addresses
replication `ユーザーがプライマリノードの` .pgpass `から欠落している問題を修正しました。 *replication_user* `エントリーは、` *failover_manager* `として` *efm* `または` *patroni* `のいずれかによって管理される` *replica* `および` *primary* `ノードの両方の` .pgpass `ファイルに必要です。以前は、` replica `ノードの.pgass ファイルにのみ追加されていたため、スイッチオーバー後にプライマリノードへの接続問題が発生しました。	48224

次のページに続く

表 21 – 前のページからの続き

Description	Addresses
TPA が PGD6 クラスターの加入者専用グループのルーティングを有効にしようとする問題を修正しました。以前は、TPA は、PGD6 展開中に誤って加入者専用グループのルーティングを有効にしようとしていました。このプロパティは、PGD6 のサブスクライバ専用グループの定義のコア部分を形成するため、編集できません。	49673
PEM 10.1.1 で PEM 展開が失敗する問題を修正しました。 TPA タスク `Register PEM backend database server for monitoring and configuration`は、 *pem.setup* SQL ファンクションを明示的に呼び出します。これは PEM のパブリック API の一部とはみなされず、シグネチャが 10.1.0 と 10.1.1 の間で変更されたため、TPA のこのファンクションへの呼び出しが失敗します。この修正では、インストールされている PEM サーバーバージョンに応じてファンクションコールを調整することにより問題を解決します。この変更に対応するために、インストールされた PEM バージョンに関するファクトを収集する新しい`pem/server/facts` Ansible ロールを導入しました。これは、TPA が Postgres Expert が存在しない PEM 10 で`postgresexpert.sql`を実行しようとしないことも意味します。	
複数のフェイルオーバーマネージャーを使用する場合に`harp_dcs_user`が作成されない問題を修正しました	49150
`bdr_node_groups`をオーバーライドできない問題を修正しました。 `tpaexec configure`の--overrides-from 引数で提供されたファイルが `cluster_vars`で`bdr_node_groups`を設定する場合、このコンテンツは自動的に作成されるノードグループに追加されます。	50550
TPA が PGD6 ノードの DSN で `pgdproxy`ユーザーを使用する問題を修正しました。 PGD バージョン 6 には、PGD プロキシに代わるビルトイン接続マネージャーがあるため、 *pgdproxy* Postgres ユーザーは DSN で使用しないでください。 Connection Manager への接続文字列を定義するために、新しい変数 `bdr_connection_manager_route_dsn`が導入されました。ユーザーは `config.yml`で `bdr_connection_manager_dsn_attributes`を追加定義して、DSN で追加の接続パラメーターを指定できます。	
複数の Barman サーバーを使用する Patroni クラスターの`ignore_slots`の問題を修正しました。 Patroni クラスターの場合、TPA は、クラスター内の Barman ノードに対応する ignore_slots 設定を動的に生成するようになりました。この変更により、Patroni は、各 Barman ノード用に作成された物理スロットを削除しません。これは、複数のバックアップサーバーがある Patroni クラスターにとって特に重要です。例、共有 Barman ノードを含む。	50748

次のページに続く

表 21 – 前のページからの続き

Description	Addresses
TPA Docker イメージの使いやすさを修正し、信頼性を向上させました。 TPA の Python 3.12 最小要件を満たし、最新の依存関係との互換性を保証するためにベースイメージを Debian Trixie にアップグレードしました。 - *--use-community-ansible* `フラグはサポートされなくなったため、削除しました。 - `*openssh-client* `パッケージを含めて、`*ssh-keygen* `の欠落による `*tpaexec* `の障害を防ぎ、すぐに使える機能を強化しました。 - ビルドタイムバージョニングの導入 `*ARG* `変数を追加して、ビルド中にホストから `*git describe* `の出力を渡し、バージョン追跡を向上させるために/opt/EDB/TPA/VERSION`として保存します。 - 最新のセットアップ手順とベストプラクティスを反映するように`docker/README.md`および`docs/src/INSTALL-docker.md`を修正しました。	
Barman が`switch-wal`を実行できない問題を修正しました。 Postgres 15+の場合、 Barman ユーザーが`pg_checkpoint`ロールで作成されるようになりました。これにより、Barman はスーパーユーザーでなくても CHECKPOINT を実行できます。`barman switch-wal --force`は失敗しません。	

2.8 Trusted Postgres Architect 23.38.1 release notes

リリース日 2025 年 6 月 25 日

Trusted Postgres Architect 23.38.1 は、次の問題を解決するバグ修正リリースです。

2.8.1 バグ修正

Description	Addresses
ベアプラットフォームを使用する PGD6 の configure コマンドの問題を修正しました。 PGD6 の新しくリリースされたアーキテクチャ PGD-S および PGD-X での tpaexecconfigure コマンドのバグを修正しました。`--platform bare`を使用してクラスターを生成しようとすると、不明なプラットフォームエラーが発生します。この修正により、configure コマンドはベアプラットフォームの PGD6 構成ファイルを正常に生成します。	49673

次のページに続く

表 22 – 前のページからの続き

Description	Addresses
pgpass_users が.pgpass ファイルに正しく追加されない問題を修正しました。以前は、 *generate_password: true* `で指定され、 `*pgpass_users* `リストに含まれる `*postgres_users* `リスト内のユーザーは、初期展開で `*~postgres/.pgpass* `ファイルに追加されません。これは、 *pgpass* タスクが実行されたときにユーザーのパスワードがまだ存在せず、したがって、ユーザーがスキップされたためです。これは、すべての *Postgres* ユーザーが作成された後、*pgpass* タスクを再度呼び出すことにより修正されます。 `*repmgr* `および `*replication* `ユーザーは、以前は `*default_pgpass_users* `リストに含まれていました。これらは `*pgpass_users* `に追加され、使用中のレプリケーションマネージャーの必要に応じて*.pgpass* ファイルに追加されます。 `*postgres_user* デフォルトでは `postgres` または `enterprisedb` は、`default_pgpass_users`リストの一部です。これが `postgres_user`を含まない `config.yml`の `pgpass_users`リストによってオーバーライドされる場合、PEM 対応クラスターは、暗号化された `postgres_user`パスワードを必要とするため、エージェントの登録に失敗します。これは、PEM エージェントタスクの一部として `postgres_user`を `~postgres/.pgpass`ファイルに追加することにより修正されます。	
無効な cluster_vars 辞書により、展開の失敗または予期しない構成が発生する問題を修正しました。以前は、無効な辞書が `config.yml`の `cluster_vars`辞書として設定されている場合他の `cluster_vars`変数を参照する `cluster_vars`変数など、TPA は空の辞書のデフォルト値にフォールバックすることにより Ansible エラーを飲み込んでいた。これにより、すべての `cluster_vars`変数が未定義になるため、それぞれが TPA デフォルト値に設定されました。結果のクラスターは、ユーザーが `config.yml`ファイルで指定したものとは完全に異なります。これは、構成ファイルのロード時に `cluster_vars`ディクショナリが定義され空でないことをアサートすることにより修正されます。また、最終的な救済として、 `cluster_vars`変数がデフォルトで空の辞書に設定されなくなりました。これにより、Ansible は `group_vars`の作成時にエラーをスローして終了できます。	

2.9 Trusted Postgres Architect 23.38.0 release notes

リリース日 2025 年 6 月 9 日

Trusted Postgres Architect 23.38.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.9.1 ハイライト

- Expanded と Essential アーキテクチャの両方での EDB Postgres 分散バージョン 6 の展開のサポート
- CVE-2025-2291 の緩和を含む PgBouncer 実装の改善
- PEM 10.1 のサポート

2.9.2 機能強化

Description	Addresses
PGD6 アーキテクチャのサポート。 TPA は、 PGD6 に基づいて PGD-X および PGD-S アーキテクチャを使用してクラスターを構成および展開できるようになりました PGD-S アーキテクチャは PGD Essential を実装し、PGD-X アーキテクチャは PGD Expanded を実装します。これらのアーキテクチャには、合理的なデフォルト構成があり、さまざまな構成オプションを受け入れて動作をカスタマイズします。 PGD 6 展開には pgd-proxy が含まれません。代わりに、PGD のビルトイン接続マネージャーが構成されます。新しいアーキテクチャのテストサポートが追加されます。	
クラスター内に両方が存在する場合、 Barman を監視するように PEM を構成します。クラスターが PEM を有効にして `--enable-pem`オプションを使用して構成され、 Barman ノードを含む場合、次のアクションが自動的に実行されるようになりました。 - `enable_pg_backup_api`は `config.yml`で `true`に設定されます - `pem-agent`ロールは Barman ノードに割り当てられます - Barman エンドポイントローカル PEM エージェントに登録されています。これらの変更により、セットアップが簡素化され、PEM と Barman 間のシームレスな統合が保証されます。	
AWS での Rocky Linux 9 のサポートを追加しました。 TPA は、 `aws`プラットフォームで Rocky Linux 9.5 を使用したクラスターの構成をサポートするようになりました。バージョンが指定されない場合、これは AWS の Rocky Linux のデフォルトバージョンになりました。	
SLES 15 で pg_backup_api のサポートを追加しました。 PEM モニタリングが有効で、 Barman ノードがクラスター内に存在する場合、TPA は pg_backup_api が SUSE Linux Enterprise Server 15 SLES 15 で構成されます。	

2.9.3 の変更

Description	Addresses
PEM 10.1 以降では、`PEM_DB_PORT`を文字列として扱います。 PEM 10.1 では、Web アプリケーションからバックエンドサーバーまでのマルチホスト接続文字列のサポートを追加します。この変更をサポートするために、PEM の `config_setup.py`ファイルの `PEM_DB_PORT`パラメーターが整数ではなく文字列になりました。 TPA は HA PEM 構成の展開をまだサポートしていませんが、PEM バージョンが 10.1 以上の場合、TPA はこのパラメーターを文字列として正しく設定するようになりました。	
`repo`がタスクから除外される場合、TPA はリポジトリチェックをスキップするようになりました。 `repo`タグは除外できますが、以前は `sys/repositories`ロールの下のタスクのみをスキップしていました。また、どのリポジトリを使用するかをチェックする初期化タスクをスキップし、それらにアクセスするための資格情報が提供されていることを確認します。	

次のページに続く

表 24 – 前のページからの続き

Description	Addresses
`postgres_package_version`の動作を改善しました。`postgres_package_version`設定により、TPA は EDB Postgres Advanced Server または EDB Postgres Extended Server をインストールしている Debian または Ubuntu システムに、さまざまな postgres 関連コンポーネントの選択したバージョンをインストールします。これにより、新しいパッケージバージョンがリポジトリで表示される場合の依存関係解決の問題を回避します。	
新しいタスクセレクター `create_postgres_system_user`および `create_pgd_proxy_system_user`を追加しました。 postgres_user および pgd_proxy_user オペレーティングシステムユーザーをスキップできる新しいタスクセレクターを追加しました。これにより、クラスターは、NIS などの集中ユーザー管理によって作成されたリモートユーザーを使用できます。これは config.yml で設定できます。*cluster_vars: excluded_tasks: - create_postgres_system_user - create_pgd_proxy_system_user*	48601,44388
TPA は、スイッチオーバー中に PgBouncer を M1 repmgr クラスターの新しいプライマリにリダイレクトします。 TPA は、`repmgr_redirect_pgbouncer`が true に設定されている repmgr + PgBouncer クラスターで`switchover`コマンドを使用した後、PgBouncer インスタンスが新しいプライマリノードにリダイレクトされるようになりました。`tpaexec switchover`コマンドは、PgBouncer インスタンスが新しいプライマリノードに接続するようになりました。最初のスイッチオーバーが実行されて最初のプライマリノードに戻った後、新しい`revert_redirect`変数を追加変数として設定することもできます。	
カスタムリポジトリ定義で `key_id`/`gpgkey`をオプショナルにしました。`key_id`および`gpgkey`パラメーター apt および yum カスタムリポジトリー定義の場合、基になるモジュールでは必要ありません。インストール前にこれを提供するのが簡単ではないユースケースがあります。この変更により、TPA は、カスタムリポジトリ定義でそれらを提供することを必須にしません。	
PGD CLI 5.7.0+の`tpaexec test`の`verify-settings`チェックを修正しました。 PGD CLI コマンド`pgd verify-settings`の出力は、PGD 5.7.0 で変更されました。 PGD CLI の 5.7.0 より上のバージョンを使用する場合、TPA は出力を正しく解析するようになりました。 verify-settings は、将来の PGD リリースで他のコマンドとともに非推奨になることに注意してください。これらのコマンドは、非推奨が発生するまで、新しいコマンドを呼び出すラッパーになります。	
`barman_role`に権限を付与するためのロジックが改善されました。 TPA は`postgresql_privs`モジュールを使用して`barman_role`に付与を適用するように、必要な場合にのみ変更が適用されます。さらに、 PGD クラスターでは、TPA は 2 番目の展開で`bdr_database`を使用するため、 DDL は PGD によってクラスター全体でレプリケートされます。	
config.yml の`cluster_vars`変数テンプレートを文書化しました。`config.yml`で不適切にテンプレート化された場合に定義されていないインベントリ変数に関連する予期しない動作による混乱を避けるために、`cluster_vars`で定義された変数の正しいテンプレート手順を作業例で説明するドキュメントを追加しました。	48797

次のページに続く

表 24 – 前のページからの続き

Description	Addresses
TPA は、configure 中に無効な CIDR が `--network`フラグに渡された場合、ArchitectureError を送出するようになりました。デフォルトでは、Python 標準ライブラリ *ipaddress* **`パッケージは、 *CIDR* の *'* 厳密な *'* 解釈を強制します。これにより、使用される *IP* は範囲のネットワークアドレスである必要があります。以前は、ホストビットを含む**--network`フラグに渡される IP は、ValueError が発生したためスタックトレースをダンプしていました。その例外がキャッチされ、ArchitectureError が発生して、`--network`パラメーターに関する明確なメッセージをユーザーに表示します。	
EFM application.name プロパティのデフォルト値を追加しました。ノードに EFM `application.name`プロパティが設定されていない場合、TPA はデフォルトとして Postgres `cluster_name`プロパティを使用します。 EFM は、スイッチオーバーを実行するとき、または新しいスタンバイデータベースを構築するときにこの値を使用します。	
TPA は、SUSE で EDB リポジトリセットアップスクリプトを使用するようになりました。以前は、TPA は繰り返しの展開で機能しないため、SUSE で EDB リポジトリセットアップスクリプトを使用しませんでした。スクリプトがインストールしようとしているリポジトリが既に存在し、一意の名前が必要なため、Zypper は発生します。リポジトリが既にインストールされている場合、リポジトリのセットアップスクリプトタスクがスキップされるようになったため、この問題は発生しません。	

2.9.4 バグ修正

Description	Addresses
TPA は、専用のデータベースに`pgbouncer_get_auth()`ファンクションを作成します。 `pgbouncer_get_auth()`ファンクションは`pg_catalog`スキーマで作成され、`pgbouncer_auth_user`に実行許可されています。このファンクションはすべてのデータベースで作成されましたが、これは PgBouncer には必要ありませんでした。このファンクションが`pg_catalog`スキーマで作成された場合、`pg_dump`によって作成されたダンプに含まれないため、`pgd node upgrade`プロセス中に障害が発生する場合があります。後のタスクは、このファンクションで`GRANT`を実行しようとしますが、ファンクションは最初にダンプされていないため復元されないため、失敗します。現在、このファンクションは単一のデータベースでのみ作成され、`config.yml`の`pgbouncer_auth_database`変数の下で名前付けられます。これが含まれない場合はデフォルトの`pgbouncer_auth_database`になります。 `pgbouncer`ロールを持つ少なくとも 1 つのインスタンスがクラスターに含まれる場合にのみ作成されます。将来の TPA リリースはそのスキーマからファンクションを削除する可能性があるため、データベースが`pg_catalog`スキーマでこのファンクションを定義している場合、展開およびアップグレード中に警告も発行されます。 PgBouncer`auth_query`が使用する`pgbouncer_get_auth()`ファンクション自分自身が更新されて、CVE-2025-2291 に対処しました。この脆弱性により、有効期限が切れたパスワードを使用した認証が可能になりましたが、a_query メカニズムは、ユーザーパスワードの PostgreSQL で設定された`VALID UNTIL`属性を考慮していませんでした。	42911, 45068
`--check`オプションを使用すると、一部のタスクが誤ってスキップされる問題を修正しました。 HARP のない PGD クラスターでは、`Read current configuration file if exists`タスクをチェックモードで実行して、次の HARP チェックタスクを正しくスキップするために利用できる情報があることを確認する必要があります。ただし、デフォルトでは、Ansible はチェックモード中に`shell`モジュールを使用してタスクをスキップします。これは、このタスクが実行されず、後続のタスクで擬似障害が発生することを意味します。このタスクを実行する必要があることを Ansible に通知します。	
`cluster_vars['patroni_conf_settings']['bootstrap']['dcs']`を介して`ignore_slots`に追加された設定が最終的な構成にマージされないバグを修正しました。	
レプリカロールを持つ BDR ノードの RAFT チェックがアップグレード中にスキップされない問題を修正しました。`subscriber-only`ノードの物理レプリケーションは、フェールオーバーマネージャーとして`repmgr`をインストールし、`subscriber-only`ノードを`primary`として指定し、別の BDR データノードを`backup`としてリストすることにより、`PGD`クラスターで実現できます。このバックアップノードには、`replica`ロールが付与されます。 TPA は BDR データノードで RAFT が有効になっていることを想定しているため、この構成では PGD アップグレードプロセスが失敗しますが、物理レプリカ BDR データノード`replica`と`bdr`ロールの両方はデザインによってそうではありません。修正として、アップグレードプロセスの特定の BDR 固有のタスクは、`replica`ロールを持つノードをスキップし、アップグレードを成功させます。	46186

2.10 Trusted Postgres Architect 23.37.0 release notes

リリース日 2025 年 3 月 24 日

Trusted Postgres Architect 23.37.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.10.1 ハイライト

- upgrade コマンドは、Patroni および EFM クラスターでの Postgres のマイナーバージョンのアップグレードをサポートするようになりました
- repmgr クラスターのアップグレードも改善されました

2.10.2 機能強化

Description	Addresses
M1 + Patroni クラスターのマイナーバージョンの Postgres のアップグレード。 TPA は、フェールオーバーマネージャーとしてパトロニを使用する M1 クラスターで Postgres を最新のマイナーバージョンにアップグレードできるようになりました。アップグレードプロセスは、クラスター内の Barman サーバーで Barman を停止し、クラスター内のレプリカをアップグレードします。次に、一時的なプライマリとして 1 つのレプリカに切り替え、元のプライマリで Postgres をアップグレードし、元のプライマリに戻します。 Patroni のクラスターの処理は、プロセス中に一時停止され、その後再開されます。次に、 Barman が再起動され、クラスターヘルスチェックが実行されます。	102662
M1 + repmgr クラスターのマイナーバージョンの Postgres アップグレードを改善しました。 Witness ノードは、以前はアップグレードから省略されましたが、レプリカとともにアップグレードされるようになりました。 Postgres サービスの再起動の信頼性が向上し、ノードでのパッケージのアップグレードが完了した直後に常に実行されます。	
M1 + EFM クラスターのマイナーバージョンの Postgres のアップグレード。 TPA は、フェールオーバーマネージャーとして EFM を使用する M1 クラスターで Postgres を最新のマイナーバージョンにアップグレードできるようになりました。アップグレードプロセスは、クラスター内の Barman サーバーで Barman を停止し、クラスター内のレプリカをアップグレードします。次に、一時的なプライマリとして 1 つのレプリカに切り替え、元のプライマリで Postgres をアップグレードし、元のプライマリに戻します。 EFM エージェントは、さまざまなサーバーで適切なタイミングで起動および停止されます。次に、 Barman が再起動され、クラスターヘルスチェックが実行されます。	103595
出力内の変更されたタスクから未変更のタスクを分離します。 TPA のデフォルトの出力プラグインでは、「ok」を返すが変更のないタスクが、変更を報告したタスクから分離され、黄色で強調表示されます。	

次のページに続く

表 26 – 前のページからの続き

Description	Addresses
Docker インスタンスのプロビジョニング解除の速度が向上しました。 Docker インスタンスのプロビジョニングを解除する場合、TPA はコンテナを停止する代わりに kill し、すべてのインスタンスに並列して実行します。	
EFM 5.x の `check.num.sync.period`プロパティを追加しました。 EFM 5.0 以降、プライマリエージェントがプライマリデータベースで num_sync を変更する必要があるかどうかをチェックする頻度を定義する新しいプロパティ`check.num.sync.period`があります。 `config.yml`で指定できるようになりました。	

2.10.3 の変更

Description	Addresses
TPA は、構成が変更されない場合でも、EFM 構成ファイルが削除される場合、コピーするようになりました。 *efm.nodes* `または `*efm.properties* `構成ファイルがトップレベルの *EFM* ディレクトリに存在しない場合、構成変更がなくても、`*efm upgrade-conf*`コマンドは/raw`ディレクトリからそれらをコピーします。これにより、 `upgrade-conf`コマンドを実行してファイルをコピーする前に構成の変更が必要だった以前の動作が修正されます。	
TPA は、EDB リポジトリ設定の URI にアクセスできることを確認します。 EDB リポジトリは、EDB Repos 2.0 ドキュメントに従って、setup スクリプトを使用してセットアップされます。cURL 出力を bash にパイピングして実行します。ただし、ユーザーが存在しない`EDB_SUBSCRIPTION_TOKEN`またはリポジトリを cURL に渡すと、bash は空の入力を実行するため、終了コードは静かに飲み込まれ、0 に置き換えられます。このバージョンでは、セットアップスクリプトのダウンロードを続行する前に、要求がリポジトリの GPG キーエンドポイントにディスパッチされ、`404`応答が返されないようにします。さらに、リポジトリが既に設定されている場合、EDB リポジトリのセットアップに関連するタスクがスキップされるようになりました。	
BDR-Always-ON が BDR バージョン 5 で構成されている場合、TPA は ArchitectureError を発生するようになりました。 *-a BDR-Always-ON* `で--bdr-version 5`が渡された場合、 `tpaexec configure`中にアーキテクチャエラーが発生し、 BDR バージョン 5 を `PGD-Always-ON`で使用する必要があることをユーザーに警告します。	

2.10.4 バグ修正

Description	Addresses
デプロイ中の `shared_preload_libraries` 計算を修正しました。構成を変更しない 2 番目の展開シナリオでも、`shared_preload_libraries` の不要で制御されない書き換えをトリガーし、その後 Postgres サービスの再起動が必要になるリストの順序付けの Ansible の処理の制限を修正しました。	
Patroni の構成時に不要なレプリケーションスロットが作成される問題を修正しました。 Patroni クラスターをセットアップするときに、etcd 専用ノードごとにレプリケーションスロットが作成されました。未使用のスロットにより WAL が蓄積されるため、これにより問題が発生しました。スロットは DB サーバー用にのみ作成されるようになりました。	
patronictl switchover `コマンドの使用法を修正しました。 *TPA* は、 `*patronictl switchover* `コマンドを使用する場合、非推奨の`--master` パラメーターの代わりに `--leader` を正しく使用するようになりました。	
PEM を含む BDR-Always-ON クラスターのインスタンスのカウントに関する問題を修正しました。特定の状況で、TPA が PEM サーバーを使用した BDR-Always-ON クラスターのインスタンス数を誤って計算し、「tpaexec configuration」が「StopIteration」で失敗するバグを修正しました。	
PEM サーバーが EFM クラスターのクラスタープライマリとして誤って選択される問題を修正しました。クラスターが pem サーバーで作成された場合、そのバックエンドは EFM によって監視されないため、クラスター全体の Postgres プライマリを検出するときにそのノードを含めないでください。この動作は、TPA によって正しく実装されるようになりました。	45279
.pgpass ファイルの重複行を修正しました。 'tpaexecdeploy' を再実行すると、同じユーザーの.pgpass に余分な行が追加されるバグを修正しました。	

2.11 Trusted Postgres Architect 23.36.0 release notes

リリース日 2025 年 2 月 19 日

Trusted Postgres Architect 23.36.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.11.1 ハイライト

- Ubuntu 24.04 および SLES 15 SP6 のサポート
- すべてのクラスターコンポーネントのバージョン指定子
- EDB Failover ManagerEFM 5 のサポートの向上

2.11.2 機能強化

Description	Addresses
すべてのクラスターコンポーネントのパッケージバージョン指定子のサポートを追加しました。次のソフトウェアパッケージは、 *tpaexec configure* `コマンドへの--xxx-package-version`オプションを受け入れるようになりました。これは、生成された `config.yml` - barman - pgbouncer - beacon-agent - etcd - patroni - pem-server - pem-agent - pg_backup_api - pgd_proxy - pgdcli - repmgr に `xxx_package_version`を設定します。	
Ubuntu Noble 24.04 のサポートを追加しました。 TPA は Ubuntu 24.04 で実行され、この OS をターゲットホストとしてサポートします。 **Ubuntu 24.04 の EDB パッケージは、2025 年 3 月末に一般提供されます。その前に、EDB パッケージを必要とする展開は失敗する可能性があります。**	
EFM 5 の「auto resume」プロパティのサポートを追加しました。 EFM 5.0 以降、`auto.resume.period`プロパティは 2 つのプロパティに分割されています。1 つは起動ケース用、もう 1 つは db 障害ケース用です。この変更により、使用されている `efm_version`に基づいて、正しいプロパティが追加されます。	
EFM 5 'backup.wal' プロパティのサポートを追加しました。 EFM 5.0 から、新しいプロパティ backup.wal が追加されました。この変更により、EFM のバージョンが 5 以上の場合、新しいプロパティが追加されます。	
SLES 15 SP6 のサポートを追加しました。構成時に SLES が要求されると、TPA は SLES 15 SP6 をインストールします。 Docker および EC2 イメージは SP6 になり、systemd-sysvcompat パッケージは SLES にインストールされるため、ローカルブート時スクリプトは引き続き機能します。	

2.11.3 の変更

Description	Addresses
RHEL 同様のシステムで PGDG の最新の Barman を使用します。 TPA は、壊れたパッケージの回避策として、RHEL 同様のシステムに PGDG からインストールする場合、以前はデフォルトの Barman 3.9 を使用していました。より新しい barman パッケージは OK であるため、yum に最新のパッケージをインストールさせます。	
RedHat および Debian のセットアップスクリプトを介して EDB リポジトリをセットアップします。この変更により、TPA ノードでの EDB リポジトリのセットアップが簡素化され、*dnf*、*yum*、および `apt`パッケージマネージャーの単一のタスクになります。これにより、EDB リポジトリをターゲットホストに追加する際の予期しない HTTP エラーの発生が減少します。	
efm_conf_settings `のドキュメントを修正しました。以前、ドキュメントには*「*TPA* で認識されるかどうかにかかわらず、*efm_conf_settings* を使用してパラメーターを設定できます。必要に応じて、*efm.properties* に表示されるとおりに値を正確に引用する必要があります。ただし、 *`efm.properties.j2* `テンプレートは次の値を使用します *`efm_conf_settings* `から *Ansible* 辞書として変換するため、エントリは *`key: value* `フォームで記述する必要があります。 *`yaml cluster_vars: efm_conf_settings: notification.level: WARNING ping.server.ip: <well known address in the network>*	

次のページに続く

表 30 – 前のページからの続き

Description	Addresses
Python の依存関係をバージョン 3.12 にバンプします。 TPA では Python 3.12 が必要になりました。 EDB Repos 2.0 から TPA をインストールすると、適切な Python インタープリターが自動的にインストールされます。	
PGD のアップグレード中にプロトコルバージョンの更新を待ちます。 PGD3 から PGD5 へのアップグレード中に、プロトコルバージョンの更新に時間がかかる場合があります。 PGD 5 特定の構成変更は、プロトコルバージョンが変更される前に試行されると失敗します。このような障害を回避するために待機を追加しました。	
新しいノードが既存の EFM クラスターに追加されるときに、 *<cluster-name>.nodes* を更新します。 *TPA* の以前のバージョンでは、新しい *EFM* ノードが *config.yml* に追加されると、クラスター内の既存の *EFM* ノードの *Allowed node host list* にリストされませんでした。 *efm upgrade-conf* を実行し、/raw/<clustername>.properties`および `/raw/<clustername>.nodes`から変更を伝播するタスクは、これらのファイルのいずれかが変更されたときに実行されるようになりました。これにより、新しい EFM ノードが `<clustername>.nodes`ファイルに書き込まれ、`efm cluster-status`が `Allowed node host list`に含めます。	
postgres ユーザーの `.bashrc`ファイルに `PGPORT`を含めます。 *PGPORT* 環境変数は、*postgres* ユーザーの.bashrc`ファイルの一部としてエクスポートされます。選択した `postgres_flavour`、または構成ファイルで指定されている場合は `postgres_port`が使用するポート値がデフォルトになります。	

2.11.4 バグ修正

Description	Addresses
データベースが構成どおりではなくデフォルトのパラメーターで作成される問題を修正しました。 TPA の以前のバージョンでは、それを必要としない拡張機能別名「モジュール」の `CREATE EXTENSION`ステートメントを抑制するために、*postgres_databases* ハッシュ全体が変更されました。これは、データベース *name* と *extensions* のリストのみを含む新しいハッシュが作成され、データベースの他のすべての構成設定を無視するためバグが発生しました *owner*、*template*、`encoding`など。これにより、データベースは構成どおりではなくデフォルトのパラメーターで作成されることになりました。これを修正するには、拡張機能のリストからモジュールが削除され、結果のリストが CREATE EXTENSION を実行するタスクに渡されます。	44539
Patroni クラスターでの `shared_preload_libraries`の問題を修正しました。 shared_preload ライブラリのエントリが Patroni によって正しく処理されるようになりました。これにより、`config.yml`を介して pglogical 拡張機能を Patroni クラスターに追加すると失敗するバグが修正されます。	

次のページに続く

表 31 – 前のページからの続き

Description	Addresses
拡張機能用の追加のリポジトリセットアップコマンドがないディストリビューションの展開が失敗するバグを修正しました。 TPA の以前のバージョンでは、`postgis`が `postgres_databases`のデータベースの `extra_postgres_extensions`または `extensions`リストに追加されると、`repository_setup_commands`のリストが空であったため、Debian、SLES、および Ubuntu での展開は失敗していました。 RHEL のみが run`crb enable`への追加コマンドを持っています。この空のリストは `command`モジュールに渡され、`no command given`で失敗し、展開が失敗しました。ディストリビューションに実行する追加コマンドがない場合、`Automatically run additional repository setup commands for recognized extensions`タスクがスキップされるようになりました。	
`repmgr_use_slot`が 0 に設定されている場合でも、TPA がレプリケーションスロットを作成しようとした問題を修正しました	41776
`pgd_proxy_dsn_attributes`および `pgd_cli_dsn_attributes`のデフォルトとして `bdr_client_dsn_attributes`を設定します。 pgd-proxy および pgd-cli は Go で書かれ、Go ドライバーを使用するため、libpq がサポートするパラメーターキーワードの完全なセットをサポートしていません。クラスターが pgd-proxy および/または pgd-cli をインストールし、Go ドライバーがサポートしていないパラメーターで `bdr_client_dsn_attributes`を構成している場合、2 つの新しい構成変数 `pgd_proxy_dsn_attributes`および `pgd_cli_dsn_attributes`を含める必要があります。 Go ドライバーがサポートしています。逆に、 pgd-proxy および pgd-cli がインストールされ、`bdr_client_dsn_attributes`が Go 非互換のパラメーターを含まない場合、これらのツールの接続文字列は `bdr_client_dsn_attributes`の属性で構成されます。これにより、ユーザーが `bdr_client_dsn_attributes`のパラメーターが使用されることを想定していても、 `pgd_proxy_dsn_attributes`および `pgd_cli_dsn_attributes`が定義されていない場合に空の文字列がデフォルトであった予期せぬ動作が修正されました。	44819
etcd のホームディレクトリの標準形式を使用します。 etcd ユーザーを作成するときに、TPA は後続のスラッシュなしでホームディレクトリを参照するようになりました、 'useradd' のような他のツールの使用法と一致します。	
cluster_ssh_port `設定が *Barman* によって尊重されない問題を修正しました。 *Barman* 構成は、 *`config.yml`* の *`cluster_ssh_port`* を介して設定されたカスタム *ssh* ポートを使用できるようになりました。設定されていない場合はデフォルトの *22* です。 *`-p / --port`* フラグは、それぞれ *`barman.d.conf`* および *`barman-wal-restore/`* `barman-wal-archive' コマンドの `ssh`コマンドに含まれるようになりました。	

2.12 Trusted Postgres Architect 23.35.0 release notes

リリース日 2024 年 11 月 25 日

Trusted Postgres Architect 23.35.0 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

2.12.1 ハイライト

- STIG / CIS コンプライアンスのためのオプション。
- PGD 軽量アーキテクチャのサポート
- Postgis は認識される拡張機能になりました。
- Docker `configure` は、静的 IP アドレスを使用して名前付けネットワークを作成します。
- ARM アーキテクチャ用の RedHat Enterprise Linux 9 のサポート。
- PostgreSQL、 EDB Postgres Extended、および EDB Postgres Advanced Server 17 のサポート。

2.12.2 機能強化

Description	Addresses
STIG / CIS コンプライアンスをサポートします。 TPA は、STIG および CIS セキュリティ標準の多くの要件に準拠するように構成されたクラスターを作成するコマンドラインオプションをサポートするようになりました。これらのオプションにより、TPA は、関連する標準で定義されているように postgresql.conf 設定を設定し、必要な拡張機能をインストールし、ファイルシステムの権限やユーザー接続制限などのシステム動作の他の側面を構成し、FIPS 暗号標準などの他の要件をチェックします。 TPA は直接課すことはできません。このように生成されたクラスターは、標準に適合することは TPA によって認定されていませんが、適合するクラスターを作成する基礎作業の多くが自動化されています。	
PGD Lightweight アーキテクチャのサポートを追加します。 TPA は、物理レプリケーションからの移行を容易にするように設計された 2 つの場所の 3 つのノードプライマリに 2 ノード、ディザスターリカバリーに 1 つで構成される PGD Lightweight アーキテクチャを生成できるようになりました。ユーザーは`tpaexec configure lw -a Lightweight --postgresql 15`を実行できるようになりました。	
`configure`に Docker でユーザー定義のネットワークを作成してもらいます。Docker が選択されたプラットフォームである場合、configure コマンドは、名前付けネットワークと静的 IP アドレスを config.yml に自動的に追加します。ネットワーク名はクラスター名と同じであり、アドレス範囲は--network オプションの既存のセマンティクスに従いますが、場所ごとに 1 つではなくクラスター全体で 1 つのサブネットのみが使用されることを除きます。ユーザーがサブネットプレフィックスを指定しない場合、TPA はプレフィックスを選択しようとします。その結果、クラスター全体に適合する十分な大きさのサブネットが得られます。config.yml で名前付けネットワークが指定されている限り、キー `ip_address`を使用して Docker インスタンスの静的 IP を指定できるようになりました。	

次のページに続く

表 32 – 前のページからの続き

Description	Addresses
新しいクラスターのバックアップノードとして既存の Barman ノードを使用するための実験的なサポートを追加しました。既存の Barman ノードを新しいクラスターのバックアップノードとして使用する場合、ユーザーはプラットフォームを `bare`に設定し、ベアインスタンスに通常どおり提供されるその他の情報を使用して、 Barman インスタンスの vars に `barman_shared: true`を設定できます。Barman ユーザーは既に以前の展開からプロセスを実行しているため、この変更により、TPA は usermod の問題が原因で失敗するいくつかの構成手順をスキップできます。共有 Barman インスタンスはベアインスタンスとして扱われるため、ターゲット PostgreSQL インスタンスへの Barman ユーザーのアクセスを含む必要なアクセスが既に設定されている必要があります。 Barman ユーザーのキーを元のクラスターから新しいクラスターにコピーすることは、これを実現するために使用できます。詳細については、TPA ドキュメントの Barman セクションを参照してください。	
`postgis`を認識される拡張機能のリストに追加します。 PostGIS パッケージは、ユーザーが `postgres_extensions`または `postgres_databases`の下で名前付けた拡張機能のリストのエントリとして `postgis`を指定すると、自動的に追加されます。また、RHEL 互換ディストリビューションの CRB Code Ready Builder リポジトリを有効にして、PostGIS 依存関係をインストールできるようにします。	
PEM エージェントが EFM ノードに登録されているときに EFM プローブを有効にします。 *--efm-install-path* **`および`** --efm-cluster-name`フラグは、PEM サーバーが EFM ノードに登録されるときに設定されます。 *Streaming Replication*、*Failover Manager Node Status*、および `Failover Manager Cluster Info`プローブは、PEM エージェントが EFM ノードに登録されているときに有効になります。	
ARM アーキテクチャ向けの RedHat Enterprise Linux 9 をサポートします。パッケージは RHEL 9 ARM64 をターゲットとして公開され、TPA はこのアーキテクチャと OS を使用した展開をサポートしています。また、サポートされる AWS イメージのリストを更新して、Amazon が提供する RedHat 9 ARM64 AMI を含めます。 ARM64 EC2 インスタンスのデフォルトの `instance_type`が `a1`から `t4g`に更新されました。これは、バースト可能な汎用ワークロードで使用可能な現行世代のプロセッサです。	
PostgreSQL、EDB Postgres Extended、および EDB Postgres Advanced Server 17 をサポート。 PostgreSQL、EDB Postgres Extended、および EDB Postgres Advanced Server バージョン 17 を使用するようにクラスターを構成できます。Barman は、postgres サーバーパッケージをインストールして使用するときに `pg_receivewal`バイナリを取得する必要がなくなりました。 EDB Postgres Advanced Server 17 または EDB Postgres Extended 17。バイナリがこれらのバージョンのクライアントパッケージに追加されているため。 TPA は、クラスターが failover_manager として `repmgr`で構成されている場合、Postgres 17 で使用できないため、アーキテクチャエラーを発生します。サポートされているバージョンを反映するようにドキュメントを更新しました。	

次のページに続く

表 32 – 前のページからの続き

Description	Addresses
efm Postgres の `password_encryption`アルゴリズムをユーザー設定可能にします。構成可能な`efm_user_password_encryption`変数を公開します。これは、ユーザー要件に応じて `'md5' または'scram-sha-256' のいずれかに設定する必要があります。これは、 *`pg_hba.conf`の `efm* Postgres ユーザーの `auth-method`と、その暗号化されたパスワードを生成するために使用されるアルゴリズムを制御します。*stig `に構成された `compliance `で展開されたクラスターでは、FIPS 対応オペレーティングシステムでは `md5 `の使用が許可されていないため、 `pg_hba.conf`の 'efm' Postgres ユーザーの `auth-method* は`scram-sha-256`に設定されます。	
ホスト名とともに複数のアドレスを指定できるようにします。 *tpaexec configure `に*--hostnames-from`オプションを使用する場合、各行に 2 つの IP アドレスを含めることができるようになりました。これらは、生成された config.yml に public_ip および private_ip として含まれます。	

2.12.3 の変更

Description	Addresses
postgres.service.j2 テンプレートの非推奨の `PermissionStartOnly`を削除します。`PermissionsStartOnly`は非推奨であり、`ExecStartPost=+/bin/bash...`構文を介して実現されます	
barman Postgres ユーザーはスーパーユーザーではなくなりました。特定の必要な特権が Postgres ロール`barman_role`に付与され、これが`barman` Postgres ユーザーに付与されます。これにより、スーパーユーザーとして `barman`ユーザーを作成する必要がなくなります。このロールは、`postgres/createuser`を使用して`granted_roles`リストに追加することにより、他の Postgres ユーザーに付与することもできます。 `barman_role`は、 Barman タスクの一部として作成されます。Barman を使用しない場合、このロールは作成されません。したがって、このロールに特権を付与するタスクは、`barman_role`ユーザー名が作成される Postgres ユーザーのリストにある場合にのみ実行されます。 'barman' ユーザーは、ロール属性として明示的に指定された `NOSUPERUSER`を持つようになりました。クラスターが以前の TPA バージョンスーパーユーザーとして作成されたバーマンユーザーで展開された場合、このバージョンで展開すると `barman`ユーザーから `superuser`ロール属性を削除します。	
HARP で etcd を使用する場合、新しいオプション `harp_local_etcd_only`を追加します。 HARP で etcd を使用する場合に使用可能な新しいオプションの var `harp_local_etcd_only`を追加します。このオプションは、 HARP マネージャーにローカル etcd ノードに接続するように指示します。この推奨事項は、コンセンサスプロトコルとして`bdr`が使用されるときに同じことを行うことにより学習されたベストプラクティスに従います。複数のエンドポイントを追加するデフォルトモードでは、場合によってパフォーマンスの問題が発生する可能性があります。このオプションは、ユーザーにさらなる制御を与えるために追加されます。	

次のページに続く

表 33 – 前のページからの続き

Description	Addresses
postgres-monitor スクリプトを改善します。 postgres-monitor スクリプトを改善して回復可能なエラーをより適切に管理し、ネットワークエラーの再試行を追加して、postgres サービスが完全に開始するための十分な時間を許可できなかった場合に障害を返さないようにします。	
`efm`ロールを持つノードのみをクラスター`efm.nodes`ファイルに追加します。以前は、`pemserver`および`barman`ノードは、EFM 機能に関連しない場合、EFM の`Allowed node host list`に追加されました。ロールのリストに`efm`があるノードのみを含めるように`efm.node`構成を書き込むタスクをリファクタリングしました。	

2.12.4 バグ修正

Description	Addresses
pgd-proxy 構成検証の tpaexec テストを修正します。現在の pgd-proxy 構成が予想される構成と一致することを確認するテストが、バージョン<5.5.0 では失敗するバグを修正しました。この修正により、TPA はバージョン 5.5.0 で追加された構成キーを照会しようとしません。	
EFM 互換性のために追加された`primary_slot_name`が`bdr_init_physical`と干渉する場合を修正します。`primary_slot_name`は、プライマリノードで構成されて、古いプライマリが EFM スイッチオーバー中にレプリケーションに物理スロットを使用するようにします。ただし、'bdr_init_physical' はノードの初期化にそれを使用しようとしますが、スロットが PGD インストールに存在しないため、無期限にハングします。`failover_manager`が EFM の場合、この`primary_slot_name`は条件付きで明示的に設定されるようになり、不必要な設定を回避します。	
正しい`bash-completion`パッケージバージョンをダウンロードします。`config.yml`で`pgdcli_package_version`が指定されている場合、`packages_for`フィルタは誤って`pgdcli_package_version`をパッケージ名に追加するため、`bash-completion`パッケージの名前は誤って命名されます。これにより、存在しないパッケージをダウンロードしようとします。`bash-completion`パッケージのバージョンは`pgdcli_package_version`から独立しているため、`packages_for`フィルターの後のリストに追加されます。	
場合によっては、タスクが成功した後でもエラーメッセージが繰り返される問題を修正します。 TPA は、各タスクの後にエラーメッセージスタックをクリアして、メッセージが誤って繰り返されないように保証	
Patroni クラスターへのレプリカの追加を妨げた問題を修正します。 Patroni クラスターの新しいレプリカがレプリケーションスロットに関連するエラーで失敗する問題を修正しました。	

次のページに続く

表 34 – 前のページからの続き

Description	Addresses
M1 アーキテクチャの場合、barman ノードに `pem-agent` ロールを最大 1 回追加します。 *--enable-pem* `および`--enable-pg-backup-api` が `tpaexec configure` に渡される場合、 `witness` でもある場合、 `pem-agent` は `barman` ノードに 2 回追加されます。両方の `if` ステートメントを統合して、条件を 1 回のみ評価することにより修正しました。	
`pem_python_executable` を `pkg` ロールの外部に設定します。ユーザーが `pkg` セレクターを除外すると、 `pem_python_executable` ファクトが設定されていないために後の PEM 関連タスクが失敗するバグを修正しました。	

2.13 Trusted Postgres Architect 23.34.1 release notes

リリース日 2024 年 9 月 9 日

Trusted Postgres Architect 23.34.1 は、次の問題を解決するバグ修正リリースです。

Type	Description
Bug Fix	`efm-witness` ロールを持つノードのスイッチオーバーが失敗した後に deploy を実行する問題を修正しました。 EFM ノードの `upstream-primary` は、Postgres から収集されたファクトを使用して決定されます。これは、以前は Postgres がないため、 `efm-witness` ロールを持つノードでは失敗していました。上流プライマリを決定するタスクは、 `primary` または `replica` ロールを持つノードでのみ実行されます。

2.14 Trusted Postgres Architect 23.34 release notes

リリース日 2024 年 8 月 22 日

Trusted Postgres Architect 23.34 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	TPA には、efm がインストールされ、構成ディレクトリとユーザーが作成された後、efm が構成される前に実行される `efm-pre-config` フックが追加されました。カスタム efm ヘルパースクリプトのインストールに使用できます。
Enhancement	TPA には、タスクごとに 1 行の情報を表示する独自の出力プラグインがあり、1 行でも情報のないタスクを省略します。線は、TPA の制御フローを可視できるようにインデントされており、成功、スキップ、および無視されたホストの色分けされた数が含まれています。 より完全な出力は、ご使用の環境で TPA_USE_DEFAULT_OUTPUT=true に設定するか、コマンドラインに-v スイッチを追加することにより、オンにできます。

次のページに続く

表 36 – 前のページからの続き

Type	Description
Enhancement	TPA では、PEM エージェントを登録するときに、`cluster_vars`の `pemagent_registration_opts`の下にリストすることにより追加オプションを指定できるようになりました。
Enhancement	cluster_vars または pem-server インスタンス vars `pem_server_ssl_certificate`および`pem_server_ssl_key`の下の`config.yml`に PEM サーバーで使用する証明書とキーペアの名前を含めることにより、PEM サーバーで使用する独自の Web サーバー証明書を提供できるようになりました。 TPA は、クラスターディレクトリの`ssl/pemserver`ディレクトリから PEM サーバーにそれらをコピーし、それに応じて Apache/httpd を構成します。
Enhancement	TPA は、新しいクラスター展開で EFM `upgrade-conf`コマンドを実行して、`<cluster_name>.properties`ファイルと`<cluster_name>.nodes`ファイルの両方に追加されるコメントとインラインドキュメントを活用するようになりました。
Enhancement	TPA は、 *bind.address* `値の *IP* アドレスの代わりにホスト名解決を使用して *EFM* クラスターをセットアップできるようになりました。 これは、*configure* コマンドの`--efm-bind-by-hostname`オプションまたは config.yml の cluster_vars セクションの `efm_bind_by_hostname: true/false`で呼び出すことができます。
Enhancement	TPA は、EFM 4.9 で追加された EFM プロパティ `enable.stop.cluster: boolean, default true`、*priority.standbys: default ''*、*detach.on.agent.failure: boolean, default false*、`pid.dir: default ''`の設定をサポートするようになりました。
Enhancement	TPA は、 pgBouncer から Postgres への接続に `cert`認証を使用するように pgBouncer を構成できるようになりました。これは、 cluster_vars で `pgbouncer_use_cert_authentication`を true に設定することにより有効にします。有効にすると、pgBouncer に接続するユーザーの認証方法も `md5`から`scram-sha-256`に変更されます。
Change	TPA は、AWS での RAID 作成をサポートしなくなりました。
Change	upstream_primary を解決するための EFM 依存関係を削除しました。以前は、スイッチオーバー後の展開で、現在のプライマリの EFM が照会されていました。 EFM が実行されていない場合、これは失敗します。 Postgres を介して収集された cluster_facts は、スイッチオーバー後の現在のプライマリを決定するために使用され、EFM への依存関係を削除します。
Change	EFM クラスターでは、スイッチオーバー後に `upstream_primary`が正しく更新されるようになりました。その結果、レプリカに正しい`auto.reconfigure`設定が設定されます。スタンバイが新しいプライマリに従うようになりました。
Bug Fix	Patroni API にアクセスするときに TPA がプロキシ設定を誤って適用する問題を修正しました。 Ansible のデフォルトでは、プロキシが定義されている場合、それを使用します。これは、プロキシも Patroni API への呼び出しをインターセプトおよび中断するため、インターネットからパッケージをダウンロードするためにプロキシを必要とするエアギャップ環境のかなり一般的な場合では機能しません。
Bug Fix	TPA がすべてのインスタンスで `pemserver`インスタンスにのみ適切な PEM エージェントパラメータを設定する問題を修正しました。

次のページに続く

表 36 – 前のページからの続き

Type	Description
Bug Fix	クラスターノードのターゲットとして SLES オペレーティングシステムを使用する場合、pgd-proxy および pgdcli のデフォルトパッケージ名に欠落していたエントリを追加しました。
Bug Fix	TPA が既存のノードで postgres をリロード/再起動して構成変更を再読み取りすることに失敗し、したがって、新しいノードがクラスターに接続できないという問題を修正します。
Bug Fix	レプリカからバックアップを取得する場合、WAL ファイルの待機がタイムアウトになることにより、初期バックアップの取得時にバーマンが失敗する場合がある問題を修正しました。これは、WAL セグメントのスイッチを強制する前に、バーマンがベースバックアップを完了するのを待機することにより修正されます。
Bug Fix	上流_プライマリ postgres にデフォルト以外の postgres ポートを使用する場合でも、`repmgr witness register`コマンドが正しい postgres_port 値で使用されていることを確認します。
Bug Fix	PGD クラスター内のサブスクライバー専用ノードとそのレプリカのインスタンスレベルで設定されている場合、 `repmgr`への failover_manager オーバーライドが正しく機能しない問題を修正しました。
Bug Fix	誤った cgroup 検出の 2 つのケースを修正しました。MacOSX では、`/proc/mounts`を読み取ろうとしなくなりました。 `/sys/fs/cgroup`が `ro`であるが、その下にマウントが `rw`であるシステムでは、TPA がこれを正しく検出するようになりました。
Bug Fix	pgd-proxy に新しく追加された `read_listen_port`オプションの pgd-proxy ノードで実際の構成セットを検証できることを確認します。
Bug Fix	PEM 9.7.0 での展開を妨げる問題を修正しました。 PEM 9.7.0 はパッケージレベルで Apache に依存しないため、Web サーバーとして Apache を使用するには、パッケージを明示的にインストールします。

2.15 Trusted Postgres Architect 23.33 release notes

リリース日 2024 年 6 月 24 日

Trusted Postgres Architect 23.33 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	TPA は、ARM64 CPU アーキテクチャで Debian 12 Bookworm をサポートするようになりました。

次のページに続く

表 37 – 前のページからの続き

Type	Description
Enhancement	PGD バージョン 5.5 では、プロキシノードを読み取りエンドポイントとして構成でき、読み取り専用クエリをシャドウノードに転送します。 TPA は、`config.yml`の `default_pgd_proxy_options`および `pgd_proxy_options`の下の`read_listen_port`パラメーターを設定することにより、この構成オプションをサポートします。このパラメーターは、 PGD バージョンが 5.5 以上の場合、デフォルトで含まれます。 `--proxy-listen-port`および`proxy-read-listen-port`引数を`tpaexec configure`コマンドに渡すことにより、ポート番号を指定することもできます。
Enhancement	TPA は、ロール'beacon-agent' を割り当てるか、*configure* `**とともに**--enable-beacon-agent`オプションを使用することにより、Postgres ノードでのビーコンエージェントの展開と構成をサポートするようになりました。
Enhancement	Patroni 展開での `postgres_wal_dir`のサポートを追加しました。 TPA 構成でカスタム `postgres_wal_dir`が指定されている場合、TPA はそのオプションを Patroni 構成ファイルの対応する設定に中継します。このようにして、Patroni が TPA から独自にスタンバイを再構築する必要がある場合、スタンバイはカスタム WAL ディレクトリで適切にセットアップされます。
Enhancement	Patroni クラスターに PgBouncer ノードを追加すると、TPA は`on_role_change`コールバックで Patroni を構成するようになりました。そのコールバックは、フェールオーバーおよびスイッチオーバーイベントに応答して、PgBouncer ノードのプライマリ接続情報を更新します。
Enhancement	EDB は、PGDG から `patroni`パッケージをリビルドする代わりに、独自の`edb-patroni`パッケージを生成するようになりました。 TPA では、ユーザーが`patroni`パッケージと`edb-patroni`パッケージを選択できるようになりました。選択は、新しい TPA 設定`patroni_package_flavour`を介して行われます。
Change	PGDG リポジトリの壊れた Barman 3.10 パッケージをワークアラウンドために、RHEL ファミリーシステムで PGDG リポジトリを使用している場合、TPA は Barman のバージョン 3.9 をインストールするようになりました。この動作は、 config.yml で barman_package_version を明示的に設定することによりオーバーライドできます。
Change	Patroni がフェールオーバーマネージャーの場合、`haproxy_bind_address`は`0.0.0.0`に設定されます。これは、Postgres ノードと HA プロキシノード間の通信を妨げる `127.0.0.1`の一般的なデフォルトに関する問題を解決します。ユーザーはこの値を、クラスターネットワーキングに適したより制限的なものに変更する必要があります。
Change	タスクセレクターが、展開の最終段階で一貫して適用されるようになりました。テストのタスクセレクターの一貫性が向上し、ドキュメントのタスクセレクターの例が正しくなりました。すべての展開時フックに、対応するタスクセレクターがあります。
Change	`barman_package_version`が設定されている場合、TPA は barman-cli パッケージおよび Barman 自分自身を検索するときにそれを参照するようになりました。これにより、`barman_package_version`が使用されたときにダウンローダーを使用するクラスターで障害が発生した不一致が解決されます。

次のページに続く

表 37 – 前のページからの続き

Type	Description
Bug Fix	BDR データベース内の機能に必要な権限が、監視ノードの HARP DCS ユーザーに付与されていない問題を修正しました。
Bug Fix	「読み取り専用ファイルシステム」で Docker プロビジョニングが失敗する問題を修正しました。最近の OS イメージを実行している docker コンテナで cgroup1 を実行しているホストシステムでは、`tpaexec provision`はコンテナのプロビジョニングに失敗し、「mkdir /sys/fs/cgroup/tpa.scope: read-only file system」のようなエラーメッセージを表示する場合がありました。 TPA はこのケースを検出して回避します。
Bug Fix	TPA は、プロビジョニング前にユーザーが`tpaexec cmd`または`tpaexec ping`を実行した場合、明確なエラーメッセージを提供するようになりました。

2.16 Trusted Postgres Architect 23.32 release notes

リリース日 2024 年 5 月 15 日

Trusted Postgres Architect 23.32 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	M1 アーキテクチャは、*tpaexec configure* `への追加引数`--location-names`、*--primary-location*、*--data-nodes-per-location*、*--witness-only-location*、および`--single-node-location`をサポートするようになりました。 これらの引数を組み合わせることで、`config.yml`を編集することなく、ほとんどの一般的なレイアウトを指定できます。
Enhancement	TPA は、展開中に chrony をインストールし、Amazon Time Sync サービスをポイントする AWS を除き、すべてでデフォルト構成を維持します。
Enhancement	TPA は、IBM PowerPPC64le で RHEL 8 および 9 をサポートするようになりました。
Enhancement	*tpaexec relink* `に`--force`オプションを追加しました。デフォルトでは、再リンクはターゲットファイルが既に存在する場合を変更しません。`--force`では、再リンクは既存のすべてのターゲットファイルを削除し、それらを再作成します。
Enhancement	TPA が Debian 12 x86 をサポートするようになりました。
Enhancement	`pg_failover_slots`が認識される拡張機能になりました
Enhancement	*sql_profiler*、*edb_wait_states*、および `query_advisor`拡張機能が `pem-agent`ノードに自動的に含まれるようになりました。 pem-agent ノードのデフォルトの拡張機能のリストは、`config.yml`に`pemagent_extensions`のリストを含めることによりオーバーライドできます。 このリストが空の場合、拡張機能は自動的に含まれません。

次のページに続く

表 38 – 前のページからの続き

Type	Description
Change	TPA は、RHEL 7 を除くすべてのシステムの cgroups 2 を実行しているホストに Docker クラスターをプロビジョニングできるようになりました。新しいシステム RHEL 9 または Ubuntu 22.04 では、TPA は cgroups 2 スコープを使用してホストとコンテナ間の分離を追加します。
Change	AWS AMI バージョンを最新バージョンに更新しました。
Bug Fix	AWS で Debian 10 への展開がメッセージ `The repository 'http://cdn-aws.deb.debian.org/debian buster-backports Release' does not have a Release file`で失敗する問題を修正しました。 debian 10 Buster のバックポートリポジトリは、deb.debian.org で利用できなくなりましたが、標準の AWS AMI はまだそれを参照しているため、apt 操作を試行する前に/etc/apt/sources.list をそれに応じて変更します。
Bug Fix	`assign_public_ip:no`が設定されている場合、AWS での展開が失敗する問題を修正しました。
Bug Fix	barman と postgres のカスタムユーザーを使用しようとしたときに混合エラーを発生させ、展開の失敗を引き起こしたさまざまなロールの問題を修正しました。
Bug Fix	キー `ip_address`を使用して IP アドレスを定義した場合、最初の展開以降の展開が `Unrecognised host=... in primary_conninfo`のようなエラーで失敗する場合がある問題を修正しました。
Bug Fix	`tpaexec upgrade`がエラーを発生させ、`tpaexec relink`の役に立たない使用メッセージを表示する方法でリリンクスクリプトを呼び出すことができるエラーを修正しました。
Bug Fix	Postgres をリロードするタスクがスキップされ、restore_command オーバーライドが `postgresql.auto.conf`から削除されない問題を修正しました
Bug Fix	ソースから BDR をコンパイルしようとする前に、TPA がソースディレクトリに変更しなかった問題を修正しました。
Bug Fix	指定されたクラスターに必要でない場合でも、TPA が有効な 2ndQuadrant トークンを必要とする問題を修正しました。
Documentation	tower/AAP のドキュメントを更新して、AAP 実行環境の作成に関する手順を含めます。

2.17 Trusted Postgres Architect 23.31 release notes

リリース日 2024 年 3 月 19 日

Trusted Postgres Architect 23.31 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Bug Fix	構文エラーのために展開が失敗する可能性がある重大なバグを修正しました。

2.18 Trusted Postgres Architect 23.30 release notes

リリース日 2024 年 3 月 19 日

Trusted Postgres Architect 23.30 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
New Feature	TPA は、Ansible Automation Platform 2.4+コントローラバージョン 4+で使用されるカスタムの'ExecutionEnvironment' イメージを提供するようになりました。このイメージには、AAP を介して展開を実行するために必要なすべてが含まれています。このイメージは、ansible-builder と python-alpine の軽量ベースイメージを使用してビルドされます。
Enhancement	TPA は、拡張機能のサブセットのパッケージ名と共有プリロードライブラリエントリを自動的に追加するようになりました。 これらの特定の拡張機能の場合、`postgres_databases`のデータベースエントリの`extra_postgres_extensions`リストまたは`extensions`リストでは拡張機能名前のみが必要です。
Enhancement	EDB Advanced Storage Pack パッケージと共有プリロードライブラリエントリは、ユーザーが拡張機能として指定し、`postgres_version`が 15 以上の場合、`bluefin`に自動的に追加されます。
Enhancement	インスタンスに新しい'provision_only' オプションを追加しました。インスタンスの config.yml に`provision_only: true`がある場合、通常どおりプロビジョニングされますが、 `tpaexec deploy`によって認識されるインベントリには追加されません。
Change	TPA の以前のバージョンでは、ソースノードのデータベース構造を監視ノードに同期していました。これは必要ではなく、同期されたスキーマは使用または更新されません。これが発生するのを防ぐため、TPA は、監視ノードの`bdr.join_node_group()`を呼び出すときに「synchronize_struction」を「none」に明示的に設定するようになりました。
Change	Ansible タグではなくカスタムセレクターを使用して、タスクの選択的な実行がサポートされるようになりました。特定のセレクター `tpaexec deploy . --included_tasks=barman`に一致するタスクのみを実行するには。特定のセレクターに一致するタスクをスキップするには *tpaexec deploy . --excluded_tasks=ssh* タスクセレクターは、 config.yml で`excluded_tasks`または`included_tasks`変数を指定して使用できます。
Change	Ansible 2.9 は、コミュニティディストリビューションも 2ndQuadrant フォークもサポートされなくなりました。 *tpaexec deploy* `に--skip-tags`オプションを使用しているユーザーは、新しい`--excluded_tasks`オプションに移動する必要があります。

2.19 Trusted Postgres Architect 23.29 release notes

リリース日 2024 年 2 月 15 日

Trusted Postgres Architect 23.29 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	クラスターボルトのパスワードをシステムキーリングに保存するためのサポートを追加しました。これは、`keyring_backend`が`system`新しいクラスターのデフォルトに設定されている場合、Python キーリングモジュールを活用して、サポートされているシステムキーリングにボールトパスワードを保存します。この変更は、既存のクラスター、または config.yml で `keyring_backend`から`legacy`を設定するクラスターには影響しません。
Enhancement	*tpaexec setup* `の--ansible-version`引数は、有効な ansible バージョンとして`8`または`9`、および既存の `2q`または `community`、どちらも ansible 2.9 を暗黙的に受け入れます。デフォルトは`8`になりました。 ansible 9 のサポートは実験的であり、Python 3.10 以降が必要です。
Bug Fix	config.yml で既に定義された edb_repositories が、再構成中に保持されない問題を修正しました。エアギャップ環境での bdr4 から pgd5 へのアップグレードシナリオを修正します。
Bug Fix	TPA の `postgres-monitor`は、「データベースシステムはまだ接続を受け入れていません」というメッセージを回復可能なエラーとして認識するようになりました。
Bug Fix	TPA は、アップグレード時にレプリカの `postgres/config/final`ロールを正しくスキップするようになりました。
Bug Fix	Debian および Ubuntu システムでパッケージダウンローダーを使用する場合、パッケージ名のワイルドカードが尊重されない問題を修正しました。
Bug Fix	ダウンローダーは、Debian および Ubuntu システムでパッケージを取得する前に apt-get update を実行するようになりました。
Bug Fix	PGD クラスターで CAMO が有効になっている場合、TPA はトランザクションストリーミングを無効にするようになりました。
Bug Fix	TPA は、selinux が有効になっている Barman サーバーを正しく構成するようになりました。

2.20 Trusted Postgres Architect 23.28 release notes

リリース日 2024 年 1 月 23 日

Trusted Postgres Architect 23.28 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	新しいオプション`postgres_log_file`を追加しました。このオプションは、Postgres ログファイルを設定します、stderr または syslog を介してロギングするかどうか。デフォルトは'/var/log/postgres/postgres.log' であり、以前にハードコードされた値です。
Enhancement	新しいフック`barman-pre-config`を追加しました。このフックは、Barman がインストールされ、そのユーザーがセットアップされた後、構成される前に呼び出されます。これは、証明書ファイルまたは barman ユーザーの存在を必要とするその他のタスクのインストールに使用できますが、 Barman が開始される前に行う必要があります。
Enhancement	`config.yml`の AWS インスタンスのキー`elastic_ip`は、このインスタンスに割り当てるために、既に割り当てられた Elastic IP アドレスに設定できます。
Change	Patroni クラスターでは、TPA は、プライマリのみを設定して Patroni にレプリカをセットアップさせるのではなく、クラスターの制御を Patroni に引き渡す前にレプリカをセットアップするようになりました。
Change	新しいクラスターの場合、TPA は`bdr_superuser`ロールに属する`harp_manager_user`デフォルトでは`harpmanager`の設定で指定されたユーザーを作成し、`postgres`スーパーユーザーの代わりにこのユーザーとして動作するように HARP Manager を設定します。これは、ユーザが`config.yml`で`harp_manager_user`を明示的に設定することによりこの動作をオーバーライドしない限り、TPA が HARP Manager ユーザーとして`postgres`を使用し続ける既存のクラスターに影響しません。
Bug Fix	TPA が誤って EFM クラスターに repmgr をインストールしようとする問題を修正しました。
Bug Fix	デプロイが成功したにもかかわらず、2q リポジトリに関する警告が表示されると、TPA がゼロ以外の終了コードを返す問題を修正しました。
Bug Fix	TPA は、Debian ファミリーシステムでオフライン使用のためのパッケージをダウンロードするときにワイルドカードを正しく解釈するようになりました。
Bug Fix	PGDG リポジトリからインストールするときに、TPA が repmgr の誤ったパッケージ名を使用しようとする問題を修正しました。
Bug Fix	selinux とカスタム barman ホームディレクトリを使用する場合の barman 接続障害を修正しました。
Bug Fix	TPA は、ディレクトリ名と異なる場合、`show-password`および`store-password`コマンドで正しいクラスター名を使用するようになりました
Bug Fix	使用できない 2ndQuadrant リポジトリキーが必要な場合、TPA はきれいにエラーアウトするようになりました。
Bug Fix	--cluster-prefixed-hostnames オプションが使用される場合、TPA はホスト名を正しくサニタイズするようになりました。
Bug Fix	TPA は、ローカルリポジトリを使用してクラスターをアップグレードするときに、パッケージがリモートホストに正しくコピーされるようになりました。

2.21 Trusted Postgres Architect 23.27 release notes

リリース日 2023 年 12 月 19 日

Trusted Postgres Architect 23.27 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	TPA は、Docker で Oracle Linux 7、8、および 9 をサポートするようになりました。
Change	TPA は Python 3.9-3.11 を必要とし、適切なインタープリターを提供するように `edb-python-39` パッケージに依存しています。
Change	TPA は、デフォルトで 2ndQuadrant リポジトリを構成せず、代わりに EDB Repos 2.0 から適切なリポジトリを選択します。
Change	TPA は、新しい `--replace-2q-repositories` 引数を `tpaexec reconfigure` に提供するようになりました。これは、既存の config.yml から 2q リポジトリを削除し、クラスターの postgres フレーバーと BDR バージョンに適切な EDB リポジトリを追加します。
Change	TPA は、より多くのオブジェクトにファイルシステムのアクセス許可を明示的に設定するようになりました。
Change	config.yml で新しい変数 `disable_repository_checks` を true に設定して、PGD-Always-ON アーキテクチャを展開するときに EDB リポジトリの通常のチェックをバイパスできます。
Change	TPA は、スイッチオーバーの場合に使用されるプライマリノードでも primary_slot_name を生成して、スイッチされたプライマリが新しいプライマリに物理スロットを持つようにします。
Change	TPA は、メジャー PGD アップグレードの準備をするために tpaexec reconfigure コマンドを実行するときに、古い BDR バージョンの既存の構成オプションを使用して、 CAMO 対応パートナーの commit_scope が生成されるようになりました。また、より良いデフォルトも選択します。
Bug fix	postgres 変数が検証ルールによりパトローニによって拒否される問題を修正しました。
Bug fix	ユーザーが `sslmode=verify-full` で単一の `barman_client_dsn_attributes` を設定できない問題を修正しました。
Bug Fix	TPA は、メモリ不足エラーを回避するために、より低いデフォルトの `maintenance_work_mem` を割り当てます。

2.22 Trusted Postgres Architect 23.26 release notes

リリース日 2023 年 11 月 30 日

Trusted Postgres Architect 23.26 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	TPA は、Docker プラットフォームで Oracle Linux 9 をサポートするようになりました。
Enhancement	`--cluster-prefixed-hostnames`オプションを`tpaexec configure`に追加しました。これにより、複数の Docker クラスターをホストするマシンでのホスト名の衝突を簡単に回避できます。
Change	Mac OS X で Docker ビルドを有効にするパッケージを追加しました。
Change	クラスター内に複数の PEM サーバーが存在する場合、PEM サーバーで実行されているエージェントはローカルサーバーに登録します。
Change	CAMO を使用した PGD 5 クラスターの場合。 TPA は、デフォルトでタイムアウトを 60 秒に設定し、 require_write_lead を true に設定します。
Bug Fix	PGD 3 クラスターを PGD 5 にアップグレードするときに CAMO 構成が正しくセットアップされない問題を修正しました。アップグレードが CAMO クラスターで完全にサポートされるようになりました。
Bug Fix	HARP ノードをフェンシングまたはアンフェンシングするときに、`bdr_node_name`ではなく`hostname`が使用される問題を修正しました。
Bug Fix	デプロイを抑制するオプションで`deploy`が呼び出されたときに、`provision`が自動的に実行される問題を修正しました。

2.23 Trusted Postgres Architect 23.25 release notes

リリース日 2023 年 11 月 14 日

Trusted Postgres Architect 23.25 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	TPA は、 PGD 3.7 から PGD 5.3 以上への自動アップグレードをサポートするようになりました。 ***CAMO* を使用したクラスターのアップグレードはまだサポートされていないことに注意してください。**
Enhancement	TPA が EDB Advanced Server 16 および EDB Extended Server 16 をサポートするようになりました。
Change	PGD 4 から PGD 5 へのアップグレードで導入されたアップグレードプロセスのさまざまな改善が、 BDR-Always-ON アップグレードにバックポートされました。
Change	TPA は、SLES への PEM のインストールをサポートするようになりました。
Change	TPA は、一部のファイルシステムオブジェクトを作成するときにアクセス許可を明示的に設定するようになりました。これは、将来のリリースではすべてのファイルシステムオブジェクトに拡張される予定です。
Change	TPA は v1 の pgd-cli 構成ファイルにシンボリックリンクを追加するため、`-f スイッチを介してパスを指定せずに実行できるようになりました。
Change	TPA は`alter_node_kind` PGD ファンクションを呼び出して、 BDR バージョン 4.3 以降を使用する BDR -Always-ON クラスターのノード種類が正しく設定されていることを確認します。

次のページに続く

表 45 – 前のページからの続き

Type	Description
Change	今後のデフォルトのクラスター構成は、SLES 15 が要求されたときに SLES 15 SP5 を選択するようになりました以前の SP4。
Bug Fix	tpaexec-deps ユーザーの `tpaexec setup`コマンド中にチェックサムエラーが発生する問題を修正しました。
Bug Fix	pemworker が呼び出されたときに `pem_server_group`が正しく適用されず、サーバーが PEM で予想通りにグループ化されなかった問題を修正しました。
Bug Fix	`sar`が Barman インスタンス以外のインスタンスで実行されるようにスケジュールされていない sys / sysstat ロールの問題を修正しました。

2.24 Trusted Postgres Architect 23.24 release notes

リリース日 2023 年 10 月 17 日

2ndQuadrant Ansible フォークの使用のサポートは、2024 年 4 月に TPA から削除され、GitHub リポジトリはアーカイブされます。

Community Ansible に切り替える必要があります。これがデフォルトになりました。大部分のユーザーにとって、この変更は透過的です。

2ndQuadrant Ansible で`--skip-tags`を使用している場合、これは TPA およびコミュニティ Ansible ではサポートされていないことに注意してください。2ndQuadrant Ansible が削除される前に、Community Ansible と互換性のある`--skip-tags`の代替を提供する予定です。

Type	Description
Change	*tpaexec setup* **`は、*2ndQuadrant Ansible* ではなくコミュニティ *Ansible* を使用することがデフォルトになりました。オプション**--use-2q-ansible`は、2ndQuadrant ansible の使用を強制するために使用できます。これは現在非推奨であり、将来のリリースで削除されます。 *--skip-tags* **`を使用している場合、*:ref: `Ansible Tower <TPA installation>*** を参照してください。
Change	config.yml の`edb_repositories`からリポジトリが削除されている場合、`tpaexec deploy`はノードから削除するようになりました。
Change	TPA は、 harp-proxy と harp-manager が同じノードで実行されている場合を検出し、 harp-proxy に別の構成ファイルを使用します。
Change	`upgrade`コマンドは、ターゲットインスタンスのローカルリポジトリを更新するようになりました。
Bug Fix	TPA が pg_basebackup 呼び出しの `postgres_wal_dir`を尊重しない問題を修正しました
Bug Fix	TPA は、PGD クラスター内のサブスクライバー専用ノードのフェールオーバーマネージャーとして repmgr を受け入れ、このようなノードの物理レプリケーションを許可します。
Bug Fix	TPA による Ubuntu 22.04 Docker イメージのビルドを妨げたタイプミスを修正しました。

次のページに続く

表 46 – 前のページからの続き

Type	Description
Bug Fix	TPA は、構成時に BDR-Always-ON アーキテクチャ、 EDB Postgres Extended フレーバー、および PEM のサポートされていない組み合わせを拒否するようになりました。

2.25 Trusted Postgres Architect 23.23 release notes

リリース日 2023 年 9 月 21 日

Trusted Postgres Architect 23.23 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	TPA は PostgreSQL 16 をサポートするようになりました。 PostgreSQL 16 パッケージは、サポートされているすべてのリポジトリでまだ利用できないことに注意してください。したがって、これに該当するまで、すべての構成が機能するわけではありません。
Change	Postgres 16 以降が選択されている場合、TPA はデフォルトで 2ndQuadrant リポジトリを追加しません。この場合、TPA は `tpa_2q_repositories: []`を明示的に設定します。
Change	EFM は、利用可能なプラットフォームでデフォルトで JDK 11 を使用するように構成されました。
Change	EDB リポジトリが使用されていない場合、TPA は PGDG からパッケージを除外しません。以前は Barman と psycopg2 が除外されていました。
Change	SLES へのインストールをサポートするために etcd および Patroni のパッケージ名を追加しました。
Bug Fix	PEM サーバーの Apache HTTPD サービスがブート時に開始されない問題を修正しました。
Bug Fix	不正な権限で pg_backup_api テストが実行され、失敗する問題を修正しました。
Bug Fix	pg_backup_api の Apache HTTPD サービスがブート時に開始されない問題を修正しました。
Bug Fix	`bdr.standby_slot_names`および `bdr.standby_slots_min_confirmed`チェックが bdr3 クラスターで誤ったスキーマを使用した問題を修正しました。
Bug Fix	拡張機能の構成キーが間違った形式でパトローニに渡され、「警告 予期しないパラメーターの削除」が発生する問題を修正しました。
Bug Fix	`docker_images`の中間ベースイメージオプションを使用した場合、結果のイメージ名が正しくない問題を修正しました。

2.26 Trusted Postgres Architect 23.22 release notes

リリース日 2023 年 9 月 6 日

Trusted Postgres Architect 23.22 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Change	TPA はオープンソースプロジェクトになりました。 GitHub から GPLv3 ライセンスに基づいてソースのクローンを作成できます。

2.27 Trusted Postgres Architect 23.21 release notes

リリース日 2023 年 9 月 5 日

Trusted Postgres Architect 23.21 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Change	デフォルトの M1 構成は、EDB ソフトウェアが選択されている場合は EDB Repos 2.0 を使用し、それ以外の場合は PGDG が使用されます。これは、新しいクラスターにのみ影響します。
Change	M1 アーキテクチャで `tpaexec configure`を実行する場合、フェールオーバーマネージャーを明示的に選択する必要があります。
Bug fix	TPA が誤って上流インスタンスではなくレプリカに'subscriber-only' を設定することを必要とした、PGD サブスクライバ専用ノードの作成に関する問題を修正しました。
Bug fix	TPA は、 'bare' プラットフォームオプションを使用している場合でも、コンテナに展開するときに適用できないタスクをスキップするようになりました。以前は、これらは'docker' が選択された場合にのみスキップされました。
Bug fix	`/etc/edb`の権限に関する問題を修正しました。展開された PGD5 クラスターのデータノードに pgd-proxy ロールを追加すると、pgd-proxy-config.yml を開く権限がないため pgd-proxy の起動に失敗します。
Bug fix	厳密な umask が設定されている場合、`/var/log/postgres`が不適切な権限 0600 になる可能性がある問題を修正しました
Bug fix	PEM に正しく登録された Barman インスタンスで `tpaexec deploy`を繰り返すと、PEM エージェントの Barman 構成が失われる問題を修正しました。

2.28 Trusted Postgres Architect 23.20 release notes

リリース日 2023 年 8 月 1 日

Trusted Postgres Architect 23.20 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
New feature	TPA は、新しいコマンド `tpaexec reconfigure`を実行して変更された config.yml を生成し、 `tpaexec upgrade`を実行してアップグレードを実行することにより、 PGD 4 から PGD 5 へのアップグレードをサポートするようになりました。
Enhancement	TPA インストールでチェックサムを実行し、パッケージで配布されたものと一致することを確認する新しいサブコマンド `tpaexec info validate`を追加しました。
Change	`update-postgres`コマンドは、より一般的な`upgrade`コマンドに置き換えられました。
Change	RHEL 8 または 9 に Failover Manager の OpenJDK をインストールするときに、TPA は `tzdata-java`を明示的に追加するようになりました。これは this OpenJDK bug の回避策です。
Change	TPA は、AWS で利用可能な最新の Debian AMI このリリースの時点で最新を使用するようになりました。
Change	config.yml が変更された場合、TPA は`tpaexec deploy`または`tpaexec upgrade`の一部として `tpaexec provision`を自動的に実行するようになりました。
Bug fix	pgd-proxy ノードで pgd-cli を実行しているときに TPA が存在しないユーザーを使用しようとするバグを修正しました。
Bug fix	`tpaexec relink`による変更が Git リポジトリに正しくコミットされないバグを修正しました。

2.29 Trusted Postgres Architect 23.19 release notes

リリース日 2023 年 7 月 12 日

Trusted Postgres Architect 23.19 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
New feature	TPA では、サブスクライバ専用 PGD ノードの物理レプリカを作成できるようになりました。
New feature	TPA は、HTTP(S) HARP および PGD プロキシ正常性プローブの構成をサポートするようになりました。
New feature	TPA では、M1 アーキテクチャを使用したフェールオーバーマネージャーとして Patroni を選択できるようになりました。このサポートは実験的なものであり、運用環境での使用はまだ推奨されていません。
Enhancement	TPA では、常に最新バージョンを使用するのではなく、edb-pgd-proxy および edb-bdr-ユーティリティの特定のバージョンを設定できるようになりました。
Change	Debian に似たシステムでは、パッケージ選択コードは、該当する場合、特定のパッケージに-dbgsym ではなく-dbg を使用するようになりました。

次のページに続く

表 51 – 前のページからの続き

Type	Description
Change	レプリケーションスロットを構成するときに、TPA は有効な文字のみが primary_slot_name で使用されるようになりました。以前は、TPA はデフォルトとして inventory_hostname を使用していましたが、これにはハイフンを含めることができます。これらはアンダースコアに置き換えられます。
Change	デフォルトのフェールオーバーマネージャーのバージョンは 4.7 になりました。
Bug fix	TPA 23.18 で PGD 3.7 から 4 へのアップグレードが失敗する問題を修正しました。
Bug fix	PEM 展開時に TPA が TLS 証明書のコモンネームにアンダースコアを含める問題を修正しました。これは無効であり、一部のプラットフォームでは障害が発生します。
Bug fix	誤った etcd サービス名が Debian 系プラットフォームで使用され、TPA が etcd を起動できない問題を修正しました。
Bug fix	TPA が RHEL 8 に etcd パッケージをインストールできない問題を修正しました。
Bug fix	configuration 中にメッセージ `Failed to commit files to git: b"`が表示される問題を修正しました。
Bug fix	`generate_password: false`のときに TPA が Postgres ユーザーパスワードを誤って生成および上書きする問題を修正しました。
Bug fix	AWS でのボリュームマップの作成がリージョンを考慮していないため、eu-west-1 以外のリージョンを使用すると障害が発生する問題を修正しました。

2.30 Trusted Postgres Architect 23.18 release notes

リリース日 2023 年 5 月 23 日

Trusted Postgres Architect 23.18 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	TPA は、初期レプリカの作成に repmgr の代わりに pg_basebackup を使用するようになりました。
Enhancement	TPA は、エアギャップ展開のためのローカルリポジトリの作成を除きて、SLES 15 をサポートするようになりました。
Enhancement	TPA は、PGD5 のマイナーバージョンのアップグレードをサポートするようになりました。
Enhancement	TPA は、 `tpaexec test`が実行されるときに改善されたテストを実行するようになりました。
Bug fix	TPA がサポートされていないディストリビューションで古い 2ndQuadrant リポジトリを使用しようとした問題を修正しました。
Bug fix	TPA が pg_receivewal が必要な Barman インスタンスに pg_receivewal をインストールしない問題を修正しました。

次のページに続く

表 52 – 前のページからの続き

Type	Description
Bug fix	TPA がプロビジョニング中に AWS ホストでデバイスをブロックするシンボリックリンクの作成に断続的に失敗し、展開が失敗する問題を修正しました。

2.31 Trusted Postgres Architect 23.17 release notes

リリース日 2023 年 5 月 10 日

Trusted Postgres Architect 23.17 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Enhancement	新しい`--pgd-proxy-routing`パラメーターを configure コマンドに追加しました。これは、`global`または`local`に設定できます。ローカルルーティングは、すべての PGD-プロキシを独自の場所内の書き込みリーダーにルートさせます。グローバルルーティングは、すべてのプロキシを単一の書き込みリーダーにルートさせます。この書き込みリーダーは、すべての場所で使用可能なすべてのデータノードの間で選択されます。
Change	configure コマンドから`--active-locations`パラメーターを削除しました。
Enhancement	TPA が Ubuntu 22.04 をサポートするようになりました
Change	RHEL 7 および 8 で使用される AWS AMI を更新しました。
Bug fix	TPA が既存の Postgres ユーザーからグループを誤って削除する問題を修正しました。
Bug fix	git コミットが失敗したときに TPA が役に立たないエラーメッセージを出力する問題を修正しました。
Bug fix	グループ名が誤ってサニタイズされ、大文字が小文字ではなくアンダースコアに変換される問題を修正しました。
Bug fix	CAMO 構成後に必要なときに Postgres が再起動されない問題を修正しました。
Bug fix	etcd の変更に関する問題を修正し、べき等になったことを確認し、後続の展開での etcd の不必要な再起動を回避します。

2.32 Trusted Postgres Architect 23.16 release notes

リリース日 2023 年 3 月 21 日

Trusted Postgres Architect 23.16 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Change	デフォルトの PGD-Always-ON クラスターは、2 つのデータノードと 1 つの監視ノードを含む関連サブグループを持つ 1 つの場所になりました。

次のページに続く

表 54 – 前のページからの続き

Type	Description
Change	TPA は、デフォルトですべてのデータノードに pgd-proxy を展開するようになりました。
Enhancement	別のプロキシインスタンスを作成する新しいオプション `--add-proxy-nodes-per-location N`を追加しました
Enhancement	`--data_nodes_per_location`が偶数の場合、TPA は監視ノードを自動的に追加し、2 つの場所のみのクラスターを指定した場合に警告を出力するようになりました
Change	ロケーションを追加するのではなく、既に名前付けられた`--location-names`でのロケーションを監視専用として指定するため、パラメーター *--add-witness-only-location* の名前が`--witness-only-location`に変更されました。
Change	`tpaexec configure`時に Postgres のフレーバーとバージョンを明示的に指定する必要があります
Enhancement	Postgres フレーバーとバージョンの新しい CLI 略語を追加しましたたとえば *--postgresql 14* `または--edbpge 15`
Enhancement	さまざまなサポートされている EDB ソフトウェアリポジトリの処理とドキュメントの向上
Change	TPA には、PGD-Always-ON クラスターのデフォルトで PGDG リポジトリが含まれません
Bug fix	EDB Failover Manager がデフォルトで EPAS のフェールオーバーマネージャーとして選択されない問題を修正しました
Bug fix	pglogical が M1 アーキテクチャに不必要にインストールされる問題を修正しました

2.33 Trusted Postgres Architect 23.15 release notes

リリース日 2023 年 3 月 15 日

Trusted Postgres Architect 23.15 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Minor change	依存関係マッピングの変更。

2.34 Trusted Postgres Architect 23.14 release notes

リリース日 2023 年 2 月 23 日

Trusted Postgres Architect 23.14 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Bug fix	PGD 3 および 4 構成のパッケージリストが正しく入力されなかったエラーを修正します。 TPA-365
Change	HARP の config.yaml で複数行の BDR DCS 構成を使用する TPA-360、RT90034

2.35 Trusted Postgres Architect 23.13 release notes

リリース日 2023 年 2 月 22 日

Trusted Postgres Architect 23.13 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Bug fix	PGD-Always-ON および `--epas`を使用して古い EDB リポジトリを有効にしないでください。
Bug fix	PGD-Always-ON および `--postgres-version 15`のエラーを修正します。
	NA

2.36 Trusted Postgres Architect 23.12 release notes

リリース日 2023 年 2 月 21 日

Trusted Postgres Architect 23.12 の新機能、機能強化、バグ修正、およびその他の変更には次のものが含まれます。

Type	Description
Feature	コミットスコープに基づいた Commit At Most OnceCAMO 構成サポートを含む、 EDB Postgres Distributed 5 の完全なサポートを導入します。
Feature	EDB Postgres 拡張リポジトリとパッケージのサポートを導入します。
Enhancement	マルチリージョン AWS クラスターの構成の予備サポート。 マルチリージョンクラスターには、VPC と VPC の手動セットアップが必要です。
Enhancement	`--active-locations`のプロキシルーティングおよびそのためサブグループ RAFT を自動的に有効にします。 configure オプションを削除して、サブグループ RAFT をグローバルに有効にします。
Bug fix	EDB_SUBSCRIPTION_TOKEN がログに記録されていないことを確認します。
Bug fix	ユーザーが tpa_2q_repositories への products/default/release リポジトリの追加を抑制できるようにします。

次のページに続く

表 58 – 前のページからの続き

Type	Description
Bug fix	利用可能な場合、ノードが bdr_child_group をサブスクライブしていることを確認します。 複数のサブグループを持つクラスターでは、TPA はインスタンスがトップグループとサブグループの両方のレプリケーションセットにサブスクライブされることを想定していなかったため、ノードのサブスクライブしたレプリケーションセットから後者を誤って削除します。
Bug fix	Postgres が起動しない場合、有用なエラーで確実に失敗します。 Ansible のバグが原因で、Postgres が一部のインスタンスで起動しないが、他のインスタンスで起動した場合、デプロイメントは失敗しませんたとえば、構成の違いのため。展開を続行すると、後で障害が発生したホストの cluster_facts にアクセスしようとすると、エラーが発生しました。
Bug fix	BDR 4.3 以降の監視では `bdr.alter_node_replication_sets()`を呼び出さないでください。 これは、監視ノードがカスタムレプリケーションセット構成で処理されない BDR バージョンの新しい制限に調整します。
Bug fix	ハードコードされた「barman」参照を置き換えて、barman_{user,group}設定を使用して barman ユーザーとホームディレクトリをカスタマイズできるようにします。
Bug fix	必要に応じて、 postgres_databases[*].extensions の下で記載されている拡張機能の shared_preload_libraries エントリを追加します。
Bug fix	`pgaudit`が shared_preload_libraries の `bdr`の前に表示されないことを確認します既知のクラッシュを回避するため。
Bug fix	pgd-cli 構成ファイルの構文エラー DSN 引用符を修正します。
Bug fix	pgd-proxy 構成のエンドポイントを並べ替えて、ファイルの書き換えを回避します。 これには、次の展開で pgd-proxy の再起動が必要になる可能性がありますが、その後の展開での不要な将来の書き換え/再起動が回避されます。
Bug fix	RH システムにローカルリポジトリから rsync をインストール中のエラーを修正します。
Bug fix	Debian システムでの PEM 9 の Apache WSGI モジュール構成のエラーを修正します。
Bug fix	bdr 拡張子が意図的に作成された場合、未使用であっても削除しないでください。
	NA

2.37 Trusted Postgres Architect 23.1 to 23.11 release notes

2.37.1 TPA 23.11

リリース日 2023-01-31

重要な変更

- TPA-180 PGD-Always-ON アーキテクチャ今年後半にリリースの実験的なサポートを導入します。PGD-Always-ON アーキテクチャは、次の BDR バージョン 5 を使用します。初期サポートは内部目的に追加され、今後のリリースで改善されます。

マイナー変更

- TPA-349 依存関係バージョンのバンプ暗号化バージョン 38.0.4 から 39.0.0 のバンプ jq バージョン 1.3.0 から 1.4.0 のバンプ
- TPA-345 ドキュメント内の TPAexec 参照を TPA に変更します。製品を参照するときは、’TPAexec’ の代わりに’TPA’を使用するようにドキュメントを更新します。

2.37.2 TPA 23.10

リリース日 2023-01-04

マイナー変更

- TPA-161 `harp_manager_restart_on_failure` 設定デフォルト false を導入して、harp-manager systemd サービスの障害時のプロセスの再起動を有効にします

バグ修正

- TPA-281 AWS クラスターのプロビジョニングを解除するときに FMS セキュリティグループを削除する削除されていない依存関係が原因でクラスターの VPC のプロビジョニング解除の失敗を修正します。
- TPA-305 Tower の事前生成されたシークレットに enterprisedb_password を追加します
- TPA-306 PEM_PYTHON_EXECUTABLE が存在する場合は、/usr/bin/python3 に優先します PEM 9.0 での展開中の Python モジュールのインポートエラーを修正します。
- TPA-219 BDR インスタンスで pem-agent にデフォルトで bdr_database を監視させます

2.37.3 TPA 23.9

リリース日 2022-12-12

バグ修正

- TPA-301 Tower クラスターの cluster_dir の自動検出を修正 Tower プロジェクトディレクトリに基づいて cluster_dir を設定する場合、デプロイ先のインスタンスではなく、コントローラ上のディレクトリの存在を正しく確認するようになりました。
- TPA-283 Ansible Tower に必要な psutil への依存関係を追加します。
- TPA-278 rsyslog 構成から「umask 0」指示子を削除します。これにより、以前は rsyslogd.pid などの誰でも読み取り可能なファイルが作成されます。

- TPA-291 Postgres サーバーパッケージをインストールして Barman インスタンスで pg_receivewal を取得するときは、postgres_package_version 設定を尊重します。

2.37.4 TPA 23.8

リリース日 2022-11-30

重要な変更

- TPA-18 サポート Ansible Tower 3.8 このリリースでは、Ansible Tower 3.8 を介して `bare` クラスターつまり既存のサーバーでの `deploy.yml` のみの実行をサポートしています。Tower サーバーに TPAexec をインストールし、`tpaexec setup` を実行して、Tower テンプレートで使用できる仮想環境を作成して TPAexec プレイブックを実行します。`--use-ansible-tower` および`--tower-git-repository` 構成オプションを使用して、Tower 互換のクラスター構成を生成します。詳細については、Ansible Tower を参照してください。

マイナー変更

- TPA-238 クラスターディレクトリを git リポジトリとして初期化する TPAexec を実行するシステムで git が利用可能な場合、 `tpaexec configure` はデフォルトでクラスターディレクトリ内の git リポジトリを初期化します。git が利用できない場合、以前と同様に続行されます。リポジトリの作成を回避するにはたとえば、既存のリポジトリ内にクラスタディレクトリを保存する場合、`--no-git` オプションを使用します。

2.37.5 TPA 23.7

リリース日 2022-11-09

重要な変更

- TPA-234 Ansible 2.9 のコミュニティリリースをサポート TPAexec は、Ansible 2.9 の 2ndQuadrant/ansible フォークを必要としました。このリリースでは、代わりに Ansible のコミュニティリリースを `tpaexec setup --use-community-ansible` で使用することを選択できます。今のところ、デフォルトは引き続き 2ndQuadrant/ansible を使用することです。これは将来のリリースで変更される予定です。2ndQuadrant / ansible のサポートがドロップされ、Ansible が新しいデフォルトになります。

マイナー変更

- TPA-209 Postgres 15 のリリースに続いて、有効な `tpaexec configure` オプションとして`--postgres-version 15` を受け入れます

- TPA-226 `--hostnames-from` ファイルの IP アドレスの受け入れ以前は、`tpaexec configure` に渡されるファイルには、行ごとに 1 つのホスト名が含まれることが想定されていました。各ホスト名の後にオプションの IP アドレスを含めることもできるようになりました。存在する場合、このアドレスは config.yml の対応するインスタンスの `ip_address` として設定されます。独自の`--hostnames-from` ファイルを指定すると、ホスト名がデフォルトでランダム化されなくなります。

- TPA-231 新しい bdr-pre-group-join フックを追加しますこのフックは、各ノードが BDR ノードグループに参加する前に実行されます。TPAexec が提供するデフォルトのレプリケーションセット構成を変更するために使用される場合があります。
- TPA-130 Community.postgresql の postgresql_user モジュールを使用します SCRAM パスワードを使用するときにタスクステータスを正しく報告するには、Community.postgresql コレクションの更新されたモジュールが必要です。デフォルトモジュールは常に changed を報告します。
- TPA-250 さまざまな Python 依存関係の最新バージョンへのアップグレード

バグ修正

- TPA-220 .bashrc の LD_LIBRARY_PATH が「:」で始まらないことを確認します
- TPA-82 BDR の削除を回避する -internal ${group_name}_ext レプリケーションセット
- TPA-247 AWS での「'str object' には属性 'node_dsn'」エラーを修正します。コードは `hostvars[hostname]` を中間変数に割り当てず、後で通常の dict と同様に動作することを想定しています時々のみ動作します。これにより、PEM が有効になっている AWS クラスターで報告された 23.6 の回帰が修正されますが、コードベース全体の他の同様のエラーも修正されます。
- TPA-232 インベントリーで生成されたシークレットへのシンボリックリンクを作成する際の競合状態を排除し、「Error while linking: [Errno 17] File attempts」エラーを発生させました
- TPA-252 すべての BDR ノードを監視専用レプリケーションセットに公開する復元コードこのコードブロックは、TPA-193 に対して行われたリファクタリング作業の一部として、v23.6 リリースで誤って削除されました。

2.37.6 TPA 23.6

リリース日 2022-09-28

重要な変更

- TPA-21 AWS 展開には、メンテナンスされていない boto2 の代わりに boto3 AWS クライアントライブラリを使用します。これにより、SSO ログインやその他の便利な機能が有効になります。
- TPA-202 harp-config フックを追加します。この展開時フックは、HARP がインストールおよび構成された後、HARP がインストールされているすべてのノードで開始される前に実行されます。

バグ修正

- TPA-181 RHEL 7 でデフォルトの Python バージョンを 2 に設定します。以前は、tpaexec は RHEL 7 と Python 3 のサポートされていない組み合わせで config.yml を生成できました。
- TPA-210 既存のセキュリティグループを使用して aws 展開を修正します。このような展開は、プロビジョニング時に失敗していましたが、予想通りに動作するようになりました。
- TPA-189 プロビジョニング解除時に group_vars ディレクトリを削除します。これにより、ダングリングのシンボリックリンクが原因で後続のプロビジョニングが失敗する原因となった問題が修正されます。

- TPA-175 共有メモリセグメントをそのままにするように systemd を正しく構成します。これはソースビルドにのみ影響します。
- TPA-160 haproxy および PEM のバージョン設定を許可します。これは、特定のバージョンが指定された場合でもパッケージの最新バージョンがインストールされるバグを修正します。
- TPA-172 正しいホストセットに EFM をインストールします。EFM は、PEM サーバーなどの他の理由で postgres がインストールされているサーバーではなく、クラスターのメンバーである postgres サーバーにのみインストールする必要があります。
- TPA-113 PEM エージェント登録をシリアル化します。これにより、複数のホストが pemworker –register-agent を同時に実行しようとした場合の競合状態が回避されます。

2.37.7 TPA 23.5

リリース日 2022-08-23

重要な変更

- TPA-81 Ubuntu 22.04 Jammy の tpaexec および tpaexec-deps パッケージを公開する
- TPA-26 単一ノードでの harp-proxy および harp-manager のインストールをサポートします。harp-proxy と harp-manager サービスの両方をクラスター内の同じターゲットノードで実行できるようになりました。

2.37.8 TPA 23.4

リリース日 2022-08-03

バグ修正

- TPA-152 は、AWS ホストでの Debian インスタンスの最初のブート中のロケール検出に関する問題を修正します。ホストは最初のブートを完了できず、SSH キーネゴシエーションの問題と、展開中に見つからないディスクのエラーとして現れます。この問題は 23.3 で導入され、TPA-38 に関連しています

2.37.9 TPA 23.3

リリース日 2022-08-03

重要な変更

- TPA-118 harp-manager 構成の 2 つの新しいオプションを公開しました。最初の設定は dcs request_timeout と同様に HARP `harp_db_request_timeout` を設定しますが、データベース接続用であり、2 番目の `harp_ssl_password_command` は、SSL 対応データベース接続の ssl キーの復号化に使用される ssl パスワードの難読化を解除するために使用されるコマンドを指定します

マイナー変更

- TPA-117 tpaexec config.yml の package_version オプションでのワイルドカードの使用に関するドキュメントの更新を追加します。これにより、deploy 操作中に予期しないパッケージのアップグレードが発生する可能性があるという警告が導入されます。詳細については、tpaexec-configure.md のドキュメントを参照してください
- TPA-38 Debian のすべてのバージョン、および RHEL 8 以降のロケールファイルを追加します。Barman などの一部の EDB ソフトウェアでは、ユーザーロケールを en_US.UTF-8 に設定する必要があります。一部のユーザーは、ロケール、文字セット、または言語をローカル地域に変更したい場合があります。この変更により、libc が提供する OS ファイルが、ユーザーデータスクリプトを使用したファーストブート中に AWS インスタンスにインストールされます。デフォルトロケールは en_US.UTF-8 です。詳細については、 platform_aws.md ドキュメントを参照してください
- TPA-23 クラスターサービス Barman、HARP、repmgr、PgBouncer、および EFM の syslog のログ構成を追加します。指定されたログサーバーは、これらのサービスの/var/log/hosts ディレクトリで受信したログファイルを保存します
- TPA-109 BDR の特定のバージョンに基づいて、ロックタイムアウト構文の選択を中心とした pgbench ロールのコードのマイナーリファクタリング

バグ修正

- TPA-147 ソースインストール方法を使用するクラスターでは、Debian および Rocky Linux の一部の欠落パッケージが観察されました。Debian は、krb5 および lz4 のライブラリヘッダーを受信します。RedHat 派生 OS では、「Development Tools」パッケージグループの必須パッケージと libcurl ヘッダーが追加されました
- TPA-146 Postgres 9.6 をインストールするクラスターのパッケージ選択方法の小さな修正
- TPA-138 local-repoconfigure オプションを有効にする「ベア」プラットフォームを使用するクラスターでの警告メッセージに対処します。OS はベアプラットフォームの TPAexec によって管理されないため、local-repo 構造を作成するようにユーザーに通知する必要があります。これにより、以前は未処理のエラーが発生し、構成の進行状況が停止しました
- TPA-135 “docker”プラットフォームおよび Rocky Linux イメージで--use-local-repo-only を使用する場合、ノード上の既存の yum リポジトリ構成の初期削除は、コマンド find および xargs が欠落しているため失敗します。この変更により、findutils パッケージがソースリポジトリに存在する場合、最初にインストールされます
- TPA-111 --enable-pem オプションが指定されている場合、PEM サーバーにエージェントを登録するために使用されるコマンドの冗長な追加引数を削除します。以前は、最初の引数現在削除された引数が 2 番目の引数によってオーバーライドされるため、問題は発生しませんでした。
- TPA-108 Postgres がソースからインストールされている場合、ポストマスターシンボリックリンクの SELinux ファイルコンテキストを復元します。以前は、systemd デーモンが Postgres データ bin ディレクトリに保存されているシンボリックリンクを読み取ることができないため、ソースから postgres をインストールしている SELinux 対応 OS を使用しているクラスターは、Postgres の再起動に失敗していました。これは、AWS で最近採用された Rocky Linux イメージを使用したテストで発見されました。デフォルトで SELinux が有効になり、強制モードになっています

2.37.10 TPA 23.2

リリース日 2022-07-13

重要な変更

- Barman および PEM で使用する Postgres Backup API のサポートを追加します。`--enable-pg-backup-api` オプションを介してアクセスできます。
- SSL 証明書は、Postgres Backup API プロキシサービスのサーバー証明書など、サービスごとに作成できるようになりました。証明書は`/etc/tpa/<service>/<hostname>.cert` に配置されます。これらの証明書は、クラスター用に生成された CA 証明書で署名することもできます。
- BDR の Etcd の配置 - Always-ON アーキテクチャ ‘harp_consensus_protocol: etcd’を使用する場合、次の各インスタンスのロールに’etcd’を明示的に追加します。*BDR* **プライマリ**’*bdr*’**ロール** BDR 論理スタンバイ’ bdr’ +’readonly’ロール) **ブロンズレイアウトの場合のみ** *BDR Witness* ‘*bdr*’ +’*witness*’**ロール** ゴールドレイアウトの場合のみ Barman’barman’ロールクレジット: Gianni Ciolli gianni.ciolli@enterprisedb .com

マイナー変更

- 製品ブランドの変更を反映するように、configure 引数`--2q` を`--pgextended` に置き換えます。既存の構成は、予想される動作を保持します。
- Debian の古い安定版に付属している docker のバージョン 18 を使用する場合の Docker プラットフォームの互換性チェックのエラーレポートを改善しました。
- 不足しているコマンドを CLI ヘルプドキュメントに追加します。
- configure コマンドのエラーレポートが改善されました。
- ソースから BDR 5 をビルドするための初期サポートを追加します。クレジット: Florin Irion florin.irion@enterprisedb.com
- Postgres の古いバージョンから EDB 製品への移行の継続的な互換性を確保するための変更。

バグ修正

- download-packages コマンドを使用してローカルリポジトリにデータを入力するときに、etcd のパッケージが欠落することを意味する問題を修正しました。
- efm フェールオーバーマネージャーの使用とそのパッケージの依存関係の選択に影響を与える問題を修正しました

2.37.11 TPA 23.1

リリース日 2022-06-21

このリリースでは、アップグレード後に `tpaexec setup` を実行する必要があります、それ以外の場合はエラーで失敗します。

パッケージのインストール動作の変更

以前のバージョンでは、正確なバージョンが明示的に指定されていない限り、`tpaexec deploy` を実行すると、インストールされたパッケージをアップグレードする可能性がありました postgres_package_version 設定など。ただし、これは決して安全、サポート、または推奨されるアップグレード方法ではありません。特に、展開中のパッケージのアップグレード後に、サービスが安全かつ正しく再起動されていない可能性があります。

このリリース以降、`tpaexec deploy` は、インストールされたパッケージをアップグレードすることはありません。最初の展開は、必要なパッケージをすべてインストールします特定のバージョンが設定されている場合、または利用可能な最新のパッケージをインストールします。後続の実行では、パッケージがインストールされたことを確認し、それ以上は何も行いません。これは、予測可能で安全な新しいデフォルト動作です。

コンポーネントを更新する必要がある場合は、`tpaexec update-postgres` を使用します。このリリースでは、コマンドは Postgres および BDR または pglogical などの Postgres 関連パッケージ、および HARP、pgbouncer、etcd などの特定の他のコンポーネントを更新できます特定のクラスターに該当する場合。将来のリリースでは、より多くのコンポーネントのアップグレードを安全にサポートする予定です。

重要な変更

- HARP ブートストラップ構成が変更された場合にのみ「harpctl apply」を実行します 警告これにより、`tpaexec deploy` を実行したときに既存のクラスターで単一の harp サービスがリスタートします。これは、リストが一貫した順序になるように config.yml が変更され、意図しない変更が回避されるためです将来の展開
- `tpaexec download-packages` コマンドを追加して、クラスターに必要なすべてのパッケージを local-repo ディレクトリにダウンロードして、エアギャップ/切断された環境のクラスターインスタンスにコピーできるようにします。詳細については、air-gapped.md および local-repo.md を参照してください。
- 新しい BDR の`--harp-consensus-protocol <etcd|bdr>` 構成オプションが必要です-Always-ON クラスターコンセンサスプロトコルの選択がデータセンター/場所間のネットワークレイテンシーによっては、フェイルオーバーパフォーマンスに悪影響を与える可能性があるため、TPAexec はここでデフォルト値を提供しなくなり、ユーザーの状態が向上します特定のクラスターに最適なプロトコルを選択します。これは、新しく生成されたクラスターの構成に影響しますが、harp_consensus_protocol を明示的に設定せずに以前のデフォルトの `etcd` を使用する既存のクラスターには影響しません。

マイナー変更

- 古いバージョンが利用できないディストリビューションに EFM 用の openjdk-8 の代わりに openjdk-11 をインストールする
- `harp_log_level` 設定たとえば cluster_vars の下を受け入れて、デフォルトの harp-manager および harp-proxy ログレベル info をオーバーライドします
- 復元力を向上させるために、個々のエンドポイント DSN のリストの代わりに単一のマルチホスト BDR DCS エンドポイント DSN を使用するように harp-proxy を構成します
- harp-manager の BDR DCS のローカル Unix ソケット DSN から余分な接続属性たとえば ssl *を省略します

バグ修正

- harp-manager と harp-proxy の構成が変更された場合に、harp-manager と harp-proxy が再起動されていることを確認します
- 読み取り専用の harp_dcs_user に必要な追加の新しい権限を付与することにより、harp プロキシエラーを修正します
- CAMO が有効になっているときに BDR4 トランザクションストリーミングを無効にする bdr.enable_camo が設定されている場合、bdr.default_streaming_mode を無効にする必要があります。これは、BDR4 の CAMO で保護されたトランザクションと互換性がありません。これにより、CAMO 対応 BDR4 クラスターいずれにせよストリーミングが有効になった状態では動作しませんでした。サーバーが再起動します。

第3章　TPA installation

TPA を使用するには、パッケージまたはソースからインストールし、`tpaexec setup` コマンドを実行する必要があります。このドキュメントでは、TPA パッケージをインストールする方法について説明します。EDB サブスクリプションプランを契約しており、EDB リポジトリにアクセスできる場合、これらの手順に従う必要があります。ソースから TPA をインストールするには、*Installing TPA from source* を参照してください。

サポートされているプラットフォームについては、*Distribution support* を参照してください。

> **注釈**
>
> システムの日付と時刻が正しいことを完全に確認してください。そうしないと、さまざまな機能が失敗します。ネットワーク時間、例 *sudo ntpdate pool.ntp.org* を使用することをお勧めします

3.1 クイックスタート

まず、 subscribe to an EDB repository 。

3.1.1 TPA のインストール

```
sudo <your-package-manager> install tpaexec
```

3.1.2 追加の依存関係をインストールする

```
sudo /opt/EDB/TPA/bin/tpaexec setup
```

3.1.3 インストールの検証一般のユーザーとして実行

```
/opt/EDB/TPA/bin/tpaexec selftest
```

各ステップのより詳細な説明を以下に示します。

3.2 TPA をインストールする場所

サポートされているプラットフォームを使用している限り、TPA はワークステーションからインストールして実行できます。これは、学習、ローカルテスト、またはデモンストレーションの目的に適しています。TPA は *Docker のインストール* をサポート

独自のワークステーションで完全な展開を実行したい場合。

運用環境で使用する場合、専用の永続的な仮想マシンで TPA を実行することをお勧めします。これにより、クラスターディレクトリが保持され、将来のクラスター管理または更新にチームが利用できるためです。また、TPA のコピーを 1 つだけ更新する必要があり、単一の TPA ホストからターゲットインスタンスへのネットワークアクセスを提供するだけであることも意味します。

3.3 TPA パッケージのインストール

TPA をインストールするには、最初に subscribe to an EDB repository を行う必要があります。TPA は、すべての EDB リポジトリで使用できます。

次のとおりに TPA をインストールします。

3.3.1 Debian または Ubuntu にインストールする

```
sudo apt-get install tpaexec
```

3.3.2 RHEL、Rocky、AlmaLinux、または Oracle Linux にインストールする

```
sudo dnf install tpaexec
```

3.3.3 SLES

```
sudo zypper install tpaexec
```

これにより、TPA が/opt/EDB/TPA にインストールされます。また、他の必要なパッケージ Python 3.12 以降がインストールされていることも確認します。

ここで sudo について言及しているのは、どのコマンドが root 権限を必要とするかを示すためです。他の手段を使用して、root としてコマンドを実行できます。

3.4 TPA Python 環境のセットアップ

次に、 `tpaexec setup` を実行して、分離された Python 環境を作成し、必要なすべてのモジュールの正しいバージョンをインストールします。

> **注釈**
>
> 20.04 より前の Ubuntu バージョンでは、 *sudo -H tpaexec setup* を使用して、*tpaexec configure* 中の後続の権限エラーを回避してください

```
sudo /opt/EDB/TPA/bin/tpaexec setup
```

`tpaexec setup` は、適切なバージョンの requirements.txt ファイルを自動的に使用して、システムに適切なモジュールをインストールします。

これは/opt/EDB/TPA に書き込むため、root として実行する必要がありますが、プロセスは、インストールしているシステム全体の Python モジュール Ansible を含むには影響しません。

通常 `tpaexec` コマンドを実行するユーザーの `PATH` に/opt/EDB/TPA/bin を追加します。たとえば、これを.bashrc または同等のシェル構成ファイルに追加できます。

```
export PATH=$PATH:/opt/EDB/TPA/bin
```

3.5 インターネットまたはネットワークアクセスエアギャップなしで TPA をインストールする

このセクションでは、EDB リポジトリ、Python パッケージインデックス、またはその両方にアクセスできないサーバーに TPA をインストールする方法について説明します。このような環境で TPA を使用する方法については、*Managing clusters in a disconnected or air-gapped environment* を参照してください。

3.5.1 TPA パッケージのダウンロード

TPA をインストールする必要があるサーバーから EDB リポジトリに直接アクセスできない場合、インターネットに接続されたマシンからパッケージをダウンロードして転送できます。これを行うには、いくつかの方法があります。

インターネットに接続されたマシンがターゲットと同じオペレーティングシステムを使用している場合、`yumdownloader` RHEL ライクまたは `apt download` Debian ライクを使用してパッケージをダウンロードすることをお勧めします。

または、 EDB Repos にアクセスし、見出し**ブラウザから EDB ソフトウェアパッケージをダウンロード** の下で **Enterprise** 、 **Standard** 、または **Community 360** のいずれかを選択することにより、ブラウザから任意のプラットフォーム用パッケージをダウンロードできます。。 TPA をインストールするには、次のパッケージが必要です。

- tpaexec
- tpaexec-deps
- edb-python312

ダウンロードしたパッケージをターゲットサーバーに転送したら、プラットフォームに適切なツールを使用してインストールする必要があります。

3.5.2 Python パッケージインデックスへのアクセスを使用しないインストール

`tpaexec setup` を実行すると、通常は Python パッケージインデックスから Python パッケージをダウンロードします。環境が別のインデックスを提供しない限り、デフォルトは公式の PyPI です。使用可能なパッケージインデックスがない場合は、インストールした `tpaexec` と同じ方法で `tpaexec-deps` パッケージをインストールする必要があります。 `tpaexec-deps` パッケージ tpaexec と同じリポジトリから入手できますは、ダウンロードされるすべてをバンドルしているため、ネットワークアクセスなしでインストール

できます。tpaexec setup を実行する前にパッケージをインストールするだけで、バンドルされたコピーが自動的に使用されます。

3.6 TPA インストールの検証

上記の手順がすべて完了したら、次のコマンドを実行してローカルインストールを確認します。

```
tpaexec selftest
```

そのコマンドがエラーなしで完了した場合、TPA インストールは使用できる状態です。

3.7 TPA のアップグレード

TPA の新しいリリースにアップグレードするには、次のことを行う必要があります。

1. 最新の tpaexec パッケージをインストールする
2. 最新の tpaexec-deps パッケージ必要な場合、上記を参照してください。
3. tpaexec setup を再度実行します

上記のように TPA パッケージリポジトリをサブスクライブしている場合、apt-get update && apt-get upgrade または yum update を実行すると、これらのパッケージの利用可能な最新バージョンをインストールする必要があります。それ以外の場合は、利用可能な任意の手段でパッケージをインストールできます。

tpaexec の新しいバージョンがインストールされるときは、tpaexec setup を再度実行することをお勧めします。一部の新しいリリースはこれを厳密に必要としない場合がありますが、他のリリースはそれなしでは動作しません。

3.8 Ansible バージョン

TPA は、デフォルトで ansible-core 2.16 を使用します。

第4章　Open source TPA

4.1 Trusted Postgres ArchitectTPA とは何ですか?

TPA は、EnterpriseDB (EDB) によって開発されたオーケストレーションツールで、Ansible を使用して EDB の推奨事項に従って Postgres クラスターを展開します。

TPA は、Postgres の展開とサポートに関する長年の苦労して得た経験に基づいて、EDB が追従するベストプラクティスを具体化しています。これらの推奨事項は、運用環境と同様に簡単なテストベッドのセットアップに適用されます。

4.2 次のステップ

- *Installing TPA from source*
- *A First Cluster Deployment*
- TPA's full documentation online

4.3 TPA オープンソースのよくある質問

4.3.1 EDB の顧客でない場合、これを使用できますか?

はい、TPA は、GPLv3 ライセンスの下のオープンソースプロジェクトです。オープンソースソフトウェア、EDB 独自の製品、またはその組み合わせで構成されるクラスターの展開をサポートします。

4.3.2 問題を報告できますか?

はい、EDB をご利用の場合は、サポートにご連絡ください。それ以外の場合は、GitHub Issue を開きます。

4.3.3 貢献できますか?

もちろん、ご意見をお待ちしておりますが、コーディングを開始する前にイシューをオープンしてください。私たちは TPA ができること/すべきことをかなり選択しているため、新機能よりもバグ修正が受け入れられる可能性が高いです。

第5章　Known issues

このページには、TPA の現在のバージョンに影響を与える既知の問題をリストしています。既知の場合、各問題の回避策が提供されます。

5.1 複数のロケーションを持つ EFM クラスターにロケーション間の HBA ルールがありません TPA-1247

5.1.1 詳細

複数の場所で EFM クラスターを展開し、1 つのノードで `efm cluster-status` コマンドを実行すると、`FATAL: password authentication failed for user "efm"` が、ノードと同じ場所にないすべてのノードの **Promote Status** セクションに表示されることがわかります。コマンドが実行されました。

これは、TPA が、EFM ユーザーがある場所のノードから別の場所のノードに接続するために必要な HBA ルールを追加できないために発生します。

5.1.2 回避策

`postgres_hba_settings` クラスター変数を使用して、`efm` ユーザーおよび `replication` ユーザーが任意のノードから他のノードに接続できるように、ノードごとに 1 つの HBA ルールを指定します。例

```
postgres_hba_settings:
  - hostssl all efm 192.168.57.12/32 md5
  - hostssl all efm 192.168.57.13/32 md5
  - hostssl all efm 192.168.58.11/32 md5
  - hostssl all efm 192.168.58.12/32 md5
  - hostssl all efm 192.168.58.13/32 md5

  - hostssl replication replication 192.168.57.12/32 md5
  - hostssl replication replication 192.168.57.13/32 md5
  - hostssl replication replication 192.168.58.11/32 md5
  - hostssl replication replication 192.168.58.12/32 md5
  - hostssl replication replication 192.168.58.13/32 md5
```

5.2 TPA 23.34 以前で展開された PGD クラスターには primary_slot_name 設定があり、bdr_init_physical が動作しない TPA-1229

5.2.1 詳細

TPA 23.35 は、EFM で使用するレプリケーションスロット名が PGD クラスターで `primary_slot_name` として誤って設定される問題を解決しました。 TPA 23.35 以降で展開されたクラスターには、この問題は発生しません。ただし、以前のバージョンで展開された既存のクラスターは、新しい TPA バージョンで展開が実行された後でも、誤った `primary_slot_name` 設定を保持します。

この不正な `primary_slot_name` の影響は、bdr_init_physical を使用してクラスターに新しいノードを追加またはノードをリビルドしようとすると、`Waiting for PostgreSQL to accept connections...` でハングし、ログに致命的エラー `replication slot ... does not exist` が表示されることです。

5.2.2 回避策

ファイル`/opt/postgres/data/conf.d/8901-primary_slot_name.conf` を削除し、Postgres をリロードすることにより、クラスター内の各 Postgres ノードからこの誤った設定を手動で削除できます。TPA 23.35 以降を使用している限り、復元されることはありません。

5.3 forward_ssh_agent: true を使用したクラスターでの SSH 構成テンプレートエラー

5.3.1 詳細

TPA 23.43 は、クラスターディレクトリの ssh_config ファイルのテンプレートを変更し、`forward_ssh_agent` が True に設定されている場合、生成された構成ファイルが無効になるバグを導入しました。この設定のデフォルトは False で、通常はテスト用のソースコードからクラスターのコンポーネントをインストールする場合にのみ True に設定されます。

5.3.2 回避策

`tpaexec provision` を実行して ssh_config ファイルを生成した後、テキストエディタでファイルを編集します。`IdentitiesOnly yes ForwardAgent yes` を読み取る行、通常はファイルの 7 行目を見つけ、最初の「yes」の後に改行を追加して 2 行に分割します。ファイルは `tpaexec provision` によって上書きされるため、プロビジョニング後に config.yml に変更を行った場合に再編集する必要があります。

5.4 pemagent_registration_opts が提供する PEM エージェント登録オプションは agent.cfg に反映されないため、有効になりません TPA-1376

5.4.1 詳細

`pemagent_registration_opts` オプションは、PEM エージェントの登録中に追加のコマンドラインオプションを pemworker ユーティリティに渡すために使用されます。これらのオプションの一部は登録の即時

行為に影響を与えるもの、一部は PEM データベースに書き込まれるデータに影響を与えるもの、その他はエージェントの動作を決定するために使用される生成された agent.cfg ファイルの内容に影響を与えるものです。 TPA の現在の実装では、agent.cfg ファイルはテンプレートバージョンで登録された直後に上書きされます。つまり、pemagent_registration_opts の結果としてこのファイルに加えられた変更は失われます。

5.4.2 回避策

指定したオプションが agent.cfg と相互作用しないことが確認できる場合を除き、pemagent_registration_opts の使用は避けてください。デプロイ後フックを使用して、必要な変更を agent.cfg に適用します。

5.5 TPA を使用して展開すると、PEM 10.4.x Web アプリケーションに「内部サーバーエラー」が表示される TPA-1415

5.5.1 詳細

PEM 10.4 は、ほとんどのオペレーティングシステムで以前の PEM バージョンより新しい Python バージョンを使用します。TPA は PEM のパッケージ依存関係であるため、必要な Python バージョンをインストールしますが、以前のバージョンの mod_wsgi モジュールをインストールするため、Web アプリケーションを実行できません。

5.5.2 回避策

最も完全な回避策は、必要な mod_wsgi パッケージを PEM サーバーインスタンスに明示的に追加することです。必要なパッケージは

PEM documentation で見つかります。 RHEL 9 の例

```
instances:
...

- name: pem-server
  ...
  packages:
    RedHat:
    - edb-python312-mod-wsgi
```

第6章　Installing TPA from source

このドキュメントでは、ソースコードリポジトリのコピーから TPA を使用する方法について説明します。

> **注釈**
>
> EDB 顧客がソフトウェアの EDB サポートを受けるには、 *TPA installation* 必要があります。

ソースから TPA を実行するには、パッケージが処理するすべての依存関係たとえば Python 3.12+をインストールするか、ソースと *Running TPA in a Docker container* をダウンロードする必要があります。(どちらの方法も Linux および macOS では正常に動作します。)

6.1 前提条件

ソースから TPA をインストールする前に、システムで `git` が利用できることを確認してください。

`pip` および venv モジュールを含む **Python 3.12** インタープリターが TPA コントローラホストに存在する必要があります。

- **RHEL 9** の場合、CodeReady Linux BuilderCRB リポジトリを有効にして、必要なビルド依存関係をインストールする必要がある場合があります。
- **SLES** の場合、Python 3.12 モジュールが有効になっていることを確認します。
- オペレーティングシステムリポジトリが Python 3.12 を提供していない場合は、pyenv を使用してインストールすることをお勧めします。

> **注釈**
>
> *openvpn* や `patch` などの古い依存関係は、標準の TPA インストールでは必要なくなり、これらの手順から削除されました。

6.2 クローンとセットアップ

前提条件がインストールされると、リポジトリのクローンを作成できます。

```
git clone https://github.com/enterprisedb/tpa.git ~/tpa
```

これにより、ホームディレクトリに `tpa` ディレクトリが作成されます。

ssh を使用してチェックアウトする場合は、次を使用します。

```
git clone ssh://git@github.com/EnterpriseDB/tpa.git ~/tpa
```

新しく作成したクローン内にある bin ディレクトリを、次のパスに追加します。

```
export PATH=$PATH:$HOME/tpa/bin
```

この行を.bashrc ファイルまたは使用するシェルの他のプロファイルファイルに追加します。

次を実行して、作業用の tpa 環境を作成できるようになりました。

```
tpaexec setup
```

これにより、TPA が将来的に使用する Python 仮想環境が作成されます。必要なパッケージはすべてこの環境にインストールされます。これが正しく構成されたことをテストするには、次を実行します。

```
tpaexec selftest
```

これで、tpaexec がインストールされました。

6.3 依存関係

6.3.1 Python 3.12+

TPA には、ほとんどの最新のディストリビューションで利用可能な Python 3.12 以降が必要です。お持ちでない場合は、pyenv を使用して、システムパッケージに影響を与えることなく、希望のバージョンの Python をインストールできます。

```
#  First, install pyenv and activate it in ~/.bashrc

#  See https://github.com/pyenv/pyenv#installation

#  (e.g., `brew install pyenv` on MacOS X)

pyenv install 3.12.0
```

```
Downloading Python-3.12.0.tar.xz...
- > https://www.python.org/ftp/python/3.12.0/Python-3.12.0.tar.xz
Installing Python-3.12.0...
Installed Python-3.12.0 to /home/ams/.pyenv/versions/3.12.0
```

```
pyenv local 3.12.0
pyenv version
```

```
3. 12.0 (set by /home/ams/pyenv/.python-version)
```

```
pyenv which python3
```

```
/home/ams/.pyenv/versions/3.12.0/bin/python3
```

```
python3 --version
```

```
3. 12.0
```

pyenv をまだ使用していない場合は、 pyenv documentation の説明に従って、pyenv を.bashrc の PATH に追加し、eval "$(pyenv init -)" を呼び出してください。

6.3.2 仮想環境オプション

デフォルトでは、tpaexec setup はビルトイン Python 3 -m venv を使用して$TPA_DIR/tpa-venv の下に venv を作成し、 tpaexec が呼び出されるたびに自動的にアクティブにします。

tpaexec setup --venv /other/location を実行して、新しい venv の別の場所を指定できます。

デフォルトの venv の場所を使用することを強くお勧めします。別の場所を使用する場合は、環境変数 TPA_VENV をその場所に設定する必要があります。たとえば、次の行を.bashrc または他のシェル起動スクリプトに追加します。

```
export TPA_VENV="/other/location"
```

第7章　CREATING A CLUSTER

7.1 A First Cluster Deployment

この短いチュートリアルでは、単純な *M1* デプロイメントをローカル Docker インストールに展開する手順を説明します。最後までに、4 つのコンテナ、1 つのプライマリデータベース、2 つのレプリカ、およびバックアップノードが構成され、探索できる準備が整います。

この例では、Ubuntu システムで TPA を実行しますが、ほとんどの Linux システムで考慮事項は同様です。

7.1.1 TPA のインストール

あなたが EDB 顧客の場合、EDB のリポジトリから TPA パッケージを直接インストールする *TPA installation* に従ってください。

TPA のオープンソースユーザーであれば、Github.com からダウンロードできる *Installing TPA from source* があります。

これらのガイドに従って、ここに戻ります。

7.1.2 Docker のインストール

前述のように、展開例を Docker に展開します。まだ Docker をインストールしていない限り、それをセットアップする必要があります。

Debian または Ubuntu では、次のコマンドを実行して Docker をインストールします。

```
sudo apt update
sudo apt install docker.io
```

他の Linux ディストリビューションについては、Docker Engine Install page を参照してください。

次を使用してユーザーを docker グループに追加します。

```
sudo usermod -aG docker <yourusername>
newgrp docker
```

> **警告**
>
> ユーザーに Docker デーモンと対話できる機能を付与すると、Docker ホストでの root を簡単に取得できます。信頼できるユーザーのみが Docker デーモンにアクセスできる必要があります。

7.1.3 TPA を使用した構成の作成

このプロセスの次のステップは、構成を作成することです。 TPA は、configure コマンドを介して作業のほとんどを実行します。あなたがしなければならないのは、コマンドラインフラグとオプションを指定して、展開するものを大まかに選択することです。tpaexec configure コマンドは次のとおりです。

```
tpaexec configure demo --architecture M1 --platform docker --postgresql 15 --enable-
↪repmgr --no-git
```

これにより、 *M1* を含む demo という構成が作成されます。したがって、プライマリ、レプリカ、バックアップノードがあります。

--platform docker は、この構成をローカルの Docker インスタンスで作成する必要があることを TPA に指示します。すべてのコンテナと OS 要件をプロビジョニングします。他のプラットフォームには、アマゾンウェブサービスと同じことを行う *awscli* を使用してボリューム属性を変更する と *bare(-metal servers)* が含まれます。オペレーティングシステムのプロビジョニングをスキップし、既に構成された Linux ホストへのソフトウェアのインストールに直接進みます。

--postgresql 15 では、Community Postgres バージョン 15 を使用するように TPA に指示します。ソフトウェアの選択に関していくつかのオプションがありますが、これはオープンソースユーザーにとって最も簡単なデフォルトです。

--enable-repmgr を追加すると、TPA に Replication Manager を使用してレプリケーションとフェイルオーバーをハンドリングするように展開を構成するように指示します。

最後に、--no-git は、git を介して構成をリビジョン制御できる TPA の機能をオフにします。

このコマンドを実行すると、どうやらコマンドラインでは何も起こりません。ただし、demo というディレクトリが作成され、新しい展開のブループリントである config.yml ファイルを含むいくつかのファイルが含まれています。

デプロイメントのプロビジョニング

これで、新しい展開を実行するコンテナまたは仮想マシンを作成する準備が整いました。これは、provision コマンドで実現できます。実行します

```
tpaexec provision demo
```

構成の展開の準備に必要なさまざまな操作を介した TPA の動作が表示されます。

Deploying

プロビジョニングが完了したら、展開に進むことができます。これにより、必要に応じて、オペレーティングシステムとシステムパッケージがインストールされます。次に、要求された Postgres アーキテクチャをインストールし、必要な構成をすべて実行します。

```
tpaexec deploy demo
```

構成を展開するために必要なさまざまな操作を介した TPA の動作が表示されます。

テスト

新しいデータベースで pgbench を実行する tpaexec `test` コマンドを使用して、新しく展開された構成をすばやくテストできます。

```
tpaexec test demo
```

接続しています

psql プロンプトを表示するには、最も簡単なルートは、docker または SSH を使用して、コンテナまたは構成に応じて VM またはホストのいずれかにログインすることです。走る

```
tpaexec ping demo
```

展開内のすべての接続可能なホストに ping を実行すると、次のような出力が得られます。

```
unfair | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
uptake | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
quondam | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
uptight | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

SUCCESS で応答したノードの 1 つを選択します。この例では、 `uptake` を使用します。

docker のみを使用する予定がある場合は、`docker exec -it uptake /bin/bash` コマンドを使用して、適切なホスト名に置き換えます。

すべての種類の TPA 展開で機能する別のオプションは、SSH を使用することです。これを行うには、最初に現在のディレクトリを作成した構成ディレクトリに変更します。

たとえば、構成の名前付けは demo なので、そのディレクトリに移動します。そこで、`ssh -F ssh_config ourhostname` を実行して接続します。

```
cd demo
ssh -F ssh_config uptake
```

```
Last login: Wed Sep  6 10:08:01 2023 from 172.17.0.1
[root@uptake ~]#
```

どちらの場合も、コンテナに root ユーザーとしてログインします。

sudo -iu postgres を使用して、ユーザーを postgres ユーザーに変更できるようになりました。 postgres として、 psql を実行できます。 TPA はそのユーザーを.pgpass ファイルで既に構成しているため、パスワードを提示する必要はありません。

```
sudo -iu postgres
psql
```

```
[root@uptake ~]# sudo -iu postgres
postgres@uptake:~ $ psql
psql (15.4)
Type "help" for help.

postgres=#
```

そして、データベースに接続しています。

SSH を使用せずにホストシステムからコンテナに接続できます。 ssh_config ファイルから接続するホストの IP アドレスを取得します。

```
grep "^ *Host" demo/ssh_config
```

```
Host *
Host uptight
    HostName 172.17.0.9
Host unfair
    HostName 172.17.0.4
Host quondam
    HostName 172.17.0.10
Host uptake
    HostName 172.17.0.11
```

Uptake に接続するので、IP アドレスは 172.17.0.11 です。

また、 postgres ユーザーのパスワードも取得する必要があります。tpaexec show-password demo postgres を実行して、システムから保存されたパスワードを取得します。

```
tpaexec show-password demo postgres
```

```
a9LmI1X^uMOpPoEnLuRdL%L$oRQak3om
```

Postgresql クライアントがインストールされていると仮定すると、次のことを実行できます。

```
psql --host 172.17.0.11 -U postgres
```

```
Password for user postgres:
```

前に取得したパスワードを入力します。

```
psql (14.9 (Ubuntu 14.9-0ubuntu0.22.04.1), server 15.4)
WARNING: psql major version 14, server major version 15.
         Some psql features might not work.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256,␣
↪compression: off)
Type "help" for help.

postgres=#
```

これで、Docker ホストから、TPA 展開された Docker コンテナのいずれかで実行されている Postgres に接続されました。

7.2 Cluster configuration

`tpaexec configure` コマンドは、プロビジョニング/展開/テストサイクルの後続のステージで必要な YAML クラスター構成ファイルを生成します。

7.2.1 クイックスタート

```
tpaexec configure ~/clusters/speedy --architecture M1 \
        --postgresql 14 \
        --failover-manager repmgr
```

このコマンドは、`~/clusters/speedy` という名前のディレクトリを作成し、M1 という名前のアーキテクチャのレイアウトに従う `config.yml` という名前の構成ファイルを生成します。シングルプライマリ、N レプリカ。新しいディレクトリに git リポジトリを作成し、生成された `config.yml` を含む初期コミットを作成します。

また、コマンドは、構成を変更するためのさまざまなオプション選択したアーキテクチャまたはプラットフォームに固有の一部を受け入れますが、デフォルトは賢明で、すぐに使用できるように設計されています。生成された config.yml を読み、ニーズに合わせて構成を微調整することをお勧めします。(これが *Instance configuration* の概要です。)

config.yml を完全に手動で書くことも可能ですが、生成されたファイルを編集するほうが簡単です。

7.2.2 構成オプション

最初の引数は、クラスターディレクトリ、たとえば、`speedy` または`~/clusters/speedy` である必要があります。クラスターの名前は、どちらの場合も speedy になります。すべてのクラスターを共通のディレクトリたとえば、上記の例では`~/clusters` に保持することをお勧めします。

アーキテクチャ、たとえば *M1* 、 *BDR-Always-ON* 、または *PGD-X* を選択するには、次の引数が`--architecture <name>` である必要があります。アーキテクチャの完全なリストについては、`tpaexec info architectures` を実行してください。

次に、インストールする *Postgres のフレーバーとバージョン* を指定する必要があります。

上記の引数は常に必須です。上記の例のように、ここで説明するオプションの残りの部分は安全に省略できます。デフォルトでは使用可能な結果が得られます。

共通オプションのリストについては、 `tpaexec help configure-options` を実行します。

アーキテクチャ固有のオプション

選択したアーキテクチャにより、他のどのオプションが受け入れられるかが決まります。通常、各アーキテクチャは、いくつかの一意のオプションと以下で説明する汎用オプションを受け入れます。

たとえば、M1 では `--location-names l1 l2` を使用して、2 つの名前付けの場所にノードを含むクラスターを作成できます。アーキテクチャが受け入れるオプションのリストについては、アーキテクチャのドキュメントを参照してください。または、場合によっては必要となります。

プラットフォームオプション

次に、 `--platform <name>` を使用してプラットフォーム、たとえば *awscli を使用してボリューム属性を変更する* または *bare(-metal servers)* を選択できます。

アーキテクチャは特定のプラットフォームをサポートする場合と、サポートしない場合があります。そうしない場合、クラスターの構成に失敗します。

プラットフォームの選択は、特定のオプションの解釈に影響を与えます。たとえば、aws を選択した場合、`--region <region>` および`--instance-type <type>` への引数は有効な AWS region name である必要があります

と EC2 instance type

それぞれ。詳細については、プラットフォームのドキュメントを参照してください。

プラットフォームを明示的に選択しない場合、現在デフォルトは aws です。

注 TPA は、さまざまなプラットフォーム上のインスタンスを使用したクラスターの作成を完全にサポートしていますが、`tpaexec configure` は現在このような構成を生成できません。config.yml を編集して、複数のプラットフォームを指定する必要があります。

所有者

`--owner <name>` を指定して、クラスターを何らかのプラットフォーム固有の手段 AWS タグなどで関連付けるには、クラスターに責任を負う人の名前を関連付けます。これは、クラウドプラットフォームにとって特に重要です。デフォルトでは、所有者は `tpaexec provision` を実行しているユーザーのログイン名に設定されます。

あなたのイニシャル、または「名、姓」、またはあなたを一意に識別するその他のものを使用できます。

リージョン

`--region <region>` を指定してリージョンを選択します。

このオプションは、クラウドプラットフォームの場合にのみ意味があります。AWS のデフォルトは eu-west-1 です。

注 TPA は、マルチプルのリージョンにまたがるクラスターの作成を完全にサポートしていますが、`tpaexec configure` は現在このような構成を生成できません。config.yml を編集して、複数のリージョンを指定する必要があります。

ネットワーク構成

> **注釈**
>
> これらのオプションは、TPA が既存のサーバーのネットワーク構成を変更しない「ベア」プラットフォームでは意味がありません。

`--network 192.168.0.0/16` を指定して、別のネットワークからサブネットを割り当てます。AWS クラスターでは、これは VPC CIDR に対応します。詳細については、*awscli を使用してボリューム属性を変更する* ドキュメントを参照してください。

`--subnet-prefix 26` を指定して、別のサイズのサブネット、この場合は/28 ではなく/26 のサブネットを割り当てます。

> **注釈**
>
> "docker"プラットフォームが選択されている場合、TPA はアーキテクチャに関係なく、常にクラスター全体を単一のサブネットに配置します。このサブネットは、ここで説明するロジックに従って生成されますが、 *subnet-prefix* が指定されない場合、TPA は `config.yaml` のインスタンス数に対応するのに十分な大きさのサブネットサイズを自動的に選択します。

`--exclude-subnets-from <directory>` を指定して、既存のクラスター config.yml ファイルで既に使用されているサブネットを除外します。この引数は、ディレクトリごとに複数回指定できます。

インスタンスタイプ

`--instance-type <type>` を指定してインスタンスタイプを選択します。

このオプションは、クラウドプラットフォームの場合にのみ意味があります。AWS のデフォルトは t3.micro です。

ディスク領域

`--root-volume-size 64` を指定して、ルートボリュームのサイズを GB 単位で設定します。プラットフォームによっては、ルートボリュームに必要な最小サイズが存在する場合があります。

`--postgres-volume-size <size>` および`--barman-volume-size <size>` オプションは、Postgres と Barman の個別のボリュームをサポートするアーキテクチャとプラットフォームで、Postgres および Barman ボリュームのサイズを設定するために使用できます。

これらのオプションは、TPA がボリュームサイズを制御できない「ベア」プラットフォームでは意味がありません。

ホスト名

デフォルトでは、`tpaexec configure` は、数十の名前の事前承認リストから必要な数のホスト名をランダムに選択します。ほとんどのクラスターではこれで十分です。

`--hostnames-from <filename>` を指定して、1 行に 1 つの名前を含むファイルからホスト名を選択します。ファイルには、クラスター内のインスタンスと同じ数以上の有効なホスト名が含まれている必要があります。各行には、名前の後にオプションの IP アドレスを含めることができます。存在する場合、このアドレスは `config.yml` の対応するインスタンスの `ip_address` として設定されます。2 つの IP アドレスが存在する場合、1 つ目は `public_ip` として設定され、2 つ目は `private_ip` として設定されます。

> **注釈**
>
> ホスト名ファイルを介してか`config.yml` で直接 IP アドレスを明示的に指定する場合、それらがクラスターネットワークの CIDR 範囲内にあることを確認する必要があります。 Docker および AWS プラットフォームの場合、これは必要があることを意味します ネットワーク構成

IP アドレスを指定する場合。

クラスターネットワーク CIDR レンジを指定しない場合、TPA は、指定された IP アドレスを参照せずにランダムに 1 つを選択します。これにより、展開が失敗するか、予想したものとは異なる IP アドレスのクラスターが発生します。

`--hostnames-sorted-by="--dictionary-order"` を使用して、`--random-sort` デフォルト以外の sort(1) オプションを選択します。

`--hostnames-unsorted` を使用して、ホスト名をまったくソートしません。この場合、ホスト名ファイルで見つかった順序で割り当てられます。これは、ホスト名ファイルが明示的に指定された場合のデフォルトです。

`--cluster-prefixed-hostnames` を使用して、各ホスト名がクラスターの名前で始まるようにします。これは、同じホストで複数の Docker クラスターを実行している場合、ホスト名の衝突を回避するのに役立ちます。

ホスト名には、文字 a〜z、数字 0〜9、および’-’のみを含めることができます。選択したプラットフォームによっては、FQDN である場合があります。ホスト名は小文字である必要があります。大文字は内部で小文字に変換され、config.yml たとえば `upstream: hostname` でのこれらのホスト名への参照は小文字バージョンを使用する必要があります。

7.2.3 ソフトウェアの選択

ディストリビューション

`--distribution <name>` を指定してディストリビューションを選択します。

選択したプラットフォームにより、どのディストリビューションが利用可能か、およびデフォルトで使用されるディストリビューションが決まります。

一般に、「Debian」、「RedHat」、「Ubuntu」、および「SLES」を使用して、適切なイメージを選択できる必要があります。

オプションで、--os-version <version number> を渡すことにより、ディストリビューションのバージョンを含めることができます。

このオプションは、TPA がどのディストリビューションをインストールするかを制御できない「ベア」プラットフォームでは意味がありません。

EDB リポジトリ

TPA は、サブスクリプションを介してアクセスしている EDB ソフトウェアリポジトリを有効にできます。デフォルトでは、TPA はアーキテクチャが必要とする製品リポジトリをインストールします。

TPA が EDB リポジトリを使用する方法の詳細な説明は、*Configuring EDB Repos 2.0 repositories* および各アーキテクチャのページで利用できます。

--edb-repositories repository …を指定して、各インスタンスにインストールする EDB リポジトリの完全なリストを指定します。

このオプションは注意して使用してください。TPA は、組み合わせが適切であることを確認せずに名前付けリポジトリを構成します。

このオプションを使用するには、TPA を実行する前に export EDB_SUBSCRIPTION_TOKEN=xxx を行う必要があります。EDB トークンは、enterprisedb.com/repos から取得できます。

ローカルリポジトリのサポート

--enable-local-repo を使用して、パッケージをターゲットインスタンスにシップするローカルパッケージリポジトリを作成します。

ネットワークアクセスが制限されている環境では、代わりに--use-local-repo-only を使用してローカルリポジトリを作成し、ターゲットインスタンスで他のすべてのパッケージリポジトリを無効にして、パッケージがローカルリポジトリからのみインストールされるようにできます。

Creating and using a local repository に関するページに詳細が記載されています。

ソフトウェアバージョン

Postgres のフレーバーとバージョン

TPA は、 PostgreSQL、 EDB Postgres Extended、および EDB Postgres Advanced Server バージョン 13〜18 をサポートしています。

インストールする Postgres のフレーバーまたはディストリビューションとメジャーバージョンの両方を指定する必要があります。たとえば、

- --postgresql 14 は PostgreSQL 14 をインストールします
- --edb-postgres-extended 15 は EDB Postgres Extended 15 をインストールします
- --edb-postgres-advanced 15 --redwood は、EPAS 15 を「Redwood」モードでインストールします
- --edb-postgres-advanced 15 --no-redwood は、非 Redwood モードで EPAS 15 をインストールします

EPAS をインストールする場合、--redwood または--no-redwood モードで動作するかどうか、つまり Oracle 互換性機能を有効にするか無効にするかを指定する必要があります。

EDB Postgres Extended または Postgres Advanced Server のインストールには、有効な *Configuring EDB Repos 2.0 repositories* が必要です。

パッケージバージョン

デフォルトでは、常にすべてのパッケージの最新バージョンをインストールします。これは通常、望ましい動作ですが、一部のテストシナリオでは、次のオプションのいずれかを使用して特定のパッケージバージョンを選択する必要がある場合があります。

1. --postgres-package-version 10.4-2.pgdg90+1
2. --repmgr-package-version 4.0.5-1.pgdg90+1
3. --barman-package-version 2.4-1.pgdg90+1
4. --pglogical-package-version '2.2.0*'
5. --bdr-package-version '3.0.2*'
6. --pgbouncer-package-version '1.8*'
7. --beacon-agent-package-version 1.56.2-1
8. --etcd-package-version 9.8.0-1.el8
9. --patroni-package-version 4.0.0-1PGDG.rhel8 10. --pem-server-package-version 9.7.0-1.el9 11. --pem-agent-package-version 9.7.0-1.el9 12. --pg-backup-api-package-version 2.0.0-1.el8 13. --pgd-proxy-package-version 5.0.0-1 14. --pgdcli-package-version 5.6.1

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

--extra-packages p1 p2 …または--extra-postgres-packages p1 p2 …を指定して、追加パッケージをインストールすることもできます。前者はシステムパッケージとともにインストールするパッケージをリストし、後者は postgres パッケージとともに後でインストールするパッケージをリストします。前者のリストで Postgres に依存するパッケージを記載すると、Postgres がまだインストールされていないため、インストールは失敗します。引数は変更せずにインストールのためにパッケージマネージャーに渡されます。

--extra-optional-packages p1 p2 … オプションは--extra-packages と同様に動作しますが、名前付けたパッケージをインストールできない場合、これはエラーではありません。

ワイルドカードの使用に関する既知の問題

config.yml に永続的に追加する場合、*_package_version でワイルドカードを使用すると、RHEL 8 以降のノードまたは Rocky Linux などの dnf を使用する派生 OS で tpaexec deploy の実行中に、インストールされたソフトウェアの予期しない更新が発生する可能性があることに注意してください。既にパッケージがインストールされている既存のクラスターで deploy が実行されている場合、ansible は完全なパッケージバージョンと一致しない場合があります。たとえば、bdr_package_version の値が 3.6* に設定されて

いる場合、ansible はこれを PGD のインストールされたバージョンと照合できず、パッケージがインストールされていないと仮定し、次のパッケージの利用可能な最新バージョンをインストールしようとします。構成済みのリポジトリ内の同じ名前、例 3.7.

私たちはこの制限を ansible dnf モジュールのバグとして認識しており、TPA の将来のリリースでこれに対処したいと考えています。

ソースからのビルドとインストール

`--install-from-source postgres` を指定すると、Postgres はパッケージからインストールされるのではなく、git リポジトリからビルドおよびインストールされます。デフォルトでは、これは適切な `REL_nnn_STABLE` ブランチをビルドします。

`--install-from-source postgres bdr5` を使用してソースから両方のコンポーネントをビルドおよびインストールするか、`--install-from-source bdr5` を使用して Postgres のパッケージを使用できますが、ソースから PGD v5 をビルドおよびインストールします。デフォルトでは、これは PGD の `main` ブランチをビルドします。

別のブランチをビルドするには、対応する引数に `:branchname` を追加します。例、`--install-from-source 2ndqpostgres:dev/xxx` 、または `pglogical:bug/nnnn` 。

代わりにソースインストールに置き換えることを選択したパッケージに依存するパッケージをインストールできない場合があります。たとえば、PGD v3 パッケージは pglogical v3 パッケージに依存しているため、ソースリポジトリから pglogical をインストールし、パッケージから PGD をインストールすることはできません。同様に、ソースから Postgres をインストールし、パッケージから pglogical をインストールすることはできません。

7.2.4 オーバーライド

オプションで`--overrides-from a.yml …`を指定して、生成された config.yml にマージする設定を含む 1 つ以上の YAML ファイルをロードできます。

ここで指定されたファイルは、最初に Jinja2 テンプレートとして展開され、結果は YAML データ構造としてロードされ、config.yml アーキテクチャとプラットフォームのデフォルトとコマンドラインからの引数で構成される引数に再帰的にマージされます。このプロセスは、指定された追加オーバーライドファイルごとに繰り返されます。これは、1 つのファイルで定義された設定が後続のファイルに可視されることを意味します。

たとえば、オーバーライドファイルには次のものが含まれる場合があります。

```
cluster_tags:
  some_tag: "{{ lookup(env, SOME_ENV_VAR) }}"

cluster_vars:
  synchronous_commit: remote_write
  postgres_conf_settings:
    effective_cache_size: 4GB
```

これらの設定は、config.yml にある `cluster_tags` および `cluster_vars` を強化します。設定は再帰的にマージされるため、`cluster_tags` には、デフォルトの Owner タグと `some_tag` の両方が含まれることになりま

す。同様に、effective_cache_size 設定はその変数をオーバーライドし、他の postgres_conf_settings 存在する場合は影響を受けません。つまり、config.yml で特定のサブキーを設定またはオーバーライドすることはできますが、cluster_tags またはその他のハッシュを完全に空にしたり置き換えたりすることはできません。

マージはハッシュ構造にのみ適用されるため、このメカニズムを使用して config.yml 内の instances のリストを変更することはできません。環境によくある設定で cluster_vars および instance_defaults を強化するのが最も役立ちます。

とはいえ、このメカニズムはいかなる制限も強制するものではないため、十分な注意を払ってください。オーバーライドを含む場合とない場合の 2 つの構成を生成し、2 つの config.yml ファイルを比較して、すべてのオーバーライドの効果を理解することをお勧めします。

7.2.5 Ansible Tower

--use-ansible-tower および--tower-git-repository オプションを使用して、Ansible Tower での展開に適応したクラスターを作成します。詳細は、*TPA and Ansible Tower/Ansible Automation Platform* を参照してください。

7.2.6 PGAI エージェント

--enable-beacon-agent および--beacon-agent-project-id オプションを使用して、PGAI エージェント beacon-agent としてパッケージ化されます。これにより、 EDB Postgres AI コンソールでクラスターを表示できます。詳細は、*Configuring the PGAI agent* を参照してください。

7.2.7 Git リポジトリ

デフォルトでは、クラスターにちなんで名前付けられた初期ブランチで git リポジトリが作成され、コミットメッセージで使用した configure オプションを使用して単一のコミットが作成されます。$PATH に git がない場合、tpaexec はエラーを発生させませんが、リポジトリは作成されません。git リポジトリの作成を抑制するには、 --no-git オプションを使用します。 Ansible Tower クラスターでは、git リポジトリが必要で、tpaexec provision によってまだ存在しない場合は後で作成されることに注意してください。

7.2.8 ボールトパスワードのキーリングバックエンド

TPA は、クラスター固有の Ansible vault パスワードを生成します。このパスワードは、クラスター用に生成された他の機密変数 Postgres ユーザーパスワード、Barman ユーザーパスワードなどを暗号化するために使用されます。

キーリングバックエンド system は、gnome-keyring や secret-tool を含む Python キーリングモジュールでサポートされているバックエンドのリストから、システムで最適なキーリングバックエンドを活用します。

デフォルトでは、新しいクラスターの system キーリングを使用してボールトパスワードを保存します。provision の**前**、config.yml ファイルの keyring_backend: system を削除すると、以前のデフォルトに戻してボールトのパスワードをプレーンテキストファイルに保存します。

keyring_backend: system を使用すると、ボールトパスワードの一意のストレージ名を保存するために使用される config.yml に vault_name エントリーも生成されます。TPA はデフォルトで UUID を生成しますが、命名スキーム要件はありません。

キーリングバックエンドは、--keyring-backend オプションを指定することにより、tpaexec configure 中に指定できます。許可される値は legacy および system です。

> **注釈**
>
> 同じ`cluster_name`を使用する複数のクラスターで`keyring_backend: system`と同じベースの config.yml ファイルを使用する場合、構成ファイルを別の場所にコピーすることにより、値ペア *vault_name* 、*cluster_name* がクラスターコピーごとに一意であることを確認します。

--compliance stig または--compliance cis オプションを使用して、STIG または CIS 標準に準拠するのに適した構成でクラスターを生成します。詳細は、*Compliance* を参照してください。これらのオプションは、クラスターが関連する標準を満たすことを保証するものではないことに注意してください。これらは、TPA によって制御可能な標準のこれらの側面に準拠するように設計された構成を TPA に生成するだけです。

7.2.9 例

次のコマンドを実行するとどうなるか見てみましょう。

```
tpaexec configure ~/clusters/speedy --architecture M1 \
        --distribution Debian \
        --os-version 12 \
        --platform aws --region us-east-1 --network 10.33.0.0/16 \
        --instance-type t2.medium --root-volume-size 32 \
        --postgres-volume-size 64 --barman-volume-size 128 \
        --postgresql 14 \
        --failover-manager repmgr
```

出力はないため、エラーはありません。クラスターディレクトリが作成され、データが設定されました。

```
ls -lh ~/clusters/speedy/
```

```
total 8.0K
drwxrwxr-x 2 haroon haroon 4.0K Aug 17 02:33 commands
- rw-rw-r-- 1 haroon haroon 1.5K Aug 17 02:33 config.yml
lrwxrwxrwx 1 haroon haroon   53 Aug 17 02:33 deploy.yml -> /home/haroon/tpa/
↪architectures/M1/deploy.yml
```

クラスター構成は config.yml にあり、その近傍は直接対話する必要のないアーキテクチャ固有のサポートファイルへのリンクです。構成は次のとおりです。

```
- --
architecture: M1
cluster_name: speedy
cluster_tags: {}
```

(次のページに続く)

(前のページからの続き)

```
keyring_backend: system
vault_name: cfae3da3-ec00-46cd-ab05-e153f1c788db

cluster_rules:
  - cidr_ip: 0.0.0.0/0
    from_port: 22
    proto: tcp
    to_port: 22
  - cidr_ip: 10.33.120.80/28
    from_port: 0
    proto: tcp
    to_port: 65535
ec2_ami:
  Name: debian-11-amd64-20240104-1616
  Owner: "136693071363"
ec2_instance_reachability: public
ec2_vpc:
  us-east-1:
    Name: Test
    cidr: 10.33.0.0/16

cluster_vars:
  edb_repositories: []
  failover_manager: repmgr
  postgres_flavour: postgresql
  postgres_version: "14"
  preferred_python_version: python3

locations:
  - Name: main
    az: us-east-1a
    region: us-east-1
    subnet: 10.33.120.80/28

instance_defaults:
  default_volumes:
    - device_name: root
      encrypted: true
      volume_size: 32
      volume_type: gp2
    - device_name: /dev/sdf
      encrypted: true
      vars:
        volume_for: postgres_data
      volume_size: 64
```

(次のページに続く)

(前のページからの続き)

```
      volume_type: gp2
  platform: aws
  type: t2.medium
  vars:
    ansible_user: admin

instances:
  - Name: uproar
    backup: kinsman
    location: main
    node: 1
    role:
      - primary
  - Name: unravel
    location: main
    node: 2
    role:
      - replica
    upstream: uproar
  - Name: kinsman
    location: main
    node: 3
    role:
      - barman
      - log-server
      - witness
    upstream: uproar
    volumes:
      - device_name: /dev/sdf
        encrypted: true
        vars:
          volume_for: barman_data
        volume_size: 128
        volume_type: gp2
```

次のステップは、 *tpaexec provision* を実行することです

または、 **クラスター構成** の構成をカスタマイズする方法または *Instance configuration* の構成方法の詳細を参照してください。

7.3 tpaexec provision

プロビジョニングは、クラスターに必要なインスタンスとその他のリソースを作成します。

このプロセスの正確な詳細は、クラスターの構成中に選択したアーキテクチャたとえば *M1* とプラットフォームたとえば *awscli* **を使用してボリューム属性を変更する**の両方に依存します。

プロビジョニングステージの最後に、基本的なオペレーティングシステムがインストールされた必要な数のインスタンスができます。TPA は、ssh 経由でアクセスできますルートに sudo を使用します。

7.3.1 前提条件

クラスターをプロビジョニングする前に、クラスター構成 を使用してクラスター構成を生成する必要があります

そして、必要に応じて config.yml を編集して構成を微調整します。

プラットフォーム依存の追加の手順が必要になる場合があります。たとえば、AWS API アクセスキーペアを取得して EC2 インスタンスをプロビジョニングするか、LXD または Docker をセットアップしてコンテナをプロビジョニングする必要があります。詳細については、プラットフォームのドキュメントを参照してください。

7.3.2 クイックスタート

```
tpaexec provision ~/clusters/speedy
```

```
PLAY [Provision cluster]  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ****  **** **
...

TASK [Set up EC2 instances]  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ****  ****  ***
changed: [localhost] => (item=us-east-1:uproar)
changed: [localhost] => (item=us-east-1:unravel)
changed: [localhost] => (item=us-east-1:kinsman)
...

TASK [Generate ssh_config file for the cluster]  ****  ****  ****  ****  ****  ****
↪****  ****  ***
changed: [localhost]

PLAY RECAP  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ****  ****  ****  ****
localhost                  : ok=163  changed=35   unreachable=0    failed=0
↪skipped=44   rescued=0    ignored=2

real    4m42.726s
user    0m39.101s
sys     0m15.687s
```

このコマンドは、大量の出力を生成しますさらに詳細な出力が必要な場合は、コマンドに-v 、-vv などを追加します。出力は、クラスターディレクトリの ansible.log にも記録されます。これは、環境変数 ANSIBLE_LOG_PATH を目的のログファイルのパスと名前に設定することによりオーバーライドできます。

エラーなしで完了した場合は、 *tpaexec deploy* の実行に進んでソフトウェアのインストールと構成を行うことができます。

7.3.3 オプション

クラウドインスタンスをプロビジョニングするときは、インスタンスが、それらを担当する人間まで直接追跡可能であることを確認することが特に重要です。デフォルトでは、TPA は `tpaexec provision` を実行しているユーザーのログイン名が所有するものとして EC2 インスタンスにタグ付けします。

名前を変更するには`--owner <name>` を指定しますたとえば、ユーザー名が postgres や ec2-user のような汎用的なものである場合。イニシャル、「名、姓」、またはその他を一意に識別するために使用できます。

指定した他のオプションは Ansible に渡されます。

7.3.4 インスタンスへのアクセス

プロビジョニングが完了すると、インスタンスに SSH 接続できるようになります。インスタンスが起動して SSH ホストキーをインストールできるようにするために少し遅延した後。上記の出力に示されているように、tpaexec は使用できる ssh_config ファイルを生成します。

```
cd ~/clusters/speedy
cat ssh_config
```

```
Host *
    Port 22
    IdentitiesOnly yes
    IdentityFile "id_speedy"
    UserKnownHostsFile known_hosts tpa_known_hosts
    ServerAliveInterval 60

Host uproar
    User admin
    HostName 3.88.255.205
Host unravel
    User admin
    HostName 54.80.99.142
Host kinsman
    User admin
    HostName 54.165.229.179
```

ホストにログインするには、コマンド `ssh -F ssh_config` の後にホスト名を使用します。例 `ssh -F ssh_config uproar`

ジャンプホストを介してインスタンスに接続する必要がある場合は、`config_yml` の `jumphost` トップレベルセクションでその詳細を指定します。

```
jumphost:
    name: myjumphost
    user: my_username
    identity_file: ~/.ssh/id_ed25519
```

TPA は、ジャンプホスト用に生成された ssh 構成に Host セクションを作成し、各ホストに ProxyJump 指示子を追加します。

プロビジョニングの直後に *tpaexec deploy* を実行できます。インスタンスが起動するまで、必要なだけ待機します。インスタンスが起動するのを待ったり、展開を開始する前に ssh 接続したりする必要はありません。

7.3.5 生成されたファイル

プロビジョニングプロセス中に、多数の新しいファイルがクラスターディレクトリに作成されます。

```
ls ~/clusters/speedy
```

```
total 240
- rw-r--r-- 1 ams ams 193098 Aug  4 17:59 ansible.log
drwxr-xr-x 2 ams ams   4096 Aug  4 17:38 commands
- rw-r--r-- 1 ams ams   1442 Aug  4 17:54 config.yml
lrwxrwxrwx 1 ams ams     51 Aug  4 17:38 deploy.yml ->
                            /opt/EDB/TPA/architectures/M1/deploy.yml
drwxr-xr-x 2 ams ams   4096 Aug  4 17:38 hostkeys
- rw------- 1 ams ams   1675 Aug  4 17:38 id_speedy
- rw------- 1 ams ams   1438 Aug  4 17:38 id_speedy.ppk
- rw-r--r-- 1 ams ams    393 Aug  4 17:38 id_speedy.pub
drwxr-xr-x 4 ams ams   4096 Aug  4 17:50 inventory
- rw-r--r-- 1 ams ams   2928 Aug  4 17:50 tpa_known_hosts
- rw-r--r-- 1 ams ams    410 Aug  4 17:50 ssh_config
- rw-r--r-- 1 ams ams   3395 Aug  4 17:59 vars.json
drwxr-xr-x 2 ams ams   4096 Aug  4 17:38 vault
```

私たちはすでに ssh *config* **ファイルを研究しました。これは、** *'id* *files クラスター用に生成された SSH キーペアと tpa_known_hosts インスタンスにインストールされた hostkeys/' の署名を参照します。

vars.json ファイルは、--cached の後続の呼び出しで tpaexec provision によって使用される場合があります。

inventory/ ディレクトリには、静的および動的インベントリーファイルと、config.yml のグループおよびホスト変数定義が含まれています。

```
cat inventory/00-speedy
```

```
[tag_Cluster_speedy]
uproar ansible_host=3.88.255.205 node=1 platform=aws
```

(次のページに続く)

(前のページからの続き)

```
unravel ansible_host=54.80.99.142 node=2 platform=aws
kinsman ansible_host=54.165.229.179 node=3 platform=aws
```

```
cat inventory/group_vars/tag_Cluster_speedy/01-speedy.yml
```

```
cluster_name: speedy
cluster_tag: tag_Cluster_speedy
edb_repositories: []
failover_manager: repmgr
keyring_backend: system
postgres_flavour: postgresql
postgres_version: 14
preferred_python_version: python3
ssh_key_file: id_speedy
tpa_version: v23.33-24-g4c0909d1
```

```
cat inventory/host_vars/kinsman/01-instance_vars.yml
```

```
ansible_user: admin
location: main
region: us-east-1
role:

- barman

- log-server

- witness
upstream: uproar
volumes:

- device: /dev/xvda

- device: /dev/sdf
  volume_for: barman_data
```

config.yml の変数を変更し、プロビジョニングを再実行すると、これらのファイルが更新されます。構成を変更しない場合、何も起こりません。config.yml に新しいインスタンスを追加して再実行すると、既存のインスタンスに影響を与えることなく新しいインスタンスが表示されます。

7.4 tpaexec deploy

展開は、クラスターのサーバーに Postgres およびその他のソフトウェアをインストールおよび構成するプロセスです。これには、レプリケーション、バックアップなどの設定が含まれます。

展開ステージの最後に、Postgres は、 repmgr、Barman、pgbouncer などの他のコンポーネントとともに実行されます選択したアーキテクチャに応じて。

7.4.1 前提条件

`tpaexec deploy` を実行する前に、**クラスター構成** を実行してクラスター構成を生成し、*tpaexec provision* でサーバーをプロビジョニングしておく必要があります。

Configuring EDB Repos 2.0 repositories を使用している場合は、展開の前に `export EDB_SUBSCRIPTION_TOKEN=xxx` を行う必要があります。これを忘れた場合、エラーメッセージがすぐに思い出させます。

7.4.2 クイックスタート

```
tpaexec deploy ~/clusters/speedy -v
```

```
Using /opt/EDB/TPA/ansible/ansible.cfg as config file

PLAY [Basic initialisation and fact discovery]  ****  ****  ****  ****  ****  ****
↪****  ****  ****  ***
...

PLAY [Set up TPA cluster nodes]  ****  ****  ****  ****  ****  ****  ****  ****  ****
 ****  ****  ****  **** **
...

PLAY RECAP  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ****  ****  ****  ****  ***
zealot                     : ok=281  changed=116  unreachable=0    failed=0
keeper                     : ok=284  changed=96   unreachable=0    failed=0
quaver                     : ok=260  changed=89   unreachable=0    failed=0
quavery                    : ok=260  changed=88   unreachable=0    failed=0
quirk                      : ok=262  changed=100  unreachable=0    failed=0

real    7m1.907s
user    3m2.492s
sys     1m5.318s
```

このコマンドは、大量の出力を生成し、時間がかかる場合があります主に、tpaexec を実行しているホストとクラスター内のホスト間の遅延、およびインスタンスがインストールする必要があるパッケージをダウンロードするのにかかる時間によって異なります。。デプロイメント中に少なくとも 1 つの`-v` を使用することをお勧めします。出力は、クラスターディレクトリの `ansible.log` にも記録されます。

ホスト、タスク、および変更されたタスクの正確な数は、もちろん異なる場合があります。

apply コマンドは自分自身にオプションを取りません。クラスター名の後に指定したオプションは、変更せずに Ansible -v などに渡されます。

Ansible に慣れている人は、時折赤い「failed」タスク出力がスクロールするのを心配するかもしれません。プロセスがその後すぐに停止しない場合、エラーは重要ではなく、コードは自動的に回復しますのでご安心ください。

デプロイが完了したら、*tpaexec* **テストを実行する** を実行してインストールを確認できます。

7.4.3 選択的な展開

deploy_hosts をインスタンス名のコンマ区切りリストに設定することにより、展開をホストのサブセットに制限できます。

```
tpaexec deploy ~/clusters/speedy -v -e deploy_hosts=keeper,quaver
```

これにより、指定されたインスタンスで展開が実行されますが、最初に他のホストでいくつかのタスクを実行して、クラスターの状態に関する情報を収集します。

deploy_hosts 設定は、TPA がサポートしていない Ansible の--limit オプションを使用することに代わる推奨される代替品です。

7.4.4 deploy.yml

展開プロセスはアーキテクチャ固有です。次に、さまざまな *Instance configuration* の概要を示します。Ansible プレイブックに精通している場合は、tpaexec がクラスターのインスタンスにさまざまなロールを適用するのを理解することができます。

config.yml とは異なり、deploy.yml は編集するように設計されておらず、通常はアーキテクチャディレクトリへのリンクです。独自の Ansible タスクを実行するようにデプロイメントプロセスを拡張する場合でも、*TPA hooks* によってこれを行う必要があります。これにより、特定のアーキテクチャ内の将来の実装の変更から保護されます。

7.5 tpaexec test

次に、展開されたクラスターに対してアーキテクチャ固有のテストを実行して、インストールを確認します。このステージの最後に、完全に機能するクラスターが完成します。

tpaexec test を実行する前に、 tpaexec configure 、tpaexec provision 、および tpaexec deploy を正常に実行している必要があります。

7.5.1 クイックスタート

```
tpaexec test ~/clusters/speedy -v
```

出力は、クラスターディレクトリの ansible.log に再びログに記録されます。

このコマンドが成功した場合、クラスターは機能します。

おめでとうございます。

第8章　ARCHITECTURES

8.1 PGD-S

> **注釈**
>
> このアーキテクチャは、 Postgres Distributed 6 専用です。 PGD 5 が必要な場合は、 *PGD-Always-ON* を使用してください。

このアーキテクチャには、EDB サブスクリプションが必要です。すべてのソフトウェアは *Configuring EDB Repos 2.0 repositories* から供給されています。

8.1.1 クラスター構成

構成オプションの概要

このアーキテクチャの `tpaexec configure` の呼び出し例を以下に示します。

```
tpaexec configure ~/clusters/pgd-s \
    --architecture PGD-S
    --edb-postgres-extended 15 \
    --platform aws --instance-type t3.micro \
    --distribution Debian \
```

help コマンドを使用して、使用可能なすべてのオプションをリストできます。

```
tpaexec configure --architecture PGD-S --help
```

以下の表では、PGD-S の必須オプションと追加の重要なオプションについて説明しています。オプションの詳細については、次のセクションで説明します。

必須オプション

Options	Description
--architecture (*-a*)	`PGD-S`に設定する必要があります
Postgres flavour and version (e.g. *--postgresql 15*)	有効な *Postgres* のフレーバーとバージョン 。 Postgres 14-18 をサポートします。

次のページに続く

表 1 – 前のページからの続き

Options	Description

追加オプション

Options	Description	Behaviour if omitted
--platform	*aws*、*docker*、`bare`のいずれか。	Defaults to *aws*.
--bdr-database	レプリケーションに使用されるデータベースの名前。	Defaults to *bdrdb*.
--layout	*standard*`または`*near-far*	Defaults to *standard*
--add-subscriber-only-nodes	追加する加入者専用ノードの数最大 10。	Defaults to 0
--read-write-port	接続マネージャーが読み取り/書き込み接続をリッスンするポート。	Left empty in config.yml, allowing default of the postgres port + 1000
--read-only-port	接続マネージャーが読み取り専用接続をリッスンするポート。	Left empty in config.yml, allowing default of the read-write port + 1
--http-port	接続マネージャーが http API 接続をリッスンするポート。	Left empty in config.yml, allowing default of the read-write port + 2
--use https	Connection Manager の http API の https を有効にする	https is not enabled

PGD-S 構成の詳細

PGD-S クラスターには 3 つのデータノードがあります。 `standard` レイアウトでは、ノードはすべて同じ場所にあります。 `near-far` レイアウトでは、2 つのノードがプライマリロケーションにあり、もう 1 つがセカンダリロケーションにあります。PGDdocumentation を参照してください。

2 つのレイアウトの詳細については、

また、クラスターには、1 つのバーマンノードと最大 10 のサブスクライバー専用ノードが含まれ、`--add-subscriber-only-nodes` パラメーターによって制御されます。これらは常にプライマリの場所にあります。

オプションで`--bdr-database dbname` を指定して、BDR が有効になったデータベースの名前を設定できますデフォルト bdrdb 。

クラスター構成 で説明されるオプションを指定することもできます。

8.2 PGD-X

> **注釈**
>
> このアーキテクチャは、 Postgres Distributed 6 専用です。 PGD 5 が必要な場合は、 *PGD-Always-ON* を使用してください。

このアーキテクチャには、EDB サブスクリプションが必要です。すべてのソフトウェアは *Configuring EDB Repos 2.0 repositories* から供給されています。

8.2.1 クラスター構成

構成オプションの概要

このアーキテクチャの `tpaexec configure` の呼び出し例を以下に示します。

```
tpaexec configure ~/clusters/pgd-x \
        --architecture PGD-X \
        --edb-postgres-extended 15 \
        --platform aws --instance-type t3.micro \
        --distribution Debian \
        --pgd-routing global \
        --location-names dc1 dc2 dc3 \
        --witness-only-location dc3 \
        --data-nodes-per-location 2
```

help コマンドを使用して、使用可能なすべてのオプションをリストできます。

```
tpaexec configure --architecture PGD-X --help
```

以下の表では、PGD-X の必須オプションと追加の重要なオプションについて説明しています。オプションの詳細については、次のセクションで説明します。

必須オプション

Options	Description
--architecture (*-a*)	`PGD-X`に設定する必要があります
Postgres flavour and version (e.g. *--postgresql 15*)	有効な *Postgres のフレーバーとバージョン* 。 Postgres 14-18 をサポートします。
--pgd-routing	`global`または`local`のいずれかである必要があります。

追加オプション

Options	Description	Behaviour if omitted
--platform	*aws*、*docker*、`bare`のいずれか。	Defaults to *aws*.
--location-names	場所名のスペース区切りのリスト。場所の数は、指定された名前の数と同じです。	TPA will configure a single location with three data nodes.
--witness-only-location	ロケーション名は、`location-names`のメンバーである必要があります。	No witness-only location is added.
--data-nodes-per-location	各場所のデータノードの数は、少なくとも 2 である必要があります。	Defaults to 3.
--enable-camo	各ロケーションに 2 つのデータノードを CAMO パートナーとして設定します。	CAMO will not be enabled.
--bdr-database	レプリケーションに使用されるデータベースの名前。	Defaults to *bdrdb*.
--enable-pgd-probes	`health/is-ready`などの pgd-プロキシの http(s) API エンドポイントを有効にして、プロキシの状態をプローブできるようにします。	Disabled by default.
--read-write-port	接続マネージャーが読み取り/書き込み接続をリッスンするポート。	Left empty in config.yml, allowing default of the postgres port + 1000
--read-only-port	接続マネージャーが読み取り専用接続をリッスンするポート。	Left empty in config.yml, allowing default of the read-write port + 1

PGD-X 構成の詳細

PGD-X クラスターは、多くの場所、好ましくは奇数で構成され、それぞれが同じ数のデータノード、やはり好ましくは奇数です。`--location-names` を指定しない場合、デフォルトでは、3 つのデータノードを持つ単一の場所を使用します。

クラスターの場所名は`--location-names dc1 dc2` …として指定されます。ロケーションは、この定義がユースケースにとって意味のある方法で、レベルの冗長性を提供する独立したデータセンターを表します。たとえば、AWS リージョン、独自のデータセンター、またはサーバーがホストされている場所を識別するためのその他の指定。

`--data-nodes-per-location N` を使用して、各場所のデータノードの数を指定します。最小数は 2 で、デフォルトは 3 です。

ロケーションごとに偶数のデータノードを指定すると、TPA は各ロケーションに追加の監視ノードを自動的に追加します。これにより、コストを削減しながら、信頼できるコンセンサスを確立する機能が維持されます。監視のハードウェア要件は、データノードと比較して最小限です。

2 つのロケーションのみからなるクラスターは、いずれかのロケーションに障害が発生した場合、グローバルコンセンサスを確立する能力を完全に失います。3 番目の監視専用の場所を追加することをお勧めします。これにはデータノードは含まず、監視ノードのみが含まれ、これもコンセンサスを確実に確立する

ために使用されます。--witness-only-location loc を使用して、いずれかの場所を監視者として指定します。

ユースケースによっては、 --pgd-routing local または global を指定して、Connection Manager が書き込みリーダーに接続をルーティングする方法を構成する必要があります。ローカルルーティングは、すべての接続マネージャーを独自の場所内の書き込みリーダーにルートさせますジオシャーディングアプリケーションに適しています。グローバルルーティングは、すべての接続マネージャーを、すべての場所で使用可能なすべてのデータノードの中から選択される単一の書き込みリーダーにルートします。

オプションで--bdr-database dbname を指定して、BDR が有効になったデータベースの名前を設定できますデフォルト bdrdb 。

オプションで--enable-camo を指定して、各リージョンの 2 つのデータノードを CAMO パートナーとして設定できます。

オプションで--enable-pgd-probes [{http, https}] を指定して、プロキシの状態を簡単にプローブできる http API エンドポイントを有効にすることができます。

クラスター構成 で説明されるオプションを指定することもできます。

8.3 PGD-Always-ON

> **注釈**
>
> このアーキテクチャは Postgres Distributed 5 専用です。 PGD 4 または 3.7 が必要な場合は、*BDR-Always-ON* を使用してください。

Always-ON 構成の EDB Postgres Distributed 5 は、テストおよび運用環境での使用に適しています。

このアーキテクチャには、EDB サブスクリプションが必要です。すべてのソフトウェアは *Configuring EDB Repos 2.0 repositories* からソースされます。

8.3.1 クラスター構成

構成オプションの概要

このアーキテクチャの tpaexec configure の呼び出し例を以下に示します。

```
tpaexec configure ~/clusters/pgd-ao \
        --architecture PGD-Always-ON \
        --edb-postgres-extended 15 \
        --platform aws --instance-type t3.micro \
        --distribution Debian \
        --pgd-proxy-routing global \
        --location-names dc1 dc2 dc3 \
        --witness-only-location dc3 \
        --data-nodes-per-location 2
```

help コマンドを使用して、使用可能なすべてのオプションをリストできます。

```
tpaexec configure --architecture PGD-Always-ON --help
```

以下の表では、PGD-Always-ON の必須オプションと追加の重要なオプションについて説明しています。オプションの詳細については、次のセクションで説明します。

必須オプション

Options	Description
--architecture (*-a*)	`PGD-Always-ON`に設定する必要があります
Postgres flavour and version (e.g. *--postgresql 15*)	有効な *Postgres* のフレーバーとバージョン 。
--pgd-proxy-routing	`global`または`local`のいずれかである必要があります。

追加オプション

Options	Description	Behaviour if omitted
--platform	*aws*、*docker*、`bare`のいずれか。	Defaults to *aws*.
--location-names	場所名のスペース区切りのリスト。場所の数は、指定された名前の数と同じです。	TPA will configure a single location with three data nodes.
--witness-only-location	ロケーション名は、`location-names`のメンバーである必要があります。	No witness-only location is added.
--data-nodes-per-location	各場所のデータノードの数は、少なくとも 2 である必要があります。	Defaults to 3.
--add-proxy-nodes-per-location	各場所のプロキシノードの数。	PGD-proxy will be installed on each data node.
--enable-camo	各ロケーションに 2 つのデータノードを CAMO パートナーとして設定します。	CAMO will not be enabled.
--bdr-database	レプリケーションに使用されるデータベースの名前。	Defaults to *bdrdb*.
--enable-pgd-probes	`health/is-ready`などの pgd-プロキシの http(s) API エンドポイントを有効にして、プロキシの状態をプローブできるようにします。	Disabled by default.
--proxy-listen-port	プロキシノードがトラフィックを書き込みリーダーにルーティングするポート。	Defaults to 6432
--proxy-read-only-port	プロキシノードが読み取り専用トラフィックをシャドウノードにルーティングするポート。	Defaults to 6433

PGD-Always-ON 構成の詳細

PGD-Always-ON クラスターは、多くの場所、好ましくは奇数で構成され、それぞれが同じ数のデータノード、やはり好ましくは奇数です。`--location-names` を指定しない場合、デフォルトでは、3 つのデータノードを持つ単一の場所を使用します。

クラスターの場所名は`--location-names dc1 dc2 …`として指定されます。ロケーションは、この定義がユースケースにとって意味のある方法で、レベルの冗長性を提供する独立したデータセンターを表します。たとえば、AWS リージョン、独自のデータセンター、またはサーバーがホストされている場所を識別するためのその他の指定。

`--data-nodes-per-location N` を使用して、各場所のデータノードの数を指定します。最小数は 2 で、デフォルトは 3 です。

ロケーションごとに偶数のデータノードを指定すると、TPA は各ロケーションに追加の監視ノードを自動的に追加します。これにより、コストを削減しながら、信頼できるコンセンサスを確立する機能が維持されます。監視のハードウェア要件は、データノードと比較して最小限です。

2 つのロケーションのみからなるクラスターは、いずれかのロケーションに障害が発生した場合、グローバルコンセンサスを確立する能力を完全に失います。3 番目の監視専用の場所を追加することをお勧めします。これにはデータノードは含まず、監視ノードのみが含まれ、これもコンセンサスを確実に確立するために使用されます。`--witness-only-location loc` を使用して、いずれかの場所を監視者として指定します。

デフォルトでは、すべてのデータノードすべての場所で、接続ルーティングのために PGD-プロキシも実行されます。代わりに別の PGD-プロキシインスタンスを作成するには、`--add-proxy-nodes-per-location 3` または追加する数のプロキシを使用します。

ユースケースによっては、 `--pgd-proxy-routing local` または `global` を指定して、PGD-Proxy が書き込みリーダーに接続をルーティングする方法を構成する必要があります。ローカルルーティングは、すべての PGD-プロキシを独自の場所内の書き込みリーダーにルートさせますジオシャーディングアプリケーションに適しています。グローバルルーティングは、すべてのプロキシを単一の書き込みリーダーにルートさせます。この書き込みリーダーは、すべての場所で使用可能なすべてのデータノードの間で選択されます。

オプションで`--bdr-database dbname` を指定して、BDR が有効になったデータベースの名前を設定できますデフォルト bdrdb 。

オプションで`--enable-camo` を指定して、各リージョンの 2 つのデータノードを CAMO パートナーとして設定できます。

オプションで`--enable-pgd-probes [{http, https}]` を指定して、プロキシの状態を簡単にプローブできる http API エンドポイントを有効にすることができます。

クラスター構成 で説明されるオプションを指定することもできます。

8.4 PGD Lightweight

> **注釈**
>
> このアーキテクチャは Postgres Distributed 5 専用です。 PGD 4 または 3.7 が必要な場合は、*BDR-Always-ON* を使用してください。

軽量構成の EDB Postgres Distributed 5 は、テストおよび実稼働での使用に適しています。

このアーキテクチャには、EDB サブスクリプションが必要です。すべてのソフトウェアは *Configuring EDB Repos 2.0 repositories* からソースされます。

8.4.1 クラスター構成

構成オプションの概要

このアーキテクチャの `tpaexec configure` の呼び出し例を以下に示します。

```
tpaexec configure ~/clusters/pgd-lw \
    --architecture Lightweight \
    --edb-postgres-extended 15 \
    --platform aws --instance-type t3.micro \
    --distribution Debian \
    --location-names main dr \
```

help コマンドを使用して、使用可能なすべてのオプションをリストできます。

```
tpaexec configure --architecture Lightweight --help
```

以下の表では、PGD-Always-ON の必須オプションと追加の重要なオプションについて説明しています。オプションの詳細については、次のセクションで説明します。

必須オプション

Options	Description
--architecture (*-a*)	`Lightweight`に設定する必要があります
Postgres flavour and version (e.g. *--postgresql 15*)	有効な *Postgres* のフレーバーとバージョン 。

追加オプション

Options	Description	Behaviour if omitted
--platform	*aws*、*docker*、`bare`のいずれか。	Defaults to *aws*.
--location-names	場所名のスペース区切りのリスト。場所の数は、指定された名前の数と同じです。	TPA will configure a single location with three data nodes.
--add-proxy-nodes-per-location	各場所のプロキシノードの数。	PGD-proxy will be installed on each data node.
--bdr-database	レプリケーションに使用されるデータベースの名前。	Defaults to *bdrdb*.
--enable-pgd-probes	`health/is-ready`などの pgd-プロキシの http(s) API エンドポイントを有効にして、プロキシの状態をプローブできるようにします。	Disabled by default.
--proxy-listen-port	プロキシノードがトラフィックを書き込みリーダーにルーティングするポート。	Defaults to 6432
--proxy-read-only-port	プロキシノードが読み取り専用トラフィックをシャドウノードにルーティングするポート。	Defaults to 6433

軽量構成の詳細

PGD Lightweight クラスターは 2 つのロケーションで構成され、2 つのノードを含むプライマリアクティブロケーションと、単一のノードを含むディザスターリカバリー dr ロケーション。

クラスターの場所名は`--location-names primary dr` として指定されます。ロケーションは、この定義がユースケースにとって意味のある方法で、レベルの冗長性を提供する独立したデータセンターを表します。たとえば、AWS リージョン、独自のデータセンター、またはサーバーがホストされている場所を識別するためのその他の指定。

デフォルトでは、すべてのデータノードすべての場所で、接続ルーティングのために PGD-プロキシも実行されます。代わりに別の PGD-プロキシインスタンスを作成するには、`--add-proxy-nodes-per-location 3` または追加する数のプロキシを使用します。

グローバルルーティングは、すべてのプロキシを単一の書き込みリーダーにルートさせます。この書き込みリーダーは、すべての場所で使用可能なすべてのデータノードの間で選択されます。

オプションで`--bdr-database dbname` を指定して、BDR が有効になったデータベースの名前を設定できますデフォルト bdrdb 。

オプションで`--enable-pgd-probes [{http, https}]` を指定して、プロキシの状態を簡単にプローブできる http API エンドポイントを有効にすることができます。

クラスター構成 で説明されるオプションを指定することもできます。

8.5 BDR-Always-ON

Always-ON 構成の EDB Postgres 分散 3.7 または 4。テストおよびプロダクションでの使用に適しています。

このアーキテクチャには、EDB サブスクリプションが必要です。すべてのソフトウェアは *Configuring EDB Repos 2.0 repositories* からソースされます。

BDR-Always-ON アーキテクチャには、 `--layout` configure オプションで選択できる 4 つのバリアントがあります。

1. ブロンズ 2 × bdr+プライマリ、bdr+witness、barman、2 × harp-プロキシ
2. シルバー: ブロンズ、bdr + Witness が bdr +プライマリに昇格し、バーテンダーが別の場所に移動した
3. gold それぞれ 2 × bdr+プライマリ、2 × harp-プロキシ、および barman を備えた 2 つの対称的な場所。プラス 3 番目の場所に bdr + Witness
4. プラチナ: ゴールド、ただし 1 つの bdr+読み取り専用ロジカルスタンバイが各メインの場所に追加されます

EDB の Postgres DistributedAlways On アーキテクチャ

whitepaper を確認できます。

詳細なレイアウト図については、

このアーキテクチャは、PGD バージョン 3.7 および 4 で使用することを対象としています。

8.5.1 クラスター構成

構成オプションの概要

このアーキテクチャの `tpaexec configure` の呼び出し例を以下に示します。

```
tpaexec configure ~/clusters/bdr \
         --architecture BDR-Always-ON \
         --platform aws --region eu-west-1 --instance-type t3.micro \
         --distribution Debian \
         --edb-postgres-advanced 14 --redwood
         --layout gold \
         --harp-consensus-protocol bdr
```

help コマンドを使用して、使用可能なすべてのオプションをリストできます。

```
tpaexec configure --architecture BDR-Always-ON --help
```

以下の表では、BDR-Always-ON の必須オプションと追加の重要なオプションについて説明します。オプションの詳細については、次のセクションで説明します。

必須オプション

Option	Description
--architecture (*-a*)	`BDR-Always-ON`に設定する必要があります。
Postgres flavour and version (e.g. *--postgresql 14*)	有効な *Postgres* のフレーバーとバージョン 。

次のページに続く

表 9 – 前のページからの続き

Option	Description
--layout	*bronze*、*silver*、*gold*、`platinum`のいずれか。
--harp-consensus-protocol	*bdr*、`etcd`のいずれか。

追加オプション

Option	Description	Behaviour if omitted
--platform	*aws*、*docker*、`bare`のいずれか。	Defaults to *aws*.
--enable-camo	各ロケーションに 2 つのデータノードを CAMO パートナーとして設定します。	CAMO will not be enabled.
--bdr-database	レプリケーションに使用されるデータベースの名前。	Defaults to *bdrdb*.
--enable-harp-probes	`health/is-ready`などの harp の http API エンドポイントを有効にして、harp の状態をプローブできるようにします。	Disabled by default.

BDR-Always-ON 構成の詳細

`--layout layoutname` を指定して、サポートされている BDR ユースケースバリエーションのいずれかを設定する必要があります。許可される引数は、ブロンズ、シルバー、ゴールド、およびプラチナです。ブロンズ、ゴールド、およびプラチナレイアウトには、Raft コンセンサスマジョリティの奇数のノードを保証する PGD 監視ノードがあります。Witness ノードはデータレプリケーションに参加しません。

`--harp-consensus-protocol protocolname` を指定する必要があります。サポートされているプロトコルは bdr および etcd です。詳細は、*Configuring HARP* を参照してください。

オプションで`--bdr-database dbname` を指定して、PGD が有効になっているデータベースの名前を設定できますデフォルト bdrdb 。

オプションで`--enable-camo` を指定して、各リージョンの PGD プライマリインスタンスのペアを互いの CAMO パートナーに設定できます。

オプションで`--enable-harp-probes [{http, https}]` を指定して、harp の状態を簡単にプローブできる http API エンドポイントを有効にすることができます。

BDR-Always-ON アーキテクチャでは、デフォルトで HARP2 が有効になっていることに注意してください。

クラスター構成 で説明されるオプションを指定することもできます。

8.6 M1

単一のプライマリノードと、バックアップとフェールオーバー管理を含む多数のスタンバイノードへの物理レプリケーションを備えた Postgres クラスター。

このアーキテクチャは、運用に適していますが、そのシンプルさと独自のコンポーネントなしで構成できる機能により、テスト、デモンストレーション、学習にも適しています。

このアーキテクチャでサブスクリプション専用の EDB ソフトウェアを選択した場合、EDB Repos 2.0 からソースされ、 *Configuring EDB Repos 2.0 repositories* が必要です。

8.6.1 フェイルオーバー管理

M1 アーキテクチャには、常にフェールオーバーマネージャーが含まれます。サポートされているオプションは repmgr、EDB Failover ManagerEFM、および Patroni です。すべての場合、フェールオーバーマネージャーはデフォルトで構成され、プライマリが使用できなくなった場合にレプリカがプライマリに代わって昇格します。

アプリケーションのフェイルオーバー

M1 アーキテクチャは、通常、アプリケーショントラフィックをプライマリに再ルーティングする自動機能を提供しません。この機能をクラスターに追加するには、いくつかの方法があります。

TPA で

- フェールオーバーマネージャーとして repmgr を選択し、PgBouncer を有効にすると、`config.yml` の `cluster_vars` の下に `repmgr_redirect_pgbouncer: true` ハッシュを含めることができます。これにより、repmgr は PgBouncer を自動的に再構成して、フェールオーバー時にトラフィックを新しいプライマリにルーティングします。
- フェールオーバーマネージャーとして Patroni を選択し、PgBouncer を有効にすると、Patroni はフェールオーバー時にトラフィックを新しいプライマリにルーティングするように PgBouncer を自動的に再構成します。
- フェールオーバーマネージャーとして EFM を選択した場合、 `config.yml` の `cluster_vars` の下の `efm_conf_settings` ハッシュを使用できます `config.yml` から *configure EFM to use a virtual IP address (VIP)* 。これは、常にプライマリノードにルーティングされる追加の IP アドレスです。
- クラスターとアプリケーションの間に適切なプロキシまたはロードバランサーを配置し、*TPA hooks* を使用して、選択したフェールオーバーマネージャーをフェールオーバー時に新しいプライマリへのルートで更新するように構成します。
- マルチホスト接続文字列を使用するなど、アプリケーション自分自身でフェールオーバーを処理します。

バックアップフェイルオーバー

TPA は、いかなる種類の「バックアップフェイルオーバー」も構成しません。バックアップしている Postgres ノードがダウンしている場合、ノードがオンラインに戻るまでバックアップは単に停止します。バックアップを新しいプライマリに手動で接続するには、`config.yml` を編集して `backup` ハッシュを新しいプライマリインスタンスに追加し、`tpaexec deploy` を再実行します。

8.6.2 クラスター構成

構成オプションの概要

このアーキテクチャの `tpaexec configure` の呼び出し例を以下に示します。

```
tpaexec configure ~/clusters/m1 \
        --architecture M1 \
        --platform aws --region eu-west-1 --instance-type t3.micro \
        --distribution Debian \
        --postgresql 14 \
        --failover-manager repmgr \
        --data-nodes-per-location 3
```

help コマンドを使用して、使用可能なすべてのオプションをリストできます。

```
tpaexec configure --architecture M1 --help
```

以下の表では、M1 の必須オプションと追加の重要なオプションについて説明します。オプションの詳細については、次のセクションで説明します。

必須オプション

Parameter	Description
--architecture (*-a*)	`M1`に設定する必要があります。
Postgres flavour and version (e.g. *--postgresql 15*)	有効な *Postgres* **のフレーバーとバージョン** 。
One of: - *--failover-manager {efm, repmgr, patroni}` - `--enable-efm`  - `--enable-repmgr` - `--enable-patroni*	*Configuring EFM* 、 *Repmgr* **リダイレクト** *pgbouncer* 、および *Patroni cluster management commands* からフェイルオーバーマネージャーを選択します。

追加オプション

Parameter	Description	Behaviour if omitted
--platform	*aws*、*docker*、`bare`のいずれか。	Defaults to *aws*.
--location-names	場所名のスペース区切りのリスト。場所の数は、指定された名前の数と同じです。	A single location called "main" is used.
--primary-location	プライマリサーバーが置かれる場所。`location-names`のメンバーである必要があります。	The first listed location is used.

次のページに続く

表 12 – 前のページからの続き

Parameter	Description	Behaviour if omitted
--data-nodes-per-location	1 以上の数字。各場所では、1 つのノードがクラスターのプライマリノードから直接ストリーミングするように構成され、他のノードが存在する場合はそのノードからストリーミングします。	Defaults to 2.
--witness-only-location	ロケーション名は、`location-names`のメンバーである必要があります。この場所には、単一の監視ノードのみが設定されます。	No witness-only location is added.
--single-node-location	ロケーション名は、`location-names`のメンバーである必要があります。この場所には、単一のデータノードのみが設定されます。	No single-node location is added.
--enable-haproxy	2 つの追加ノードがロードバランサーレイヤーとして追加されます。フェールオーバーマネージャーとして Patroni でのみサポートされています。	HAproxy nodes will not be added to the cluster.
--enable-pgbouncer	PgBouncer は、プライマリの接続をプールするように Postgres ノードで構成されます。	PgBouncer will not be configured in the cluster.
--patroni-dcs	patroni の分散構成ストアバックエンドを選択します。現時点では、オプションは`etcd`のみです。フェールオーバーマネージャーとして Patroni でのみサポートされています。	Defaults to *etcd*.
--efm-bind-by-hostname	efm が IP アドレスの代わりにホスト名を使用してクラスター`bind.address`を構成するようにします。	Defaults to use IP addresses

M1 構成の詳細

クラスター構成 で説明されるオプションを指定することもできます。

第9章　PLATFORMS

9.1 aws

TPA は、AWS EC2 での運用クラスターのプロビジョニングを完全にサポートしています。

9.1.1 API アクセス設定

AWS API を使用するには、次のことを行う必要があります。

- Obtain an access keypair
- Add it to your configuration

例

```
cat > ~/.aws/credentials
```

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

tpaexec がそれを使用して ec2 リソースをプロビジョニングできるように、IAM ユーザーは少なくとも次の権限セットを持っている必要があります。

```
ec2:AssociateRouteTable
ec2:AttachInternetGateway
ec2:AuthorizeSecurityGroupIngress
ec2:CreateInternetGateway
ec2:CreateRoute
ec2:CreateRouteTable
ec2:CreateSecurityGroup
ec2:CreateSubnet
ec2:CreateTags
ec2:CreateVpc
ec2:DeleteKeyPair
ec2:DeleteRouteTable
ec2:DeleteSecurityGroup
ec2:DeleteSubnet
ec2:DeleteVpc
ec2:DescribeImages
```

(次のページに続く)

(前のページからの続き)

```
ec2:DescribeInstanceStatus
ec2:DescribeInstances
ec2:DescribeInternetGateways
ec2:DescribeKeyPairs
ec2:DescribeRouteTables
ec2:DescribeSecurityGroups
ec2:DescribeSubnets
ec2:DescribeTags
ec2:DescribeVolumes
ec2:DescribeVpcAttribute
ec2:DescribeVpcClassicLink
ec2:DescribeVpcClassicLinkDnsSupport
ec2:DescribeVpcs
ec2:DisassociateRouteTable
ec2:ImportKeyPair
ec2:ModifyVpcAttribute
ec2:RevokeSecurityGroupIngress
ec2:RunInstances
ec2:TerminateInstances
iam:AddRoleToInstanceProfile
iam:CreateInstanceProfile
iam:CreateRole
iam:DeleteInstanceProfile
iam:DeleteRole
iam:DeleteRolePolicy
iam:GetInstanceProfile
iam:GetRole
iam:GetRolePolicy
iam:ListAttachedRolePolicies
iam:ListGroups
iam:ListInstanceProfiles
iam:ListInstanceProfilesForRole
iam:ListRolePolicies
iam:ListRoles
iam:ListUsers
iam:PassRole
iam:PutRolePolicy
iam:RemoveRoleFromInstanceProfile
kms:CreateGrant
kms:GenerateDataKeyWithoutPlaintext
s3:CreateBucket
s3:GetBucketVersioning
s3:GetObject
s3:GetObjectTagging
s3:ListAllMyBuckets
```

(次のページに続く)

(前のページからの続き)

```
s3:ListBucket
s3:ListBucketVersions
s3:PutBucketOwnershipControls
s3:PutObject
s3:PutObjectAcl
```

9.1.2 はじめに

サービスは物理的に regions and availability zones に分割されます。アベイラビリティーゾーンは、リージョンコードに続いて 1 文字の eu-west-1a などで表されますが、その名前は、異なる AWS アカウントの異なる場所を参照する場合があり、アカウント間の解釈を調整する方法はありません。

AWS リージョンは互いに完全に分離され、リソースを共有しません。リージョン内のアベイラビリティーゾーンは物理的に分離され、論理的にほとんど分離されていますが、低遅延リンクで接続され、特定のネットワークリソースを共有できます。

ネットワーキング

AWS のすべてのネットワーク構成は、 Virtual Private Cloud のコンテキストで発生します

リージョン内で。 VPC 内で、 subnets を作成できます

インターネットゲートウェイ、ルーティングテーブルなどとともに、特定のアベイラビリティーゾーンに関連付けられています。

Security Groups は、いくつでも作成できます

インスタンスに許可されるインバウンドおよびアウトバウンドトラフィックのルール、プロトコル、宛先ポートレンジ、およびソースまたは宛先 IP アドレスレンジの観点から。

インスタンス

AWS EC2 は、さまざまな instance types を提供します

異なる価格/パフォーマンスポイントで異なるハードウェア構成を使用します。特定のアベイラビリティーゾーンのサブネット内に、EC2 instances を作成できます。

AMI と呼ばれるディストリビューションイメージに基づいて、1 つ以上の EBS volumes をアタッチします

インスタンスに永続ストレージを提供します。SSH public key を登録することにより、インスタンスに SSH 接続できます。

インスタンスには、サブネット内のプライベート IP アドレスが常に割り当てられます。サブネット構成によっては、ephemeral public IP address が割り当てられる場合があります。

これは、インスタンスがシャットダウンすると失われ、再起動されたときに別のエフェメラル IP が割り当てられます。代わりに、Elastic IP と呼ばれる静的リージョン固有のルーティング可能 IP アドレスを割り当てることができます。

任意のインスタンスに。

外部からインスタンスに到達できるようにするには、ルーティング可能な IP アドレスを持つだけでなく、VPC のネットワーク構成インターネットゲートウェイ、ルーティングテーブル、セキュリティグループもアクセスを許可するように調整する必要があります。

9.1.3 構成

ここでは、 `tpaexec configure` を介して指定できる、または config.yml で直接定義できる AWS 固有の設定の簡単な説明を示します。

リージョン

`--region` または`--regions` で使用するクラスターの 1 つ以上のリージョンを指定できます。TPA は、それぞれに関連する必要な vpc エントリを生成し、可能な場合は異なるアベイラビリティーゾーンを使用してロケーションをこれらのリージョンに均等に分散します。

`regions` は `locations` とは異なり、各ロケーションはリージョンおよびこのリージョン内のアベイラビリティーゾーンに属します。`regions` は AWS 固有のオブジェクト、`locations` はクラスターオブジェクトです。

注複数のリージョンを指定する場合、ネットワーク構成を手動で編集する必要があります。

- `ec2_vpc` エントリに AWS vpc ピアリングを使用できるように、オーバーラップしない cidr ネットワークが必要です。デフォルトでは、TPA はすべての cidr を `10.33.0.0/16` に設定します。詳細については、 *VPC 必須* を参照してください。
- 各 `location` は、それらが属している `ec2_vpc` `cidr` と一致する `subnet` で更新する必要があります。詳細については、**サブネットオプショナル** を参照してください。
- TPA は `cluster_rules` の下で基本的なルールを使用してセキュリティグループを作成します。これらは、各 `subnet` cidr の `ec2_vpc` cidr と一致するように更新する必要があります。詳細については、**セキュリティグループオプショナル** を参照してください。
- `tpaexec deploy` の前に VPC ピアリングを手動で設定する必要があります。`tpaexec configure` を実行し、config.yml で `vpc-id` を使用する前に、VPC と必要な VPC ピアリングを作成することをお勧めします。詳細については、*VPC 必須* を参照してください。

VPC 必須

使用する VPC を指定する必要があります。

```
ec2_vpc:
  Name: Test
  cidr: 10.33.0.0/16
```

これはデフォルト構成であり、指定された CIDR で Test という名前の VPC が存在しない場合は作成し、それ以外の場合は既存の VPC を使用します。

VPC を作成するには、名前と cidr の両方を指定する必要があります。VPC 名のみを指定した場合、一致する VPC が存在しない場合、TPA は失敗します。

TPA が VPC を作成すると、tpaexec deprovision はそれを削除しようとしますが、既存の VPC はそのままにしておきます。AWS にはアカウントごとの VPC 数に 1 桁のデフォルト制限があるため、新しい VPC を作成する前によく考えてください。

より詳細なマッチングが必要な場合、またはさまざまなリージョンでさまざまな VPC を指定する場合、展開されたフォームを使用できます。

```
ec2_vpc:
  eu-west-1:
    Name: Test
    cidr: 172.16.0.0/16
  us-east-1:
    filters:
      vpc-id: vpc-nnn
  us-east-2:
    Name: Example
    filters:
      [filter expressions]
```

AMI 必須

使用する AMI を指定する必要があります。

```
ec2_ami:
  Name: xxx
  Owner: self
```

より正確なマッチングのためにフィルタ仕様を追加できます。

```
ec2_ami:
  Name: xxx
  Owner: self
  filters:
    architecture: x86_64
    [more key/value filters]
```

デフォルトでは、tpaexec configure は--distribution 引数に基づいて、適切な ec2_ami を選択します。

サブネットオプショナル

すべてのインスタンスは、そのサブネットを CIDR 形式で、または subnet-xxx ID として指定する必要があります。オプションで、作成する各サブネットの名前とアベイラビリティーゾーンを指定できます。

```
ec2_vpc_subnets:
  us-east-1:
    192.0.2.0/27:
      az: us-east-1b
```

(次のページに続く)

(前のページからの続き)

```
      Name: example1
    192.0.2.100/27:
      az: us-east-1b
      Name: example2
```

セキュリティグループオプショナル

デフォルトでは、クラスターのセキュリティグループを作成します。1 つ以上の既存のセキュリティグループを使用するには、次のように設定します。

```
ec2_groups:
  us-east-1:
    group-name:
      - foo
```

デフォルトのセキュリティグループのルールをカスタマイズする場合は、`cluster_rules` を設定します。

```
cluster_rules:

- cidr_ip: 0.0.0.0/0
  from_port: 22
  proto: tcp
  to_port: 22

- cidr_ip: 192.0.2.0/27
  from_port: 0
  proto: tcp
  to_port: 65535

- cidr_ip: 192.0.2.100/27
  from_port: 0
  proto: tcp
  to_port: 65535
```

この例では、任意のアドレスからの ssh ポート 22、および特定の IP レンジからの任意のポートでの TCP 接続を許可します。注 from_port および to_port は、ソースと宛先ではなく、ポートの数値範囲を定義します。

カスタムルールを設定するか、既存のセキュリティグループを使用する場合、クラスター内のインスタンスが必要に応じて相互に通信を許可されていることを確認する必要がありますたとえば、Postgres の TCP/5432 を許可します。

インターネットゲートウェイオプショナル

デフォルトでは、次を設定しない限り、すべての VPC のインターネットゲートウェイを作成します。

```
ec2_instance_reachability: private
```

より詳細な制御のために、次を設定できます。

```
ec2_vpc_igw:
  eu-west-1: yes
  eu-central-1: yes
  us-east-1: no
```

SSH キーオプション

```
#  Set this to change the name under which we register our SSH key.

#  ec2_key_name: tpa_cluster_name

#
#  Set this to use an already-registered key.

#  ec2_instance_key: xxx
```

S3 バケットオプション

TPA では、AWS クラスターをプロビジョニングするために S3 バケットへのアクセスが必要です。このバケットは、SSH ホストキーなどのファイルを一時的に保存するために使用されますが、他のクラスターデータバックアップなどに使用される場合があります。

デフォルトでは、TPA はプロビジョニングするクラスターに edb-tpa-<aws-account-user-id> という名前の S3 バケットを使用します。バケットが存在しない場合、TPA にバケットを作成するかどうかを確認するメッセージが表示されます。

代わりに既存の S3 バケットを使用するには、次のように設定します

```
cluster_bucket: name-of-bucket
```

確認プロンプトを表示せずにデフォルトのバケット名を受け入れるように cluster_bucket: auto を設定することもできます。

クラスターのプロビジョニングを解除するときに、TPA は S3 バケットを削除しません。バケットを自分で削除するには、次を実行します。

```
aws s3 rb s3://<bucket> --force
```

インスタンスのプロビジョニングに使用している IAM ユーザーには、このバケットへの読み取りおよび書き込みアクセスが必要です。プロビジョニング中に、tpaexec は、インスタンスプロファイルを介して cluster_bucket への読み取り専用アクセスをインスタンスに提供します。

Elastic IP アドレス

Elastic IP アドレスを使用するには、config.yml で `assign_elastic_ip` を `true` に設定します。クラスター内のすべてのインスタンスに影響を与えるように `instance_defaults` で、または必要に応じて別のインスタンスに個別に。デフォルトでは、これにより、新しい Elastic IP アドレスが割り当てられ、新しいインスタンスに割り当てられます。割り当て済みだがまだ割り当てられていない Elastic IP アドレスを使用するには、 `elastic_ip: 34.252.55.252` を使用して、割り当てられたアドレスに置き換えます。

インスタンスプロファイルオプショナル

```
#  Set this to change the name of the instance profile role we create.

#  cluster_profile: cluster_name_profile

#
#  Set this to use an existing instance profile (which must have all the

#  required permissions assigned to it).

#  instance_profile_name: xxx
```

9.2 bare(-metal servers)

config.yml に `platform: bare` を設定します

このプラットフォームは、ベアメタルサーバーやクラウドプラットフォーム AWS を含む既にプロビジョニングされたサーバーを含む、SSH を介してアクセス可能なサーバーをサポートすることを目的としています。

各ターゲットサーバーの IP アドレスとユーザー名を定義する必要があります。

```
instances:
  - node: 1
    Name: igor
    platform: bare
    public_ip: 192.0.2.1
    private_ip: 192.0.2.222
    vars:
      ansible_user: xyzzy
```

次のことを確認する必要があります

1. TPA は `ansible_user` としてインスタンスに ssh できます
2. `ansible_user` には、インスタンスで sudo アクセスがあります

9.2.1 SSH アクセス

上記の例では、TPA は xyzzy@192.0.2.1 に ssh 接続してインスタンスにアクセスします。

デフォルトでは、TPA は ssh-keygen を実行して、クラスターディレクトリに新しい SSH キーペアを生成します。プライベートキーの名前は id_cluster_name であり、公開キーは id_cluster_name.pub に保存されます。

インスタンスが受け入れる別のキーを使用するように ssh_key_file: /path/to/id_keyname を設定するか、生成されたキーからのアクセスを許可するようにインスタンスを構成する必要がありますたとえば、id_cluster_name.pub のコンテンツを~xyzzy/.ssh/authorized_keys に追加する ssh-copy-id を使用します。

また、ssh がインスタンスのホストキーを検証できることを確認する必要があります。クラスターディレクトリの known_hosts ファイルにエントリを追加するか、TPA 生成ホストキーをクラスターディレクトリの hostkeys/ssh_host_ *_key* からインスタンスの/etc/ssh にインストールできます。生成された tpa_known_hosts ファイルには、これらのキーのエントリが含まれています。

たとえば、生成されたユーザーキーを使用して ssh 接続しますが、既存のホストキーを保持するには、次のことができます。

```
cd ~/clusters/speedy
ssh-copy-id -i id_speedy xyzzy@192.0.2.1
ssh-keyscan -H 192.0.2.1 >> tpa_known_hosts
```

tpaexec ping ~/clusters/speedy を実行して、動作しているかどうかを確認します。まだない場合は、コマンドに-vvv を追加して、完全な ssh コマンドラインを確認します。（注 Ansible は ssh を呼び出して、インスタンスで bash -c 'python3 && sleep 0' のようなコマンドを実行します。デバッグ中に ssh コマンドを手動で実行する場合、これを'id' などの出力を生成して終了するコマンドに置き換えます。）

詳細については

- *ssh_key_file*
- *Managing SSH host keys*

9.2.2 ディストリビューションのサポート

TPA は、ターゲットインスタンスで実行されているディストリビューションを検出しようとしますが、サポートされていない場合は失敗します。TPA は現在、bare インスタンスで Debian 11/12; または Bullseye/bookworm 、Ubuntu 22.04/24.04; または jammmy/noble 、および RHEL/Rocky/AlmaLinux 8.x/9.x をサポートしています。

9.2.3 IP アドレス

任意のインスタンスに public_ip 、private_ip 、またはその両方を指定できます。

TPA は、これらの IP アドレスを 2 つの方法で使用します。1 つ目は、インスタンスに ssh して、展開中にコマンドを実行します。次に、クラスター内の通信を設定するためなど、/etc/hosts または Postgres の primary_conninfo を設定します。

public_ip を指定すると、デプロイ中にインスタンスに ssh 接続するために使用されます。private_ip を指定すると、クラスター内の通信を設定するために使用されます。両方を指定した場合、public_ip は展開中に使用され、 private_ip はクラスター通信に使用されます。

いずれかのみを指定した場合、アドレスは両方の目的に使用されます。たとえば、別のネットワーク上のサーバーに public_ip のみを設定したり、閉じたネットワーク内で TPA を実行している場合は private_ip のみを設定したりできます。(パブリック/プライベートを使用する代わりに、1 つの IP アドレスのみを指定する必要がある場合は、ip_address を設定できます。)

9.2.4 新しく始める

bare プラットフォームのクラスターで新しくスタートするには、適切な外部ツールを使用してサーバーを再インストール、再イメージ、または再プロビジョニングし、このドキュメントで説明しているプロセスを繰り返します。新しいサーバーの IP アドレスが異なる場合、または複雑な ssh セットアップがある場合は、*tpaexec deprovision* を実行してローカルに作成されたすべてのファイルを削除してから *tpaexec provision*

それらを再作成し、上記と同様にこのドキュメントのプロセスを繰り返します。

9.3 Docker

TPA は Docker コンテナを作成し、それらにクラスターを展開できます。現時点では、通常の VM であるかのように systemd およびその他のサービスを実行するようにコンテナをセットアップします。

Docker コンテナへの展開は、さまざまなクラスター構成をテストする簡単な方法です。実稼働使用を目的としたものではありません。

9.3.1 概要

構成時にプラットフォームを選択するだけです。

```
tpaexec configure clustername --platform docker [···]
tpaexec provision clustername
tpaexec deploy clustername
```

9.3.2 オペレーティングシステムの選択

標準の--os Debian/Ubuntu/RedHat/SLES configure オプションを使用して、コンテナに使用するディストリビューションを選択します。TPA は、このディストリビューション用に独自の systemd 対応イメージをビルドします。これらのイメージには、tpa/redhat:8 など、tpa/ 接頭辞が付いた名前が付けられます。

代わりに--os-image some/image:name を使用して、既存の systemd 対応イメージを指定します。例、centos/systemd

イメージ CentOS 7 ベースは、この方法で使用できます。

9.3.3 Docker のインストール

最新の安定した Docker-CE パッケージを使用して TPA をテストします。

このドキュメントでは、動作する Docker インストールがあり、イメージのプルやコンテナの作成などの基本的な操作に精通していることを前提としています。

install Docker および get started のヘルプが必要な場合は、Docker documentation を参照してください。

MacOS X では、次のことができます install "Docker Desktop forMac"

そして、アプリケーションメニューから Docker を起動します。

Cgroup

すべての *Distribution support* は、cgroups バージョン 2 のサポートを提供します。古いオペレーティングシステムを使用している場合を除き、Docker に展開するときは、ホストマシンで常に cgroups バージョン 2 を使用する必要があります。

!!!重要

古いシステムを使用する手順は、純粋にテストまたはこれらのプラットフォームからの移行をサポートするために提供されています。

Debian ファミリーの Linux ディストリビューションの場合

```
echo GRUB_CMDLINE_LINUX=systemd.unified_cgroup_hierarchy=false > \
  /etc/default/grub.d/cgroup.cfg
update-grub
reboot
```

RedHat ファミリーの Linux ディストリビューションの場合

```
grubby --args=systemd.unified_cgroup_hierarchy=false --update-kernel=ALL
reboot
```

MacOS の場合

1. ~/ ライブラリー/Group\ Containers/group.com.docker/settings.json を編集し、次の置き換えを行います`"deprecatedCgroupv1": false` → `"deprecatedCgroupv1": true`
2. Docker デスクトップアプリを再起動します

アクセス許可

TPA は、それを実行しているユーザーが Docker デーモンへのアクセス権限を持っていることを想定しています通常は、`/var/run/docker.sock` を所有する `docker` グループのメンバーであること。次のようなコマンドを実行して、アクセスがあるかどうかを確認します。

```
docker version --format {{.Server.Version}}
```

```
19. 03.12
```

> **警告**
>
> ユーザーに Docker デーモンと対話できる機能を付与すると、Docker ホストでの root を簡単に取得できます。信頼できるユーザーのみが Docker デーモンにアクセスできる必要があります。

特権コンテナ

デフォルトでは、TPA は Linux 機能フラグを追加せずに、非特権モードで Docker コンテナをプロビジョニングします。このようなコンテナは、ホストファイアウォールルール、ファイルシステム、ブロックデバイス、またはホストで真の root 権限を必要とするその他のほとんどのタスクを管理できません。

コンテナを特権モードで実行する必要がある場合は、それを必要とするインスタンスまたは `instance_defaults` でグローバルに `privileged` ブール変数を設定します。例

```
instance_defaults:
  privileged: true
```

> **警告**
>
> 特権モードでコンテナを実行すると、 root ユーザーまたは root を取得できるプロセスは、カーネルモジュールのロード、ホストファイアウォールルールの変更、コンテナ名前空間のエスケープ、または実際のホスト「root」ユーザーと同じように動作できます。本当に必要な場合を除き、特権モードでコンテナを実行しないでください。

`security_opts` および `no-new-privileges` フラグ

tpaexec は、プロセスが特権を増加できない制限モードで Docker コンテナを起動できます。setuid バイナリは制限されているなど。 `instance_defaults` またはコンテナごとの変数 `docker_security_opts` を使用して tpaexec でこれを有効にします。

```
instance_defaults:
  docker_security_opts:
    - no-new-privileges
```

`docker run` の`--security-opts` への他の引数も受け入れられます。SELinux のユーザーとロール。

Linux 機能フラグ

tpaexec は、 `docker_cap_add` リスト変数を使用して Linux 機能フラグを Docker の制御を公開します。このリスト変数は、コンテナごとまたは `instance_defaults` で設定できます。機能フラグの詳細については、 `man capabilities` 、`docker run` ドキュメント、および Ansible `docker_containers` モジュールのドキュメントを参照してください。

Docker の--cap-drop は、 docker_cap_drop リストを介してサポートされています。

たとえば、コンテナを非特権として実行しながら、システムクロックを変更する機能を付与するには、次のように書くことができます。

```
instance_defaults:
  privileged: false
  docker_cap_add:
    - sys_time
  docker_cap_drop:
    - all
```

Docker ストレージ構成

多くのホスト上のデフォルトの Docker 構成は lvm-loop ブロックストレージを使用し、運用展開には適していません。 docker info を実行して、使用しているストレージドライバーを確認します。ループバックスキームを使用している場合、次のようなものが表示されます。

```
Storage Driver: devicemapper
 ...
 Data file: /dev/loop0
```

ストレージ構成の詳細については、Docker ドキュメントを参照してください。

- Storage Drivers
- Configuring lvm-direct for production

Docker MTU 設定

デフォルトでは、Docker ネットワークの最大転送ユニット MTU は 1500 バイトです。これがホストシステムのネットワークインターフェイスの MTU より大きい場合、そのインターフェイスを介して Docker コンテナに接続をルーティングする際に問題が発生する可能性があります。ipconfig | grep mtu、ip |grep mtu などのコマンドを使用して、ネットワークインターフェイスの MTU を確認できます。以下に示すように、config.yml のネットワークに適切なドライバーオプションを追加することにより、TPA によってプロビジョニングされた Docker ネットワークの MTU を変更できます。

```
docker_networks:
  - ipam_config:
      - subnet: 10.33.214.192/28
    name: tpa-docker
    driver_options:
      com.docker.network.driver.mtu: 1400
```

> **警告**
>
> MTU は、Docker ネットワークが最初にプロビジョニングされるときにのみ設定できます。 config.yml の後続の変更は効果ありません。

9.3.4 Docker コンテナ管理

クラスター内のすべての Docker コンテナは、 `start-containers` および `stop-containers` コマンドを使用して、一緒に起動および停止できます。

```
tpaexec start-containers clustername
tpaexec stop-containers clustername
```

これらのコマンドは、コンテナをプロビジョニングまたはプロビジョニング解除し、さらにはコンテナに接続しません。これらは、将来の使用に備えて維持する必要がある Docker クラスターを一時的に使用しない場合に、リソースを節約することを目的としています。

クラスター内にプロビジョニングされた Docker コンテナの概要を表示するには、起動または停止にかかわらず、`list-containers` コマンドを使用します。

```
tpaexec list-containers clustername
```

第10章　CUSTOMIZATIONS

10.1 Cluster configuration

TPA では、クラスターの構成を変更する方法は、config.yml を編集し、プロビジョニング/展開/テストサイクルを実行することです。このプロセスは、べき等であるように、構成の変更またはインスタンスの変更に応じてのみ変更を行うように注意深く設計されています。

クラスター構成 コマンドは、賢明な config.yml ファイルを生成しますが、最も一般的なトポロジと構成オプションのみをカバーしています。デフォルト以外のものが必要な場合、またはクラスターをプロビジョニングした後に変更を行う必要がある場合は、とにかく config.yml を編集する必要があります。

このページは、使用可能な構成メカニズムの概要を示しています。特定の *Instance configuration* の詳細を含む別のページがあります。

10.1.1 config.yml

`config.yml` ファイルは、目的のクラスター構成のすべての側面を表す YAML format テキストファイルです。次に、2 つのインスタンスを持つクラスターの最小限の例を示します。

```
cluster_name: speedy

cluster_vars:
  postgres_version: 14

instances:
  - node: 1
    Name: one
    role: primary
    platform: docker
    vars:
      ansible_user: root
      x: 42

  - node: 2
    Name: two
    role: replica
    platform: docker
    upstream: one
    vars:
```

(次のページに続く)

(前のページからの続き)

```
    ansible_user: root
    x: 53
```

これら 3 つの定義は、クラスター構成の中心となります。ファイルには、他の多くの定義プラットフォーム固有の詳細を含む場合がありますが、1 つのインスタンスまたはクラスター全体に設定された instances と vars のリストは、すべての TPA 構成の基本ビルディングブロックです。

すべて クラスター構成

オプションは、何らかの方法で config.yml 変数に変換されます。1 つのオプションは複数の変数に影響を与える場合がありますたとえば、--bdr-version は postgres_version 、edb_repositories 、extra_postgres_extensions などを設定できますが、コマンドを実行してできることはエディターでいつでも実行できます。

YAML 構文の観点では、config.yml 全体が cluster_vars や instances などのキーを含むハッシュを表します。**各キーが 1 度だけ定義されていることを確認する必要があります。**誤って cluster_vars を繰り返した場合、2 番目の定義が前者を完全にオーバーライドし、次の展開はシャドウされた変数が欠落しているために意図しない変更を行う可能性があります。

TPA は、クラスタートポロジ全体の一貫性をチェックしますたとえば、ロール「レプリカ」でインスタンスを宣言する場合、その上流インスタンスの名前も宣言し、そのインスタンスが存在する必要があります。ただし、それは妨げるものではありませんインスタンスに好きな変数を設定します。十分な注意を払い、変更を運用環境にロールアウトする前にテスト環境で変更を試す必要があります。

10.1.2 変数

Ansible の用語では、ほとんどの構成設定は「インベントリー変数」です。TPA は、プロビジョニング中にインベントリーで cluster_vars を group_vars クラスター全体に適用し、各インスタンスの vars を host_vars に変換し、デプロイメントはインベントリー値を使用します。config.yml を変更した後、**``tpaexec deploy`` の前に** tpaexec provision を実行することを忘れないでください。

クラスター全体、個々のホスト、またはその両方に変数を設定できます。ホスト変数はグループ変数をオーバーライドします。実際には、cluster_vars で x: 42 を設定することは、すべてのホストの vars で設定することと同じです。デプロイメント中に x を必要とするホストには、いずれの場合も値 42 が表示されます。ホストは常に最も具体的な値を参照するため、グループにいくつかのデフォルト値を設定し、必要に応じて特定のインスタンスにオーバーライドすると便利です。

可能な限り、 cluster_vars で変数を定義し、特定のインスタンスでそれらをオーバーライドすると、確認と変更が簡単な簡潔な構成が得られます。繰り返しが少なくなります。それを超えて、特定の設定がグループ変数またはホスト変数としてより意味があるかどうかは、あなたが決定する必要があります。

10.1.3 クラスター変数

cluster_vars の下のキーは、有効な YAML タイプにマップでき、Ansible インベントリのグループ変数に直接変換されます。

```
cluster_vars:
  postgres_version: 14
```

(次のページに続く)

(前のページからの続き)

```
  edb_repositories:
    - enterprise
    - postgres_distributed
  postgres_conf_settings:
    bdr.trace_replay: true
```

警告

テンプレートで使用される変数は、*config.yml* のトップレベル *cluster_name* 変数と同じレベルで定義する必要があります

以下の例を参照してください。

この場合、tpaexec provision は 3 つの変数文字列、リスト、およびハッシュを group_vars/tag_Cluster_name/01-cluster_name.yml のインベントリに書き込みます。

10.1.4 インスタンス変数

このドキュメントでは、config.yml の特定のインスタンスに対して定義された変数を参照するために「インスタンス変数」という用語を使用しています。例、これは一般的なインスタンス定義です。

```
instances:
  - Name: unwind
    node: 1
    backup: unkempt
    location: a
    role:
      - primary
      - bdr
    volumes:
      - device_name: root
        encrypted: true
        volume_size: 16
        volume_type: gp2
      - device_name: /dev/xvdf
        encrypted: true
        vars:
          volume_for: postgres_data
        volume_size: 64
        volume_type: gp2
    platform: aws
    type: t3.micro
    vars:
      ansible_user: ec2-user
      postgres_conf_directory: /opt/postgres/conf
```

このインスタンスの vars で定義された変数はすべてインベントリーのホスト変数になりますが、インベントリー内のすべてのホスト変数は vars のみから取得されるわけではありません。 platform 、location 、volumes 、role などの他の一部のインスタンス設定もホスト変数としてインベントリにコピーされますが、代わりに vars または cluster_vars でこれらの設定を定義することはできません。

vars の外部の設定は、インスタンスのプロパティたとえば Name および node またはクラスターのトポロジ内のその場所たとえば role 、backup またはそれらはプラットフォーム固有の属性たとえばインスタンス type および volumes などを記述する場合があります。 。 vars では定義できないことを除き、これらのインスタンスの「設定」とインスタンス「変数」を区別する必要があることはほとんどありません。

この場合、tpaexec provision は多くのホスト変数を host_vars/unwind/01-instance_vars.yml のインベントリに書き込みます。

10.1.5 instance_defaults

これは、config.yml の繰り返しをさらに削減するメカニズムです。cluster_vars として定義できないインスタンス設定に最も役立ちます。例、次のように書くことができます。

```
instance_defaults:
  platform: aws
  type: t3.micro
  tags:
    AWS_ENVIRONMENT_SPECIFIC_TAG_KEY: some_mandated_value

instances:
  - node: 1
    Name: one
  - node: 2
    Name: two
  - ...
```

instance_defaults の下で指定したものは、 instances のすべてのエントリーのデフォルトとして機能します。この例では、各インスタンスの platform および type のスペルアウトを保存し、すべてのインスタンスを別のタイプに変更するのが簡単になります。インスタンスが別の値を指定している場合、もちろんデフォルトより優先されます。

instance_defaults は、instances を定義する際に使用するマクロ機能であると考えると役立つでしょう。最終的にインベントリに書き込まれるものは、instances の展開された定義のみから来ます。 cluster_vars または instance_defaults に何かを入れるかどうかを決定しようとしている場合、それはおそらく前者に属します変数たとえば、platform または type など_cannot_これは、多くのプラットフォーム固有のプロパティなどに当てはまります AWS リソースタグ) は、プロビジョニングでのみ使用され、展開中は使用されません。

instance_defaults メカニズムは、インスタンスの vars を埋めるために使用することを妨げるものではありません。デフォルトのハッシュ値は、instances エントリーで指定されたハッシュとマージされます。ただし、cluster_vars で同じデフォルトを設定し、必要に応じてインスタンスにオーバーライドするよりもこれを行うことに特に利点はありません。よくわからない場合は、cluster_vars を使用してください。

10.1.6 ロケーション

config.yml で locations のリストを指定することもできます。

```
locations:

- Name: first
  az: eu-west-1a
  region: eu-west-1
  subnet: 10.33.110.128/28


- Name: second
  az: us-east-1b
  region: us-east-1
  subnet: 10.33.75.0/24

instances:

- node: 1
  Name: one
  location: first
…
```

インスタンスが location: first または location: 0 を指定する場合、その場所の設定はそのインスタンスのデフォルトとして機能します。ここでも、instance_defaults と同様に、インスタンスはその場所から継承するデフォルトをオーバーライドできます。また、この機能を使用して、インスタンスの vars を入力できます。これは、インスタンスの半分にのみ適用されるデフォルトと、残りの半分に異なる値がある場合に役立ちます。上記の例のプラットフォーム固有の設定の場合。

ロケーションは、インスタンスが「オプトイン」できる設定のコレクションを表します。それらを使用して、さまざまなデータセンター、AWS リージョン、Docker ホスト、または完全に他のものを表すことができます。TPA は、特定の解釈を想定または強制しません。

10.2 Instance configuration

このページでは、クラスターインスタンスでの展開プロセスをカスタマイズするために TPA が提供するさまざまな制御の概要を、より詳細なドキュメントへのリンクとともに紹介します。

デプロイメントの詳細に入る前に、クラスター構成 を読んで、config.yml のクラスター変数、インスタンス変数、およびその他のメカニズムがどのように連携して、簡潔でレビューが簡単な構成を書くことができるようにする方法を理解すると役立ちます。

10.2.1 システムレベルの構成

TPA が最初に行うことは、Python がブートストラップされ、Ansible モジュールディストリビューション固有のプロセスを実行する準備ができていることを確認することです。次に、以下の *Postgres* **のフレーバーとバージョン** に進む前に、さまざまなシステムレベルの構成タスクを完了します。

- *Distribution support*
- *TPA Python* **環境のセットアップ** `preferred_python_version`
- *Environment variables* 例 `https_proxy`

パッケージリポジトリ

展開前 を使用できます

パッケージリポジトリが構成される前にタスクを実行します。

- *Configuring YUM repositories*

RHEL、Rocky、および AlmaLinux の場合

- *Configuring APT repositories*

Debian および Ubuntu の場合

- *Configuring EDB Repos 2.0 repositories*

任意のシステムで

- *Creating and using a local repository*

パッケージをターゲットインスタンスにシップするため

ポストリポジトリ を使用できます

パッケージリポジトリが構成された後にタスクを実行するにはたとえば、パッケージをインストールする前にリポジトリ構成の問題を修正します。

パッケージのインストール

リポジトリが構成されると、パッケージは展開のさまざまな段階でインストールされます。システムパッケージのバッチから始めます。

- *Installing packages*

例、 acl、openssl、sysstat

Postgres およびその他のコンポーネント Barman、repmgr、pgbouncer などは、クラスター構成に応じて個別にインストールされます。これらは、以下の独自のセクションに文書化されています。

その他のシステムレベルのタスク

- *Filesystem configuration* RAID、LUKS セットアップを含む
- *Uploading artifacts* ファイル、ディレクトリ、tar アーカイブ
- *Setting sysctl values*
- *Configuring /etc/hosts*
- *Managing SSH host keys*
- *Locale*

デプロイメントを完全にスキップする

TPA がインスタンスの展開プロセスの一部を実行しないようにするには、つまり、TPA にインスタンスをプロビジョニングして、そのままにしておく場合、インスタンスの `provision_only` 設定を `config.yml` で `true` に設定します。この設定により、TPA は、 `tpaexec deploy` が認識するインベントリからインスタンスを完全に省略します。

10.2.2 Postgres

Postgres 構成は、repmgr や pgbouncer などの他のコンポーネントのセットアップと連携する拡張プロセスです。それは Postgres 自分自身をインストールすることから始まります。

バージョンの選択

ソフトウェアバージョン を使用して Postgres フレーバーとバージョンを選択するか、config.yml で `postgres_version` を設定して、インストールする Postgres メジャーバージョンを指定します。

作業用クラスターをセットアップするために実際に必要な作業はこれだけです。このページのその他の部分はすべてオプショナル。必要に応じて、展開のすべての側面を制御できますが、デフォルトは注意深く調整されて、開始点として合理的なクラスターを提供します。

インストール

デフォルトの `postgres_installation_method` では、アーキテクチャのニーズに応じて、選択した Postgres のバージョンのパッケージをさまざまな拡張機能とともにインストールします。

- *Installing Postgres-related packages*

例、 pglogical、 BDR など

- *Postgres source installation*

開発およびテスト用

どのインストール方法を選択するかにかかわらず、TPA は最小限の労力で同じクラスター構成を提供できます。

構成

- *The postgres Unix user*
- *Running initdb*
- *pg_hba.conf*
- *pg_ident.conf*
- *postgresql.conf*

postgres-config を使用できます

Postgres 構成ファイルがインストールされた後にタスクを実行するには追加の構成ファイルをインストールします。

Postgres の構成が配置されると、TPA は、アーキテクチャの詳細に従って、Barman、repmgr、pgbouncer、haproxy などの他のコンポーネントのインストールと構成を開始します。

10.2.3 その他のコンポーネント

- **共有 *Barman* または共有 *PEM* クラスター**
- ***Repmgr* リダイレクト *pgbouncer***
- *etcd*
- *Patroni cluster management commands*
- *Configuring haproxy*
- *Configuring HARP*
- *Configuring EFM*

サービスの構成と開始

TPA は、各サービスの systemd サービスユニットファイルをインストールします。Postgres のサービスの名前は `postgres.service` で、`systemctl start postgres` で開始または停止できます。

最初の展開では、Postgres サービスが開始されます。 `tpaexec deploy` を再度実行している場合、行った構成変更によっては、サービスがリロードまたは再起動される場合があります。もちろん、サービスが既に実行されており、変更がない場合は、放置されます。

いずれの場合も、この手順の最後に Postgres は実行されます。

10.2.4 Postgres を起動した後

- *Creating Postgres users*
- *Creating Postgres tablespaces*
- *Creating Postgres databases*
- *pglogical*

- *Configuring .pgpass*

postgres-config-final を使用できます

起動後の Postgres 構成が完了した後にタスクを実行するため SQL クエリを実行してオブジェクトを作成またはデータをロードするためなど。

- *EDB Postgres Distributed configuration*

展開後 を使用できます

展開プロセスが完了した後にタスクを実行します。

10.3 Building from source

> **警告**
>
> このオプションは、開発者および上級ユーザーを対象としています。 EDB によって構築およびテストされたソフトウェアのみが EDB によってサポートされています。 Self-Managed Supported Open Source Software を参照してください。

特定のコンポーネントのパッケージインストールを他のコンポーネントのソースビルドと組み合わせることもできます。たとえば、パッケージから Postgres をインストールし、ソースから pglogical と PGD をコンパイルできますが、パッケージの依存関係により、ソースから pglogical をインストールし、パッケージから PGD をインストールできません。

ソースビルドは、開発、テスト、およびサポート操作で使用することを目的としています。

10.3.1 クイックスタート

2ndQPostgres、pglogical3、および bdr はすべて安定したブランチからビルドされたものを使用してクラスターをスピンアップします。

```
tpaexec configure ~/clusters/speedy -a BDR-Always-ON          \
    --layout bronze                                             \
    --harp-consensus-protocol etcd                              \
    --install-from-source                                       \
      2ndqpostgres:2QREL_13_STABLE_dev                          \
      pglogical3:REL3_7_STABLE                                  \
      bdr3:REL3_7_STABLE
```

上記と同様ですが、公式 git リポジトリから 2ndQPostgres ソースコードをビルドし、指定されたローカルワークツリーを使用して pglogical および BDR をビルドするクラスターをセットアップします。この機能は Docker に固有です。

```
tpaexec configure ~/clusters/speedy                           \
    --architecture BDR-Always-ON --layout bronze                \
    --harp-consensus-protocol etcd                              \
```

(次のページに続く)

(前のページからの続き)

```
    --platform docker                                          \
    --install-from-source 2ndqpostgres:2QREL_13_STABLE_dev     \
      pglogical3 bdr3                                          \
    --local-source-directories                                 \
      pglogical3:~/src/pglogical bdr3:~/src/bdr
```

クラスターを展開した後、後続の実行で `tpaexec deploy … --skip-tags build-clean` を使用して、ビルドディレクトリを再利用できます。それ以外の場合、ビルドディレクトリはビルドを開始する前に空にされます。

カスタムの場所とビルドパラメーターを使用して Postgres、pglogical、BDR、およびその他のコンポーネントをビルドする方法の詳細な説明を読んでください。

10.3.2 構成

ソースビルドの構成には 2 つの側面があります。

特定のソースの組み合わせを実行するクラスターのみが必要な場合は、`tpaexec configure` を実行して、選択したコンポーネントをダウンロード、コンパイル、およびインストールするための合理的なデフォルトで構成を生成します。上記の例に示すように、Postgres または Postgres Extended、pglogical、および BDR をビルドし、ビルドするブランチ名を指定できます。

基になるメカニズムは、コマンドラインオプションで許可される以上の機能を備えています。config.yml を編集することにより、さまざまなソースリポジトリのクローンを作成したり、ビルドの場所を変更したり、別の configure または build パラメーターを指定したり、build コマンドを完全に再定義したりできます。したがって、Postgres、pglogical、および BDR 以外のものをビルドできます。

使用可能なオプションはここに文書化されています。

- *Postgres source installation*
- ソースからのビルドとインストール

10.3.3 ローカルソースディレクトリ

TPA を使用して、Git リポジトリからではなくローカルソースディレクトリから Postgres および/または拡張機能をビルドする Docker コンテナをプロビジョニングできます。

`--install-from-source` を使用して、ビルドするものを宣言しているとします。

```
tpaexec configure ~/clusters/speedy                            \
    --architecture BDR-Always-ON --layout bronze               \
    --harp-consensus-protocol etcd                             \
    --platform docker                                          \
    --install-from-source 2ndqpostgres:2QREL_13_STABLE_dev     \
      pglogical3:REL3_7_STABLE bdr3:REL3_7_STABLE              \
      …
```

デフォルトでは、これは Postgres Extended、pglogical3、および bdr3 の既知のリポジトリのクローンを作成し、指定されたブランチをチェックアウトして、ビルドします。ただし、--local-source-directories を追加して、代わりにホストマシンからソースを直接取得するように指定できます。

```
tpaexec configure ~/clusters/speedy                                \
    --architecture BDR-Always-ON --layout bronze                    \
    --harp-consensus-protocol etcd                                  \
    --platform docker                                               \
    --install-from-source 2ndqpostgres:2QREL_13_STABLE_dev          \
      pglogical3 bdr3                                               \
    --local-source-directories                                      \
      pglogical3:~/src/pglogical bdr3:~/src/bdr                     \
    …
```

この構成でも Postgres Extended はリポジトリからインストールされますが、ホスト上の指定されたディレクトリから pglogical3 および bdr3 ソースを取得します。これらのディレクトリは、git リポジトリがクローン作成されるのと同じ場所にある Docker コンテナに読み取り専用でバインドマウントされ、デフォルトのツリー外ビルドが通常のように続行されます。

コンポーネントのローカルソースディレクトリを指定する場合、ビルドするブランチを指定することはできません。上記の例の--install-from-source の pglogical3:REL3_7_STABLE と pglogical3 を参照してください。ソースディレクトリはコンテナに読み取り専用でマウントされるため、TPA はそれを変更することはできません。git pull も git checkout も。ローカルにチェックアウトしたブランチ、コミットされていない変更などを取得します。

--local-source-directories を使用すると、config.yml に Docker ボリューム定義のリストが含まれます。

```
local_source_directories:
  - /home/ams/src/pglogical:/opt/postgres/src/pglogical:ro
  - /home/ams/src/bdr:/opt/postgres/src/bdr:ro
  - ccache-bdr_src_36-20200828200021:/root/.ccache:rw
```

ccache

TPA は、すべての種類のソースビルドにデフォルトで ccache をインストールします。ローカルソースディレクトリで Docker クラスターを使用している場合、デフォルトでは、新しい Docker ボリュームがクラスターのコンテナに接続され、共有 ccache ディレクトリとして機能します。このボリュームはホストから完全に分離され、クラスターがプロビジョニング解除されるときに削除されます。

--shared-ccache /path/to/host/ccache configure オプションを使用して、有効期間の長い共有 ccache ディレクトリを指定します。このディレクトリは、コンテナに r/w バインドマウントされ、そのコンテンツはホストとコンテナの間で共有されます。

デザインとして、ホストでコンパイルされたバイナリをコンテナに直接インストールする方法はありません。

10.3.4 リビルド

ソースからビルドされたコンポーネントを使用してクラスターを展開した後、`tpaexec rebuild-sources` コマンドを使用して `tpaexec deploy` を再実行せずに、これらのコンポーネントをすばやくリビルドできます。これにより、コンテナ上の git リポジトリからビルドされたコンポーネントに対して `git pull` が実行され、すべてのコンポーネントがリビルドされます。

10.4 TPA hooks

TPA は、ユーザーの介入なしで完全な機能のクラスターをセットアップでき、カスタムリポジトリとパッケージ、カスタム Postgres 構成 pg_hba.conf と postgresql.conf の両方を含むさまざまな *Instance configuration* を既に提供しています。

フックスクリプトを作成して、使用可能な構成設定では満たされない特定のニーズに対応できます。フックを使用すると、展開中に任意の Ansible タスクを実行できます。

フックは TPA の究極の拡張メカニズムであり、それらを使用してできることに制限はありません。それらは注意して使用し、追加のメンテナンス負担が発生することに注意してください。TPA 開発者はフックコードについての洞察がなく、予想される段階でフックを呼び出すことを超えて、リリース間の互換性を保証できません。

10.4.1 概要

クラスターディレクトリの `hooks` サブディレクトリの下に特定の名前のファイルを作成すると、TPA は以下で説明するように、展開プロセスのさまざまな段階でそれらを呼び出します。

```
mkdir ~/clusters/speedy/hooks
cat > ~/clusters/speedy/hooks/pre-deploy.yml
```

```
- --


-  debug: msg="hello world!"
```

フックスクリプトは `include_tasks` で呼び出されるため、Ansible タスクのリストを含む YAML ファイルであることが想定されますプレイのリストを含むプレイブックではありません。以下に特に記載されていない限り、フックは展開内のすべてのホストに対して無条件に実行されます。

10.4.2 汎用フック

展開前

TPA は、Python のブートストラップとロケールの構成の後、リポジトリの構成とパッケージのインストールの前に、`hooks/pre-deploy.yml` を呼び出します。これは、独自のコードを実行できる最初の段階です。

このフックを使用して、*Configuring APT repositories* または *Configuring YUM repositories* でできることを超えて、カスタムリポジトリー構成をセットアップできます。

ポストリポジトリ

TPA は、パッケージリポジトリの構成後に hooks/post-repo.yml を呼び出します。これを使用して、パッケージのインストールを開始する前にリポジトリー構成を修正できます。

pre-initdb

TPA は、*Running initdb* が存在しない場合、 *Running initdb* を決定する前に hooks/pre-initdb.yml を呼び出します。通常、このフックを使用する必要はありませんが、これを使用して PGDATA を自分自身で作成すると、TPA は initdb をスキップします。

postgres-config

TPA は、pg_hba.conf および conf.d のファイルを含む Postgres 構成ファイルを生成した後、サーバーが起動する前に hooks/postgres-config.yml を呼び出します。

このフックを使用して、たとえば、 conf.d の下に追加の構成ファイルを作成できます。

postgres-config-final

TPA は、Postgres を起動し、ユーザー、データベース、および拡張機能を作成した後、hooks/postgres-config-final.yml を呼び出します。このフックを使用して、SQL コマンドを実行できます。たとえば、カスタム拡張機能の構成を実行したり、データベースオブジェクトを作成したりできます。

barman-pre-config

TPA は、 Barman をインストールして Barman ユーザーを設定した後、Barman 構成を生成する前に hooks/barman-pre-config.yml を呼び出します。

このフックを使用して、たとえば、Barman 証明書ファイルまたはマウントポイントに関連するタスクを実行できます。

efm-pre-config

TPA は、efm のインストール、構成ディレクトリの作成、および efm ユーザーのセットアップの後、efm 構成を生成する前に hooks/efm-pre-config.yml を呼び出します。

このフックの使用例は、efm ヘルパースクリプトをインストールすることです。

efm-post-config

TPA は、efm 構成ステップが完了した後、hooks/efm-post-config.yml を呼び出します。

このフックの使用例は、efm 構成プロセスを終了する前に考慮する必要がある追加の手順です。

harp-config

TPA は、 HARP 構成ファイルを生成した後、HARP サービスが開始される前に hooks/harp-config.yml を呼び出します。

このフックを使用して、たとえば、TPA のビルトインインターフェイスによって提供されない HARP プロキシのカスタマイズを実行できます。

このフックは、PGD ノードを含む HARP パッケージをインストールするノードで実行されることに注意してください。

pgd-proxy-config

TPA は、 PGD PROXY 構成ファイルを生成した後、 PGD PROXY サービスが開始される前に、 hooks/pgd-proxy-config.yml を呼び出します。

このフックを使用して、TPA のビルトインインターフェイスでは提供されていない PGD プロキシのカスタマイズを実行できます。

このフックは、PGD PROXY パッケージをインストールするノードで実行されることに注意してください。

展開後

TPA は、展開の最後に hooks/post-deploy.yml を呼び出します。

この段階の後は、好きなことを続けることができます。

このフックを使用して、TPA 展開中に生成または変更された構成ファイルを変更すると、次の tpaexec deploy が変更を上書きするリスクがあります TPA はフックが何をしたのかを知りません。

10.4.3 PGD フック

これらのフックは、PGD 展開に固有です。

bdr-pre-node-creation

TPA は、インスタンスに初めて PGD ノードを作成する前に、すべてのインスタンスで hooks/bdr-pre-node-creation.yml を呼び出します。必要なすべての PGD ノードが既に存在する場合、フックは呼び出されません。

bdr-post-group-creation

TPA は、 first_bdr_primary インスタンスで PGD ノードグループを作成した後、すべてのインスタンスで hooks/bdr-post-group-creation.yml を呼び出します。必要な PGD グループが既に存在する場合、フックは呼び出されません。

bdr-pre-group-join

TPA は、レプリケーションセットを作成、変更、または削除し、必要なサブスクリプションを構成した後、ノードに参加する前に、すべてのインスタンスで `hooks/bdr-pre-group-join.yml` を呼び出します。

このフックを使用して、SQL コマンドを実行し、ノード参加が開始する前に必要になるレプリケーションセットの構成とサブスクリプションに対してその他の調整を実行できます。

たとえば、PGD 監視レプリケーションセットを調整して、新しいテーブルを自動的に追加し、一般的に DDL フィルタを作成できます。

10.4.4 その他のフック

postgres-pre-update、postgres-post-update

Upgrading your cluster コマンドは、パッケージをインストールする前に特定のインスタンスで `hooks/postgres-pre-update.yml` を呼び出し、パッケージのインストールが完了した後に `hooks/postgres-post-update.yml` を呼び出します。どちらのフックも、更新されるインスタンスでのみ呼び出されます。

これらのフックを使用して、環境に合わせて更新プロセスをカスタマイズできますたとえば、他のパッケージをインストールし、TPA が管理しないサービスを停止および再起動します。

10.4.5 新しいフック

EDB は、必要に応じて TPA に新しいフックを追加します。ユースケースが既存のフックでカバーされていない場合は、私たちに連絡して問題について話し合ってください。

第11章　CLUSTER MANAGEMENT

11.1 Upgrading your cluster

`tpaexec upgrade` コマンドは、TPA クラスターで実行されているソフトウェアをアップグレードするために使用されます。`tpaexec deploy` はアップグレードを実行しません。

このコマンドは、以前の `tpaexec update-postgres` コマンドを置き換えます。

> **注釈**
>
> TPA は、共有 Barman および/または共有 PEM 構成を持つクラスターでの `tpaexec upgrade` コマンドの使用をまだサポートしておらず、将来のリリースでこの機能を追加します。

config.yml に変更を加えた場合、それらの変更を適用する方法は、`tpaexec provision` に続いて `tpaexec deploy` を実行することです。

このルールの例外は、 `tpaexec deploy` が既にインストールされているパッケージの別のバージョンのインストールを拒否することです。代わりに、`tpaexec upgrade` を使用してソフトウェアアップグレードを実行する必要があります。

次のコンポーネントは、 **任意の**アーキテクチャでアップグレードできます。

- Postgres
- *Repmgr* **リダイレクト** *pgbouncer*
- **共有** *Barman* **または共有** *PEM* **クラスター**
- *PG Backup API*
- **共有** *Barman* **または共有** *PEM* **クラスター**

サーバーとエージェントの両方

次のコンポーネントは **M1** アーキテクチャでアップグレードでき、使用されるフェールオーバーマネージャーに応じて異なります。

- *efm-pre-config*
- *Patroni cluster management commands*
- *etcd* パトローニ用
- *Repmgr* **リダイレクト** *pgbouncer*

次のコンポーネントは、使用する BDR バージョンに応じて、 **BDR -Always-ON / PGD-Always-ON** アーキテクチャでアップグレードできます。

- *pgdcli* BDR 4 の場合は v1、PGD5 の場合は v5
- *pgd-proxy-config* PGD5 のみ
- M1 アーキテクチャと、M1 に該当するすべてのフェールオーバーマネージャー、`upgrade` は、`Postgres` 、対応するフェールオーバーマネージャー `EFM` 、`Patroni` または `repmgr` 、および選択された非アーキテクチャ固有のコンポーネントのマイナーバージョンのアップグレードを実行できます。
- PGD アーキテクチャでは、 `upgrade` は、Postgres と BDR 拡張機能、および明示的にオプトインされている場合 `pgd-cli` および `pgd-proxy` のマイナーバージョンのアップグレードを実行します。
- PGD アーキテクチャでは、 `reconfigure` コマンドとの組み合わせでのみ、`upgrade` は BDR 拡張機能のメジャーバージョンのアップグレードを実行できます。

他のクラスターコンポーネントのアップグレードのサポートが TPA `v23.41.0` に追加されました _特定のコンポーネント `EFM` などは例外です。詳細については、各コンポーネントのドキュメントの `upgrade` セクションを参照してください。

アップグレードするときは、アップグレードを開始する前に常に barman を使用してバックアップをとり、アップグレードのために確保されている時間中に実行されるスケジュールされたバックアップを無効にする必要があります。

一般に、TPA はインスタンスごとに続行し、影響を受けるサービスの停止、新しいパッケージのインストール、必要に応じて構成の更新、サービスを再起動し、ランタイム構成の変更を実行してから、次のインスタンスで同じことを行う前に。プロセス中の任意の時点で、クラスターのノードの 1 つだけが使用不可になります。

クラスターを PGD-Always-ON にアップグレードする場合、既存の PGD-Always-ON クラスターをアップグレードする場合、または PGD-S または PGD-X クラスターのマイナーアップグレードを実行する場合、オプション `-e enable_proxy_monitoring=true` を `tpaexec upgrade` コマンドラインに追加して、アップグレードします。有効にすると、bdr データベースに追加のテーブルが作成され、アップグレードの実行中に監視データがそこに書き込まれます。モニタリングを有効にすることによるパフォーマンスへの影響は非常に小さいため、有効にすることをお勧めします。

11.1.1 アップグレードする前に

このセクションでは、`tpaexec upgrade` を実行する前にクラスターで実行する必要があるチェックについて説明します。これらを順番に行うと、ライブクラスターに影響を与える前に、最も一般的な問題をキャッチします。

クラスターの状態を確認する

アップグレードは、正常な状態にあるクラスターで実行するのが最も簡単です。`tpaexec test <cluster>` を実行して、標準のヘルスチェックを実行します。アップグレードプロセス自分自身はアップグレード前のヘルスチェックを実行しますが、事前に問題を検出すると、それに対処する時間が長くなります。

BDR 対応アーキテクチャの場合、すべてのノードがカタログに表示されることも確認します。

```
SELECT node_name, peer_state_name, peer_target_state_name
  FROM bdr.node_summary;
```

すべてのエントリは ACTIVE または読み取り専用スタンバイの場合は STANDBY を報告する必要があります。続行する前に、他の状態を調査します。

repmgr を使用する M1 クラスターの場合、プライマリで repmgr cluster show を実行し、すべてのノードが実行中および接続されていると表示されることを確認します。

バックアップの検証

アップグレードの前に、最近の Barman バックアップが存在し、復元可能な状態にあることを確認してください。Barman ホストの場合

```
sudo -u barman barman list-backup <server>
sudo -u barman barman check <server>
```

ここで失敗したステータスは、開始する前に調査する必要があります。アップグレードが進行すると、Barman WAL レシーバーが停止されるため、アップグレードウィンドウ中のバックアップランドスケープは、事前に存在したものに固定されます。

レプリケーションラグの検証

クラスターは高レベルで正常な場合がありますが、個々のレプリカは遅延している場合があります。アップグレードは、各インスタンスを順番にフェンスします。レプリカが大幅に遅れている場合、フェンスを解除した後に追いつくのに時間がかかり、全体のアップグレード時間が長くなります。

PGD クラスターの場合

```
SELECT slot_name, active, restart_lsn, write_lag, flush_lag, replay_lag
  FROM bdr.node_slots;
```

M1 クラスターの場合、プライマリで

```
SELECT application_name, state, write_lag, flush_lag, replay_lag
  FROM pg_stat_replication;
```

分ではなく秒単位で測定される遅延を目指します。開始する前に、大幅に遅れているレプリカを調査します。

パッケージバージョンのピンとレビュー

アップグレード中にインストールされるものを制御するには、tpaexec upgrade を実行する前に config.yml でパッケージバージョンを明示的に設定することをお勧めします。関連する変数は、postgres_package_version 、bdr_package_version 、pgd_proxy_package_version 、pgdcli_package_version 、およびコンポーネント固有の同等品です。

明示的なピンがないと、TPA は各コンポーネントの利用可能な最新のパッケージをインストールします。これにより、意図しないメジャーバージョンのアップグレードが発生する可能性があります。パッケージバージョンの選択 を参照

詳細については。

> **注釈**
>
> *config.yml* のバージョンを更新した後、 *tpaexec provision* を実行してインベントリを再生成します。

アップグレードするすべてのコンポーネントのターゲットバージョンのリリースノートを確認しますが、現在のバージョンとターゲットバージョン間のリリースノートも確認します。EDB の PGD および Postgres リリースノートでは、バージョン固有のアップグレードの考慮事項、つまりアップグレードの前後に変更する必要がある設定、非推奨、および動作の変更を呼びかけている場合があります。同じことは、アップグレードの実行に使用している TPA の新しいバージョンの TPA 独自のリリースノートにも適用されます。

ステージング環境でアップグレードをテストする

`tpaexec upgrade` は、クラスターをインスタンスごとに処理します。アップグレードはローリング中です。一度に 1 つのインスタンスのみがフェンスオフされます。ただし、合計ウォールクロック時間は、クラスターサイズ、アップグレードするコンポーネントの数、各インスタンスのストレージとネットワークの特性だけでなく、アップグレードシナリオが問題なく完了するかどうかにも依存します。

実稼働クラスターのアーキテクチャ、Postgres のフレーバーとバージョン、パッケージのピン留め、およびカスタムフックと一致する非実稼働環境でアップグレードを再現することを強くお勧めします。これにより、ライブトラフィックに影響を与える前に、予期せぬ事態構成済みリポジトリ内のパッケージの欠落、予期しない構成のドリフトをキャッチします。

スケジュールされたジョブを無効にする

アップグレードウィンドウ中にクラスターに対して実行されるようにスケジュールされているものはすべて一時停止する必要があります。代表的な商品

- データベースにヒットする Cron ジョブアナリティクス、レポート。
- アップグレードを認識しない外部バックアップツール。
- 長時間実行される移行またはバッチジョブ。

TPA 自分自身は、開始する前に Barman WAL レシーバーを停止し、その後再起動します。Barman を手動で無効にする必要はありません。

11.1.2 アップグレード中

このセクションでは、`tpaexec upgrade` の実行中に予想されることと、TPA の外部から注意すべきことについて説明します。

予想されること

TPA は、クラスターを一度に 1 インスタンスずつ処理します。インスタンスごとに、アップグレードは次のとおりです。

1. インスタンスをフェンスオフして、プロキシまたはフェールオーバーマネージャーが新しい接続の送信を停止します。
2. そのインスタンスで影響を受けるサービスを停止します。
3. 新しいパッケージをインストールします。
4. 必要に応じて構成ファイルを更新します。
5. サービスを再起動し、起動するまで待機します。
6. インスタンスのフェンスを解除します。

各インスタンスがフェンスされている間、およびサービスに戻ったときに、短い接続の中断が発生することを想定してください。接続プーリングと再試行ロジックを使用するアプリケーションは、通常、これらの移行中にクライアントに表示されるエラーを認識しないはずです。

非リーダーノードの各ノードがアップグレードされている間、書き込みトラフィックは、現在の書き込みリーダーを介して継続されます。書き込みリーダー自分自身がアップグレードされるたびに、短いリーダー選択の中断が発生します。PGD-S および PGD-X は、書き込みリーダーが最後にアップグレードされるように手配します。PGD-Always-ON はインベントリの順序でノードをアップグレードするため、現在のリーダーの番が巡って来るたびにリーダー選挙が発生します。

アプリケーション側で見るべきもの

アップグレードの実行中、各フェンス/アンフェンスの移行で短い一時停止が発生するのは通常です。複数のインスタンスにわたる持続的な接続障害はそうではありません。これらが表示された場合は、TPA が次のインスタンスに進む前に停止して調査します。

アップグレード中にクラスターに対してアプリケーションが実行されている場合、以下を監視します。

- アプリケーションエラー率一時的な再試行が予想されます。持続的なエラーはそうではありません。
- 接続プールメトリック 使用可能な接続の一時的な低下が予想されます。
- レプリケーションスロットの遅延各インスタンスがアップグレード後に追いつくと、一時的に増加する場合があります。

11.1.3 アップグレード後

このセクションでは、`tpaexec upgrade` が完了したら実行する必要がある検証について説明します。

インストールされたパッケージバージョンの確認

新しいパッケージがすべてのインスタンスにインストールされることを確認します。RHEL ファミリーホストの場合

```
tpaexec cmd <cluster> all -m shell -a rpm -qa | grep -E "^(postgres|edb-)"
```

Debian / Ubuntu ホストの場合

```
tpaexec cmd <cluster> all -m shell -a dpkg -l | grep -E "(postgres|edb-)"
```

同じパッケージバージョンは、同じロールのすべてのインスタンスで報告される必要があります。

サービスとレプリケーションの検証

すべてのサービスが正常に再起動したことを確認します。

```
tpaexec cmd <cluster> all -m shell -a systemctl is-active postgres
```

PGD クラスターの場合、 BDR が完全に再コンバートしたことを確認します。

```
SELECT node_name, peer_state_name FROM bdr.node_summary;
```

repmgr を使用した M1 クラスターの場合

```
tpaexec cmd <cluster> role_primary -m shell -a repmgr -f /etc/repmgr.conf cluster show
```

新しいバックアップを取ります

重要なアップグレードの後、クラスターの新しい Barman バックアップを取得します。

```
tpaexec cmd <cluster> role_barman -m shell -a sudo -u barman barman backup <server>
```

これにより、アップグレード後の状態を反映する既知の正常な復元ポイントが得られます。

スケジュールされたジョブを再度有効にする

アップグレードの前に cron ジョブ、バッチジョブ、または外部スケジュールされたタスクを無効にした場合、ここでそれらを再度有効にします。TPA 独自の Barman WAL レシーバーは、自動的に再起動されます。

tpaexec テストを実行する

最終的なスモークチェックとして、標準のテストスイートを実行します。

```
tpaexec test <cluster>
```

これにより、接続、レプリケーション、ロールマッピング、およびアップグレード前に実行されたコンポーネント固有のチェックが実行されます。クリーンテストの実行は、アップグレードが完了したことの最も簡単なエンドツーエンドの確認です。

11.1.4 失敗したアップグレードからの復旧

`tpaexec upgrade` が障害を報告した場合、クラスターは、障害が発生したときに処理されていたインスタンスで既知の状態のままです。後続のインスタンスはタッチされません。

ロールバックする場合、EDB サポートに電話する場合

ロールバックは、TPA が不可逆的なカタログ変更を適用する前にのみ安全です。通常は、メジャーバージョンのアップグレードで発生する BDR 側の構成の更新です。これらの変更が適用されると、正しいパスは逆方向ではなく前方になります。

続行するかロールバックするかがわからない場合は、停止して EDB サポートに連絡してください。

- 失敗した実行の `ansible.log` 。
- `tpaexec test <cluster>` の出力。
- どのインスタンスがアップグレードを完了し、どのインスタンスが完了していないかの説明。
- 使用される config.yml。
- オプションで EDB lasso レポートまたはその他の意味のあるログ。

11.1.5 よくある落とし穴

このセクションには、過去にユーザーを魅了したいくつかの特定の状況が統合されています。

固定されたパッケージバージョンにより no-op アップグレードが発生する

`postgres_package_version` または他の `xxx_package_version` がインストールされたパッケージと既に一致する場合、 `tpaexec upgrade` はコンポーネントを「既に目的のバージョン」として報告し、何も行いません。この症状は、実行が変更なしですぐに完了し、目的のバージョンがインストールされていないことです。

修正 `config.yml` を更新して目的の新しいバージョンを設定し、`tpaexec provision` 、続いて `tpaexec upgrade` を再実行します。詳細は、 パッケージバージョンの選択 を参照してください。

共有 Barman または共有 PEM クラスター

TPA は、現在、共有 Barman または共有 PEM 構成を使用するクラスターに対する `tpaexec upgrade` の実行をサポートしていません。この制限は、早期に中止されるアップグレードの前提条件によって適用されます。サポートが追加されるまで、これらのクラスターシェイプのアップグレードは TPA の外部で手動で実行する必要があります。*After a manualmajor-version Postgres upgrade* を参照してください

その後にクラスターを調整する方法については。

11.1.6 コンポーネントの選択

> **注釈**
>
> コンポーネントのアップグレードは厳密にオプトインです

```
tpaexec upgrade ~/clusters/speedy
```

更新する特定のコンポーネントを選択するために、`--components` フラグはコンマ区切りリストを取ります

```
tpaexec upgrade ~/clusters/speedy \
   --components=postgres,pgd-proxy,pgdcli,pgbouncer,pg-backup-api,barman
```

クラスター内の該当するすべてのコンポーネントを更新する必要がある場合は、`all` をフラグに渡すことができます

```
tpaexec upgrade --components=all
```

component	value	architecture
Barman	barman	all
PEM	pem-server,pem-agent	all
PgBackupAPI	pg-backup-api	all
PgBouncer	pgbouncer	all
EFM	efm	M1 with *failover_manager=efm*
etcd	etcd	M1 with *failover_manager=patroni*
Patroni	patroni	M1 with *failover_manager=patroni*
RepMgr	repmgr	M1 with *failover_manager=repmgr*
PGD Cli	pgdcli	BDR-Always-ON, PGD-Always-ON
PGD-Proxy	pgd-proxy	PGD-Always-ON
Postgres,EPAS,PGE	postgres	All
All	all	All

11.1.7 パッケージバージョンの選択

デフォルトでは、クラスターを作成したときにパッケージバージョン Postgres、PGD、または pglogical などを明示的に指定しなかった場合、`tpaexec upgrade` はインストールされたパッケージの最新の利用可能なバージョンに更新します。

この場合、config.yml を編集し、バージョン設定を更新し、`tpaexec provision` を再実行する必要があります。更新により、選択したバージョンのパッケージがインストールされます。以下に示すように、コマンドラインでバージョンを指定することにより、特定のバージョンに更新することもできます。

```
tpaexec upgrade ~/clusters/speedy -vv      \
  --components=postgres,pgbouncer          \
  -e postgres_package_version="16.10*"     \
  -e pgbouncer_package_version="1.24*"     \
  -e bdr_package_version="5.9.0*"
```

ここでのバージョン構文は、OS ディストリビューションとパッケージマネージャーによって異なることに注意してください。特に、yum は`*xyz*` ワイルドカードを受け入れますが、apt は `xyz*` のみを理解します

上記の例のように。

PGD クラスターの場合、パッケージマネージャーの依存関係の解決に依存して正しい依存関係を選択するのではなく、3 つのコンポーネントすべて Postgres、PGD、pglogical の正確なバージョンを明示的に指定することをお勧めします。

運用環境で実行する前に、QA 環境でアップグレードをテストすることを強くお勧めします。

11.1.8 構成

特定の場合、マイナーバージョンのアップグレードでは config.yml を変更する必要はありません。`config.yml` で `postgres_package_version` が定義されていない場合、`tpaexec upgrade` が実行されると、Postgres を使用可能な最新のマイナーバージョンに正常にアップグレードします。それが正確に何を意味するかは、クラスターの詳細によって異なります。

他のコンポーネントのマイナーバージョンのアップグレードを制御するには、`tpaexec upgrade` を実行する前に `config.yml` で特定の `xxx_package_version` が指定されていることを確認し、`--components=x,y,z` フラグまたは`--components=all` を使用して特定のコンポーネントをアップグレードするように明示的にオプトインすることをお勧めします。クラスター)。`config.yml` で `xxx_package_version` を固定せずに `tpaexec upgrade` を実行し、一部またはすべてのコンポーネントをアップグレードすることをオプトインすると、インストールされたコンポーネントパッケージのメジャーバージョンのアップグレードが発生する可能性があります。互換性が損なわれる可能性があるため、TPA はサポートしていません。

場合によっては、アップグレードには、新しいパッケージのインストールとサービスの再起動を超える追加の手順が含まれる場合があります。たとえば、BDR4 から PGD5 にアップグレードするには、新しいパッケージリポジトリをセットアップし、プロセス中に BDR ノードとグループ構成に特定の変更を加える必要があります。

このような場合、ソフトウェアのアップグレードを有効にするプロセスの一部として必要な複雑な手順がある場合、`tpaexec upgrade` はこれらの手順を実行します。たとえば、上記のシナリオでは、新しい PGD5 パッケージリポジトリを構成します通常、これは deploy でも行います。

ただし、アップグレードプロセス自分自身に直接必要な変更のみを行います。たとえば、config.yml を編集して新しい Postgres ユーザーまたはデータベースを追加する場合、これらの変更はアップグレード中に行われません。混乱を避けるために、ソフトウェアのアップグレードプロセスを開始する前に、無関係の保留中の変更を `tpaexec deploy` することをお勧めします。

11.1.9 BDR-Always-ON から PGD-Always-ON へのアップグレード

BDR-Always-ON から PGD-Always-ON つまり BDR3/4 から PGD5 にアップグレードするには、最初に `tpaexec reconfigure` を実行します。

```
tpaexec reconfigure ~/clusters/speedy\
  --architecture PGD-Always-ON\
  --pgd-proxy-routing local
```

このコマンドは、config.yml を読み取り、クラスターをアップグレードするために必要な変更を行い、新しい config.yml を書き込みます。呼び出しの詳細については、*tpaexec reconfigure* を参照してください。変更を確認した後、`tpaexec upgrade` を実行してアップグレードを実行します。

```
tpaexec upgrade ~/clusters/speedy\
```

または、プロキシモニタリングを有効にしてアップグレードを実行するには、

```
tpaexec upgrade ~/clusters/speedy\
  -e enable_proxy_monitoring=true
```

`tpaexec upgrade` は `tpaexec provision` を自動的に実行して、Ansible インベントリーを更新します。アップグレードプロセスでは、次のことを行います。

1. クラスターをアップグレードするためのすべての前提条件が満たされていることを確認します。
2. クラスター内の各インスタンスについて、正しいリポジトリが構成されていること、および必要な postgres パッケージがそれらの中で利用可能なことを確認します。
3. クラスター内の BDR ノードごとに、一度に 1 つずつ
   - harp-プロキシがそれに接続を送信しないことを保証にノードをフェンスオフします。
   - BDR4 を PGD5 に置き換えるなど、postgres を停止、更新、および再起動します。
   - ノードのフェンスを解除して、接続を再度受信できるようにします。
   - このノードに適用される pgbouncer および pgd-cli を更新します。
4. クラスター内のインスタンスごとに、 BDR v5 専用の BDR 構成を更新します
5. クラスター内のプロキシノードごとに、一度に 1 つずつ
   - pgd-proxy.
   - をセットアップします harp-proxy.
   - を停止します pgd-proxy を起動します。
6. harp-proxy とそのサポートファイルを削除します。

11.1.10 PGD-Always-ON から PGD-X へのアップグレード

`PGD-Always-ON` クラスターの `PGD-X` へのアップグレードは、**重要なアーキテクチャの進化** であり、単純な **ソフトウェアの更新**を超えた変更を含みます。これは、_注意深く調整された多段階プロセス_であり、最終的なソフトウェアのアップグレードを実行する前に、個別のフェーズでクラスターを再構成する必要があります。このプロシージャーは、最初に `pgd-proxy` をビルトイン `Connection Manager` に置き換えることにより、`PGD 5` クラスターの接続処理を最新化します。このステップは、現在ライブクラスターでの手動操作を必要としますが、将来の TPA リリースで自動化が計画されています。次に、クラスターを新しい `PGD-X` に移行します。アーキテクチャ。

アップグレードプロセスは、クラスターを 3 つの異なる状態を介して移行します。

1. **開始** PGD 5.9+ (`PGD-Always-ON`) `PGD-Proxy` を使用
2. **中級** PGD 5.9+ (`PGD-Always-ON`) ビルトイン `Connection Manager` を使用するようになりました
3. **最終** PGD 6 (`PGD-X Architecture`)

前提条件

開始する前に、次の要件を満たしていることを確認してください。

- **クラスターバージョン** クラスターは PGD バージョン 5.9 以降を実行している必要があります。以前の 5.x バージョンを使用している場合は、`tpaexec upgrade` を使用して最初に最新のマイナーバージョンにアップグレードします。PGD-Always-ON クラスターのマイナーバージョンのアップグレードの詳細については、セクション#pgd-always-on を参照してください。
- **バックアップ**クラスターの現在のテストされたバックアップがあります。
- **オーバーライドの確認**インスタンスレベルのプロキシオーバーライドたとえば `pgd_proxy_options` などについて `config.yml` を確認しました。これらは自動的に移行できないため、手動での介入が必要です。
- **共同ホストプロキシ** PGD 5 クラスターは、共同ホストプロキシを使用して構成する必要がありますここで `pgd-proxy` ロールは `bdr` ロールと同じインスタンスにあります。スタンドアロンプロキシインスタンスが存在すると、`switch2cm` コマンドが中止されます。移行を続行する前に、クラスターからスタンドアロンプロキシインスタンスを削除する必要があります。

ステージ 1 ビルトイン接続マネージャーへの移行

最初のステージは、 PGD 5.9+ クラスターを再構成して、外部 `pgd-proxy` の使用から最新のビルトイン Connection Manager に切り替えることです。 TPA は、 `tpaexec switch2cm` コマンドを提供して、最小限のダウンタイムでこの移行を自動化します。

次のコマンドを実行して、`config.yml` ファイルを更新します。これにより、ビルトイン Connection Manager を有効にするために必要な設定が追加されます。

このアクションは構成ファイルのみを変更します。データベースクラスターの実行状態はまだ変更されません。

新しいバージョンを書き込む前に、`reconfigure` は、現在のファイルたとえば `config.yml.~1~` のバックアップを自動的に保存し、安全な復元ポイントを提供します。

呼び出しの詳細については、 *tpaexec reconfigure* を参照してください。

```
tpaexec reconfigure ~/clusters/speedy --enable-connection-manager
```

ステップ 1.2 接続マネージャーに切り替える

`tpaexec switch2cm` コマンドを実行して、`pgd-proxy` からビルトイン Connection Manager への移行を実行します。このコマンドは、 `tpaexec provision` を自動的に実行して Ansible インベントリーを更新し、最小限のダウンタイムですべてのノードをスイッチします。

```
tpaexec switch2cm ~/clusters/speedy
```

`switch2cm` コマンドは、次の操作を実行します。

1. 接続マネージャー設定で Ansible インベントリーを更新します
2. 各ノードごと

- ノードをフェンスして新しい接続を防止します
- PostgreSQL を再起動して接続マネージャー構成をロードします
- pgd-proxy サービスを停止します
- PostgreSQL を再度起動して接続マネージャーがポートをバインドできるようにします b_tran_7 接続マネージャーがリッスンを開始するのを待機します
- ノードのフェンスを解除し、接続を確認します

このプロセスは、公式 *EDB Connection Manager Migrationprocedure* に従います。

ステージ 1 完了

このステージの最後に、ビルトイン Connection Manager で実行される PGD クラスターが作成されます。これは中間状態であり、ステージ 2 に直接進む必要があります。マイナーバージョンアップグレードの tpaexec upgrade は、この中間状態では **サポートされていません** が、 PGD 6 へのアップグレードが完了するまで tpaexec deploy を実行することをお勧めします。

ステージ 2 アーキテクチャの PGD-X へのアップグレード

クラスターが Connection Manager で実行されたら、最後の構成手順に進んで、PGD 6 アップグレードの準備をできます。

> **注釈**
>
> *Stage 1* が正常に完了し、ビルトイン `Connection Manager` で実行されているクラスターからこのプロセスを開始する必要があります。

次のコマンドを実行して、新しいアーキテクチャに合わせて config.yml を更新します。これにより、クラスターアーキテクチャタイプが変更され、BDR バージョンが 6 に設定され、古い古い設定が削除されます。

このアクションは構成ファイルのみを変更します。データベースクラスターの実行状態はまだ変更されません。

```
tpaexec reconfigure ~/clusters/speedy --architecture PGD-X
```

ステップ 2.2 ソフトウェアのアップグレードを実行する

config.yml の最終的な変更を確認した後、標準の tpaexec upgrade コマンドを実行できます。これにより、すべてのノードでソフトウェアのアップグレードが実行され、クラスターが PGD 6 になります。

```
tpaexec upgrade ~/clusters/speedy
```

または、プロキシモニタリングを有効にしてアップグレードを実行するには、

```
tpaexec upgrade ~/clusters/speedy\
  -e enable_proxy_monitoring=true
```

tpaexec upgrade は tpaexec provision を自動的に実行して、Ansible インベントリーを更新します。アップグレードプロセスでは、次のことを行います。

1. クラスターをアップグレードするためのすべての前提条件が満たされていることを確認します。
2. クラスター内の各インスタンスについて、正しいリポジトリが構成されていること、および必要な postgres パッケージがそれらの中で利用可能なことを確認します。
3. クラスター内の BDR ノードごとに、一度に 1 つずつ
   - ノードをフェンスオフして、接続がないようにします。
   - PGD5 を PGD6 に置き換えるなど、postgres を停止、更新、および再起動します。
   - ノードのフェンスを解除して、
   - このノードに適用される pgbouncer および pgd-cli を更新します。
4. BDR v6 専用の BDR 構成を適用します

アップグレードの完了

クラスターは PGD-X アーキテクチャで PGD 6 を実行し、通常のように tpaexec deploy と tpaexec upgrade の両方で完全に管理可能です。

11.1.11 PGD-S または PGD-X

既存の PGD6 PGD-S または PGD-X クラスターを最新の利用可能なソフトウェアバージョンにアップグレードする場合、アップグレードプロセスは次を実行します。

1. クラスターが正常であること、およびノードが構成済みのポートでリッスンしていることを確認します。
2. アップグレードするノードにローカルリポジトリを含むリポジトリが構成および更新されていることを確認します。
3. 更新されたパッケージをインストールできることの確認
4. クラスター内の各 BDR ノードを一度に 1 つずつアップグレードします。

重要: 高可用性を保証するために、書き込みリーダーがアップグレードされるノードの中にある場合、アップグレードされる最後のノードになります。

- 接続を受け入れないようにノードをフェンスします
- postgres
- postgres と更新 PGD パッケージ
- ノードのフェンスを解除して、接続を再度受信できるようにします
- BDR クラスターが再確立されたことを確認します Raft コンセンサス使用 b_tran_6 アップグレードされたノードが構成済みのポートでリッスンしていることを確認します

1. クラスターヘルスチェックを再実行します
2. アップグレードされたパッケージに関する情報を出力します

11.1.12 PGD-Always-ON

既存の PGD-Always-ON PGD5 クラスターを最新の利用可能なソフトウェアバージョンにアップグレードする場合、アップグレードプロセスは次を実行します。

1. クラスターをアップグレードするためのすべての前提条件が満たされていることを確認します。共有 PEM または共有 Barman クラスターではないことを含みます。
2. クラスターに適用されるすべてのコンポーネントのアップグレード前ヘルスチェックを実行します。進行中の Barman バックアップがないことを含む、これにより WAL レシーバーが停止します
3. クラスター内の各インスタンスについて、正しいリポジトリが構成されていること、および必要な postgres パッケージがそれらの中で利用可能なことを確認します。
4. 選択したすべてのコンポーネントを目的のバージョンに更新できることを確認しますパッケージバージョンが提供されている場合
5. クラスター内の BDR ノードごとに、一度に 1 つずつ
    - pgd-proxy が接続を送信しないようにノードをフェンスオフします。
    - postgres を停止、更新、および再起動します。
    - ノードのフェンスを解除しますので、
    - pgd-proxy および pgd-cli ソフトウェアの更新明示的にオプトインした場合
6. クラスター内の該当ノードの場合、pgbouncer、barman、pg-backup-api、および PEM エージェント/PEM サーバーを更新しますノードのロールに従って
7. 必要に応じて Barman WAL レシーバーを起動し、すべてのコンポーネントのアップグレード後のヘルスチェックを実行しますクラスターに適用される場合

11.1.13 BDR-Always-ON

BDR-Always-ON クラスターの場合、アップグレードプロセスはクラスターインスタンスを 1 つずつ実行し、次のことを実行します。

1. クラスターをアップグレードするためのすべての前提条件が満たされていることを確認します。共有 PEM または共有 Barman クラスターではないことを含みます。
2. クラスターに適用されるすべてのコンポーネントのアップグレード前ヘルスチェックを実行します。進行中の Barman バックアップがないことを含む、これにより WAL レシーバーが停止します
3. クラスター内の各インスタンスについて、正しいリポジトリが構成されていること、および必要な postgres パッケージがそれらの中で利用可能なことを確認します。
4. サーバーがメンテナンス中であることを haproxy に伝えます。
5. インスタンスがアクティブサーバーの場合、pgbouncer に再接続し、アクティブなセッションが閉じられるまで待機するように要求します。
6. Postgres を停止し、Postgres、etcd、および pgdcli 該当する場合およびオプトインの場合パッケージを更新し、Postgres を再起動します。

7. 最後に、ha プロキシを介して要求を受信するためにサーバーを再度「準備ができている」としてマークします。
8. クラスター内の該当ノードの場合、pgbouncer、barman、pg-backup-api、および PEM エージェント/PEM サーバーを更新しますノードのロールに従って
9. 必要に応じて Barman WAL レシーバーを起動し、すべてのコンポーネントのアップグレード後のヘルスチェックを実行しますクラスターに適用される場合

PGD 論理スタンバイまたは物理レプリカインスタンスは、haproxy または pgbouncer の相互作用なしで更新されます。クラスター内の非 Postgres インスタンスは放置されます。

11.1.14 M1

M1 クラスターの場合、upgrade は最初にストリーミングレプリカと監視ノードを更新し、次に *tpaexec switchover*

プライマリからアップグレードされたレプリカのいずれかに移行し、プライマリを更新し、最初のプライマリノードにスイッチオーバーします。

1. クラスターをアップグレードするためのすべての前提条件が満たされていることを確認します。共有 PEM または共有 Barman クラスターではないことを含みます。
2. クラスターに適用されるすべてのコンポーネントのアップグレード前ヘルスチェックを実行します。進行中の Barman バックアップがないことを含む、これにより WAL レシーバーが停止します
3. クラスター内の各インスタンスについて、正しいリポジトリが構成されていること、および必要な postgres パッケージがそれらの中で利用可能なことを確認します。
4. ストリーミングレプリカと監視の Postgres を更新します。ノード該当する場合
5. プライマリからアップグレードされたレプリカのいずれかに *tpaexec switchover* を実行します
6. プライマリで Postgres を更新する
7. 最初のプライマリノードにスイッチオーバーします。
8. クラスター内の該当ノードの場合、pgbouncer、barman、pg-backup-api、および PEM エージェント/PEM サーバーを更新しますノードのロールに従って
9. 必要に応じて Barman WAL レシーバーを起動し、すべてのコンポーネントのアップグレード後のヘルスチェックを実行しますクラスターに適用される場合

11.1.15 アップグレードプロセスの制御

update_hosts 変数を定義することにより、クラスターのインスタンスがアップグレードされる順序を制御できます。

```
tpaexec upgrade ~/clusters/speedy \
  -e update_hosts=quirk,keeper,quaver
```

これは、シャドウサーバーが最初にアップグレードされるように、アクティブな PGD プライマリインスタンスを最後にリストすることにより、アップグレード中のリード/シャドウのスイッチオーバーを最小限に抑えるのに役立つ場合があります。

環境で追加のアクションが必要な場合、 *TPA hooks*

パッケージのインストール手順の前後にカスタム Ansible タスクを実行できます。

11.1.16 ノードのサブセットのアップグレード

update_hosts 変数を設定することにより、インスタンスのサブセットでローリングアップグレードを実行できます。ただし、この機能のサポートはアーキテクチャによって異なります。

- **M1** アーキテクチャの場合、この機能は repmgr および Patroni 管理対象クラスターで完全にサポートされています。EFM 管理対象クラスターは、EFM を除くすべてのコンポーネントの update_hosts リストを尊重します。EFM はデータノード全体で異なるバージョンを実行しているクラスターをサポートしていないため、update_hosts で指定されたノードに関係なく、すべてのデータノードは EFM バージョンをアップグレードします。
- **PGD-Always-ON/ BDR-Always-ON** の場合、これはマイナーバージョンのアップグレード中に **のみ**サポートされています。

PGD-Always-ON/ BDR-Always-ON のベストプラクティス

PGD ノードのサブセットでマイナーアップグレードを実行する場合、**RAFT リーダーノードを最後に更新することを強くお勧めします。この戦略により、クラスターが BDR の混合バージョンを実行している間のアップグレード後チェックに関する潜在的な問題を回避します。

PGD-S および PGD-X

tpaexec upgrade は、現在の書き込みリーダーが PGD-S および PGD-X クラスターの最後にアップグレードされるように自動的に調整します。Postgres / PGD ローリングフェーズは 2 つの serial: 1 プレイに分割され、1 つは現在の書き込みリーダーを除くすべての BDR データノードに、もう 1 つはリーダー自分自身を書き込みます。これらのアーキテクチャでリーダー-最後のシーケンスを達成するために追加の手順は必要ありません。

11.2 tpaexec switchover

tpaexec switchover コマンドは、*M1* のプライマリとレプリカの間で制御されたスイッチオーバーを実行します。このコマンドを実行すると、選択したレプリカが新しいプライマリに昇格し、元のプライマリは新しいレプリカになり、クラスター内の他のレプリカは新しいプライマリに従うように再構成されます。

このコマンドは、ロールを切り替える前にクラスターが正常であることを確認し、事前に repmgr サービスをシャットダウンせずに実行されるように設計されています。

これは、 --siblings-follow オプションを指定して repmgr standby switchover を実行することと同じです。

11.2.1 例

このコマンドは、 replicaname を~/clusters/speedy の新しいプライマリにします。

```
tpaexec switchover ~/clusters/speedy replicaname
```

11.2.2 アーキテクチャオプション

このコマンドは *M1* にのみ適用されます

1 つの書き込み可能なプライマリインスタンスと 1 つ以上の読み取り専用レプリカがあります。

BDR-Always-ON クラスターの場合、*BDR/HAProxy server pool management* を使用して PGD インスタンスのメンテナンスを実行します。

11.2.3 Repmgr リダイレクト pgbouncer

repmgr をフェールオーバーマネージャーとして、pgbouncer を接続プーラーとして使用し、repmgr_redirect_pgbouncer: true を設定している場合、switchover コマンドは、pgbouncer インスタンスが新しいプライマリノードにリダイレクトされます。

```
# switchover to a replica
tpaexec switchover <cluster_name> <replica_name>
# revert to initial primary
tpaexec switchover <cluster_name> <initial_primary_name> -e"revert_redirect=true"
```

11.3 BDR/HAProxy server pool management

tpaexec pool-disable-server および pool-enable-server コマンドを使用すると、 *BDR-Always-ON* の PGD インスタンスを、メンテナンスのために HAProxy アクティブサーバープールから一時的に削除し、その後に追加できます。

これらのコマンドは、デフォルトで *Upgrading your cluster* と同じプロセスに従うため、pool-disable-server は、PGD インスタンスに対するアクティブなトランザクションが完了し、pgbouncer が完了する前に新しい接続を別のインスタンスに転送するのを待機します。アクティブなセッションが終了するまで待機したくない場合は、--nowait オプションを使用します。

インスタンスで pool-disable-server の直後に pool-enable-server を実行すると、すべてのアクティブなトラフィックが別のインスタンス本質的にスイッチオーバーに移動する効果があります。これにより、インスタンスの可用性を維持しながら、長時間実行される VACUUM コマンドなどのオンラインメンテナンスタスクを実行できます。

haproxy_backend_servers の同じセットで構成された複数の HAProxy サーバーが存在する場合、このコマンドは、指定されたサーバーを関連するすべてのプロキシのプールに並行して削除または追加します。

11.3.1 例

```
tpaexec pool-disable-server ~/clusters/clockwork orange # --nowait

#  HAProxy will no longer direct any traffic to the PGD instance named

#  orange, so you can perform maintenance on it (e.g., run `tpaexec

#  rehydrate`).

tpaexec pool-enable-server ~/clusters/clockwork orange
```

サーバープールからインスタンスを削除すると、プール内の他のインスタンスに障害が発生しても、HAProxy はトラフィックをそれに転送しません。メンテナンスアクティビティが終了したら、サーバーをアクティブなプールに追加することを忘れないでください。

11.4 tpaexec rehydrate

`tpaexec rehydrate` コマンドは、更新されたマシンイメージ AMI で AWS EC2 インスタンスを再構築し、TPA によって管理されるクラスターにセキュリティパッチと OS アップグレードを迅速に展開できます。

必要な変更をすべて備えた新しい AMI が与えられると、このコマンドはインスタンスを終了し、新しいイメージを使用する新しくプロビジョニングされたインスタンスに置き換え、サーバーの構成を正確に再作成する前に古いインスタンスからデータボリュームを接続します `config.yml`)。

最新のイメージを公開し、定期的なスケジュールでサーバーをスクラッチから再構築することは、サーバーフリートに個々のセキュリティ更新を自分自身でダウンロードしてインストールできるようにするための代替手段です。これにより、各サーバーの状態を一目で追跡することが簡単になり、個々のサーバーへの手動変更を防ぎますインスタンスの置換中に一掃されます。

TPA を使用すると、リハイドレーション中にクラスター全体の中断を最小限に抑えることが簡単になりますが、プロセスには、個々のサーバーが終了および置き換えられるときに必然的にダウンタイムが含まれる必要があります。*M1* では、最初にレプリカを rehydrate から、*tpaexec switchover* を使用できます

リハイドレートする前にプライマリをレプリカに変換します。*BDR-Always-ON* では、リハイドレートする前に *BDR/HAProxy server pool management* を実行し、後で追加して戻すことができます。

Postgres および関連コンポーネントのマイナーバージョンの更新をインストールするだけの場合は、代わりに *Upgrading your cluster* コマンドを使用できます。

11.4.1 前提条件

インスタンスを rehydrate できるようにするには、 `config.yml` のデータボリュームごとに `delete_on_termination: no` および `attach_existing: yes` を指定する必要があります。新しいインスタンスには、必ず新しい EBS ルートボリュームがあります。

デフォルトでは、EC2 インスタンスを終了すると、それにアタッチされている EBS ボリュームも終了します。この場合、新しいインスタンスに再アタッチしたいため、`delete_on_termination` を無効にする必

要があります。 attach_existing 設定により、TPA は、新しいインスタンスをプロビジョニングするときに古いボリュームを検索し、見つかった場合は、実行後にインスタンスにアタッチします。

古いインスタンスを手動で停止または終了しないでください。tpaexec rehydrate コマンドは、インスタンスを安全にリハイドレートできることを確認した後にこれを実行します。

11.4.2 例

~/clusters/night に AWS クラスター構成があると仮定します。

構成を変更する

最初に、config.yml を編集し、新しい AMI を指定する必要があります。例

```
ec2_ami:
  Name: RHEL-8.3_HVM-20210209-x86_64-0-Hourly2-GP2
  Owner: 309956199498
```

各データボリュームで delete_on_termination が無効になっていることを確認します。パラメーターが存在しない場合、AWS EC2 管理コンソールからその値を確認できます。’Instances’ をクリックし、インスタンスを選択してから、[Description] タブを開き、[Block devices] までスクロールし、EBS ボリュームをクリックします。 “Delete on termination” フラグが true に設定されている場合、 付録ことができます。また、 attach_existing を確認し、まだ設定されていない場合は yes に設定します。

次に、両方の属性が正しく設定された例を示します。

```
instances:

- node: 1
  Name: vlad
  subnet: 10.33.14.0/24
  role: primary
  volumes:
  - device_name: /dev/xvdf
    volume_type: gp2
    volume_size: 16
    attach_existing: yes
    delete_on_termination: false
    vars:
      volume_for: postgres_data
      mountpoint: /var/lib/pgsql
```

ボリュームパラメーターは instance_defaults および特定のインスタンスで設定できることに注意してください。volumes: を検索し、関連するすべてのボリュームにこれらの 2 つの属性が設定されていることを確認します。

rehydration を開始します

次に、 rehydrate コマンドの構文を示します。

```
tpaexec rehydrate ~/clusters/night instancename
```

単一のインスタンス名またはインスタンス名のコンマ区切りリストを指定できますが、クラスター内のすべてのインスタンスを一度に rehydrate ことはできません。

このコマンドは、リハイドレートされるインスタンスに接続されているすべての非ルート EBS ボリュームの `delete_on_termination` フラグが false に設定されていることを確認します。これに当てはまらない場合、インスタンスが終了する前にエラーで停止します。

ボリューム属性が正しく設定されている場合、コマンドは最初に各インスタンスを終了し、次にプロビジョニングと展開を実行して、新しい AMI を使用する新しいインスタンスに置き換えます。

11.4.3 段階的に rehydration する

クラスターの継続性を維持するために、クラスターを段階的にリハイドレートすることをお勧めします。

たとえば、プライマリインスタンス、2 つのレプリカ、および Barman バックアップサーバーを備えた *M1* では、最初に Barman インスタンスと 1 つのレプリカを rehydrate、次に別のレプリカをリハイドレートし、次に *tpaexec switchover* プライマリからリハイドレートしたレプリカのいずれかに、前者を rehydrate プライマリ、およびオプションで元のプライマリにスイッチオーバーします。このシーケンスにより、1 つのプライマリと 1 つのレプリカが常に利用できるようになります。

11.4.4 付録

awscli を使用してボリューム属性を変更する

最初に、AWS 管理コンソールでインスタンスと EBS ボリュームを見つけます。’Instances’をクリックしてインスタンスを選択し、[Description] タブを開いて [Block devices] までスクロールし、EBS ボリュームを選択します。`delete_on_termination` を無効にするには、 `--region` 、`--instance-id` 、およびブロックデバイス名を正しい値に置き換えた後、次のコマンドを実行します。

```
aws ec2 modify-instance-attribute \
    --region eu-west-1 --instance-id i-XXXXXXXXXXXXXXXXX \
    --block-device-mappings \
      [{"DeviceName": "/dev/xvdf", "Ebs": {"DeleteOnTermination": false}}]
```

インスタンスのデータボリュームごとにこれを行います。少し遅延すると、管理コンソールに変更が表示されます。`tpaexec rehydrate` は、インスタンスを安全にリハイドレートできることも検出します。

第12章　MISCELLANEOUS

12.1 TPA and Ansible Tower/Ansible Automation Platform

TPA は、RedHat Ansible Automation PlatformAAP を介した展開をサポートしています。このサポートは、以下で詳細に説明するように、AAP で `deploy` および `upgrade` ステップを実行できるようにすることで機能します。デプロイまたはアップグレードを実行する前に、tpa パッケージがインストールされている別のスタンドアロンマシンで構成 `configure` コマンドとプロビジョニング `provision` コマンドを実行する必要があります。このスタンドアロンマシンで適切なオプションを使用して `configure` および `provision` を実行すると、結果のクラスターディレクトリを AAP にインポートできます。サポートはベアメタルプラットフォームに制限されています。

TPA は AAP 2.4 および AAP 2.6 でテストされています。

12.1.1 AAP の初期セットアップ

TPA が AAP を使用してクラスターを展開する前に、この初期設定を実行する必要があります。

TPA 実行環境イメージの追加 管理者

バージョン 2.4 以降、AAP はコンテナイメージを使用して Ansible プレイブックを実行します。これらのコンテナは実行環境 EE と呼ばれ、プレイブックが正常に実行するために必要な依存関係をバンドルしています。その結果、これは、必要な TPA 依存関係をすべて解決して使用するには、TPA を含む EE が必要で、展開とアップグレードの実行時に AAP が使用できることを意味します。

EDB の顧客は、EDB サポートに連絡して、EE に関するサポートを求めることができます。

AAP 管理者として、TPA 対応 EE イメージを指す使用可能な EE リストにエントリを作成します。

EDB_SUBSCRIPTION_TOKEN 資格情報タイプを作成します admin

AAP 管理者として、カスタム資格情報タイプ `EDB_SUBSCRIPTION_TOKEN` を作成して、EDB サブスクリプションアクセストークンを保持します。

1. AAP UI の [Credential Types] ページに移動します。
2. **Name** フィールドを `EDB_SUBSCRIPTION_TOKEN` に設定します。
3. 次を **入力構成** フィールドに貼り付けます。

```
fields:
- id: tpa_edb_sub_token
  type: string
```

(次のページに続く)

(前のページからの続き)

```
  label: EDB_SUBSCRIPTION_TOKEN
  secret: true
required:
- tpa_edb_sub_token
```

4. 次を **Injector Configuration** フィールドに貼り付けます。

```
env:
  EDB_SUBSCRIPTION_TOKEN: {{ tpa_edb_sub_token }}
```

5. 変更を保存します。

6. 新しく追加されたタイプ EDB_SUBSCRIPTION_TOKEN を使用して資格情報を作成します。

12.1.2 クラスターのセットアップ

tpaexec パッケージがインストールされているワークステーションで初期手順を実行します。

TPA ワークステーションで

構成

次のオプションを含む tpaexec configure コマンドを実行します。--platform bare 、--use-ansible-tower 、--tower-git-repository

```
tpaexec configure <clustername> \
         --platform bare \
         --use-ansible-tower https://aac.example.com \
         --tower-git-repository ssh://git@git.example.com/example \
         --hostnames-from <hostnamefile> \
         --architecture PGD-Always-ON \
         --pgd-proxy-routing local \
         --postgresql 16
```

--use-ansible-tower は、その時点で使用されない場合でも、パラメーターとして AAP アドレスを想定しています。--tower-git-repository は、クラスターデータを AAP にインポートするために使用されます。TPA は、ブランチ名として cluster_name を使用して独自のブランチを作成します。これにより、すべてのクラスターに同じリポジトリを使用できます。**クラスター構成** で説明しているように、 tpaexec configure の他のすべてのオプションは引き続き有効です。

config.yml の変更

config.yml には、トップレベル辞書の ansible_tower が含まれ、これにより、 tpaexec provision はクラスターを AAP 対応クラスターとして扱います。

config.yml を編集して、ノードごとに ansible_host および{private,public}_ip が定義され、ansible_host が AAP が解決可能な値に設定されていることを確認します。必要に応じて、さらに変更または追加を加えます。詳細は、**クラスター構成** を参照してください。

インベントリーおよびその他の関連ファイルを生成するには、`tpaexec provision` を実行します。

AAP UI の場合

プロジェクト

git リポジトリをソースとして使用して、AAP にプロジェクトを追加します。 TPA EE イメージを使用するようにプロジェクトのデフォルト EE を設定します。

単一のプロジェクトで複数のインベントリを使用しようとする場合、 **Allow Branch Override** が必要です。

インベントリ

空のインベントリを追加します。プロジェクトを外部ソースとして使用し、インベントリファイルとして `inventory/00-cluster_name` を使用してデータを入力します。

また、AAP で TPA の制御の外部に追加の変数を設定しない場合、**リモートインベントリソースからのローカル変数を上書きします** AAP 2.4 または **上書き変数** AAP 2.6 を使用することをお勧めします。

資格情報

`vault` 資格情報を作成します。TPA ワークステーションで `tpaexec show-vault <cluster_dir>` を使用して、ボールトのパスワードを取得できます。

AAP サーバーがインベントリノードに接続できるようにする `machine` 資格情報があることを確認します。

テンプレートの作成

テンプレートを作成するには

1. プロジェクトとインベントリを使用するテンプレートを作成します。
2. 次の必要な資格情報を含めます。
    - Vault credential
    - `EDB_SUBSCRIPTION_TOKEN` credential
    - マシン資格情報
3. 2 つの追加変数を設定します。

```
tpa_dir: /opt/EDB/TPA
cluster_dir: /runner/project
```

4. プレイブックとして `deploy.yml` を選択します。
5. クラスターを展開するには、新しいテンプレートに基づいてジョブを実行します。

12.1.3 カスタムプレイブックの実行

初期展開に加えて、既存のクラスターに対して小さなアドホックプレイブックを実行する必要がある場合があります。サービスステータスの確認、ログの検査、診断出力の収集など。AAP のアドホックコマンド機能は、プロジェクトを参照せずにインベントリーホストに対して直接実行され、クラスターディレクトリの `ssh_config` 、`ansible.cfg` 、vault、またはその他の TPA コンテキストにアクセスできないため、これには適していません。適切なアプローチは、プレイブックをクラスターの git リポジトリに追加し、2 番目のテンプレートから実行することです。

プレイブックをクラスターの `commands/` サブディレクトリに配置することをお勧めします。これには、2 つの利点があります。カスタムプレイブックを TPA 自分自身のファイルから分離し、そこに配置されたものはすべてサブコマンドとして `tpaexec` によって取得されます。同じメカニズムは、配置されたシェルスクリプトでも動作します `commands/` で)。

例、次を`<cluster_dir>/commands/check-disk-space.yml` として保存します。

```
- --

#  Sample playbook: check disk space on every cluster node.


-  hosts: all
  gather_facts: false
  tasks:
    - name: Show disk usage
      ansible.builtin.command: df -h
      changed_when: false
```

AAP が認識できるように、ファイルをリモートリポジトリのクラスターのブランチにコミットしてプッシュします。

```
git add commands/check-disk-space.yml
git commit -m "Add check-disk-space playbook"
git push
```

AAP UI では、展開テンプレートの **Duplicate template** アクションを使用して、同じプロジェクト、インベントリ、資格情報、および追加の変数を使用してコピーを作成します。新しいテンプレートを編集し、プレイブックを `commands/check-disk-space.yml` に変更します。新しいテンプレートに基づいてジョブを起動します。 `df -h` のホストごとの出力は、ジョブの詳細で各ホストをクリックし、 **Output** タブを選択することにより利用できます。

12.1.4 複数のインベントリに 1 つのプロジェクトを使用する

TPA は、関連する git リポジトリのクラスターごとに異なるブランチ名を使用します。このアプローチでは、複数のクラスターで単一のプロジェクトを使用できます。

Allow ブランチオーバーライドオプションを設定します

AAP プロジェクトで、 **Allow Branch override** オプションを有効にします。

複数のインベントリを定義する

TPA は、git リポジトリ内のクラスターごとに異なるブランチ名を使用します。インベントリごとにブランチをオーバーライドすることにより、同じプロジェクトをソースと使用して複数のインベントリを生成できます。

インベントリーごとに資格情報を定義する

ボールトのパスワードは、TPA クラスターごとに異なるため、インベントリーごとにそれに応じて設定されていることを確認します。

12.1.5 AAP で TPA を更新

AAP で TPA の更新には、いくつかの追加手順が含まれます。

TPA ワークステーションパッケージの更新

OS に応じて、OS パッケージと同様に TPA ワークステーションパッケージを更新します。*TPA installation* を参照してください。

同じバージョンタグで EE イメージを使用する

AAP の EE イメージを変更して、使用されるワークステーションパッケージバージョンと同じバージョンタグを使用します。

クラスターディレクトリで tpaexec relink を実行します

`tpaexec relink <cluster_dir> --force` を実行して、AAP を使用するクラスターが最新の状態であることを確認します。これを行う必要がある場合の例は、TPA インストールを新しいバージョンにアップグレードした後です。`relink` コマンドによってコミットされた変更は必ずプッシュしてください。

```
git status
```

```
On branch cluster_name
Your branch is ahead of tower/cluster_name by 1 commit.
  (use "git push" to publish your local commits)
...
```

```
git push tower
```

プロジェクトとインベントリの同期

ジョブの起動時にリビジョンの更新 と **起動時の更新**を使用するように設定されていない場合は、それぞれ AAP UI と関連インベントリでプロジェクトを同期します。

12.1.6 TPA の EE をビルドする

前提条件

独自の EE イメージをビルドするには、 `ansible-builder` を使用することをお勧めします。

必要なものは次のとおりです。

1. `docker` または `podman`
2. `ansible-builder` および `ansible-navigator` python ツールキット
3. TPA repo からタグ `vA.B.C` でチェックアウトした TPA ソースコード `vA.B.C` は、使用する TPA バージョンです。

環境ファイル

`ansible-builder` は、環境ファイルを使用して、作業用 EE イメージを生成します。

次に、このような TPA の環境ファイルのテンプレート例を示します。

execution-environment.yml

```
version: 3
images:
  base_image:
    name: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel9:latest
dependencies:
  python: << TPA_REPO_CLONE_FOLDER >>/requirements-aap.txt
  galaxy: << TPA_REPO_CLONE_FOLDER >>/collections/requirements.yml
options:
  package_manager_path: /usr/bin/microdnf

additional_build_steps:
  append_final:
   - RUN mkdir -p /opt/EDB/TPA
   - COPY << TPA_REPO_CLONE_FOLDER >> /opt/EDB/TPA
   - ENV PYTHONPATH="${PYTHONPATH:+${PYTHONPATH}:}/opt/EDB/TPA/lib"
```

異なるベースイメージには、より多くの `additional_build_steps` が必要になる場合があります。 ansible-builder を参照してください

高度な使用のために。

EE build コマンド

次のコマンドは、EE イメージをビルドします。

```
ansible-builder build \
  --file=execution-environment.yml \
  --container-runtime=<docker/podman> \
  --tag=<your-registry>/<your-namespace>/tpa-ee:vA.B.C \
  --verbosity 2
```

12.2 TPA, Ansible, and sudo

TPA は sudo で Ansible を使用して、ターゲットインスタンスで昇格した特権でタスクを実行します。このページでは、Ansible が sudo を使用する方法決して TPA 固有ではありません、および TPA で管理されているシステムへの影響について説明します。

TPA には root 権限が必要です。

- パッケージをインストールするためオペレーティングシステムのネイティブパッケージマネージャーを使用した必須パッケージ、および pip を使用したオプションパッケージ
- サービスを停止、リロード、および再起動するため Postgres、repmgr、efm、etcd、haproxy、pgbouncer など
- 他のさまざまなタスクを実行するためたとえば、クラスターファクトの収集、スイッチオーバーの実行、クラスターノードのセットアップ

TPA は sudo を使用できることも必要です。 `ansible_user: root` を設定することにより、root として直接 ssh インすることができますが、引き続き sudo を使用して他のユーザー postgres などとしてタスクを実行します。

12.2.1 Ansible sudo 呼び出し

Ansible が sudo を使用してタスクを実行すると、ターゲットインスタンスに次のようなプロセスが表示されます。

```
/bin/bash -c sudo -H -S -n  -u root /bin/bash -c \
  ""echo BECOME-SUCCESS-kfoodiiprztsyerriqbjuqhhbemejgpc ; \
  /usr/bin/python2"" && sleep 0
```

`sudo yum install -y xyzpkg` のようなものを期待していた人は、これに驚くことがよくあります。一般に、Ansible のほとんどのタスクは、認識可能なシェルコマンドを実行するのではなく、Python インタープリターを呼び出して Python コードを実行します。Playbook は `raw` シェルコマンドを実行できますが、TPA はそのようなタスクを Python インタープリターをブートストラップするためにのみ使用します。

Ansible モジュールには、さまざまな複雑さの Python コードが含まれており、Ansible プレイブックは YAML 形式で書かれた単なるシェルスクリプトではありません。任意の Ansible プレイブックを実行するのと同じことを行うシェルコマンドを「抽出」する方法はありません。

Ansible が sudo privilegeescalation must be general を使用する方法の重要な結果が 1 つあります。つまり、一部の管理者が慣れているように、sudoers.conf で sudo 呼び出しを特定のコマンドに制限することはできません。ほとんどのタスクは Python を呼び出すだけです。すべてのコマンドにランダムな文字列がなければ、Python への sudo アクセスを制限できますが、Python が root として実行されると、とにかくできることに対する事実上の制限がありません。

ターゲットホストで Python モジュールを実行するのは、Ansible の動作です。これはどれも TPA に固有ではなく、これらの考慮事項は他の Ansible プレイブックに等しく適用されます。

12.2.2 おすすめ

- SSH 公開キーベースの認証を使用して、ターゲットインスタンスにアクセスします。
- SSH ユーザーがパスワードなしで sudo コマンドを実行できるようにします。
- コマンドではなく時間でアクセスを制限します。

TPA は、クラスターを最初にセットアップするとき、またはメンテナンスウィンドウなどに構成を変更するために `tpaexec deploy` を再度実行するときのみにアクセスが必要です。それまでは、そのアクセスを完全に無効にすることができます ssh と sudo の両方を 1 行の変更で。

デプロイメント中に、Ansible が行うことはすべて、プレイブックの動作と指定したパラメーターに基づいて予測可能であり、各アクションは、ターゲットインスタンスのシステムログ、および tpaexec 自分自身が実行されるマシンの Ansible ログに表示されます。

Ansible の焦点は、実行できるアクションにきめの細かい制限を課すことよりも、実行時に何をするのかの可視性を提供することであるため、昇格した特権はスコープではなく時間でより適切に割り当ておよび管理されます。

12.2.3 SSH および sudo パスワード

パスワードなしの SSH キー認証とパスワードなしの sudo アクセスを設定することを**強く** お勧めしますが、パスワードを使用することも可能です。

tpaexec を実行する前に環境で `ANSIBLE_ASK_PASS=yes` および `ANSIBLE_BECOME_ASK_PASS=yes` を設定すると、Ansible はリモートサーバーのログインパスワードと sudo パスワードを入力するように求めます。次に、リモートサーバーで login/sudo パスワードプロンプトをネゴシエートし、指定したパスワードを送信しますこれにより、Playbook の実行時間が著しく長くなります。

パスワードの組み合わせを使用してアクセスを制限するよりも、必要がないときに特定のアカウントを介したアクセスを完全に無効にする方が効果的なセキュリティ制御であると考えられるため、この操作モードはお勧めしません。ssh の公開キー認証を使用すると、サーバーにアクセスできる人を効果的に制御できます。許可されたユーザーごとに 1 つのプライベートキーを保護することは、1 つまたは複数の共有パスワードを保護するよりも簡単です。また、ssh / sudo レベルでのアクセスを必要な場合に制限すると、メンテナンス期間にパスワードは追加のセキュリティを追加しません。

12.2.4 sudo オプション

sudo で Ansible を使用するには、sudoers.conf で `requiretty` を設定しないでください。

必要に応じて、ansible.cfg の `[privilege_escalation]` セクションで `become_flags` 、または環境の `ANSIBLE_BECOME_FLAGS` 、またはインベントリの `ansible_become_flags` を設定することにより、Ansible が使用する sudo オプション`-H -S -n` を変更できます。3 つの方法はすべて同等ですが、特定の必要がある場合にのみ sudo オプションを変更してください。デフォルトが選択されたのには十分な理由があります。たとえば、パスワードなしの sudo が誤って構成されている場合、`-S -n` を削除するとタスクがタイムアウトします。

12.2.5 特権エスカレーション構成の管理

デフォルトの sudo 構成

デフォルトでは、TPA はターゲットインスタンスでの sudo 関連の構成を自動的に管理します。これには、sudo パッケージが存在しない場合のインストール、さまざまなコンポーネントの sudoers ファイルの構成が含まれます。

`privilege_escalation_command` のデフォルト値は`"sudo"` で、これにより、TPA は sudo のインストールと構成を管理できます。

代替の特権エスカレーションコマンドを使用する

環境が別の特権エスカレーションコマンドを使用している場合、`cluster_vars` で `privilege_escalation_command` を設定することにより、代替を使用するように TPA を構成できます。

```
cluster_vars:
  privilege_escalation_command: other_tool  # replaces sudo; may include arguments
```

この値は、TPA が管理対象アプリケーションに対して生成する各特権エスカレーションコマンドで sudo の直接インプレース置換として使用されます。必要に応じて追加の引数たとえば `other_tool --flag` などを含めることができます。

Ansible の become メカニズムでサポートされている任意の特権エスカレーションコマンドに設定できます。Ansible privilege escalation documentation を参照してください

サポートされているメソッドの完全なリストについては、

重要: sudo は、EDB で公式にサポートされている唯一の特権エスカレーションコマンドです。代替コマンドを使用する場合、互換性と適切な構成を保証するのは自分の責任です。EDB サポートは、代替の特権エスカレーションメカニズムに関連する問題を支援する能力が制限されている場合があります。

Ansible の become メソッドと `privilege_escalation_command`

注意する必要がある 2 つの別個の特権エスカレーション設定があります。

- **``ansible_become_method``** は、ターゲットインスタンスでデプロイメントタスクを実行するときに Ansible 自分自身が特権をエスカレートする方法を制御します。これは、インベントリまたは `ansible.cfg` に設定された Ansible 変数です。

- ``**privilege_escalation_command**`` は、管理対象アプリケーション EFM、repmgr、HARP が実行時に呼び出すコマンドを制御して、Ansible とは無関係に、サービス管理操作の特権をエスカレートします。

これらは目的が異なるため、両方を一貫して構成する必要があります。環境が sudo の代わりに代替ツールを使用している場合、`cluster_vars` で `privilege_escalation_command` を設定し、**および** Ansible インベントリまたは `ansible.cfg` で `ansible_become_method` と一致するように `ansible.cfg` を構成する必要があります。

推奨されるアプローチ フックを使用する

別の特権エスカレーションコマンドを使用する必要がある場合は、*TPA hooks* を使用して特権エスカレーションメカニズムを構成することをお勧めします。フックを使用すると、展開中の特定のポイントでカスタムタスクを実行でき、カスタマイズを TPA のコア展開ロジックから分離したまま、特権エスカレーションの構成を完全に制御できます。

たとえば、リポジトリを構成した後、`post-repo` フックを使用して特権エスカレーションコマンドをインストールおよび構成したり、`pre-deploy` フックを使用して、メイン展開が開始される前に必要な権限を設定したりできます。このアプローチにより、保守性が向上し、環境固有の要件の管理が簡単になります。

手動構成要件

代替の特権エスカレーションコマンド"sudo" 以外の場合、TPA は sudo パッケージのインストールと sudoers 構成をスキップします。`tpaexec deploy` を実行する前に、選択した特権エスカレーションコマンドをすべてのターゲットシステムに手動またはフックを使用してインストールして構成する必要があります。

1. postgres ユーザー のサービス管理権限

特権エスカレーションメカニズムでは、 postgres system ユーザーが PostgreSQL および関連サービスを起動、停止、再起動、およびリロードするための systemctl コマンドを実行できるようにする必要があります。これは、フェールオーバーマネージャー repmgr、HARP、EFM が自動フェイルオーバー操作中に正常に機能するために必要です。

たとえば、sudo を使用すると、TPA は次の権限を構成します。

```
postgres ALL=(ALL) NOPASSWD: /bin/systemctl start postgresql
postgres ALL=(ALL) NOPASSWD: /bin/systemctl stop postgresql
postgres ALL=(ALL) NOPASSWD: /bin/systemctl restart postgresql
postgres ALL=(ALL) NOPASSWD: /bin/systemctl reload postgresql
```

選択した特権エスカレーションシステムで同等の権限を構成する必要があります。

2. EFM データベース機能の権限 EFM クラスターのみ

クラスターがフェールオーバーマネージャーとして EFM を使用している場合、特権エスカレーションメカニズムは、EFM システムユーザーが postgres ユーザーとして `efm_db_functions` スクリプトを実行できるようにする必要があります。これは、EFM がヘルスチェックとフェイルオーバー操作を実行するために必要です。

たとえば、sudo を使用すると、TPA は次のように構成します。

```
efm ALL=(postgres) NOPASSWD: /usr/edb/efm-X.Y/bin/efm_db_functions
```

特権エスカレーションシステムで同等の権限を構成して、efm ユーザーが postgres ユーザーとしてこのスクリプトを実行できるようにします。

12.2.6 ロギング

プレイブックの実行の場合、sudo ログは主に Python の呼び出しを表示します誰かが sudo -i を使用するときに bash の呼び出しのみを表示するのと同様に。

詳細については、syslog には、ターゲットインスタンスでの各モジュール呼び出しの正確な引数が表示されます。そのモジュールが呼び出された理由の高レベルのビューについては、コントローラの ansible.log に、そのタスクがしようとしたことと結果が表示されます。

さらに詳細な監査データの独立したソースが必要な場合は、サーバーで auditd を実行し、SELinux ログファイルを使用できます。bpftrace/bcc からさらに詳細なシステムコールレベルの情報を取得できます。たとえば、opensnoop はシステムで開かれているすべてのファイルを表示し、execsnoop はシステムで実行されたすべてのプロセスを表示します。ニーズに応じて、これらのことのいずれかまたはすべてを実行できますが、ロギングの増加に伴うオーバーヘッドの増加という明らかな注意事項。

12.2.7 ローカル権限

TPA installation

sudo は、「run これらのコマンドを root attempts で実行する」の短縮形としてのみ言及しています。TPA がインストールされ、tpaexec setup を実行すると、TPA 自分自身はローカルマシンでの昇格した特権を必要としません。ただし、Docker を使用する場合、Docker デーモンへの接続が許可されたグループに属するユーザーとして tpaexec を実行する必要があります。

12.3 TPA - PuTTY Configuration guide

Windows で PuTTY を使用して、TPA ユーティリティ **tpaexec プロビジョニング***によって作成された AWS インスタンスに ssh を介して接続するには、Amazon が生成したプライベートキー形式.pem からキーを変換する必要があります EC2 を PuTTY 形式.ppk に変換します。

```
#  Provision the cluster

[tpa]$ tpaexec provision <clustername>
```

PuTTY には、 **PuTTYgen** という名前のツールがあり、キーを必要な形式に変換できます。

- **

12.3.1 キー変換

プライベートキーを見つけます

クラスターディレクトリ<clustername> でプライベートキーを見つけます。名前付けは、**id_** が接頭辞を付けた **config.yml** で設定された **cluster_name** 変数に従って名前付けられます。例、 cluster_name が **testenv1** に設定されている場合、プライベートキーの名前は **id_testenv1** になります。

キーを.pem として保存

このファイルを Windows ファイルシステムにコピーし、.pem ファイルとして保存します。この例では **id_testenv1.pem** - テキストファイルにカットアンドペーストすると、これにうまく機能します。

キー変換

PuTTYgen を起動し、[パラメーター] の下で、生成するキーの適切なタイプを選択します。

PuTTYgen の古いバージョンの場合、 **SSH-2 RSA** を選択します。最近のバージョンの場合 select **RSA**

- SSH-1 RSA は選択しないでください

次に、 **Load** を選択します。 **PuTTY Private Key Files (*.ppk)** と記載されたボックスで、 **All Files (*.*)** を選択する必要があります。

.pem ファイルを選択し、 **Open** を選択して、 **OK** をクリックします。

Save private key を選択し、 **Yes** をクリックして、パスフレーズなしでキーを保存することに関する警告を無視します。ファイルのサフィックスが**.ppk** であることを確認し、 **.pem** ファイルと同じ名前を選択します。この例では、ファイル名は **id_testenv1.ppk** になります。

- **

12.3.2 PuTTY の構成

PuTTY を起動し、 **Category** ウィンドウから **Session** を選択します。 **Host Name** パネルに<user>@<IP address> と入力し、ポートパネルに **22** と入力します。

<user> および<IP address> は、``**tpaexec provision**`` ユーティリティによって作成される<clustername>/ssh_config ファイルにあります。

Putty **Category** ウィンドウで、 **Connection** を選択し、 **SSH** を展開して **Auth** を選択します

認証用の秘密キーファイル *とマークされたパネルで、 **Browse** をクリックし、上記で保存した.ppk ファイルを選択し、**Open** を選択します

Putty **Category** ウィンドウで、 **Session** を再度選択し、 **Saved Sessions** にセッション名を入力して、 **Save**

この保存されたセッションを選択することにより、PuTTY を介して AWS ホストに接続できるようになります。

- **

12.4 Troubleshooting

12.4.1 Python 仮想環境を再作成します

時々、Python venv は一貫性のない状態になる場合があります。その場合、最も簡単な解決策は削除して再作成することです。壊れた venv の症状には、次のようなプロビジョニング中のエラーが含まれる場合があります。

```
TASK [Write Vagrantfile and firstboot.sh]  ****  ****  ****  ****  ****  ****  **** ␣
↪****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  **** ␣
↪****  ****  ****  ****  ****  ****  ****  ****  ****  **** **
failed: [localhost] (item=Vagrantfile) => {"changed": false, "checksum":
↪"bf1403a17d897b68fa8137784d298d4da36fb7f9", "item": "Vagrantfile", "msg": "Aborting,
 target uses selinux but python bindings (libselinux-python) arent installed!"}
```

新しい仮想環境を作成するには tpaexec がデフォルトの場所にインストールされていると仮定します。

```
sudo rm -rf /opt/EDB/TPA/tpa-venv
sudo /opt/EDB/TPA/bin/tpaexec setup
```

12.4.2 資格情報に関する奇妙な AWS エラー

TPA サーバーの時刻と日付が正しくない場合、プロビジョニング中にこれと同様の AWS エラーを取得する場合があります。

```
TASK [Register key tpa_cluster in each region]  ****  ****  ****  ****  ****  **** ␣
↪****  ****  ****  ****  **** **
An exception occurred during task execution. To see the full traceback, use -vvv. The␣
↪error was: ClientError: An error occurred (AuthFailure) when calling the␣
↪DescribeKeyPairs operation: AWS was not able to validate the provided access␣
↪credentials
failed: [localhost] (item=eu-central-1) => {"boto3_version": "1.8.8", "botocore_
↪version": "1.11.8", "changed": false, "error": {"code": "AuthFailure", "message":
↪"AWS was not able to validate the provided access credentials"}, "item": "eu-
↪central-1", "msg": "error finding keypair: An error occurred (AuthFailure) when␣
↪calling the DescribeKeyPairs operation: AWS was not able to validate the provided␣
↪access credentials", "response_metadata": {"http_headers": {"date": "Thu, 27 Sep␣
↪2018 12:49:41 GMT", "server": "AmazonEC2", "transfer-encoding": "chunked"}, "http_
↪status_code": 401, "request_id": "a0d905ba-188f-48fe-8e5a-c8d8799e3232", "retry_
↪attempts": 0}}
```

解決策-時刻と日付を正しく設定します。

```
sudo ntpdate pool.ntp.org
```

12.4.3 ロギング

デフォルトでは、すべての tpaexec ログはログファイル<clusterdir>/ansible.log に保存されます

ログファイルの場所を変更するには、環境変数 ANSIBLE_LOG_PATH を目的の場所に設定します。たとえば、

```
export ANSIBLE_LOG_PATH=~/ansible.log
```

ロギングの詳細度を高めるには、 -v /-vv /-vvv /-vvvv /-vvvvv を tpaexec コマンドラインに追加するだけです。

```
tpaexec deploy <clustername> -v
```

そこで

```
- v     shows the results of modules
- vv    shows the files from which tasks come
- vvv   shows what commands are being executed on the target machines
- vvvv  enables connection debugging, what callbacks have been loaded
- vvvvv shows some additional ssh configuration, filepath information
```

12.4.4 クラスターテスト

既存のクラスターをスモークテストする簡単な方法は、次を実行することです。

```
tpaexec test <clustername>
```

これにより、クラスターコンポーネントの機能テストが実行され、続いて pgbench を使用してクラスターのパフォーマンステストが実行されます。pgbench は完了までに時間がかかるため、次のコマンドを実行してベンチマークを省略できます。

```
tpaexec test <clustername> --excluded_tasks=pgbench
```

12.4.5 TPA サーバーテスト

TPA サーバー自分自身のインストールを確認するには、次のコマンドを実行します。

```
tpaexec selftest
```

12.4.6 特定のタスクを含めるまたは除外する

障害が発生した後、またはテストを実行しているときに tpaexec のプロビジョニングまたは展開を再実行する場合、TPA の *Selective task execution* を使用してタスクを見逃すことが役立つ場合があります。

12.5 Selective task execution

12.5.1 タスクセレクターを使用する

`tpaexec deploy` への`--excluded_tasks` および`--included_tasks` オプションを使用して、完全な展開を構成するタスクのサブセットのみを実行するように TPA に指示できます。これらの引数はそれぞれ、セレクターのコンマ区切りリストとして扱われる文字列です。同様に、クラスター全体または個別のインスタンスに対して、`config.yml` の `excluded_tasks` および `included_tasks` 変数を設定できます。 `config.yml` では、コンマ区切り文字列または yaml リストを使用できます。

`excluded_tasks` によって一致するタスクは常に除外されます。`included_tasks` を指定すると、一致しないタスクは暗黙的に除外されます。

以下に詳細を示すように、一部のセレクターはどちらのリストでも使用でき、一部は `excluded_tasks` リストでのみ使用できます。 `tpaexec test` には、別のセレクターセットが使用できます。

12.5.2 例

バーマン関連のタスクを実行せずに展開するには

```
tpaexec deploy <clustername> --excluded_tasks=barman
```

repmgr 関連のタスクのみを実行しているを展開するには

```
tpaexec deploy <clustername> --included_tasks=repmgr
```

インスタンスにホスト名を設定せずに展開するには

```
tpaexec deploy <clustername> --excluded_tasks=hostname
```

ブートストラップおよび ssh タスクが実行されないようにするには、次を `config.yml` に追加します。

```
cluster_vars:
    excluded_tasks:
    - bootstrap
    - ssh
```

12.5.3 `tpaexec deploy` でサポートされているセレクター

次のセレクターは、包含または除外のいずれかがサポートされています。

- バーテンダー

Barman に関連するタスク 。

- bdr

BDR のセットアップに関連するタスク PGD クラスター内で使用される場合を含むこのセレクターが除外された場合、TPA は config.yml で指定されたとおりに拡張機能をインストールおよび構成しますが、ノードグループを作成せず、ノードに参加しようとしません。

- create_pgd_proxy_system_user

pgd_proxy のシステムユーザーを作成するタスク

- create_postgres_system_user

postgres のシステムユーザーを作成するタスク

- efm

EFM に関連するタスク。

- etcd

etcd に関連するタスク

- first-backup

最小限のバーマンバックアップが存在することを保証するタスク。

- ハプロキシ

haproxy に関連するタスク。

- ハープ

harp に関連するタスク。

- パトローニ

パトローニに関連するタスク。

- pem-agent

PEM エージェントに関連するタスク。

- pem-server

PEM サーバーに関連するタスク。

- pem-webserver

PEM サーバーでの Web サーバーの構成に関連するタスク。

- pg-backup-api

Barman の Postgres バックアップ API に関連するタスク。

- pgbouncer

PgBouncer に関連するタスク。

- pgd-proxy

PGD プロキシに関連するタスク。

- pglogical

pglogical に関連するタスク。

- pkg

システムパッケージマネージャーを使用してパッケージをインストールするタスク。

- 展開後

展開後フックが定義されている場合。

- postgres

postgres に関連するタスク。

- レプリカ

実行されるタスクと postgres レプリカとして機能するインスタンス。

- repmgr

repmgr に関連するタスク

- リスタート

サービスを再起動するタスク

- sys

postgres または関連ソフトウェアに固有のタスクの前のシステムセットアップに関連するタスク。

- zabbix-agent

zabbix エージェントに関連するタスク。

次のセレクターは、除外の場合にのみサポートされています。

- アーティファクト

Uploading artifacts に関連するタスク

- バーマン-クリーン

Barman がソースからビルドされている場合、Barman ビルドディレクトリをクリーンアップするタスク。

- barman-pre-config

barman-pre-config フックが定義されている場合。

- bdr-pre-node-creation

bdr-pre-node-creation フックが定義されている場合。

- bdr-post-group-creation

bdr-post-group-creation フックが定義されている場合。

- bdr-pre-group-join

bdr-pre-group-join フックが定義されている場合。

- ブートストラップ

残りの展開が実行される前に、Python およびその他の最小限の依存関係が存在することを確認するタスク。関連する要件を手動でインストールしたことが確かな場合にのみ、これを除外します。

- ビルドクリーン

ソースからビルドされているソフトウェアのビルドディレクトリをクリーンアップするタスク。

- build-configure

ソースからビルドされているソフトウェアを構成するタスク。

- Cloudinit

Cloud-init によって管理されるホストでのみ実行されるタスク。

- コミットスコープ

BDR コミットスコープの構成に関連するタスク。

- config

構成ファイルを作成するタスク。

- efm-pre-config

efm-pre-config フックが定義されている場合。

- efm-post-config

efm-post-config フックが定義されている場合。

- fs

インスタンスでの追加の *Filesystem configuration* のセットアップに関連するタスク。

- ホストキー

Managing SSH host keys を設定するタスク

- ホスト名

ホスト名を設定するタスク。

- ホスト

Configuring /etc/hosts のタスク

- initdb

initdb を実行するタスク

- ローカルリポジトリ

Creating and using a local repository を設定するタスク

- ロケール

Locale をインストールするタスク

- openvpn

OpenVPN を設定するタスク。

- pg-backup-api-clean

Postgres バックアップ API がソースからビルドされている場合、ビルドディレクトリをクリーンアップするタスク。

- pgbouncer-config

pgbouncer の構成ファイルを作成するタスク。

- pgpass

Configuring .pgpass ファイルを作成するタスク。

- リポ後

リポ後フックが定義されている場合。

- postgres-clean

postgres がソースからビルドされている場合、build ディレクトリをクリーンアップするタスク。

- postgres-config

postgres-config フックが定義されている場合。

- postgres-config-final

postgres-config-final フックが定義されている場合。

- 事前展開

展開前フックが定義されている場合。

- pre-initdb

pre-initdb フックが定義されている場合。

- レプリケーションセット

BDR3 クラスター上の監視専用レプリケーションセットに関連するタスク。

- repmgr-clean

repmgr がソースからビルドされている場合、ビルドディレクトリをクリーンアップするタスク。

- repmgr-configure

repmgr がソースからビルドされている場合に構成するタスク。

- リポジトリ

パッケージリポジトリを設定するタスク。

- rsyslog

rsyslog に関連するタスク。

- サービス

構成や再起動を含むシステムサービスに関連するタスク。

- src

ソースからパッケージをビルドおよびインストールするタスク。

- ssh

インスタンス間の ssh の設定に関連するタスク。

- ssh-cluster-key-config

クラスター SSH 公開キーを `ansible_user` のデフォルト root の authorized_keys ファイルに追加するタスク。これは、authorized_keys が外部で管理されているプラットフォームまたは読み取り専用であるプラットフォームでは除外できます。

- sysctl

sysctl 設定を設定およびリロードするタスク。

- sysstat

sysstat サービスに関連するタスク。

- tpa

インスタンスにインストールされる TPA 独自のファイルに関連するタスク。

- ユーザー

システムユーザーの設定に関連するタスク。

- ウォッチドッグ

パトロニクラスター上のカーネルウォッチドッグに関連するタスク。

12.5.4 `tpaexec test` でサポートされているセレクター

次のセレクターは、`tpaexec test` の実行にのみ適用されます。

- 迷彩柄

BDR または PGD クラスターでの CAMO のテストに関連するタスク。

- ddl

BDR または PGD クラスターでの DDL のテストに関連するタスク。

- 失敗

問題が検出されたかどうかを中止するタスク。障害に関係なくテストを実行するには、このセレクターを除外します。

- pgbench

pgbench を実行するタスク

- sys

システムレベルのテストを実行するタスク。

- barman、bdr、haproxy、pg-backup-api、pgbouncer、pgd-proxy、postgres、repmgr、

さまざまなソフトウェアコンポーネントをテストするタスク。

12.6 Reference

12.6.1 Running TPA in a Docker container

利用可能な *TPA installation* がなく、 *Installing TPA from source* の後に TPA を実行することが難しい場合たとえば、動作する Python 3.12 以降のインタープリターを入手するのは簡単ではないため、最後の手段は Docker イメージをビルドすることですそして、Docker コンテナ内で TPA を実行します。

Docker **のインストール** を行うために Docker コンテナで TPA を実行する必要はないことに注意してください。可能であれば、MacOS X でも TPA を直接実行することをお勧めします。

クイックスタート

システムに Docker をインストールして動作している必要があります。

次のコマンドを実行して、Github から tpaexec ソースリポジトリのクローンを作成し、tpa/tpaexec という名前の新しい Docker イメージをビルドします。

```
git clone ssh://git@github.com/EnterpriseDB/tpa.git
cd tpa
docker build -f docker/Dockerfile --build-arg TPA_VER=$(git describe) -t␣
↪tpaexec:latest .
```

作成したイメージをダブルチェックします。

```
docker image ls tpaexec
```

```
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
tpaexec      latest    3943dec4d660   20 minutes ago   658MB
```

```
docker run --rm tpaexec info
```

```
#  TPAexec v23.38.0-38-g4dc030dc1

tpaexec=/usr/local/bin/tpaexec
TPA_DIR=/opt/EDB/TPA
PYTHON=/opt/EDB/TPA/tpa-venv/bin/python3 (v3.13.5, venv)
TPA_VENV=/opt/EDB/TPA/tpa-venv
ANSIBLE=/opt/EDB/TPA/tpa-venv/bin/ansible (v2.16.14)
```

次に、実行しているシェルセッションで tpaexec のエイリアスを設定する必要があります。

```
alias tpaexec="docker run --rm -v $PWD:/work -v /var/run/docker.sock:/var/run/docker.
↪sock tpaexec"
```

これで、次のようなコマンドを実行できます。

```
tpaexec configure cluster -a M1 --postgresql 15 --failover-manager patroni --platform␣
↪docker
tpaexec deploy cluster
```

Docker のインストール

install Docker および get started のヘルプが必要な場合は、Docker documentation を参照してください。

MacOS X では、次のことができます install "Docker Desktop forMac"

そして、アプリケーションメニューから Docker を起動します。

12.6.2 Managing clusters in a disconnected or air-gapped environment

インターネットへの直接接続が許可されないセキュリティ制御された環境では、展開を完了するために TPA が必要とするすべてのパッケージを提供する必要があります。これは、クラスター内の各ノードのローカルリポジトリを介して実行できます。TPA は、*Creating and using a local repository* を介して各ノードでのカスタムリポジトリの追加をサポートし、必要なパッケージは *tpaexec download-packages* コマンドを使用してダウンロードできます。

準備

TPA をインストールできるインターネットに接続されたマシンを選択し、以下の手順に従って、既存のクラスター構成をコピーするか、新しいクラスターを作成します。

> **注釈**
>
> エアギャップサーバーに TPA がまだインストールされていない場合、次の指示に従ってください *TPA をインストールする場所*

インストールします。

切断された環境に既存のクラスターがある場合、インターネットに接続されたホストで必要なのは config.yml だけです。ディレクトリを作成し、そのファイルをコピーします。次に、そのディレクトリで `tpaexec relink` を実行して、通常は `tpaexec configure` によって作成される残りのファイルを生成します。

または、ターゲットインスタンスがネットワークアクセスを持たない環境の新しい構成を作成するには、このオプションを使用して新しいクラスターを構成します。

```
tpaexec configure --use-local-repo-only …
```

これは、 `--enable-local-repo` が行うことをすべて行い、他のすべてのパッケージリポジトリの構成を無効にします。RedHat インスタンスでは、これには、サブスクリプションベースのサービスへのアクセスの無効化も含まれます。

既存のクラスターでは、 `config.yml` で `use_local_repo_only: yes` を設定できます。

```
cluster_vars:
    use_local_repo_only: yes
```

注インターネットに接続された環境と切断された環境で個別のクラスター構成は必要ないことに注意してください、以下のオプションは両方で機能します。

Creating and using a local repository の詳細

パッケージのダウンロード

インターネットに接続されたマシンで、*Docker* **のインストール** があることを確認し、実行します。

```
tpaexec download-packages cluster-dir --os <OS> --os-version <version>
```

tpaexec download-packages の詳細な説明を参照してください。

ターゲット環境へのパッケージのコピー

生成されるリポジトリは、cluster-dir/local-repo ディレクトリに含まれます。これは、ターゲット OS の完全なパッケージリポジトリです。このディレクトリを、接続されたコントローラから、クラスターの展開に使用される切断されたコントローラにコピーします。ディレクトリをクラスターディレクトリの下の同じ場所に配置します。TPA は、 deploy の実行時にパッケージをインスタンスに自動的にコピーします。

切断された環境での展開

クラスター config.yml が **準備**で上記のように構成されていることを確認します。通常どおりに tpaexec provision および deploy を実行します。

切断された環境での更新

Upgrading your cluster コマンドを使用して、エアギャップ環境で更新を実行できます。このコマンドを実行する前に、以下の手順に従う必要があります。

TPA ノードでインターネットに直接アクセスしてローカルリポジトリのコンテンツを更新する

エアギャップ環境にインターネットアクセスがある場合は、tpaexec download-packages を安全に信頼して新しいパッケージをダウンロードし、**パッケージのダウンロード** を確認してその使用法を知ることができます。

TPA ノードでインターネットに直接アクセスせずにローカルリポジトリのコンテンツを更新する

インターネットに切断されたローカルリポジトリの更新を実行するには、2 つの方法があります。正しい **リポジトリの設定とメタデータの生成** に従って、ローカルリポジトリを手動で再作成します

選択したディストリビューションに応じて。または、新しいパッケージをローカルリポジトリに配置した後、tpaexec download-packages の--refresh-repository オプションを使用します。詳細については、**ローカルリポジトリのメタデータを再生成** を確認してください。TPA ノードクラスターに対応する Docker イメージと並んで、Docker が利用可能である必要があります。

変更が保存されたら、`tpaexec upgrade cluster-dir` を実行できます。パッケージが更新されます。

クロスバージョンの共有 Barman

単一の Barman ホストが、共有 barman ノードを管理する初期クラスターとは異なるメジャーバージョンを実行している Postgres インスタンスをバックアップする場合、追加のメジャークライアントパッケージが Barman クラスターの `local-repo` に存在する必要があります。デフォルトの `tpaexec download-packages` 実行は、クラスターのバージョンのみを認識し、共有 barman インスタンスを使用して他のクラスターから追加バージョンをプリフェッチしません。*Backing up nodes at different PostgreSQL versions* を参照

推奨されるワークフロー

12.6.3 Distribution support

TPA は、各ターゲットインスタンスで実行されているディストリビューションを検出して適応します。このページには、アクティブにサポートされているプラットフォームと、以前にサポートされていた「古いディストリビューション」がリストされています。古いプラットフォームへの展開は、必要なパッケージにアクセスできる限り機能する可能性がありますが、これは TPA のサポートされている使用とは見なされず、運用使用には適していません。

完全にサポートされているプラットフォームは、TPA を実行するホストシステムとして、および TPA が Postgres クラスターを展開するターゲットシステムの両方としてサポートされています。

Debian ARM64

- Debian 12/bookworm は完全にサポートされています

Debian x86

- Debian 12/bookworm は完全にサポートされています
- Debian 11/bulseye は完全にサポートされています
- Debian 10/buster は古いディストリビューションです
- Debian 9/stretch は古いディストリビューションです
- Debian 8/jessie は古いディストリビューションです

Ubuntu x86

- Ubuntu 24.04/noble が完全にサポートされています
- Ubuntu 22.04/jammy は完全にサポートされています
- Ubuntu 20.04/focal は古いディストリビューションです
- Ubuntu 18.04/bionic は古いディストリビューションです
- Ubuntu 16.04/xenial は古いディストリビューションです

Oracle Linux x86

- Oracle Linux 9.x が完全にサポートされています
- Oracle Linux 8.x が完全にサポートされています
- Oracle Linux 7.x は古いディストリビューションです

RedHat x86

- RHEL / Rocky/AlmaLinux/Oracle Linux 10.x が完全にサポートされています python3 のみ
- RHEL / Rocky/AlmaLinux/Oracle Linux 9.x が完全にサポートされています python3 のみ
- RHEL / CentOS / Rocky / AlmaLinux 8.x が完全にサポートされています python3 のみ
- RHEL / CentOS 7.x は古いディストリビューション python2 のみ

警告

RHEL 8.x または同等のものを実行しているインスタンスにデプロイするには、TPA コントローラも RHEL 8.x を実行している必要があります。これは、別の OS で実行されている TPA コントローラが ansible-core 2.16 より新しい ansible バージョンを実行している可能性があるため、ターゲットインスタンスで RHEL 8.x と互換性がないためです。

- RHEL / Rocky / AlmaLinux 10.x が完全にサポートされています python3 のみ
- RHEL / Rocky / AlmaLinux 9.x が完全にサポートされています python3 のみ
- RHEL / AlmaLinux 8.x が完全にサポートされています python3 のみ

SLES x86

- SLES 15 SP7 がサポートされています。

注釈

Repmgr を EDBPGE または EPAS と組み合わせて使用する M1 アーキテクチャは、SLES ではサポートされていません。この状況では、コミュニティ PostgreSQL を使用します。

一部のプラットフォームは、ここで記載されている古いディストリビューションで動作しない場合があります。たとえば、Debian 8 および Ubuntu 16.04 は *Docker* **のインストール** では利用できません。

12.6.4 TPA capabilities and supported software

- *TPA Python* **環境のセットアップ**
- *Distribution support*

サポートされているソフトウェア

TPA は、次のメジャーコンポーネントをインストールおよび構成できます。

- Postgres
- EPAS EDB Postgres Advanced Server
- 5、4、3.7 ポンド
- pglogical 3、2 オープンソース
- pgd-cli および pgd-proxy
- HARP 2
- repmgr
- Barman
- pgbouncer
- haproxy PGD 3.7 でのみサポートされています
- フェールオーバーマネージャー EFM
- Postgres Enterprise ManagerPEM

12.6.5 Reconciling changes made outside of TPA

TPA 構成を変更して実行されない TPA 作成クラスターへの変更は、`config.yml` に保存されません。これは、クラスターに TPA 構成で再作成できない変更が含まれることを意味します。

このページでは、TPA で構成が管理される方法と、構成を変更するための推奨される方法を示します。次に、作成する戦略を確認し、クラスターに手動変更を加えた結果を調整します。

手動で構成を変更する必要があるのはなぜですか?

TPA の外部で構成を変更する必要がある最も一般的なシナリオは、実行している操作が TPA でサポートされていない場合です。このような操作で最も一般的な 2 つは、ノードの削除などの破壊的な変更と、Postgres のメジャーバージョンのアップグレードです。

破壊的な変更

一般に、TPA は、クラスターの以前に展開された要素を削除しません。これらは `config.yml` から削除される場合でも。厳密な宣言型システムは、宣言と一致するようにデプロイされたアーティファクトを常に変更する必要があるため、これは人々を驚かせる場合があります。ただし、実稼働データベースに破壊的な変更を行うと、重大な結果を招く可能性があるため、サポートしないことを選択しました。

メジャーバージョンの Postgres のアップグレード

TPA は、Postgres のメジャーバージョンのアップグレードを実行するための自動メカニズムをまだ提供していません。したがって、既存のクラスターでインプレースアップグレードを実行する必要がある場合、これは pg_upgrade や pgd node upgrade などの他のツールを使用して実行する必要があります。

変更がリコンサイルされていない場合、何が起こりますか?

未リコンサイルの変更に関する一般的な問題は、既存の `config.yml` を使用して新しいクラスターを展開する場合、または問題を再現するために `config.yml` を EDB サポートに提供する場合、元のクラスターと一致しないことです。さらに、将来的に TPA を使用してそのクラスターを管理する場合、操作上の問題が発生する可能性があります。

未調整の変更の運用への影響は、変更の性質によって異なります。特に変更が破壊的かどうか、および変更がエラーを発生させるか `config.yml` のデータを無効にすることにより TPA の実行をブロックするかどうか。

非破壊的、非ブロッキングな変更

多くの場合、追加的な変更はすぐに運用上の問題なく対応されます。ユーザーを手動で追加することを検討します。新しいユーザーは存在し続けますが、TPA に関する問題はまったく発生しません。TPA を介してユーザーを管理する場合、`config.yml` で宣言できますが、手動追加ユーザーの存在は操作上の問題を発生させません。

一部の手動追加は、より微妙な効果を持つ場合があります。手動で追加された拡張機能の例を考えます。TPA は破壊的な変更を行わないため、次回 `tpaexec deploy` が実行されたときに拡張機能は削除されません。 **ただし、**、新しい拡張機能に対応するように Postgres 構成を変更した場合、TPA でサポートされているメカニズム以下を参照していない場合、これらは上書きされる可能性があります。

さらに、TPA は手動変更を反映するように `config.yml` ファイルを変更しようとせず、新しい拡張機能は `tpaexec upgrade` から省略されるため、クラスターに互換性のないソフトウェアバージョンが存在する可能性があります。

破壊的なノンブロッキングの変更

簡単に検出され、TPA のオペレーションをブロックしない破壊的な変更は、次回 `tpaexec deploy` が実行されたときに単に元に戻されます。拡張機能を手動で削除することを検討します。TPA の観点から、この状況は、ユーザーが `config.yml` ファイルに拡張機能を追加し、deploy を実行することと区別できません。そのため、TPA は、クラスターと `config.yml` が調整されるように拡張機能を追加しますが、ユーザーが意図したものとは逆の方法です。

同様に、構成パラメーターに手動で加えられた変更は、次の場合を除き、元に戻されます。

1. 手動編集用に予約された `conf.d/9999-override.conf` ファイルで作成されます。
2. `ALTER SYSTEM` SQL を使用して作成されました。または
3. `postgres_conf_settings` を追加して、 *postgres_conf_settings* を作成しました。

オプション 3 が自己文書でポータブルであるという事実を除き、方法 1 または 2 によって行われた変更を調整するための差し迫った運用上の理由はありません。

破壊的な変更をブロックする

`config.yml` 間のより根本的な不一致を作成する変更は、TPA がオペレーションの実行をブロックする可能性があります。たとえば、ベアメタルクラスター内のノードを物理的に削除すると、TPA によるそのノードへの接続は失敗します。つまり、ほとんどの TPA 操作はエラーで終了し、この違いを調整するまで TPA でクラスターを管理できません。。

構成変更を調整する方法

通常、リコンシリエーションプロセスには、クラスターの現在の状態を記述するように config.yml を変更し、tpaexec deploy を実行することが含まれます。

例 PGD ノードの分割

ベアアーキテクチャと次のような configure コマンドを使用して、最小限の PGD クラスターを展開します。

```
tpaexec configure mycluster \
  -a PGD-Always-ON \
  --platform bare \
  --edbpge 15 \
  --location-names a \
  --pgd-proxy-routing local
```

次の SQL を使用してノードを分割します。これは、任意のノードから実行できます。

select * from bdr.part_node(‘node-2’);

deploy を再実行します。エラーは発生していませんが、ノードはまだ分離されたままであることに注意してください。これは、任意のノードでコマンド pgd show-nodes を使用して確認できます。これは、TPA が PGD にノードが分離されたことを伝えるメタデータを上書きしないためです。

> **注釈**
>
> TPA、そして実際に PGD 自分自身には、'parted' 状態でノードを初期化するメカニズムがないため、*config.yml* をこのクラスター状態と調整することはできません。原則として、TPA を使用してこの分割されたクラスターを継続できますが、これはお勧めできません。ほとんどの場合、ノードの完全な削除と `config.yaml` の調整を続行することができます。

前の例では、PGD クラスターからノードを分離しましたが、ノード自分自身は無傷のままで、実行可能だが調整不能な状態で TPA によって管理されています。

ノードを完全に停止するには、node-2 に対応するサーバーの電源をオフにするだけで安全です。この段階で deploy を実行しようとすると、サーバーに到達できず早期に失敗します。

config.yml のこの変更を調整するには、node-2 に対応する instances の下のエントリーを削除するだけです。これは次のようになります。

```
- Name: node-2
  public_ip: 44.201.93.236
  private_ip: 172.31.71.186
  location: a
  node: 2
  role:
    - bdr
    - pgd-proxy
  vars:
```

(次のページに続く)

(前のページからの続き)

```
    bdr_child_group: a_subgroup
    bdr_node_options:
      route_priority: 100
```

これで、TPA を使用してこのノードを通常のように管理できるようになりました。元のクラスターには、状態が PARTED であるノードとして node-2 を参照するメタデータがまだあります。これは、クラスター機能に影響を与えないため、デフォルトでは削除されません。

注釈

config.yml から削除した後、元の `node-2` をクラスターに参加させたい場合は、*config.yml* の削除された行を復元し、Postgres を停止し、そのノードの `PGDATA` ディレクトリを削除して、*tpaexec deploy* を繰り返すことにより実行できます。上記のように、TPA は、対応するエントリが `config.yml` から削除されても、既存のデータベースを削除しないため、このアクションを手動で実行する必要があります。

TPA は、スーパーユーザーのパスワードを自動的に生成します。これは、`tpaexec show-password <cluster> <superuser-name>` を使用して表示できます。パスワードを手動で変更する場合たとえば、psql の/password コマンドを使用すると、次に `tpaexec deploy` が実行された後、パスワードが TPA によって設定されたものに戻っていることがわかります。TPA を介して変更を行い、`tpaexec deploy` の実行全体で持続させるには、`tpaexec store-password <cluster> <superuser-name>` コマンドを使用してパスワードを指定し、`tpaexec deploy` を実行する必要があります。これは、TPA を通じて作成された他のユーザーにも適用されます。

例拡張機能の追加または削除

単純なシングルノードクラスターは、次の `config.yml` を使用して展開できます。

```
- --
architecture: M1
cluster_name: singlenode

cluster_vars:
  postgres_flavour: postgresql
  postgres_version: "15"
  preferred_python_version: python3

instance_defaults:
  image: tpa/debian:11
  platform: docker
  vars:
    ansible_user: root

instances:
  - Name: nodeone
    node: 1
```

(次のページに続く)

(前のページからの続き)

```
    role:
      - primary
```

ノードに接続して apt install postgresql-15-pgvector を実行して、次の SQL コマンド CREATE EXTENSION vector; を実行することにより、pgvector 拡張機能を手動で追加できます。これは、config.yml が以前のようにクラスターを完全に説明しないという事実を超えて、操作上の問題は発生しません。ただし、次のクラスター変数を追加することにより、config.yml を調整するか、実際に単に TPA を使用して拡張機能を追加することをお勧めします。

```
cluster_vars:
  ...
  extra_postgres_packages:
   common:
   - postgresql-15-pgvector
  extra_postgres_extensions:
  - vector
```

この構成を追加した後、 SQL コマンド DROP EXTENSION vector; 、次に apt remove postgresql-15-pgvector を実行して、拡張機能を手動で削除できます。ただし、config.yml を調整せずに tpaexec deploy を再度実行すると、拡張機能は再インストールされます。 config.yml を調整するには、前に追加した行を削除するだけです。

注釈

前述のように、TPA は破壊的な変更を受け入れません。したがって、*config.yml* から行を削除するだけでは拡張機能は削除されません。この操作を手動で実行し、変更を調整する必要があります。

pg_upgrade 、pgd node upgrade 、または別のアウトオブバンドプロシージャを使用してクラスター上の Postgres をアップグレードしている場合、TPA は config.yml を自動的に更新して新しいバージョンを反映しません。後続の tpaexec deploy 実行は、クラスターがまだ以前のバージョンにあったかのように動作し続け、Postgres バージョンに依存する展開時設定 shared_preload_libraries 選択またはバージョン条件の Ansible タスクなどは再評価されません。

config.yml をクラスターの新しい状態と調整するには

1. **``config.yml`` のバージョン変数を更新します。**少なくとも、次を設定します。

```
cluster_vars:
  postgres_version: "<new-major>"
  postgres_package_version: "<new-package-version>"
```

PGD クラスターの場合、同じ手順の一部として BDR がアップグレードされた場合、bdr_version および bdr_package_version も更新します。

1. tpaexec provision を再実行して、新しいバージョンがインベントリに書き込まれます。

```
tpaexec provision <cluster>
```

1. `tpaexec deploy` を再実行して、バージョン条件付き構成が再生成されます。

```
tpaexec deploy <cluster>
```

`tpaexec deploy` は、パッケージのインストールまたはアップグレード自分自身を実行しません。既にインストールされているパッケージの別のバージョンのインストールを拒否します。ただし、構成ファイル、`shared_preload_libraries` エントリ、および `postgres_version` に依存するバージョン条件設定はリフレッシュします。

1. クラスターに関する TPA のビューが現実と一致することを確認します。

```
tpaexec cmd <cluster> all -m shell -a psql -tAc "SELECT version()"
tpaexec test <cluster>
```

すべてのインスタンスは新しい Postgres バージョンを報告する必要があり、`tpaexec test` は合格する必要があります。

> **注釈**
>
> この調整プロシージャは、手動アップグレードによりすべてのインスタンスが一貫した実行状態にあることを前提としています。一部のインスタンスのみがアップグレードされた場合、またはインスタンスが部分的な状態にある場合は、 *tpaexec deploy* を実行する前に最初にそれに対処します。 !!!

12.6.6 EDB Postgres Distributed configuration

TPA は、 EDB Postgres DistributedPGD、以前は BDR 双方向レプリケーションバージョン 3.7、4.x、5.x として知られていたものをインストールおよび構成できます。と 6.x

PGD パッケージへのアクセスは、EDB のパッケージリポジトリを介してのみです。パッケージをダウンロードするには、有効な EDB サブスクリプショントークンが必要です。

このドキュメントでは、PGD 構成のいくつかの側面について触れていますが、詳細の信頼できる説明については PGDdocumentation を参照してください。

はじめに

TPA は、Postgres 自分自身とともに PGD と依存関係をすべての PGD インスタンスにインストールします。

基本的な Postgres のセットアップを完了し、Postgres を起動した後、TPA は `bdr_database` を作成し、以下で説明するさまざまな手順を介して PGD クラスターのセットアップを続行します。

インストール

TPA は、使用中の Postgres のバージョンとフレーバー Postgres、Postgres Extended、または EPAS などに応じて、正しい PGD パッケージをインストールします。

bdr_version を設定して、PGD のどのメジャーバージョン 3、4、5、6 をインストールするかを決定します。bdr_package_version を設定して、インストールする正確なパッケージを決定しますたとえば、最新の 5.0.x をインストールする場合は‘5.0*’ 。

クラスターセットアップの概要

必要なパッケージをインストールし、PGD をロードするように Postgres を構成し、サーバーを起動した後、TPA は PGD ノード、グループ、レプリケーションセット、およびその他のリソースのセットアップを続行します。

次に、TPA が実行する手順の概要を示します。

- 参加するインスタンスごとに PGD ノードを作成します bdr.create_node()
- bdr_node_groups に応じて、1 つ以上の PGD ノードグループを作成します bdr.create_node_group() を使用
- 必要に応じて、レプリケーションセットを作成して、どの変更をレプリケートするかを正確に制御しますノードグループタイプとメンバーシップによって異なります。たとえば、サブスクライバ専用ノードと監視ノードは特別な処理が必要な場合があります。
- 個々のインスタンスで関連するノードグループに参加する
- サブグループ RAFT またはプロキシルーティングを有効にするなど、追加構成を実行します。

このプロセスには、クエリの複雑なシーケンスの実行が含まれます。一部は各インスタンスで順番に、他はパラレルで。手順を簡単に行うために、TPA は任意の PGD プライマリインスタンスをクラスターの「first_bdr_primary」として指定し、これを使用しますインスタンスを使用して、これらのクエリのほとんどを実行します。インスタンスはそれ以外の点では特別ではなく、その ID は PGD 構成自分自身にとって重要ではありません。

インスタンスロール

role に bdr を含むすべてのインスタンスは PGD インスタンスであり、暗黙的に postgres サーバーインスタンスでもあります。

ロールに readonly が含まれる PGD インスタンスは、ロジカルスタンバイノード pause_in_standby 設定で PGD ノードグループに参加します。昇格の対象です。

ロールに subscriber-only を持つ PGD インスタンスはサブスクライバー専用ノードであり、レプリケートされた変更を受信しますが、公開しません。

ロールに witness が含まれる PGD インスタンスは、監視ノードです。

上記のすべての PGD インスタンスは、暗黙的に primary インスタンスでもあります。例外は、ロールに replica を持つインスタンスです。これは、上流の PGD インスタンスの物理的ストリーミングレプリカを示します。このようなインスタンスは、推奨される PGD アーキテクチャには含まれておらず、現在 TPA ではサポートされていません。

構成設定

以下で説明する設定は、クラスター内のすべての PGD インスタンスに均一に設定されるように、通常は `cluster_vars` で設定する必要があります。一部の場合 `bdr_database` など、さまざまなインスタンスに異なる値を設定できますが、他の場合、結果は未定義ですたとえば、すべてのインスタンスが完全に同じ値の `bdr_node_groups` を持つ必要があります。

`cluster_vars` の下でクラスター全体の均一な値を設定することにより、PGD 構成を定義することを強くお勧めします。

bdr_database

`bdr_database` デフォルト bdrdb は PGD で初期化されます。

bdr_client_dsn_attributes

任意の_追加_ parameter keywords supported by libpq

`bdr_client_dsn_attributes` に含めることができます。

`host` 、`port` 、`dbname` 、および `user` は含めないでください。これらは既に接続文字列に含まれています。

これらのドライバーは、libpq C ライブラリが提供する DSN 属性の完全なセットをサポートしていません。

pgd-proxy および/または pgd-cli がインストールされ、 *bdr_client_dsn_attributes* に Go ドライバーによって_unsupported_タイムアウトなどのパラメーターが含まれている場合、2 つの新しい変数をクラスター構成に含める必要があります。

- `pgd_proxy_dsn_attributes` `pgd-proxy-conf` で接続文字列を作成するために使用されます
- `pgd_cli_dsn_attributes` `pgd-cli-conf` で接続文字列を作成するために使用されます

これらの 2 つの文字列には、Go ドライバーと互換性のあるパラメーターキーワードのみが含まれている必要があります。

`bdr_client_dsn_attributes` がサポートされていないパラメーターを含まない場合、これは無視でき、`bdr_client_dsn_attributes` は `pgd-proxy-conf` および `pgd-cli-conf` の接続文字列に含まれます。

`bdr_node_group` の設定デフォルトクラスター名に基づいて、インスタンスがどの PGD クラスターに参加するかを指定します。外部コンポーネントたとえば、pgd-proxy または harp-proxy などの特定のクラスターを識別するためにも使用されます。

bdr_node_groups

これは、グループ参加ステージ前に作成する必要がある PGD ノードグループのリストですクラスターに追加のサブグループが必要な場合。

通常、`tpaexec configure` は、選択したアーキテクチャに基づいて適切な値を生成します。

```
cluster_vars:
  bdr_node_groups:
  - name: topgroup
  - name: abc_subgroup
```

(次のページに続く)

(前のページからの続き)

```
    node_group_type: data
    parent_group_name: topgroup
    options:
      location: abc
  …
```

最初のエントリーは、クラスターの bdr_node_group 用である必要があります。

リストの後続の各エントリは parent_group_name を指定する必要があり、node_group_type オプショナルを指定する場合があります。

各エントリは、グループオプションのオプションのキー/値のマッピングを持つ場合があります。使用可能なオプションは PGD バージョンによって異なります。

bdr_child_group

インスタンスに bdr_node_groups で記載されているグループの名前に bdr_child_group が設定されている場合、bdr_node_group の代わりにそのグループに参加します。

bdr_commit_scopes

これは commit scopes のオプショナルリストです

PGD データベース PGD 4.1 以降で利用可能に存在する必要があります。

```
cluster_vars:
  bdr_commit_scopes:
  - name: somescope
    origin: somegroup
    rule: ALL (somegroup) ON received …`
  - name: otherscope
    origin: othergroup
    rule: …
  …
```

各エントリは、コミットスコープの name 、origin グループの名前、およびコミットスコープ rule を指定する必要があります。グループは bdr_node_groups のエントリに対応する必要があります。

bdr_commit_scopes を明示的に設定すると、TPA は必要に応じてコミットスコープを作成、変更、またはドロップして、データベースが構成と一致することを確認します。設定しない場合、既存のコミットスコープはそのままになります。

preferred_first_bdr_primary

これは、TPA が単一のノードでのみ実行される PGD 関連のタスクに使用するインスタンスの名前です。これには、DDL/DML の実行が含まれ、レプリケーションによって他の PGD ノードに伝播されます。また、新しい PGD ノードを初期化するためのデータの読み取りも含まれます。設定されていない場合、または使用できないインスタンスに設定されている場合、TPA は適切な候補が存在する場合にインスタンスを任意に選択します。

TPA によって選択されたインスタンスは、ファクト first_bdr_primary に保存されます。実行可能なインスタンスのリストは、ファクト first_bdr_primary_candidates に保存されます。これらのファクトはいずれも config.yml から設定できません。フックまたはデバッグで使用するためにここにノートされています。

> **注釈**
>
> *preferred_first_bdr_primary* の選択は、展開されたクラスター内の書き込みリーダーとして選択されるノードに影響を与えません。

単一のオペレーションに preferred_first_bdr_primary を設定する場合は、 config.yml で指定するのではなく、Ansible -e オプションを使用してそれを渡すことができます。例

```
tpaexec deploy . -e preferred_first_bdr_primary=instance_name
```

これは、PGD ノードを追加または再構築しており、たとえば同じデータセンター内のノードからデータを確実にクローンしたい場合に役立ちます。

同様に、各ノードに異なるインスタンスを使用して、preferred_first_bdr_primary をインスタンス変数として設定できます。たとえば、この構成は、 node1 がリビルドされる場合、 tpaexec deploy コマンドは、リビルドのデータソースとして node2 を優先することを意味します。

```
instances:
  - Name: node1
    ...
    instance_vars:
      preferred_first_bdr_primary: node2
```

この構成を使用する場合、ノードが最初に展開された後に適用する必要があります。最初の展開時に preferred_first_bdr_primary の複数の値を設定することはお勧めできません。

その他のノート

フック

TPA は、PGD クラスターセットアッププロセス中に bdr-node-pre-creation、bdr-post-group-creation、および bdr-pre-group-join *TPA hooks* を呼び出します。

データベースの照合順序

TPA は、クラスター内のすべてのインスタンスの PGD データベースが同じ照合順序 LC_COLLATE 設定があることを確認します。同じ PGD クラスター内のデータベースに異なる照合順序があると、データ損失のリスクが発生します。

PGD の古いバージョン

TPA は、 BDR v1 Postgres 9.4 のパッチ適用バージョンを使用、v2 Postgres 9.6、または v3.7 より前の PGD バージョンの展開を積極的にサポートまたはテストしなくなりました。

12.6.7 Barman

インスタンスに config.yml で barman ロールが付与されている場合、TPA は Barman サーバーとして構成して、backup 設定で名前を付けている他のインスタンスのバックアップを取得します。

```
instances:

- Name: one
  backup: two
  …


- Name: two
  role:
  - barman
  …
```

複数の postgres インスタンスは、backup と同じ名前の Barman サーバーを持つことができます。同様に、1 つの postgres インスタンスは、その backup として名前付けた Barman サーバーのリストを持つことができ、バックアップはすべての名前付けサーバーに取得されます。

デフォルトの Barman 構成は、pg_receivewal を使用して PostgreSQL に接続し、WAL の継続的なバックアップを取得し、週に 2 回 ssh での rsync を使用してインスタンスの完全なバックアップを取得します。完全なバックアップと WAL は、過去 4 週間の任意のポイントにリカバリーできるように十分な期間保持されます。

Barman パッケージバージョン

デフォルトでは、TPA は Barman の最新バージョンをインストールします。

config.yml ファイルの cluster_vars セクションの下に barman_package_version: xxx を含めることにより、インストールされる Barman パッケージのバージョンを指定できます。

```
cluster_vars:
    …
    barman_package_version: 1.56.2-1
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

WAL 復元圧縮フラグ

TPA は、インストールされた Barman バージョンを自動的に検出し、restore_command 設定で barman-wal-restore の正しい圧縮フラグを選択します。

- Barman **>= 3.12** --keep-compression を使用します -z オプションはこのリリースでは非推奨でした。
- Barman **< 3.12** -z を使用します

restore_command の構成後に Barman がインストールされるため、最初の展開では、検出ができない場合があります。この場合、TPA はデフォルトで--keep-compression になります。これは、デフォルトでインストールされる最新の Barman バージョンと一致します。古い Barman バージョンを展開する場合は、barman_package_version を明示的に設定して、正しいフラグが選択されていることを確認します。

Barman 構成

Barman サーバーの Barman ホームディレクトリは、クラスター変数 barman_home を使用して設定できます。デフォルト値は/var/lib/barman です。

各 Barman サーバーでは、グローバル構成ファイルが/etc/barman.conf として作成されます。このファイルには、多くの Barman 構成変数のデフォルト値が含まれています。バックアップされる Postgres サーバーごとに、追加の Barman 構成ファイルが作成されます。たとえば、サーバー one をバックアップするには、ファイルは/etc/barman.d/one.conf で、バックアップは Barman ホームディレクトリのサブディレクトリ one に保存されます。構成ファイルとディレクトリ名は、プロビジョニングステップの前に vars セクションで定義されたバックアップインスタンスの backup_name 設定から変更できます。

```
- Name: myPrimary
  backup: myBarman
  platform: bare
  ip_address: x.x.x.x
  node: 1
  role:
    - primary
  vars:
    backup_name: my_backup
```

次の変数はバックアップインスタンスで設定でき、接頭辞 barman_ が削除された状態で Barman の構成に渡されます。

variable	default
barman_archiver	false
barman_log_file	/var/log/barman.log
barman_backup_method	rsync
barman_compression	pigz
barman_reuse_backup	link
barman_parallel_jobs	1
barman_backup_options	concurrent_backup
barman_immediate_checkpoint	false

次のページに続く

表 1 – 前のページからの続き

variable	default
barman_network_compression	false
barman_basebackup_retry_times	3
barman_basebackup_retry_sleep	30
barman_minimum_redundancy	3
barman_retention_policy	RECOVERY WINDOW OF 4 WEEKS
barman_last_backup_maximum_age	1 WEEK
barman_pre_archive_retry_script	
barman_post_backup_retry_script	
barman_post_backup_script	
barman_streaming_wals_directory	
barman_path_prefix	_backed up node's postgres bin **dir**_
backup_name	_backed up instance's **name**_

バックアップ スケジュール

TPA は、`/etc/cron.d/barman` に cron ジョブをインストールします。これは 1 分ごとに実行され、`barman cron` を呼び出してメンテナンスタスクを実行します。

バックアップされるインスタンスごとに、そのインスタンスのバックアップを取得する別の cron ジョブを`/etc/cron.d/<backup_name>` にインストールします。このジョブは、インスタンスの `barman_backup_interval` 変数で決定されたとおりに実行され、デフォルトでは、毎週水曜日と土曜日の 04:00 にバックアップを取得します。

SSH キー

TPA は、 `postgres` および `barman` ユーザーの ssh キーペアを生成し、それぞれの~/.ssh ディレクトリにインストールし、お互いの authorized_keys ファイルに追加します。postgres ユーザーは、WAL セグメント構成されている場合をアーカイブするために barman サーバーに ssh 接続できる必要があり、barman ユーザーは Postgres インスタンスに ssh 接続して、バックアップを取得または復元できる必要があります。

`barman` および `barman_role` Postgres ユーザー

TPA は、2 人の Postgres ユーザー `barman` および `barman_role` を作成します。

TPA バージョン <`23.35` は、 `barman` Postgres ユーザーを `superuser` として作成しました。

`23.35` から `barman` ユーザーは `NOSUPERUSER` で作成されるため、既存のクラスターに再展開すると、`barman` Postgres ユーザーから `superuser` 属性が削除されます。代わりに、`barman_role` には必要な特権セットが付与され、 `barman` ユーザーには `barman_role` メンバーシップが付与されます。

これにより、Barman Manual で提供される特権セットを使用して、 `barman` ユーザーに `superuser` 属性を付与することを回避します。

共有 Barman サーバー

> **注釈**
>
> 23.35 より前の TPA バージョンを使用して作成されたクラスターで共有 Barman 機能を使用するには、以下を行う必要があります。a) 共有 Barman インスタンスの作成をサポートする TPA のバージョンにアップグレードします。 b）TPA をアップグレードした後、$first-cluster で deploy を実行して、TPA が後続のクラスターが共有 Barman ノードに対してスムーズに実行されるように必要な構成変更を加えられるように。

一部の展開では、複数のクラスターで単一の Barman サーバーを共有する必要がある場合があります。tpaexec 内の共有 Barman サーバーの展開は、既存の Barman サーバーを使用する予定がある特定のクラスター構成の Barman サーバーインスタンスの下で `vars:` を介して設定できる `barman_shared` 設定を介してサポートされています。`barman_shared` はブール変数であるため、可能な値は true および false デフォルトです。共有シナリオで Barman 構成を変更する場合、複数のクラスターにわたる構成が同期したままであることを確認して、1 つのクラスターが特定の構成を追加し、2 番目のクラスターがそれをオーバーライドするシナリオを回避する必要があります。

複数のクラスターで共有 Barman サーバーを使用するための一般的なワークフローを以下に説明します。

1. `barman` ロールを持つインスタンスを使用して TPA クラスターを作成しますこの例では、'first-cluster' と呼びます。

2. 2 番目のクラスター second-cluster 例では、この特定の Barman インスタンスを共有 Barman サーバーインスタンスとして$clusters/first-cluster から参照し、プラットフォームとして `bare` を使用するため、実行時に新しい Barman インスタンスを作成しようとしません条項。また、このクラスターがアクセスするために使用できる Barman インスタンスの IP アドレスを指定します。

```
- Name: myBarman
  node: 5
  role:
    - barman
  platform: bare
  ip_address: x.x.x.x
  vars:
    barman_shared: true
```

3. 2 番目のクラスターがプロビジョニングされたら、展開を実行する前に、ssh を介して Barman サーバーインスタンスにアクセスできることを確認します。`ssh-copy-id` を介して 2 番目のクラスターの公開キーを Barman サーバーインスタンスにコピーすることにより、このアクセスを許可でき、ssh を実行して、パスワードを指定せずにログインできることを確認します。

```
# add first-clusters key to the ssh-agent
cd $clusters/first-cluster
ssh-add id_first-clutser
cd $clusters/second-cluster
ssh-keyscan -t rsa,ecdsa -4 $barman-server-ip >> tpa_known_hosts
```

(次のページに続く)

(前のページからの続き)

```
ssh-copy-id -i id_second-cluster.pub -o UserKnownHostsFile=tpa_known_hosts $user@
↪$barman-server-ip
ssh -F ssh_config $barman-server
```

4. Barman ユーザーのキーを最初のクラスターから 2 番目のクラスターにコピーします

```
mkdir $clusters/second-cluster/keys
cp $clusters/first-cluster/keys/id_barman* clusters/second-cluster/keys
```

5. `tpaexec deploy $clusters/second-cluster` を実行します

単一の Barman サーバーは、自分自身とは異なる PostgreSQL メジャーバージョンを実行する Postgres インスタンスをバックアップできます。

`/etc/barman.d/<backup>.conf` のサーバーごとの構成ファイルは、バックアップされたノード自分自身の `postgres_bin_dir` から `path_prefix` を設定するため、Barman はバックアップする各ノードの正しいクライアントバイナリ `pg_basebackup` や `pg_receivewal` などを使用します。`/etc/barman.conf` のグローバル `path_prefix` は、Barman ホスト自分の bin ディレクトリをポイントし続け、同じバージョンのシナリオのデフォルトとして機能します。

バックアップノードの `postgres_version` が Barman ホストの `postgres_version` と異なる場合、TPA は一致する Postgres クライアントパッケージを Barman ホストに自動的にインストールするため `delegate_to` を介して、サーバーごとの `path_prefix` は実際のバイナリに解決されます。

これが機能するには、Barman ホストが追加の PostgreSQL バージョンのパッケージリポジトリにアクセスできる必要があります。EDB エンタープライズリポジトリ `edb_repositories: ["enterprise"]` は、サポートされているすべてのメジャーバージョンをキャリーし、この要件を満たします。TPA の `sys/repositories` ロールは、ホスト自分の `postgres_version` 以外のすべてのメジャーバージョンの PGDG エントリを無効にするため、PGDG のみの展開では、Barman ホストで対応する `pgdgNN` リポジトリエントリを自分で有効にする必要があります。

特定のノードで解決されたパスをオーバーライドするには、そのノードの `vars` で `barman_path_prefix` を設定します上記の変数テーブルを参照してください。

エアギャップクラスター

エアギャップクラスター `use_local_repo_only: true` では、Barman ホストにクロスバージョンクライアントをインストールするには、一致する Postgres クライアントパッケージが Barman クラスターの `local-repo` に既に存在する必要があります。 `tpaexec download-packages` は、クラスターごとに単一の Postgres メジャーのパッケージを取得するため、デフォルトでは、追加メジャークライアントパッケージはプリフェッチされません。

エアギャップ共有 Barman セットアップで追加の Postgres メジャーでノードをバックアップする予定がある場合は、追加バージョンを導入するクラスターを展開する前に、追加メジャークライアントパッケージを Barman クラスターの `local-repo` にコピーします。

ディレクトリレイアウトについては *Managing clusters in a disconnected or air-gapped environment* を、`local-repo` にデータを入力する方法については *tpaexec download-packages* を参照してください。

共有 Barman サーバーの特別な考慮事項

Barman サーバーインスタンスを共有するクラスターをセットアップするときは、注意する必要があります。このようなセットアップを試行する前に、考慮する必要がある重要な側面が多数あります。

1. Barman サーバーを共有するクラスター内の 2 つのインスタンスが同じ名前を使用していないことを確認します。`tpaexec configure` の`--cluster-prefixed-hostnames` オプションは、この点で役立つ場合があります。
2. Barman の構成と設定は、共通の Barman サーバーを使用するすべてのクラスターで同期したままにして、1 つのクラスターが特定の構成をセットアップし、構成が欠落しているか別の値を使用しているために他のクラスターがセットアップしないシナリオを回避する必要があります。
3. 異なるクラスターにわたってバックアップされるインスタンスの Postgres のバージョンは、同じである必要があります。
4. 共通の Barman サーバーを使用する異なるクラスターは、デフォルトをオーバーライドしようとするときに Barman パッケージの異なるバージョンを指定できません。

共有 Barman サーバーのサポートの向上を続けているため、これらの一部は将来のリリースで対応される可能性があります。

警告

共通の Barman ノードを共有するクラスターのプロビジョニングを解除するときは、特に注意してください。特に、Barman を展開した最初のクラスターが非ベアプラットフォームを使用する場合。 Barman ノードは最初のクラスターによって既にプロビジョニング解除され、もう存在しないため、最初に Barman をプロビジョニングおよび展開した最初のクラスターをプロビジョニング解除すると、 Barman ノードを共有する他のクラスターが事実上一貫性のない状態のままになります。

特定のパッケージバージョンにアップグレードしようとする場合は、`config.yml` の `barman_package_version` が更新されて、目的のバージョンを反映していることを確認します。目的のバージョンを追加の引数として `tpaexec upgrade` コマンドに渡すこともできます。

```
- e barman_package_version="<desired version>"
```

package version selection and upgrade のセクションを参照してください

詳細については、

アップグレードのために Barman を選択するには、 `tpaexec upgrade` コマンドに渡される`--components` フラグに `barman` または `all` が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.8 Configuring EFM

failover_manager が efm に設定されている場合、TPA は EFM をインストールおよび構成します。

efm_version を変更することにより、使用する EFM のバージョンを構成できます。

```
cluster_vars:
  …
  failover_manager: efm
  efm_version: 5.2
  …
```

tpaexec configure の実行時に定義することもできます。

```
tpaexec configure my-cluster-dir -a M1 --failover_manager efm --efm-version 5.2
```

EFM は EDB のパッケージリポジトリを介してのみ利用でき、有効なサブスクリプションが必要であることに注意してください。

EFM 構成

TPA は、適切なインスタンス固有の設定で efm.nodes および efm.properties を生成し、残りの設定はそれぞれのデフォルト値に設定されます。 TPA はまた、efm.notification.sh スクリプトも配置します。これは、基本的にデフォルトでは何も含まず、ユーザーが必要に応じて入力するかに任されています。TPA は、auto.allow.hosts および stable.nodes.file のデフォルト設定をオーバーライドして、クラスターへのエージェントの追加を簡素化します。

EFM documentation を参照してください

EFM 構成の詳細については、こちらをご覧ください。

efm_user_password_encryption

scram-sha-256 または md5 のいずれかである必要があります

efm_user_password_encryption を設定して、pg_hba.conf の efm Postgres ユーザーの auth-method の auth-method と、暗号化されたパスワードを生成するときに使用されるアルゴリズムを制御します。

> **注釈**
>
> 新しいクラスターの `efm_user_password_encryption` のデフォルト値は `scram-sha-256` です。

```
efm_user_password_encryption: "scram-sha-256" # or can be set to `md5`
```

efm_conf_settings

efm_conf_settings を使用して、特定のパラメーターを設定できます。これらは、Ansible 辞書のエントリとして key: value フォームで書き込む必要があります

*documentation on the `efm.properties* file <https://www.enterprisedb.com/docs/efm/latest/04_configuring_efm/01_cluster_properties/>`_ を参照してください

構成できる設定の詳細については、 を参照してください。

```
cluster_vars:
  efm_conf_settings:
    notification.level: WARNING
    ping.server.ip: <well known address in network>
```

efm_conf_settings の下の値を変更すると、TPA は常に EFM を再起動して変更をアクティブにします。

efm_user_is_superuser

デフォルトでは、クラスターに展開するときに、TPA はスーパーユーザー権限を持つ efm ユーザーを作成します。 config.yml の cluster_vars に efm_user_is_superuser オプションを追加することにより、この動作を無効にできます。

```
cluster_vars:
  efm_user_is_superuser: false
```

この方法では、EFM は、EFM が動作するために必要な最小限の特権を持つ efm ユーザーを作成します the EFMdocumentation を参照してください。tpaexec deploy を介してこの変数を true から false に、およびその逆に切り替えることにより、EFM のユーザー権限を変更できます。

EFM 証人

TPA は、 role に efm-witness が含まれるインスタンスに監視として EFM をインストールおよび構成します。EFM を使用する場合、暗黙的に efm-witness に変換されるため、ロール witness を使用することもできます。このようなインスタンスの場合、 upstream プロパティを指定して、指定されたプライマリデータベースインスタンスをポイントする必要があります。

Repmgr

EFM はフェールオーバーマネージャーとして機能するため、TPA は引き続き repmgr をインストールして、postgres バージョン 11 以下で postgresql レプリカをセットアップします。repmgrd つまり、この場合、repmgr のデーモンは無効なままで、repmgr の唯一のジョブは、レプリケーションセットアップ機能を提供することです。

postgres バージョン 12 以降の場合、EFM を使用するクラスターは、pg_basebackup を使用してスタンバイノードを作成し、どの形式でも repmgr を使用しません。

ノードのプロモーション

TPA は、ノードのロールとレプリケーショントポロジに基づいて、フェールオーバー中にノードが EFM によるプロモーションの対象かどうかを判断します。EFM 構成を生成するときに次のルールが適用されます。

- **Witness ノード** `efm-witness` ロールはプロモーションできません。
- **``efm-not-promotable`` ロール**を持つノードは昇格の資格がありません。これは、DR または報告ノードなどの特定のスタンバイがプライマリに昇格しないようにするために使用できます。
- **カスケードスタンバイ**プライマリから直接複製していないノードもプロモーション可能ではありません。
- 他のすべてのノードは、デフォルトでプロモーション可能であると考えられます。

スタンバイが昇格することを明示的に防ぐには、クラスター構成のノードの `roles` リストに `efm-not-promotable` を追加します。これにより、EFM は、フェールオーバーイベント中にこのノードを昇格させようとしないことが保証されます。

`tpaexec upgrade` を使用した EFM のマイナー更新

TPA は、新しいバージョンのパッケージをインストールし、構成ファイルを既存の EFM バージョンの構成ディレクトリから新しい EFM バージョンの構成ディレクトリにコピーし、クリーンアップのために既存の EFM バージョンのパッケージ、サービスファイル、およびバイナリディレクトリを削除します。

EFM アップグレードは、インストールおよび構成されている以前のバージョンソースに依存しているため、TPA は最初にこれが true を確認します。ソース EFM バージョンがインストールされていない場合、バイナリディレクトリが存在しない、または構成されていないイベント、構成ディレクトリが存在しない場合、アップグレードはエラーで終了します。アップグレードを実行するときに、ターゲット EFM バージョン `config.yml` の `efm_version` で指定に既にバイナリディレクトリと構成ディレクトリがある場合、EFM は既にインストールされ、目的のバージョンに構成されているため、アップグレードは EFM をスキップします。

> **注釈**
>
> EFM アップグレードは、つまりマイナーアップグレードでなくても、EFM 4.9 から 5.0 からサポートされています。このアップグレードを発生させるために追加の手順は必要ありません。

component selection for upgrade のセクションを参照してください

詳細については、

> **注釈**
>
> EFM は、混合バージョン環境では動作できません。この制限のため、EFM コンポーネントの部分的なアップグレードをサポートする十分な理由はありません。 *efm* がアップグレードするコンポーネントのリストの一部である場合、TPA は**常に**、EFM ノードの完全なセットで EFM ソフトウェアをアップグレードします。 TPA は、どのノードを EFM をアップグレードする必要があるかを決定するときに `update_hosts` を無視します。 !!!

12.6.9 Configuring haproxy

TPA は、 `role` に `haproxy` が含まれるインスタンスに haproxy をインストールおよび構成します。

デフォルトでは、 haproxy は `127.0.0.1:5432` で、同じインスタンスで実行されている *Repmgr リダイレクト pgbouncer* によって転送された要求をリッスンします。要求を転送する `haproxy_backend_servers` のリストを指定する必要があります。

TPA は、デフォルトで haproxy の最新バージョンをインストールします。`haproxy_package_version: 1.9.15*` たとえば、を設定することにより、特定のバージョンをインストールできます。

注 *tpaexec-configure* の package_version でのワイルドカードの使用の制限を参照してください。

Haproxy パッケージは、アーキテクチャのタイプに応じて選択されます。EDB 管理対象 haproxy パッケージを使用できますが、サブスクリプションが必要です。必要に応じて、PGDG エクストラリポジトリからのパッケージをインストールできます。

`haproxy` インスタンスで次の変数を設定できます。

Variable	Default value	Description
haproxy_bind_address	127.0.0.1*	`failover_manager`がパトロニの場合、haproxy が *0.0.0.0* をバインドするアドレス。 * ユーザーはこの値を、クラスターネットワーキングに適したより制限的なものに変更する必要があります**
haproxy_port	5432 (5444 for EPAS)	TCP ポート HA プロキシがリッスンする必要がある
haproxy_read_only_port	5433 (5445 for EPAS)	読み取り専用ロードバランサーの TCP ポート
haproxy_backend_servers	None	Postgres インスタンス名のリスト
haproxy_maxconn	`max_connections`× 0.9	バックエンドサーバーごとに許可される接続の最大数。デフォルトは、バックエンドの`max_connections`設定から取得されます
haproxy_peer_enabled	True*	既知のハプロキシホストを `peer`リストとして追加します。 *`False` `failover_manager`が harp または patroni の場合。

読み取り専用ロードバランサー

データベースへの読み取り専用アクセスを追加ポートでリッスンするように Haproxy を構成できます。現時点では、これは Patroni フェールオーバーマネージャーでのみサポートされています。バックエンドヘルスチェックは、どの postgres インスタンスが現在レプリカとして機能し、ラウンドロビンロードバランシングアルゴリズムを使用してトラフィックを送信するかを判断します。

読み取り専用ロードバランサーはデフォルトで無効になっていますが、cluster_vars 変数 `haproxy_read_only_load_balancer_enabled` を使用して有効にできます。

Patroni ヘルスチェック

Patroni がフェールオーバーマネージャーとして使用される場合、HAProxy は Patroni REST API に対してヘルスチェックを実行します。TPA は、Patroni のセキュリティ設定に基づいて、これらのチェックを自動的に構成します。

- **HTTP:** `patroni_ssl_enabled` が `false` の場合、標準の HTTP チェック `option httpchk` が Patroni API ポートデフォルト 8008 に対して使用されます。
- **HTTPS 基本認証または認証なし** `patroni_ssl_enabled` が `true` で、`patroni_authentication_mode` が `basic` の場合、HAProxy は `check-ssl` を使用し、クラスターの CA 証明書を信頼してパトロニ API サーバー証明書を検証するように構成されています `verify required ca-file <path>` を使用します。
- **HTTPS (mTLS)** `patroni_authentication_mode` が `mtls` の場合、HAProxy は `check-ssl` を使用し、クラスターの CA 証明書_and_認証用の一意のクライアント証明書/キーペアで構成されます `verify required ca-file <path> crt <path>` を使用します。 TPA は、HAProxy のこれらのクライアント証明書を自動的に生成および展開します。

サーバーオプション

TPA は、 `default-server` 行とバックエンドサーバーごとに 1 行があるバックエンドで`/etc/haproxy/haproxy.cfg` を生成します。最初のサーバーを除くすべては「バックアップ」サーバーとしてマークされます。

`haproxy_default_server_extra_options` を haproxy インスタンスのオプションのリストに設定して、`default-server` 行にオプションを追加します。そして、`haproxy_server_options` をバックエンドサーバーのオプションのリストに設定して、各バックエンドの個々のサーバー行にオプションデフォルトをオーバーライドします。TPA は、フェールオーバーマネージャー構成に基づいて、必要な `check` および `check-ssl` オプションを自動的に追加します。

ssl-dh-param-file に使用される RSA キーサイズは、 `cluster_vars` セクションに変数 `ha_proxy_dhparams_key_size` を追加することにより変更できます。

例

```
cluster_vars:
  ha_proxy_dhparams_key_size: 4096
instances:

- Name: one
  vars:
    haproxy_server_options:
    - maxconn 33

- Name: two
…

- Name: proxy
```

(次のページに続く)

(前のページからの続き)

```
role:
- haproxy
vars:
  haproxy_backend_servers:
  - one
  - two
  haproxy_default_server_extra_options:
  - on-error mark-down
  - on-marked-down shutdown-sessions
```

12.6.10 Configuring HARP

failover_manager が harp に設定されている場合、TPA は HARP をインストールおよび構成します。これは、BDR-Always-ON クラスターのデフォルトです。

HARP のインストール

harp-manager および harp-proxy パッケージを提供する必要があります。これらのパッケージへのアクセスを取得するには、EDB に連絡してください。

HARP 構成の変数

HARP documentation を参照してください

HARP 構成の詳細については、

Variable	Default value	Description
cluster_name	``	クラスターの名前。
harp_consensus_protocol	``	使用するコンセンサスレイヤー *etcd*`または *`bdr*
harp_location	*location*	このインスタンスの場所 デフォルトは `location`パラメーター
harp_ready_status_duration	*10*	ノードの準備ができている時間秒更新されない場合、ステータスは持続します。
harp_leader_lease_duration	*6*	リードマスターの秒単位の時間リースは、更新されない場合は持続します。
harp_lease_refresh_interval	*2000*	ミリ秒単位の時間リードマスターリースの更新。
harp_dcs_reconnect_interval	*1000*	ミリ秒単位で測定される間隔、切断されたノードが DCS への再接続を試行する間隔。
harp_dcs_priority	*500*	2 つのノードが等しい量の遅延とリードマスターリースを取得するその他の修飾基準がある場合、これは追加のランキング値として機能し、1 つのノードを別のノードより優先します。

次のページに続く

表 3 – 前のページからの続き

Variable	Default value	Description
harp_stop_database_when_fen	*false*	考えられるすべてのルーティングからノードを単に削除するのではなく、ノードがフェンスされたときにデータベースを停止します。
harp_fenced_node_on_dcs_fai	*false*	HARP が DCS に到達できない場合は、ノードをフェンスします。
harp_maximum_lag	*1048576*	リードマスターロックを取られる前に、以前のリードマスターとこのノードの LSN を記録した間の最大許容分散バイト単位。
harp_maximum_camo_lag	*1048576*	このノードとその CAMO パートナー間で最後に受信した LSN と適用された LSN 間の最大許容分散バイト単位。
harp_camo_enforcement	*lag_only*	CAMO キュー状態を厳密に適用するかどうか。
harp_use_unix_sock	*false*	マネージャーデータベースアクセスには UNIX ドメインソケットを使用します。
harp_request_timeout	*250*	ミリ秒単位の時間 DCS へのクエリーが成功するまでの時間。
harp_watch_poll_interval	*500*	ポーリング DCS 間のスリープするミリ秒。`harp_consensus_protocol`が `bdr`の場合にのみ適用されます。
harp_proxy_timeout	*1*	Lead Master へのビルトインプロキシ接続タイムアウト秒単位。
harp_proxy_keepalive	*5*	ビルトインプロキシがキープアライブ ping を送信する前に、リードマスターへのアイドル接続で待機する時間。
harp_proxy_max_client_conn	*75*	harp-プロキシによって受け入れられるクライアントの最大数 *max_client_conn*
harp_ssl_password_command	None	難読化された ssl パスワードを stdin で受信し、処理された ssl パスワードを stdout 経由で提供するカスタムコマンド。
harp_db_request_timeout	*10s*	dcs -> request_timeout に似ていますが、データベース自分自身への接続用です。
harp_local_etcd_only	None	すべての etcd ノードではなくローカル etcd ノードのみを含むように harp manager エンドポイントリストを制限する

harp-config hook を使用できます

HARP 構成ファイルがインストールされた後にタスクを実行するには追加の構成ファイルをインストールします。

コンセンサスレイヤー

tpaexec configure への--harp-consensus-protocol 引数は、BDR-Always-ON アーキテクチャでは必須です。

etcd

--harp-consensus-protocol etcd オプションが tpaexec configure に指定されている場合、TPA は config.yml で harp_consensus_protocol を etcd に設定し、選択したレイアウトに応じてインスタンスの適切なサブセットに etcd ロールを付与します。

HARP および harp_consensus_protocol: etcd で BDR 4 を使用する場合、少なくとも 1 つのインスタンスに config.yml で定義された etcd ロールが必要です。通常、etcd ロールは BDR プライマリインスタンスに追加されます。TPA は、プロビジョニング中にこの要件を検証し、etcd ロールが見つからない場合は明確なエラーメッセージで失敗します。

HARP v2 には、etcd v3.5.0 以降が必要です。これは、EDB が提供する products/harp/release パッケージリポジトリで入手できます。

etcd の次のパラメーターを構成できます。

Variable	Default value	Description
etcd_peer_port	2380	etcd がピア通信に使用するポート
etcd_client_port	2379	クライアントが etcd に接続するために使用するポート

bdr

tpaexec configure に--harp-consensus-protocol bdr オプションが指定されている場合、TPA は config.yml で harp_consensus_protocol から bdr を設定します。この場合、既存の PGD インスタンスはコンセンサスに使用され、追加の構成は必要ありません。

harp プロキシの別のユーザーの構成

harp プロキシで別の読み取り専用ユーザーを使用する場合は、cluster_vars の下で harp_dcs_user: username を設定することにより、それを指定できます。 TPA は harp_dcs_user 設定を使用して読み取り専用ユーザーを作成し、DCS 構成で設定します。

harp manager 用の別のユーザーの設定

harp manager で別のユーザーを使用する場合は、cluster_vars の下で harp_manager_user: username を設定することにより、それを指定できます。TPAexec はその設定を使用して、新しいユーザーを作成し、bdr_superuser ロールを付与します。

カスタム SSL パスワードコマンド

harp_ssl_password_command が提供するコマンドは、HARP が使用して、接続文字列で指定された sslpassword の難読化を解除します。 sslpassword が存在しない場合、harp_ssl_password_command は無視されます。 sslpassword が難読化されていない場合、 harp_ssl_password_command は必要なく、指定しないでください。

harp サービスの構成

harp サービスの次のパラメーターを構成できます。

Variable	Default value	Description
harp_manager_restart_on_fail	*false*	`true`の場合、`harp-manager`サービスはオーバーライドされるため、失敗時に再起動されます。デフォルトは`false`で、`harp-manager`パッケージによってインストールされるサービスに準拠します。

harp http(s) ヘルスプローブの構成

サービスの状態を監視するための API エンドポイントを提供する harp の http(s) サービスを有効にして構成できます。

Variable	Default value	Description
harp_http_options	 `enable: false` `secure: false` `host: <inventory_hostname>` `port: 8080` `probes: `*`timeout: 10s`*` `*`endpoint: "host=<proxy_name> port=<6432> dbname=<bdrdb> user=<username>"`*	http(s) API 設定を定義する harp config.yml の http セクションを構成します。

変数には、次のキーを含めることができます。

```
enable: false
secure: false
cert_file: "/etc/tpa/harp_proxy/harp_proxy.crt"
key_file: "/etc/tpa/harp_proxy/harp_proxy.key"
host: <inventory_hostname>
```

(次のページに続く)

(前のページからの続き)

```
port: 8080
probes:
  timeout: 10s
endpoint: "<valid dsn>"
```

cert_file および key_file キーは、 secure: true を使用し、独自の証明書とキーを使用する場合に必要です。

deploy を実行する前に、証明書とキーの両方がターゲットノードの指定された場所で利用できることを確認する必要があります。

クラスター固有の CA 証明書を使用して TPA に証明書とキーを生成する場合、cert_file および key_file の両方を空のままにします。 TPA CA 証明書は「ウェルノウン」になりません。この証明書を、エンドポイントをプローブする各マシンの信頼ストアに追加する必要があります。CA 証明書は、TPA ノードのクラスターディレクトリで deploy の後の<cluster_dir>/ssl/CA.crt にあります。

使用可能な API エンドポイントの詳細については、harp ドキュメントを参照してください。

12.6.11 Configuring Postgres Enterprise Manager (PEM)

TPA は、 --enable-pem コマンドラインオプションを使用して tpaexec configure コマンドが実行されると、PEM のインストールと構成を行います。

--enable-pem のデフォルト動作は、クラスター内のすべての postgres インスタンスの pem-agent ロールを有効にすることです。 pem-agent ロールは、 --enable-pg-backup-api コマンドラインオプションが--enable-pem とともに使用される場合、バーマンノードにも追加されます。

pemserver という名前の専用インスタンスもクラスターに追加されます。

PEM サーバーは postgres バックエンドを使用するため。pemserver インスタンスは、 postgres ロールを暗黙的に使用するため、pemserver が PEM バックエンドとして使用するように構成された有効な postgres クラスターを取得します。通常の postgres インスタンスで使用可能なすべての構成オプションは、PEM のバックエンド postgres インスタンスでも有効です。詳細については、以下を参照してください。

- *pg_hba.conf*
- *postgresql.conf*

PEM は EDB のパッケージリポジトリを介してのみ利用可能であるため、有効なサブスクリプションが必要であることに注意してください。

サポートされているアーキテクチャ

PEM は、 --enable-pem 構成コマンドラインオプションを介してすべてのアーキテクチャでサポートされています。ただし、EDB Postgres Extended で使用する場合の BDR -Always-ON アーキテクチャを除きます。オプションで、生成されたクラスター構成 config.yml を編集し、クラスター内の postgres インスタンスから pem-agent ロールを割り当てまたは削除して、そこで PEM を有効または無効にすることができます。

PEM コンポーネントパッケージバージョン

デフォルトでは、TPA は PEM エージェントと PEM サーバーの最新の利用可能なバージョンをインストールします。

`config.yml` ファイルの `cluster_vars` セクションの下に `pem_agent_package_version: xxx` および `pem_server_package_version: xxx` を含めることにより、インストールされる PEM エージェントと PEM サーバーパッケージのバージョンを指定できます。

```
cluster_vars:
    …
    pem_agent_package_version: 9.7.0-1.el9
    pem_server_package_version: 9.7.0-1.el9
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに `2:...` のようなエポック修飾子がある場合に必要になることがよくあります。

PEM 構成

TPA は、適切なインスタンス固有の設定で pem エージェントと pem サーバーを構成し、残りの設定はそれぞれのデフォルト値に設定されます。構成オプションの一部は、将来のある時点でユーザー構成用に公開される可能性があります。

PEM エージェントの登録時に追加オプションを渡す

TPA は、 `pemworker` ユーティリティの`--register agent` コマンドを使用して、クラスター内の各 PEM エージェントを登録します。

クラスター構成に `pemagent_registration_opts` を含めることにより、追加登録オプションのリストを渡すことができます。

例

```
pemagent_registration_opts:
  - --enable-smtp true
  - --enable-heartbeat-connection
  - --allow-batch-probes true
  - -l DEBUG1
```

PEM documentation

に、登録オプションの詳細をリストします。

pem エージェントを使用するノードの便利な拡張機能

デフォルトでは、TPA は sql_profiler、edb_wait_states、および query_advisor 拡張機能を pem-agent ロールを持つインスタンスに追加します。

pem-agent ノードのデフォルトの拡張機能のこのリストは、config.yml で pemagent_extensions を設定することによりオーバーライドできます。

このリストが空の場合、拡張機能は自動的に含まれません。

PEM サーバー SSL 認証の外部証明書の提供

デフォルトでは、PEM サーバーは自己署名証明書ペア server-pem.crt および server-pem.key を作成し、HTTPS アクセスにそれらを使用するようにウェブサーバーを構成します。

server-pem.key のサイズは、変数 pem_rsa_key_size を cluster_vars セクションに追加して変更できます。

```
(...)
cluster_vars:
  pem_rsa_key_size: 4096
```

PEM データベースによって促進される CA 証明書のサイズは、変数 pem_db_ca_certificate_key_size を cluster_vars セクションに追加して変更することもできます。

```
(...)
cluster_vars:
  pem_db_ca_certificate_key_size: 4096
```

デフォルトでは、自己署名 TLS 証明書の期間は 10 年です。期間は、cluster_vars セクションで openssl_certificate_validity を定義することにより調整できます。証明書を置き換える必要がある場合は、tpaexec deploy tpaexec deploy (...) -e pem_web_server_renew_tls_certificates=true で pem_web_server_renew_tls_certificates 変数を使用します。

独自の証明書ペアを提供するには、 ssl/pemserver という名前のクラスターディレクトリのルートの下にディレクトリを作成し、その中に証明書ペアを配置します。

```
cluster directory
├─── ssl
│    └─── pemserver
│        ├─── externally-provided.crt
│        └─── externally-provided.key
```

次に、 pem サーバーインスタンスの下の vars: またはクラスター構成ファイルの cluster_vars の値として、各ファイル名を使用して変数 pem_server_ssl_certificate および pem_server_ssl_key を設定します。

TPA は、これらのファイルの pem サーバーインスタンスへのコピーを処理し、それに応じてウェブサーバーを構成します。

```
- Name: pemserver
  location: main
  node: 4
  role:
    - pem-server
  vars:
    pem_server_ssl_certificate: externally-provided.crt
    pem_server_ssl_key: externally-provided.key
```

PEM でのインスタンスの整理

PEM は、サーバーおよび/またはエージェントの **グループ** および**クラスター**への照合をサポートしています。次のインスタンス変数を設定することにより、TPA からこのグループ化を指定できます。

pem-agent ロールはあるが、pem-server ロールがないインスタンス

- monitoring_group は、このインスタンスの Postgres サーバーが割り当てられるグループの名前を指定します。指定しない場合、デフォルトの pem_server_group の値が使用されます。
- monitoring_agent_group は、このインスタンスの PEM エージェントを割り当てるグループの名前を指定します。指定しない場合、デフォルトの pem_agent_group の値が使用されます。
- monitoring_cluster は、このインスタンスの Postgres サーバーが割り当てられるクラスターの名前を指定します。指定しない場合、クラスターは割り当て/作成されません。
- monitoring_agent_cluster は、このインスタンスの PEM エージェントを割り当てるクラスターの名前を指定します。指定しない場合、クラスターは割り当て/作成されません。

> **注釈**
>
> PEM は、特定のクラスター名が単一のグループに存在することのみを許可するため、これらの値の一部の組み合わせは実行できません。たとえば、同じ `monitoring_cluster` と `monitoring_agent_cluster` を指定したが、異なる `monitoring_group` と `monitoring_agent_group` を指定した場合、これは同じクラスター名が 2 つの異なるグループに表示されることを意味します。 TPA を使用すると、PEM はこのような不一致を防止するのではなく、処理できます。これは、一般に `deploy` は成功するが、予想したサーバーとエージェントの構成を正確に取得できない場合があることを意味します。

- pem_server_group は、この PEM サーバーに登録されている Postgres サーバーが割り当てられるデフォルトグループの名前を指定します。指定しない場合、デフォルトの PEM Server Directory が使用されます。
- pem_agent_group は、この PEM サーバーに登録された PEM エージェントが割り当てられるデフォルトグループの名前を指定します。指定しない場合、デフォルトの PEM Agents になります

pem-server ロールを持つインスタンスの Postgres サーバーと PEM エージェントは、常にそれぞれ pem_server_group および pem_agent_group に割り当てられることに注意してください。TPA は、このサーバーまたはエージェントのクラスターへの追加をサポートしていません。

共有 PEM サーバー

一部の展開では、組織内の複数のクラスターの監視と管理に単一の PEM サーバーを使用する場合があります。tpaexec 内の共有 pem サーバーの展開は、既存の pem サーバーを使用する予定がある特定のクラスター構成の pem サーバーインスタンスの下で `vars:` を介して設定できる `pem_shared` 変数を介してサポートされています。`pem_shared` はブール変数であるため、可能な値は true および false デフォルトです。pemserver インスタンスを共有として宣言する場合、pemserver インスタンスが実際には、最初に pem サーバーをプロビジョニングおよび展開した別のクラスター構成によって管理されていることを指定されたクラスター構成に伝えます。したがって、pem の postgres バックエンドを含む pem サーバーインスタンスに加えたい変更は、pemserver インスタンスが共有 pem インスタンスとして宣言されていないクラスターによって管理されます。

複数のクラスターで共有 pem サーバーを使用するための一般的なワークフローは次のとおりです。

1. `pem-server` ロールを持つ単一のインスタンスを使用して tpaexec クラスターを作成しますこの例では、’pem-cluster’ と呼びます。pem サーバーとして実行される単一のインスタンスではなく、大規模なクラスターの一部としてプロビジョニングされるシナリオでも、同じワークフローを簡単に使用できますが、例として使用する方が簡単であるため、おそらくシングルノードクラスターを使用しますメンテナンスも簡単です。

2. 他のクラスターたとえば pg-cluster では、共有 pem サーバーインスタンスとして$clusters/pem-cluster からこの特定の pem サーバーを参照し、プラットフォームとして `bare` を使用するため、新しい pemserver インスタンスを作成しようとはしていません。また、このクラスターが pemserver インスタンスにアクセスするために使用できる pemserver の IP アドレスを指定します。

```
- Name: pemserver
  node: 5
  role:
    - pem-server
  platform: bare
  public_ip: 13.213.53.205
  private_ip: 10.33.15.102
  vars:
    pem_shared: true
```

3. postgres クラスターで deploy を実行する前に、pg-cluster が ssh を介して pem サーバーインスタンスにアクセスできることを確認してください。`ssh-copy-id` を介して pg-cluster の公開キーを pem サーバーインスタンスにコピーすることにより、このアクセスを許可でき、ssh を実行して、パスワードを指定せずにログインできることを確認します。

```
# add pem-clusters key to the ssh-agent (handy for `aws` platform)
$ cd $clusters/pem-cluster
$ ssh-add id_pem-clutser
$ cd $clusters/pg-cluster
$ ssh-keyscan -4 $pem-server-ip >> known_hosts
$ ssh-copy-id -i id_pg-cluster.pub -o UserKnownHostsFile=tpa_known_hosts $user@$pem-
↪server-ip
$ ssh -F ssh_config pemserver
```

4. pem サーバーノードの postgresql 構成を更新して、新しい pg-cluster からの接続を許可します。pem-cluster の config.yml の vars: の下の pem_postgres_extra_hba_settings に新しいエントリを追加することにより、pem サーバー上の既存の pg_hba.conf を変更できます。例

```
instances:
  - Name: pemserver
    location: main
    node: 1
    role:
      - pem-server
    vars:
      pem_postgres_extra_hba_settings:
        - "# Allow pem connections from pg-cluster1.quire"
        - hostssl pem +pem_agent 10.33.15.108/32 cert
        - "# Allow pem connections from pg-cluster1.upside"
        - hostssl pem +pem_agent 10.33.15.104/32 cert
        - "# Allow pem connections from pg-cluster2.zippy"
        - hostssl pem +pem_agent 10.33.15.110/32 cert
        - "# Allow pem connections from pg-cluster2.utopic"
        - hostssl pem +pem_agent 10.33.15.109/32 cert
```

次に、 tpaexec provision $clusters/pem-cluster に続いて tpaexec deploy $clusters/pem-cluster を実行します。完了すると、新しい pg-クラスターのノードは pem サーバーバックエンドと通信できるようになります。

5. pg-cluster 内のノードの pem エージェントが pem サーバーバックエンドに接続して登録できることを確認するには、tpaexec deploy を実行する前に最初に export EDB_PEM_CREDENTIALS_FILE=/path/to/pem/credentials/file を行う必要があります。資格情報ファイルは、pem サーバーのバックエンド postgres インスタンスへのアクセス資格情報を username:password 形式で含むテキストファイルです。

```
cat pem_creds
```

```
postgres:f1I%fw!QmWevdzw#EL#$Ulu1cWhg7&RT
```

バックエンドパスワードがわからない場合は、 show-password tpaexec コマンドを使用して取得できます。

```
tpaexec show-password $pem-clusterdir $user
```

6. tpaexec deploy $clusters/pg-cluster を実行して、共有 pem サーバーインスタンスを使用しながら pem が新しい pg クラスターに展開されます。

PEM UI は https インターフェイスで実行されるため、https://\protect\T1\textdollarpem-server-ip/pem を介して PEM サーバーの実行中のインスタンスに接続できます。PEM UI のログイン資格情報は、postgresql および epas フレーバーのそれぞれに postgres または enterprisedb を使用する postgres バックエンドユーザーに設定されます。tpaexec の show-password コマンドは、バックエンドユーザーのパスワードを表示します。例

```
tpaexec show-password $clusterdir $user
```

PEM documentation を参照

PEM の構成と使用法の詳細については、こちらをご覧ください。

tpaexec upgrade を使用した PEM のマイナー更新

```
The PEM server package includes an SQL upgrade script which handles upgrading the␣
↪database according to the PEM schema version.
Upgrading from PEM `v9.8.0` to PEM `10.1.1` using `tpaexec upgrade` has been tested␣
↪and works, but is a major version upgrade.
```

PEM サーバーを特定のパッケージバージョンにアップグレードしようとする場合、config.yml の pem_server_package_version が更新されて、目的のバージョンを反映していることを確認します。

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade <cluster_dir> \
  -e pem_agent_package_version="<desired version>" \
  -e pem_server_package_version="<desired version>" \
  --components=pem-agent,pem-server
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードする PEM エージェントを選択するには、 tpaexec upgrade コマンドに渡される--components フラグに pem-agent または all が含まれていることを確認します

アップグレードする PEM サーバーを選択するには、tpaexec upgrade コマンドに渡される--components フラグに pem-server または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.12 PgBouncer

PgBouncer パッケージバージョン

デフォルトでは、TPA は PgBouncer の最新バージョンをインストールします。

config.yml ファイルの cluster_vars セクションの下の pgbouncer_package_version: xxx を含めることにより、インストールされる PgBouncer パッケージのバージョンを指定できます。

```
cluster_vars:
    …
    pgbouncer_package_version: 1.8*
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

PgBouncer の構成

TPA は、 role に pgbouncer が含まれるインスタンスに PgBouncer をインストールおよび構成します。

デフォルトでは、PgBouncer はポート 6432 で接続を監視し、pgbouncer_backend が指定されない場合、接続を 127.0.0.1:5432 アーキテクチャに応じて Postgres または *Configuring haproxy* のいずれかに転送します。

Variable	Default value	Description
pgbouncer_port	6432	pgbouncer がリッスンする TCP ポート
pgbouncer_backend	127.0.0.1	接続する Postgres サーバー
pgbouncer_backend_port	5432	`pgbouncer_backend`がリッスンするポート
pgbouncer_max_client_conn	`max_connections`× 0.9	許可される接続の最大数。デフォルトは、可能であればバックエンドの `max_connections`設定から取得されます
pgbouncer_auth_user	pgbouncer_auth_user	認証に使用する Postgres ユーザー

データベース

デフォルトでは、TPA は [databases] の下に単一のワイルドカード* エントリーを使用して/etc/pgbouncer/pgbouncer.databases.ini を生成し、すべての接続をバックエンドサーバーに転送します。次の例に示すように pgbouncer_databases を設定して、データベース構成を変更できます。

認証

PgBouncer は pgbouncer_auth_user として Postgres に接続し、構成済みの auth_query を実行してユーザーを認証します。

PgBouncer によって auth_query として使用される pgbouncer_get_auth() ファンクションは、単一のデータベース pgbouncer_auth_database に作成されます。このファンクションの実行権限は pgbouncer_auth_user に付与されます。

例

```
instances:
  - Name: one
    vars:
      max_connections: 300
  - Name: two
  - Name: proxy
    role:
```

(次のページに続く)

(前のページからの続き)

```
    - pgbouncer
  vars:
    pgbouncer_backend: one
    pgbouncer_databases:
      - name: dbname
        options:
          pool_mode: transaction
          dbname: otherdb
      - name: bdrdb
        options:
          host: two
          port: 6543
```

tpaexec upgrade を使用した PgBouncer のマイナー更新

特定のパッケージバージョンにアップグレードしようとする場合は、config.yml の pgbouncer_package_version が更新されて、目的のバージョンを反映していることを確認します。

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade <cluster_dir> \
  -e pgbouncer_package_version="<desired version>" \
  --components=pgbouncer
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードのために PgBouncer を選択するには、tpaexec upgrade コマンドに渡される--components フラグに pgbouncer または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.13 pgd-proxy

pgd-proxy パッケージバージョン

デフォルトでは、TPA は pgd-proxy の利用可能な最新バージョンをインストールします。

config.yml ファイルの cluster_vars セクションの下に pgd_proxy_package_version: xxx を含めることにより、インストールされる pgd-proxy パッケージのバージョンを指定できます。

```
cluster_vars:
    …
    pgd_proxy_package_version: 5.0.0-1
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

pgd-プロキシの構成

TPA は、 role に pgd-proxy を含むインスタンスに、PGD 5 を使用した PGD-Always-ON アーキテクチャの pgd-proxy をインストールおよび構成します。

デフォルトでは、 *PGD-Always-ON*

は、すべての場所のすべてのデータノードで pgd-proxy を実行しますが、代わりに--add-proxy-nodes-per-location 3 を使用して任意の数の追加プロキシインスタンスを作成できます。

構成

pgd-proxy は、 SQL ファンクションを介して PGD レベルで構成されます。

Hash	Function	Description
pgd_proxy_options	*bdr.alter_proxy_option()*	pgd-proxy 構成、例ポート
bdr_node_groups	*bdr.alter_node_group_option()*	プロキシのノードグループの構成、例 `enable_proxy_routing`
bdr_node_options	*bdr.alter_node_option()*	個々の PGD ノードのルーティング構成

詳細については、 PGD ドキュメントを参照してください。

pgd-proxy-config hook を使用できます

PGD PROXY 構成ファイルがインストールされた後にタスクを実行するには追加の構成ファイルをインストールします。

bdr_node_groups

pgd-proxy に関連するグループレベルのオプションは、他のノードグループオプションとともに bdr_node_groups の下で設定できます。

```
cluster_vars:
  bdr_node_groups:
  - name: group1
    options:
      enable_proxy_routing: true
```

グループで pgd-プロキシを有効にするには、enable_proxy_routing を明示的に true に設定する必要があることに注意してください。

bdr_node_options

pgd-proxy に関連するノードレベルのオプションは、PGD インスタンスの bdr_node_options の下で設定できます。

```
instances:

- Name: first
  vars:
    bdr_node_options:
      route_priority: 42
```

pgd_proxy_options

pgd-proxy インスタンスが接続されているグループまたはノードではなく、pgd-proxy インスタンス自分自身のオプションは、default_pgd_proxy_options の下、cluster_vars すべてのプロキシに適用されます、または pgd-proxy インスタンスの pgd_proxy_options の下で設定できます。

```
cluster_vars:
  default_pgd_proxy_options:
    listen_port: 6432
    read_listen_port: 6433

instances:

- Name: someproxy
  vars:
    pgd_proxy_options:
      listen_port: 9000
      read_listen_port: 9001
```

この場合、他のインスタンスは cluster_vars から listen_port 設定を取得しますが、 someproxy はそのデフォルト設定をオーバーライドし、インスタンスの vars セクションで独自の listen_port を構成します。

PGD プロキシ http(s) ヘルスプローブ

プロキシの状態をモニタするための API エンドポイントを提供する PGD プロキシの http(s) サービスを有効にして構成できます。

cluster_vars またはインスタンス vars の下の pgd_http_options は、pgd-proxy-config.yml の proxy トップセクションの http サブセクションの下にある http API を定義するすべての設定を保存します。

変数には、次のキーを含めることができます。

```
enable: false
secure: false
cert_file: "/etc/tpa/harp_proxy/harp_proxy.crt"
```

(次のページに続く)

(前のページからの続き)

```
key_file: "/etc/tpa/harp_proxy/harp_proxy.key"
host: <inventory_hostname>
port: 8080
probes:
  timeout: 10s
endpoint: "<valid dsn>"
```

cert_file および key_file キーは、 secure: true を使用し、独自の証明書とキーを使用する場合に必要です。

deploy を実行する前に、証明書とキーの両方がターゲットノードの指定された場所で利用できることを確認する必要があります。

クラスター固有の CA 証明書を使用して TPA に証明書とキーを生成する場合、cert_file および key_file の両方を空のままにします。 TPA CA 証明書は「ウェルノウン」になりません。この証明書を、エンドポイントをプローブする各マシンの信頼ストアに追加する必要があります。CA 証明書は、TPA ノードのクラスターディレクトリで deploy の後の<cluster_dir>/ssl/CA.crt にあります。

使用可能な API エンドポイントの詳細については、pgd-proxy ドキュメントを参照してください。

tpaexec upgrade を使用した pgd-proxy の更新

特定のパッケージバージョンにアップグレードしようとする場合は、config.yml の pgd_proxy_package_version が更新されて、目的のバージョンを反映していることを確認します。このバージョンは、クラスターに指定された bdr_package_version および pgdcli_package_version バージョンと一致する必要があります。bdr_package_version を定義し、config.yml から pgd_proxy_package_version および pgdcli_package_version を省略すると、3 つのコンポーネントすべてで同じ値が使用されます。

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade <cluster_dir> \
  -e pgd_proxy_package_version="<desired version>" \
  --components=pgd-proxy
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードのために pgd-プロキシを選択するには、 tpaexec upgrade コマンドに渡される--components フラグに pgd-proxy または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.14 pglogical

pglogical パッケージバージョン

デフォルトでは、TPA は pglogical の利用可能な最新バージョンをインストールします。

config.yml ファイルの cluster_vars セクションの下に pglogical_package_version: xxx を含めることにより、インストールされる pglogical パッケージのバージョンを指定できます。

```
cluster_vars:
    …
    pglogical_package_version: 2.2.0*
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

pg 論理構成

TPA は、pglogical v2 および pglogical v3 を使用して pglogical レプリケーションセットパブリケーションとサブスクリプションを構成できます。

```
instances:

- node: 1
  Name: kazoo
  …
  vars:
    publications:
    - type: pglogical
      database: example
      name: some_publication_name
      replication_sets:
      - name: custom_replication_set
        …



- node: 2
  Name: keeper
  vars:
    subscriptions:
    - type: pglogical
      database: example
      name: some_subscription_name
      publication:
        name: some_publication_name
```

(次のページに続く)

(前のページからの続き)

```
    replication_sets:
      - default
      - default_insert_only
      - custom_replication_set
    ...
```

type: pglogical でパブリケーションまたはサブスクリプションを定義すると、pglogical 拡張機能がデフォルトで作成されますが、どのバージョンをインストールするかは自分自身ですたとえば、pglogical3 の products/pglogical3/release リポジトリをサブスクライブします。

はじめに

TPA は、pglogical を使用してインスタンス間で変更をレプリケートするために必要なすべてを構成でき、config.yml の変更に基づいてレプリケーションセットアップを変更することもできます。

変更を公開するには、 publications に type: pglogical を使用してエントリーを定義します。これらの変更をサブスクライブするには、上記のように、subscriptions に type: pglogical を使用してエントリーを定義します。

Pglogical には、名前付けパブリケーションエンティティがありませんビルトイン論理レプリケーションが CREATE PUBLICATION がある意味で。config.yml のパブリケーションは、レプリケーションセットのコレクションに名前を割り当てるだけであり、サブスクリプションはこの名前を使用して、目的のプロバイダーを参照できます。

pg 論理レプリケーションを使用するには、パブリッシャーとサブスクライバーの両方が名前付けローカル pg 論理ノードが必要です。TPA はこのノードが存在しない場合、pglogical.create_node() で作成します。パブリケーションの場合、パブリケーション名が pg 論理ノード名として使用されます。特定のデータベースには 1 つの pg 論理ノードのみが存在できるため、データベースごとに publications のエントリは 1 つのみです。

ただし、pglogical サブスクリプション *do* には、独自の名前があります。TPA は、指定された name でサブスクリプションを作成し、インスタンスの名前とサブスクリプションが作成されるデータベースの名前に基づいて、pg 論理ノード名のデフォルト値を使用します。必要に応じて別の node_name を指定できます。たとえば、同じデータベース内のパブリケーションを構成している場合、そのデータベース内のすべてのサブスクリプションが同じ pg 論理ノードを共有する必要があります。

TPA は、構成の基本的な検証を行います。replication_sets を replciation_sets と書くか、定義されていないパブリケーションをサブスクライブしようとすると、エラーを指摘しますが、意味のあるパブリケーションとサブスクリプションのセットを指定するのはあなたの責任です。

TPA は、ユーザー、拡張機能、データベースを作成した後、PGD 構成の前に pglogical を構成します。*Creating Postgres users* および *Creating Postgres databases* を設定してレプリケーション用のデータベースを作成し、*postgres-config-final* を使用できます。

pglogical が構成する前にデータベースにデータを設定するフック。

出版物

publications のエントリーは、 name および database を指定する必要があり、オプションの属性を備えた名前付け replication_sets のリスト、およびテーブルまたはシーケンス名のリストを指定できます。

```
publications:

- type: pglogical
  database: example
  name: some_publication_name
  replication_sets:
  - name: default
    replicate_insert: true
    replicate_update: true
    replicate_delete: true
    replicate_truncate: true
    autoadd_tables: false
    autoadd_sequences: false
    autoadd_existing: true
  - name: custom_replication_set
    tables:
    - name: sometable
    - name: "some-schema".othertable
      columns: [a, b, c]
      row_filter: a > 42
      synchronize_data: true
    sequences:
    - name: someseq
      synchronize_data: true
    - name: "some-schema".otherseq
```

各レプリケーションセットは、replicate_insert 、autoadd_existing などのオプションの属性を指定できます。指定した場合、それらは pglogical.create_replication_set() 呼び出しに名前付けパラメーターとして含まれます。それ以外の場合、それらは省略され、代わりに pglogical のデフォルトでレプリケーションセットが作成されます。

pglogical3 の autoadd_* パラメーターを使用してレプリケーションセットに属するリレーションのリストを操作することとは別に、テーブルまたはシーケンスのリストを明示的に指定することもできます。各リレーションの名前はスキーマ修飾されており、非修飾名は public にあるものと想定されており、エントリには、上に示しているように、row_filter テーブルのみ、synchronize_data などのオプションの属性が含まれる場合があります。

サブスクリプション

subscriptions のエントリーは、 name および database を指定し、サブスクライブするパブリケーションを定義する必要があり、サブスクリプションの他のオプショナル属性を指定する場合があります。

```
subscriptions:

- type: pglogical
  database: example
  name: some_subscription_name
  node_name: optional_pglogical_node_name
  publication:
    name: some_publication_name
  # Optional attributes:
  synchronize_structure: true
  synchronize_data: true
  forward_origins: [all]
  strip_origins: false
  apply_delay: 1 second
  writer: heap
  writer_options:
    - magic
    - key=value
    - just-a-string
  # Optional attributes that can be changed for an existing
  # subscription:
  replication_sets:
    - default
    - default_insert_only
    - custom_replication_set
  enabled: true
```

サブスクリプションは publication.name を設定して、サブスクライブするパブリケーションを定義できます。その名前のクラスター全体に複数のパブリケーションがある場合、明確にするインスタンスの名前を指定できます。パブリケーションを名前で参照したい場合は、同じインスタンスに同じ名前で複数のパブリケーションを作成しないでください。

```
- type: pglogical
  …
  publication:
    name: some_publication_name
    instance: kazoo

  # OR

  provider_dsn: "host=… dbname=…"
```

パブリケーションを名前で参照する代わりに、代わりに provider_dsn を明示的に指定できます。この場合、指定された DSN は pglogical.create_subscription() に直接渡され、publication は無視されます。このメカニズムを使用して、TPA クラスターの外部のインスタンスをサブスクライブできます。

上記の例の他の属性はオプショナル。定義されている場合、それらは pglogical.create_subscription()

呼び出しに名前付けパラメーターとして含まれます。それ以外の場合、それらは除外されます。表示される一部の属性は pglogical3 に固有です。

構成の変更

パブリケーションの場合、レプリケーションセットの追加または削除、レプリケーションセットの属性の変更、またはそのメンバーシップそれに含まれるテーブルとシーケンスを変更できます。

`replicate_*` または `autoadd_*` を変更すると、TPA はそれに応じて `pglogical.alter_replication_set()` を呼び出しますが、既存のレプリケーションセットの `autoadd_existing` は変更できず、`autoadd_*` パラメーターはすべて pglogical3 固有であることに注意してください。

レプリケーションセットの `tables` または `sequences` のリストを変更すると、TPA は必要に応じて `pglogical.alter_replication_set_{add,remove}_{table,sequence}()` を呼び出すことによりこれらの変更を調整します。

ただし、レプリケーションセットのメンバーであるリレーションテーブルまたはシーケンスの `synchronize_data` または他の属性を変更した場合、TPA は変更を伝播しませんたとえば、テーブルをドロップし、別の構成で再追加します。

サブスクリプションの場合、`replication_sets` のリストを変更し、サブスクリプション `enabled: false` を有効または無効にすることのみができます。

どちらの場合も、存在するが構成に記載されていないレプリケーションセットは、サブスクライバーで `pglogical.alter_subscription_remove_replication_set()`、またはパブリッシャーで `pglogical.drop_replication_set()` を使用して削除されます。ただし、`default`、`default_insert_only`、および `ddl_sql` という名前のデフォルトのレプリケーションセットはドロップされません。

config.yml を編集する場合は、`tpaexec deploy` を実行する前に `tpaexec provision` を実行してください

PGD との相互作用

注意すれば、同じデータベースで PGD と pglogical を一緒に使用することが可能です。

PGD v3 は内部で pglogical3 を使用し、pglogical ノードが存在しない場合は作成します。データベースごとに pg 論理ノードは 1 つだけであるため、`bdr_database` で pg 論理パブリケーションを構成する場合、インスタンスの `bdr_node_name` はパブリケーションの `name` と同じである必要があります。それ以外の場合、最初にパブリケーション用にノードが作成され、後で `bdr.create_node()` はノード名の競合に関するエラーで失敗します。`bdr_database` の `subscriptions` は、同じ `node_name` も使用する必要があります。

制限事項

- 現在 `pglogical.replication_set_{add,remove}_ddl()` のサポートはありません
- 現在 `pglogical.replication_set_add_all_{tables,sequences}()` のサポートはありません
- 現在、`pglogical.alter_subscription_{interface,writer_options}()` または `pglogical.alter_subscription_{add,remove}_log()` のサポートはありません
- pglogical v1 のサポートは現在テストされていません。

12.6.15 repmgr

TPA は、 failover_manager インスタンス変数が repmgr に設定されているすべての postgres インスタンスに repmgr をインストールします。これはデフォルト設定です。

repmgr 構成ファイルのディレクトリは、デフォルトで/etc/repmgr/<version> になります。ここで、<version> は、このインスタンスにインストールされる postgres のメジャーバージョンですが、インスタンスの repmgr_conf_dir 変数を設定することにより変更できます。構成ファイル自分自身は常に repmgr.conf と呼ばれます。

デフォルトの repmgr 構成は、ロール primary とロール replica で構成されたインスタンス間の自動フェイルオーバーをセットアップします。

repmgr パッケージバージョン

デフォルトでは、TPA は repmgr の最新バージョンをインストールします。

インストールされる repmgr パッケージのバージョンは、 config.yml ファイルの cluster_vars セクションの下の repmgr_package_version: xxx を含めることにより指定できます。

```
cluster_vars:
    …
    repmgr_package_version: 4.0.5-1.pgdg90+1
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

repmgr 構成

次のインスタンス変数を設定できます。

repmgr_priority : sets priority in the config file repmgr_location : sets location in the config file repmgr_reconnect_attempts : sets reconnect_attempts in the config file, default 6 repmgr_reconnect_interval : sets reconnect_interval in the config file, default 10 repmgr_use_slots : sets use_replication_slots in the config file, default 1 repmgr_failover 構成ファイルで failover を設定します、デフォルト automatic

repmgr_conf_settings の追加設定も repmgr 構成ファイルに渡されます。

repmgr サービス構成

repmgr_service_environment 環境変数を repmgr サービスユニットファイルに設定します。環境変数は、repmgr_service_environment 変数の下で、’キー-値’の方法で定義する必要があります。以下の例を参照してください。repmgr_service_environment を使用して、TPA で認識されるかどうかにかかわらず、パラメーターを設定できます。repmgrd.service ファイルに表示されるのとまったく同じように値を引用符で囲む必要があります。

```
cluster_vars:
  repmgr_service_environment:
    LD_PRELOAD: "/usr/edb/pge15/lib/libpq.so.5:/usr/edb/pge15/lib/libpqagent86.so"
    AGENT86_SHARD_FILE: "/var/lib/pgsql/shard.dat"
```

repmgr サービスファイルで

```
root@kaput:~# cat /etc/systemd/system/repmgr.service
[Unit]
Description=Postgres replication manager
After=postgres-monitor.service
Wants=postgres-monitor.service

[Service]
Type=simple
User=postgres
Group=postgres
Environment=AGENT86_SHARD_FILE=/var/lib/pgsql/shard.dat
Environment=LD_PRELOAD=/usr/edb/pge15/lib/libpq.so.5:/usr/edb/pge15/lib/libpqagent86.
↪so
StandardOutput=syslog
ExecStart=/usr/lib/postgresql/17/bin/repmgrd -f /etc/repmgr/17/repmgr.conf --verbose -
↪-daemonize=false
ExecStop=/bin/kill -TERM $MAINPID
ExecReload=/bin/kill -HUP $MAINPID
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

PGD インスタンスの## repmgr

PGD インスタンスでは、repmgr_failover はデフォルトで manual に設定されます。

tpaexec upgrade を使用した repmgr のマイナー更新

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade <cluster_dir> \
    -e repmgr_package_version="<desired version>"\
    --components=repmgr
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードのために repmgr を選択するには、 tpaexec upgrade コマンドに渡される--components フラグに repmgr または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.16 Configuring EDB Repos 2.0 repositories

TPA は、EDB Repos 2.0 から EDB ソフトウェアをソースします。 EDB Repos 2.0 を使用するには、tpaexec を実行する前に `export EDB_SUBSCRIPTION_TOKEN=xxx` を行う必要があります。サブスクリプショントークンは the web interface から取得できます。

> **注釈**
>
> *tpaexec configure* コマンドを使用して`config.yml` ファイルを作成する場合、*edb_repositories* キーには、選択した構成に必要なリポジトリが自動的に入力されるため、編集する必要はありません。

```
cluster_vars:
  edb_repositories:
    - enterprise
    - postgres_distributed
```

この例では、'enterprise'サブスクリプションリポジトリと、EPAS および PGD 製品へのアクセスを付与する'postgres_distributed'をインストールします。Debian または Ubuntu システムでは、APT リポジトリを使用し、RedHat または SLES システムでは、それぞれ yum または zypper フロントエンドを介して rpm リポジトリを使用します。

12.6.17 Configuring APT repositories

このページでは、Debian および Ubuntu システムで APT パッケージリポジトリを構成する方法について説明します。

`apt_repositories` で名前付けリポジトリを定義し、`apt_repository_list` で名前をリストすることによりどのリポジトリを使用するかを決定できます。

```
cluster_vars:
  apt_repositories:
    Example:
      key_id: XXXXXXXX
      key_url: https://repo.example.com/path/to/XXXXXXXX.asc
      repo: >-
        deb https://repo.example.com/repos/Example/ xxx-Example main

  apt_repository_list:
    - PGDG
    - Example
```

この構成では、GPG キー `key_url` から取得した ID `key_id` を使用して、PGDG リポジトリデフォルトで既に定義されているものと新しい例リポジトリの指定された `repo` 行複数可を使用して`/etc/apt/sources.list.d` の下に新しいエントリーをインストールします。

追加のリポジトリを構成するとき、PGDG パッケージをインストールする場合は、apt_repository_list に PGDG を含めることを忘れないでください。

リポジトリをインストールしないように apt_repository_list: [] を設定できます。

12.6.18 Configuring YUM repositories

このページでは、RedHat システムで YUM パッケージリポジトリを構成する方法について説明します。

yum_repositories で名前付けリポジトリを定義し、yum_repository_list で名前をリストすることによりどのリポジトリを使用するかを決定できます。

```
cluster_vars:
  yum_repositories:
    Example:
      rpm_url: >-
        https://repo.example.com/repos/Example/example-repo.rpm

    Other:
      description: "Optional repository description"
      baseurl: https://other.example.com/repos/Other/$basearch
      gpgkey:
        https://other.example.com/repos/Other/gpg.XXXXXXXXXXXXXXX.key

  yum_repository_list:
    - EPEL
    - PGDG
    - Example
    - Other
```

この例では、YUM リポジトリを定義する 2 つの方法を示しています。

リポジトリに「repo RPM」つまり、必要な/etc/yum.repos.d/*.repo ファイルと、リポジトリから署名されたパッケージを検証するために必要な GPG キーを慣例的にインストールするパッケージがある場合、それをポイントするだけで済みます。

それ以外の場合は、説明、baseurl 、および gpgkey URL を指定できます。TPA は、この情報に基づいて/etc/yum.repos.d/Other.repo ファイルを作成します。

EPEL および PGDG リポジトリはデフォルトで定義されています。正常な操作には EPEL リポジトリが必要であるため、常に EPEL を yum_repository_list に含める必要があります。PGDG パッケージをインストールする場合は、PGDG も含める必要があります。

リポジトリをインストールしないように yum_repository_list: [] を設定できますが、EPEL パッケージの代替ソースがないと壊れます。

リポジトリーアクセスを構成する特別な手順を実行する必要がある場合は、*TPA hooks* を使用して.repo ファイルを自分自身で作成できます。

```
- name: Define Example repository
  copy:
    dest: /etc/yum.repos.d/example.repo
    owner: root
    group: root
    mode: "0644"
    content: |
      [example]
      name=Example repo
      baseurl=https://repo.example.com/repos/Example/
      enabled=1
      gpgkey=https://repo.example.com/repokey.asc
      gpgcheck=1
```

この場合、yum_repository_list にリポジトリをリストする必要はありません。

12.6.19 Creating and using a local repository

クラスターディレクトリ内にローカルリポジトリを作成すると、TPA はリポジトリ内のパッケージをクラスターインスタンスで利用できるようにします。これは、追加のパッケージをクラスターにシップする簡単な方法です。

オプションで、このリポジトリ**のみを使用するようにインスタンスを構成するように** *TPA* **に指示することもできます。つまり、他のすべてを無効にします。この場合、***rsync***、***Python*** などの基本的な依存関係から始めて、展開中に必要な**すべての*パッケージを提供する必要があります。

ローカルリポジトリを手動で作成するか、TPA にローカルリポジトリを作成してもらうことができます。両方の手順を以下に示します。

> **注釈**
>
> 特定の指示は *Managing clusters in a disconnected or air-gapped environment* 環境で使用できます。

TPA を使用したローカルリポジトリの作成

TPA には、このようなローカルリポジトリの作成に役立つツールが含まれています。具体的には、--enable-local-repo スイッチを tpaexec configure とともに使用して、ローカルリポジトリとして使用する空のディレクトリ構造を作成できます。tpaexec download-packages は、その構造に必要なパッケージを設定します。

ディレクトリ構造の作成

ローカルリポジトリでクラスターを構成するには、次のコマンドを実行します。

```
tpaexec configure --enable-local-repo …
```

これにより、クラスター構成が生成され、local-repo ディレクトリと OS 固有のサブディレクトリが作成されます。ローカルリポジトリのレイアウト については、以下を参照してください。

リポジトリにデータを設定し、メタデータを生成する

tpaexec download-packages を実行して、クラスターに必要なすべてのパッケージをローカルリポジトリにダウンロードします。結果のリポジトリには、すべてのパッケージの完全な依存関係ツリーが含まれるため、このリポジトリからクラスター全体をインストールできます。リポジトリのメタデータも自動的に作成され、すぐに使用できます。

ローカルリポジトリの手動作成

ローカルリポジトリのレイアウト

ローカルリポジトリを手動で作成するには、最初に適切なディレクトリ構造を作成する必要があります。--enable-local-repo を使用する場合、TPA は、クラスター用に選択した OS に基づいて、local-repo ディレクトリとその内に OS 固有のサブディレクトリたとえば local-repo/Debian/12 を作成します。この構造は、手動で作成したリポジトリにも使用することをお勧めします。

たとえば、RedHat 8 を実行しているクラスターは次のレイアウトになる場合があります。

```
local-repo/
`-- RedHat
    |-- 8.5 -> 8
    `-- 8
        `-- repodata
```

インスタンスごとに、TPA は local-repo の次のサブディレクトリを順番に検索し、最初に見つけたものを使用します。

- <distribution>/<version> 、例 RedHat/8.5
- <distribution>/<major version> 、例 RedHat/8
- <distribution>/<release name> 、例 Ubuntu/focal
- <distribution> 、例 Debian
- local-repo ディレクトリ自分自身。

これらのディレクトリが存在しない場合、もちろん、TPA はターゲットインスタンスにローカルリポジトリをセットアップしようとしません。

リポジトリの設定とメタデータの生成

tpaexec deploy を実行する前に、以下で詳しく説明する手順を完了する必要があります。

リポジトリにデータを設定するには、含めるパッケージを適切なディレクトリにコピーします。次に、以下で詳しく説明するように、システムに適切なツールを使用してメタデータを生成します。

> **注釈**
>
> 制御ノードつまり tpaexec を実行するマシンでメタデータを生成する必要があります。 TPA は、メタデータとパッケージをターゲットインスタンスにコピーします。

> **注釈**
>
> インスタンスが使用するサブディレクトリにメタデータを生成する必要があります。つまり、パッケージを `local-repo/Debian/12` にコピーする場合、*local-repo/Debian* ではなくそのディレクトリにメタデータを作成する必要があります。

Debian / Ubuntu リポジトリのメタデータ

Debian ベースのディストリビューションの場合、dpkg-dev パッケージをインストールします。

```
sudo apt-get update && sudo apt-get install -y dpkg-dev
```

これで、 dpkg-scanpackages を使用してメタデータを生成できます。

```
cd local-repo/Debian/bookworm

#  download/copy .deb package files

dpkg-scanpackages . | gzip > Packages.gz
```

RedHat / SLES リポジトリのメタデータ

最初に、createrepo パッケージをインストールします。

```
sudo yum install -y createrepo
```

これで、 createrepo を使用してメタデータを生成できます。

```
cd local-repo/RedHat/8

#  download/copy .rpm package files

createrepo .
```

RHEL 8+リポジトリでは、次の追加手順を実行する必要があります。

```
repo2module --module-name tpa --module-stream local --module-version 1 --module-
↪context f32 . modules.yaml
modifyrepo_c --mdtype=modules modules.yaml repodata/
dnf clean all
```

TPA がローカルリポジトリを使用する方法

リポジトリのコピー

TPA は rsync を使用して、生成されたメタデータを含むリポジトリーディレクトリの内容をターゲットインスタンス上のディレクトリにコピーします。

rsync がインスタンスでまだ利用できない場合、TPA はそれをインストールできますつまり `apt-get install rsync` または `yum install rsync` 。ただし、`use_local_repo_only` を設定している場合、rsync パッケージはローカルリポジトリに含まれる必要があります。必要に応じて、TPA は scp を使用して rsync パッケージのみをコピーし、残りをコピーする前にインストールします。

リポジトリー構成

ローカルリポジトリのコンテンツをターゲットインスタンスにコピーした後、TPA は宛先ディレクトリをローカルつまり、URL ベースではなくパスベースのリポジトリとして構成します。

たとえば、リポジトリーディレクトリに `example.deb` を提供する場合、他のリポジトリーのパッケージと同様に、`apt-get install example` を実行するだけでインストールできます。

パッケージのインストール

TPA は、提供されたコンテンツでリポジトリを構成しますが、同じパッケージが別のリポジトリから利用可能な場合、どれをインストールするかはパッケージマネージャーの判断にあります特定のバージョンを指定しない限り、通常は最新です。

ただし、`use_local_repo_only: yes` を設定すると、TPA は他のすべてのパッケージリポジトリを無効にするため、インスタンスは `local-repo` で提供したパッケージのみを使用できます。

12.6.20 Installing from source

config.yml で `install_from_source` を設定することにより、Git リポジトリからビルドおよびインストールする拡張機能のリストを定義できます。

```
cluster_vars:
  install_from_source:
    - name: ext
      git_repository_url: https://repo.example.com/ext.git
      git_repository_ref: dev/example

    - name: otherext
      git_repository_url: ssh://repo.example.com/otherext.git
      git_repository_ref: master
      source_directory: /opt/postgres/src/otherext
      build_directory: /opt/postgres/build/otherext
      build_commands:
        - "make -f /opt/postgres/src/otherext/Makefile install"
      build_environment:
        VAR: value
```

TPA はリストされた順序で拡張機能を 1 つずつビルドしてインストールするため、正しい順序でメンションすることにより、別の拡張機能 pglogical や BDR などに依存する拡張機能をビルドできます。

ビルドするには、各エントリーで `name` 、`git_repository_url` 、および `git_repository_ref` デフォルト `master` を指定する必要があります。 *Git credentials* を使用できます

Git リポジトリに対して認証するため。また、上記のように `source_directory` 、`build_directory` 、`build_environment` 、および `build_commands` を設定します。

`tpaexec deploy … --skip-tags build-clean` を実行して、展開を繰り返すときにビルドディレクトリを再利用します。それ以外の場合、古いビルドディレクトリはビルドを開始する前に空にされます。**ローカルソースディレクトリ** を構成することもできます。

開発ビルドを高速化します。

ソースビルドを実行するたびに、Postgres が再起動されます。

ビルドの依存関係

ソースからビルドする場合、TPA は、基本的な Postgres ビルドの依存関係がインストールされることを保証します。追加パッケージが必要な場合は、*Installing packages* に記載してください。例

```
cluster_vars:
  packages:
    common:
    - golang-1.16
```

12.6.21 Git credentials

このページでは、認証が必要な Git リポジトリのクローンを作成する方法について説明します。

これは、 `postgres_git_url` を *Postgres source installation* または **ソースからのビルドとインストール** に変更して拡張機能をコンパイルおよびインストールする場合に必要になる場合があります。

資格情報をターゲットインスタンスのディスクに書き込まずに認証するには、2 つのオプションがあります。

- `ssh://` リポジトリの場合、SSH キーをローカル ssh-agent に追加できます。`--install-from-source` config.yml の `forward_ssh_agent: yes` を使用する場合、エージェント転送はデフォルトで有効になります。
- `https://` リポジトリの場合、`tpaexec deploy` を実行する前に環境で `export TPA_GIT_CREDENTIALS=username:token` を実行できます。

> **注釈**
>
> この環境ではホストから ssh で Docker コンテナにアクセスできないため、macOS で Docker に展開する場合、*https://* リポジトリ URL のみを使用する必要があります。

SSH キー認証

SSH リポジトリのクローンを作成しており、SSH キーペア `id_example` および `id_example.pub` がある場合、SSH エージェント転送を使用してターゲットインスタンスで認証します。

- **``ssh-agent`` をローカルで実行する必要があります**。デスクトップ環境がこれをまだ設定していない場合は、ほとんどの場合-`pgrep ssh-agent` が実行されているかを確認します `ssh-agent bash` を実行してエージェントを有効にした新しいシェルを一時的に起動し、そのシェルから `tpaexec deploy` を実行します。
- `ssh-add /path/to/id_example` プライベートキーファイルを使用して、必要なキーをエージェントに追加します**
- `tpaexec provision` の前に config.yml のトップレベルで `forward_ssh_agent: yes` を設定することにより、SSH エージェント転送を有効にします**。`--install-from-source` を使用する場合、これはデフォルトで行われます。

展開中に、エージェントに追加したキーは、転送されたエージェント接続を介してリモートサーバーへの認証に利用できるようになります。

SSH エージェント転送は注意して使用し、できればこの目的のために専用に生成された使い捨てキーペアを使用します。ターゲットサーバー上のエージェントの Unix ドメインソケットにアクセスする権限を持つユーザーは、他のサーバーへの認証中にエージェントを組み込んでなりすましができます。

HTTPS のユーザー名/パスワード認証

ユーザー名と認証トークンまたはパスワードを使用して HTTPS リポジトリのクローンを作成する場合は、`tpaexec deploy` の前に環境で `export TPA_GIT_CREDENTIALS=username:token` だけを作成します。デプロイメント中に、これらの資格情報は `git clone` または `git pull` タスクのみで利用できるようになります。これらはターゲットインスタンスのディスクに書き込まれません。

12.6.22 Environment

`target_environment` を設定して、展開中に TPA がターゲットインスタンスに設定する必要がある環境変数を指定できますたとえば、以下に示すように HTTPS プロキシを指定します。

```
cluster_vars:
    target_environment:
        https_proxy: https://proxy.example:8080
```

TPA は、これらの設定が、展開中、およびその後のクラスター管理コマンドの実行中に、必要な他の設定とともに環境に存在することを確認します。

これらの環境設定は永続的なものではありませんが、代わりに *The postgres Unix user* を使用して postgres ユーザーの環境変数を設定できます。

12.6.23 Python environment

TPA は、ターゲットインスタンスの *Distribution support* に基づいて、どの Python インタープリターを使用するかを決定します。可能な場合は常に Python 3 を使用し、やむを得ない場合にのみ Python 2 にフォールバックします。

`tpaexec configure` コマンドは、ディストリビューションに応じて `preferred_python_version` を設定します。

Distribution	Python 2	Python 3
Debian 12/bookworm	Yes	Yes (3.11)
Debian 11/bullseye	Yes	Yes (3.9)
Debian 10/buster	Yes	Yes (3.7)
Ubuntu 24.04/noble	No	Yes (3.12)
Ubuntu 22.04/jammy	No	Yes (3.10)
Ubuntu 20.04/focal	No	Yes (3.8)
RHEL 10.x	No	Yes (3.12)
RHEL 9.x	No	Yes (3.9)
RHEL 8.x	No	Yes (3.6)
RHEL 7.x	Yes	No (3.6)
SLES 15	No	Yes (3.6)

Ubuntu 20.04、22.04、24.04、RHEL 8.x、9.x、10.x、および SLES 15 は、Python 3 でのみ使用できます。

RHEL 7.x には Python 3.6 が同梱されていますが、システム Python 3 の librpm バインディングは利用できません。

他のディストリビューションの場合、`python2` または `python3` のどちらを優先するかを決定できますが、新しいクラスターのデフォルトは `python3` です。

下位互換性

既存のクラスターとの互換性のため、`preferred_python_version` のデフォルト値は `python2` ですが、`python2` で既に展開されたシステムでも `python3` を明示的に選択できます。

```
cluster_vars:
  preferred_python_version: python3
```

TPA は、Python 3 を使用できないディストリビューションではこの設定を無視します。

インストールされたスクリプトの Python インタープリター

TPA は、クラスターノードに少数のヘルパー Python スクリプトたとえば、`/etc/tpa/postgres-monitor`、展開が Postgres が利用可能になるのを待機するために使用する`/etc/tpa/postgres-monitor` をインストールします。インストールされた各スクリプトのシバン行は、`script_python_interpreter` ファクトからテンプレート化されています。

デフォルトでは、script_python_interpreter は/usr/bin/env {{ python }} に設定されます。これは、スクリプトが実行時にノードの PATH で最初に見つけた python3 バイナリ env の下で実行されることを意味します。これは、TPA の歴史的な動作と一致し、ほとんどのホストでの正しい回答です。TPA 自分自身がパッケージインストールのターゲットとしているシステム Python を選択します。

TPA のインストールされたスクリプトを特定の Python インタープリターで実行する必要がある場合たとえば、ホストに複数の Python バージョンがあって、あいまいさを削除したい、または非システム Python を使用したいため、cluster_vars の script_python_interpreter をに設定します必要なインタープリターの絶対パス

```
cluster_vars:
  script_python_interpreter: /usr/bin/python3
```

または、非システム Python をポイントするには

```
cluster_vars:
  script_python_interpreter: /opt/python3.11/bin/python3
```

値は、シバン行の#! の後に逐語的に使用されるため、カーネルが直接 exec できるパスである必要があります。この変数を設定する場合、スクリプトをサポートするために TPA がインストールする OS パッケージたとえば python3-psycopg2 がそのインタープリターで利用できることを確認します。TPA がインストールするパッケージに影響を与える方法については、次のセクションを参照してください。

インストールされたパッケージの Python バージョン

TPA は、スクリプトが依存する Python モジュール python3-psycopg2 などを提供する複数の OS パッケージ RPM または DEB をインストールします。これらのパッケージ名を構成するために使用される名前プレフィックスは、ディストリビューションで検出された Python RHEL 7 以外の現在サポートされているすべてのディストリビューションでは python3 がデフォルトである python_pkg_prefix ファクトから取得されます。

TPA にデフォルト以外の Python のパッケージをインストールさせたい場合たとえば、システム Python とともに Python 3.11 をインストールし、TPA をデフォルトの python3-psycopg2 ではなく一致する python311-psycopg2 パッケージに依存させたいため cluster_vars に python_pkg_prefix を設定します。

```
cluster_vars:
  python_pkg_prefix: python311
```

TPA は、接頭辞を使用してパッケージ名を構成し、たとえばインストールします。python311-psycopg2、python311-barman など。

重要 python_pkg_prefix は、_TPA がインストールする OS パッケージの名前_のみを制御します。自分自身では、どの Python インタープリター TPA インストールされたスクリプト/etc/tpa/postgres-monitor などで実行されるかは変更されません。複数の Python インタープリターがインストールされているホストで python_pkg_prefix を設定する場合、ほぼ確実に script_python_interpreter 前のセクションを参照してください。一致するインタープリターの絶対パスを設定します。それ以外の場合、TPA のインストールされたスクリプトのデフォルトの/usr/bin/env ベースのシバンは、PATH を介して、python_pkg_prefix

名前付けパッケージがインストールされたものとは別の Python に解決される場合があり、インポートは実行時に失敗します。

12.6.24 Configuring /etc/hosts

デフォルトでは、TPA はクラスター内のすべてのインスタンスの IP アドレスとホスト名を使用してターゲットインスタンスの/etc/hosts に行を追加して、クラスター内の通信にお互いの名前を使用できるようにしますたとえば primary_conninfo Postgres の場合)。

extra_etc_hosts_lines のリストを指定することもできます。

```
instances:

- Name: one
  …
  vars:
    extra_etc_hosts_lines:
    - 192.0.2.1 acid.example.com
    - 192.0.2.2 water.example.com
```

デフォルトエントリがまったく必要ない場合は、代わりにインスタンスの etc_hosts_lines の完全なリストを指定できます。これらの行のみが/etc/hosts に追加されます。

```
instances:

- Name: one
  …
  vars:
    etc_hosts_lines:
    - 192.0.2.1 acid.example.com
    - 192.0.2.2 water.example.com
    - 192.0.2.3 base.example.com
```

/etc/hosts にクラスター内のインスタンスのデフォルトエントリが含まれていない場合、他の方法で名前を解決できることを確認する必要があります。

12.6.25 Filesystem configuration

TPA では、各インスタンスにアタッチされる volumes のリストを定義できます。

このリストは、プロビジョニング中に使用されるプラットフォーム固有の設定と、展開中に使用されるファイルシステムレベルの設定の両方で構成されています。

最初に、tpaexec provision は、情報を使用してボリュームを作成し、インスタンスに接続します該当する場合。詳細については、以下のプラットフォーム固有のセクションを参照してください。次に、ボリュームの単純化されたリストプラットフォーム固有ではない設定のみを含むをインスタンスのホスト変数として書き込みます。最後に、tpaexec deploy は、単純化されたリストに基づいて動作し、必要に応じてファイルシステムをセットアップおよびマウントします。

次に、AWS クラスターからの適度に複雑な例を示します。

```
instances:

- Name: one
  ...
  volumes:
  - device_name: root
    volume_type: gp2
    volume_size: 32
  - device_name: /dev/xvdf
    volume_type: io2
    volume_size: 64
    iops: 5000
    vars:
      volume_for: postgres_data
      encryption: luks
  - device_name: /dev/xvdb
    ephemeral: ephemeral0
    vars:
      mountpoint: /mnt/scratch
```

この例では、EC2 インスタンスは、32GB EBS ルートボリューム、/opt/postgres/data としてマウントされた 64GB io2 ボリューム provisioned-iops EBS ボリューム、およびインスタンスストアによって提供される/tmp/scratch ファイルシステムになります。「エフェメラル」ボリューム。その数とサイズは、インスタンスタイプによって決まります。

詳細は以下の AWS に関するセクションに記載されていますが、 `volume_type` や `volume_size` のような設定はプロビジョニング中に使用され、`volume_for` や `mountpoint` などの `vars` の下の設定は、展開中に使用するためにインベントリに書き込まれます。

default_volumes

ボリュームはインスタンスのプロパティです。プラットフォーム固有の設定が含まれるため、`cluster_vars` では設定できません。

instance_defaults

このメカニズムでは、ボリューム定義に特別な考慮事項を作成します。大規模なクラスター内のボリューム定義は非常に反復的な場合があるため、特にクラスター内のインスタンスを可能な限り互いに近づけて構成することをお勧めしますので、ここに示すように `default_volumes` を指定できます。

```
instance_defaults:
  default_volumes:
  - device_name: root
    volume_type: gp2
    volume_size: 32
  - device_name: /dev/xvdf
```

(次のページに続く)

(前のページからの続き)

```
    volume_size: 100

instances:

- Name: one
  ...

- Name: two
  volumes:
  - device_name: /dev/xvdf
    volume_size: 64
  - device_name: /dev/xvdg
    volume_size: 64
    ...

- Name: three
  volumes:
  - device_name: /dev/xvdf
    volume_type: none

- Name: four
  volumes: []
```

ここで、すべてのインスタンスには、デフォルトで 32GB のルートボリュームと 100GB の追加ボリュームがあります他に何も指定しないインスタンス one の場合と同様に。インスタンス two は同じルートボリュームがありますが、代わりに 64GB になるように/dev/xvdf をオーバーライドし、さらに別の 64GB ボリュームがあります。インスタンス three には同じルートボリュームがありますが、デフォルトの/dev/xvdf に volume_type: none を設定するため、追加ボリュームはありません。インスタンス four にはボリュームがまったくありません。

インスタンスは default_volumes で指定されたもので始まり、その volumes エントリは、同じ device_name でデフォルトエントリをオーバーライドしたり、volume_type を none に設定してボリュームを削除したり、別の名前で新しいボリュームを追加したり、デフォルトを完全に拒否したりできます。

2 つのリストをマージするこの動作は、default_volumes に固有です。instance_defaults と instances の両方で他のリストを設定すると、後者は前者を完全にオーバーライドします。

プラットフォーム AWS

AWS EC2 インスタンスでは、EBS ボリュームをアタッチできます。

```
instances:

- Name: one
  ...
  volumes:
```

(次のページに続く)

(前のページからの続き)

```
  - device_name: root
    volume_type: gp2
    volume_size: 32
    encrypted: yes
    …
  - device_name: /dev/xvdf
    volume_type: io1
    volume_size: 32
    iops: 10000
    delete_on_termination: false
    …
  - device_name: /dev/xvdg
    ephemeral: ephemeral0
    …
```

TPA は、インスタンスタイプに基づいて、root の device_name を/dev/sda または/dev/xvda に変換するため、どれを使用するかを覚えたり変更したりする必要はありません。

volume_type は、EBS ボリュームタイプ、たとえば gp2 「汎用」EBS ボリュームの場合、プロビジョニングされた IOPS ボリュームの io1 iops: 5000 も設定する必要があります。などを指定します。

volume_size は、ボリュームのサイズをギガバイト単位で指定します。

encrypted: yes を設定して、保存時の EBS 暗号化を有効にします。(これは AWS 機能であり、新しく生成された TPA 構成でデフォルトで有効になり、以下で説明する *LUKS* **暗号化** とは異なります。

delete_on_termination を false に設定して、アタッチされたインスタンスが終了したときにボリュームが破壊されないようにします。これがデフォルトの動作です。

物理的に接続された instance store volume 、以前はエフェメラルボリュームと呼ばれていたを使用するように ephemeral: ephemeralN を設定します。使用可能なインスタンスストアボリュームの数、タイプ、およびサイズは、インスタンスタイプによって異なります。すべてのインスタンスにインスタンスストアボリュームがあるわけではありません。インスタンスストアボリュームはテストまたは一時データにのみ使用し、EBS ボリュームは重要なデータに使用します。

EBS ボリュームの場合、既存のスナップショットからボリュームを接続するように snapshot: snap-xxxxxxxx を設定することもできます。十分なデータが S3 から読み込まれローカルにキャッシュされるまで、スナップショットから復元されたボリュームは非常に遅くなる場合があります。(特に、スナップショットから PGDATA を使用して新しいインスタンスをスピンアップできますが、完全なロードを処理する準備が整うまでに数時間かかることが予想されます。

ボリュームに attach_existing: yes を設定し、名前/タイプ/サイズ/iops と一致する既存の未接続の EBS ボリュームが存在する場合、インスタンスの起動時に新しいボリュームは作成されませんが、代わりに既存のボリュームがインスタンスにアタッチされます初めて始まります。再接続された EBS ボリュームは、スナップショットから作成されたボリュームのパフォーマンス制限の影響を受けません。

プラットフォームベア

TPA は、事前にプロビジョニングされた bare インスタンスにどのボリュームを接続できるかを制御できませんが、適切な device_name で volumes を定義すると、必要に応じてデバイスの mkfs および mount を処理します。

Platform Docker

Docker コンテナにはボリュームを接続できますが、それらはバインドマウントされたディレクトリであり、通常のブロックデバイスではありません。個別に初期化またはマウントする必要はありません。そのため、構成はかなり異なって見えます。

```
instances:

- Name: one
  platform: docker
  …
  volumes:
  - /host/path/to/dir:/tmp/container/path:ro
  - named_volume:/mnt/somevol:rw
```

これらのボリューム仕様は、docker run -v への引数として認識される場合があります。

ボリュームはコンテナの作成時に接続され、展開中にそれ以上のアクションはありません。

RAID アレイ

AWS EC2 インスタンスでは、RAID 0 のみが Amazon によって推奨されています。同様のコマンドで RAID ボリュームを作成できます。

```
sudo mdadm --create --verbose /dev/md0 --level=0 --name=MY_RAID --raid-devices=number_
↪of_volumes device_name1 device_name2
```

この例では、 /dev/md0 という名前のブロックデバイスを接続します。導入中の volume_for または mountpoint の処理は、他のボリュームの場合と同じように発生します。 TPA は、mkfs および mount を処理します。

```
- Name: one
  …
  volumes:
  - device_name: /dev/md0
    vars:
      volume_for: postgres_data
```

LUKS 暗号化

TPA は、LUKS で暗号化されたデバイスをセットアップできます。

```
instances:

- Name: one
  …
  volumes:
  - device_name: /dev/xyz
    vars:
      encryption: luks
      luks_volume: mappedname
      volume_for: …
```

encryption: luks セットがまだ初期化されていない場合、TPA は他のデバイスと同様にファイルシステムの作成を処理する前に、cryptsetup を使用して最初に luksFormat 、次に luksOpen を使用して/dev/mapper/mappedname の下にマップします。

データ損失の可能性を回避するため、TPA は、有効なファイルシステムが既に含まれているデバイスでの LUKS 暗号化のセットアップを拒否します。

LUKS で暗号化された volume_for: postgres_data を作成する場合、TPA はブート時に Postgres が自動的に起動しないように構成します。tpaexec start-postgres clustername を使用してボリュームをマウントし、Postgres を起動できます。および stop-postgres を使用して Postgres を停止し、ボリュームのマップを解除します。

LUKS パスフレーズはローカルで生成され、ボールトに保存されます。

ファイルシステムの作成とマウント

device に有効なファイルシステムが含まれていない場合、 mkfs で初期化されます。

```
instances:

- Name: one
  …
  volumes:
  - device_name: /dev/xyz
    vars:
      volume_for: …
      fstype: ext4
      fsopts:
        - -cc
        - -m 2
      mountopts: defaults,relatime,nosuid
      readahead: 65536
      owner: root
```

(次のページに続く)

(前のページからの続き)

```
    group: root
    mode: "0755"
```

fstype デフォルト ext4、mkfs デフォルト none に渡される fsopts 、および mount に渡されて fstab に書き込まれる mountopts を指定できます。以下を参照してください。

TPA は、デバイスの先読みをデフォルトで 16MB に設定し、リブート後も値を持続させますが、上記のようにボリュームに別の値を指定できます。

ボリュームがマウントされている場所を決定するには 2 つの方法があります。mountpoint を明示的に指定するか、 volume_for を postgres_data 、postgres_wal 、postgres_tablespace または barman_data に設定できます。TPA は設定をシステムの適切なマウントポイントに変換します。

mountpoint が決定されると、 device は指定された mountopts デフォルト defaults,noatime とともにそこにマウントされます。/etc/fstab のファイルシステムのエントリーも作成されます。

オプションでボリュームに owner 、group 、または mode を指定できます。これらの属性は mountpoint に設定されます。デプロイメントのこの非常に初期段階では、postgres ユーザーの存在を当てにできないことに注意してください。いずれの場合も、TPA は Postgres が必要とするディレクトリに適切な所有権と権限があることを別途に保証するため、自分で行う必要はありません。

12.6.26 Uploading artifacts

artifacts を定義して、ファイルをターゲットインスタンスに作成またはコピーできます。

```
cluster_vars:
  artifacts:
  - type: path
    path: /some/target/path
    state: directory
    owner: root
    group: root
    mode: "0755"
  - type: file
    src: /host/path/to/file
    dest: /target/path/to/file
    owner: root
    group: root
    mode: "0644"
  - type: archive
    src: example.tar.gz
    dest: /some/target/path
  - type: directory
    src: /host/path/a/
    dest: /target/path/b/
```

次のタイプがサポートされています。

- path を使用して、ファイルとディレクトリの所有権またはモードを作成または削除、および変更します内部で使用する Ansible の file モジュールと同じパラメーターを取得します
- file を使用してコントローラからファイルをコピーし、所有権とモードを設定します copy を使用
- archive を使用して、アーカイブから指定した場所にファイルを抽出します unarchive を使用
- directory を使用して、コントローラからターゲットインスタンスにディレクトリを r 同期します synchronize を使用

この例では、上記のアーティファクトタイプごとに 1 つのエントリーを示していますが、これらのパラメーター、または対応する Ansible モジュールが受け入れるその他のパラメーターを使用できます。

ファイルとディレクトリのターゲットインスタンスへのコピーは、十分な一般的なニーズであるため、この機能は *TPA hooks* を書き込む代わりに使用できる便利なショートカットを提供します。

12.6.27 ssh_key_file

デフォルトでは、tpaexec provision は ssh-keygen を使用して、クラスターの新しい SSH キーペアを生成しますクラスターディレクトリ内の id_cluster_name および id_cluster_name.pub という名前のファイルに。

代わりに既存のキーを使用する場合は、config.yml のトップレベルで ssh_key_file を SSH プライベートキーファイルの場所に設定できます。対応する公開キーは、同じ場所にある .pub の拡張子で利用できる必要があります。

```
ssh_key_file: ~/.ssh/id_rsa
```

このファイルがまだ存在しない場合、プロビジョニング中に ssh-keygen によって作成されます。

enabled_keys 管理のスキップ

デプロイ中に、TPA は、クラスターの公開キーを、各インスタンスの管理ユーザーの authorized_keys ファイルに追加します。 Oracle Cloud Infrastructure などの一部の環境では、このファイルは読み取り専用または外部管理される場合があります。

この手順をスキップするには、 ssh-cluster-key-config タスクセレクターを使用します。

```
tpaexec deploy <clustername> --excluded_tasks=ssh-cluster-key-config
```

または config.yml

```
cluster_vars:
  excluded_tasks:
  - ssh-cluster-key-config
```

このタスクを除外する場合、tpaexec deploy を実行する前に、インスタンスへの SSH アクセスが他の手段で既に構成されていることを確認する必要があります。

12.6.28 Managing SSH host keys

TPA は、クラスターのプロビジョニング中に SSH ホストキーのセットを生成します。これらのキーは、hostkeys サブディレクトリの下の cluster ディレクトリに保存されます。これらのホストキーは、AWS EC2 インスタンスと Docker コンテナの/etc/ssh に自動的にインストールされます。

デフォルトでは、これらのホストキーは *bare(-metal servers)* にインストールされませんが、manage_ssh_hostkeys を設定して有効にすることができます。

```
instances:

- Name: one
  …
  platform: bare
  vars:
    manage_ssh_hostkeys: yes
```

bare(-metal servers) のドキュメントの説明に従って、正しいエントリを使用してクラスターディレクトリに known_hosts を最初にセットアップする必要があります。TPA は、展開中にホストキーを置き換えます。

manage_ssh_hostkeys 設定は、ベアインスタンスの場合にのみ意味があります。生成されたホストキーは、他のすべてのインスタンスにインストールされます。

known_hosts

TPA は、すべてのホストとそのパブリックホストキーのエントリをクラスター内のすべてのインスタンスのグローバル ssh_known_hosts ファイルに追加するため、manage_ssh_hostkeys 設定があるかどうかに関係なく、ホストキー検証プロンプトなしで相互に ssh できます。

12.6.29 Postgres source installation

postgres_installation_method を src に設定している場合、TPA はソースから Postgres をコンパイルおよびインストールします。この機能は、開発とテストで使用することを目的とており、同一構成のクラスター内でパッケージビルドとソースビルドを切り替えることができます。

ここでも、デフォルトを変更する必要はありません。これにより、デバッグが有効になった作業用クラスターが得られます。

Git リポジトリ

デフォルト設定では、postgres_version に対応する REL_xx_STABLE ブランチを使用して、コミュニティ Git リポジトリから Postgres をビルドおよびインストールします。次のように、別のリポジトリまたはブランチ有効な git リファレンスを指定できます。

```
cluster_vars:
  postgres_git_url: git://git.postgresql.org/git/postgresql.git
  postgres_git_ref: REL_12_STABLE
```

デフォルトの git.postgresql.org リポジトリは認証を必要としませんが、必要に応じて、 *Git credentials* を使用できます

他のリポジトリに対して認証します。

リポジトリは postgres_src_dir デフォルト/opt/postgres/src/postgres にクローン作成されるか、ディレクトリが既に存在する場合たとえば、再展開する場合は git pull で更新されます。

ビルドのカスタマイズ

デフォルトでは、TPA はデバッグを有効にし、postgres_build_dir デフォルト/opt/postgres/build/postgres の賢明なデフォルトで Postgres を構成およびビルドします。さまざまな設定を変更して、ビルドをカスタマイズできます。

```
cluster_vars:
  postgres_extra_configure_env:
    CFLAGS: "-O3"
  postgres_extra_configure_opts:
    - --with-llvm
    - --disable-tap-tests
```

これは、 postgres_extra_configure_opts のオプションと環境で定義された postgres_extra_configure_env の設定を使用して./configure を実行します。一部のオプションはデフォルトで指定されます--with-debug ですが、対応する--disable-xxx または--without-xxx オプションで無効にできます。 --without-openssl のビルドはサポートされていません。

必要に応じて、次のデフォルトのビルドコマンドを変更することもできます。

```
cluster_vars:
  postgres_make_command: "make -s"
  postgres_build_targets:
    - "all"
    - "-C contrib all"
  postgres_install_targets:
    - "install"
    - "-C contrib install"
```

tpaexec deploy … --skip-tags build-clean を実行して、展開を繰り返すときにビルドディレクトリを再利用します。それ以外の場合、古いビルドディレクトリはビルドを開始する前に空にされます。**ローカルソースディレクトリ** を構成することもできます。

開発ビルドを高速化します。

ソースビルドを実行するたびに、Postgres が再起動されます。

追加コンポーネント

パッケージから Postgres をインストールする場合でも、ソースから拡張機能をコンパイルおよびインストールできます。ソースからのビルドとインストール の構成方法に関する別のページがあります。

ただし、拡張機能パッケージは通常、インストールされる Postgres パッケージに依存するため、ソースから Postgres をインストールする場合、ソースから拡張機能もインストールする必要があります。

パッケージのインストール

Installing Postgres-related packages に関する別のページがあります

`postgres_installation_method`: `pkg` デフォルト

12.6.30 Installing packages

TPA は、展開の早い段階で非 Postgres 関連パッケージのバッチをインストールし、次にすべての Postgres 関連パッケージを一緒にインストールし、次にオプションコンポーネントのパッケージを個別にインストールします。このページは、Postgres パッケージに依存していない sysstat や strace のようなパッケージのインストールに関するものです。

config.yml の `cluster_vars` または特定のインスタンスの `vars` の下の `packages` にエントリを追加できます。

```
cluster_vars:
  packages:
    common:
      - pkg1
      - pkg2
    Debian:
      - debpkg1
    RedHat:
      - rhpkg1
      - rhpkg2
    Ubuntu:
      - ubpkg1
    SLES:
      - slespkg1
```

上記の例では、TPA は独自の `default_packages` のリストとすべてのインスタンスの `packages.common` の下にリストされているパッケージをインストールし、インスタンスが実行されているディストリビューションに基づいて、残りのディストリビューション固有パッケージをインストールします。これらのパッケージのいずれかが使用できない場合、展開は失敗します。

Postgres に依存するパッケージはリストしないでください。*Installing Postgres-related packages* を使用

代わりに。

オプショナルパッケージ

インストールする optional_packages のリストを指定できます。これらは利用可能な場合はインストールされ、それ以外の場合は無視されます。他の設定と同様に、common エントリはすべてのインスタンスに適用されますが、他のリストは、関連するディストリビューションを実行しているインスタンスにのみ適用されます。

```
optional_packages:
  common:
    - pkg1
    - pkg2
  Debian:
    - debpkg4
```

パッケージの削除

インストールされている場合に削除する unwanted_packages のリストを指定できます。

```
unwanted_packages:
  common:
    - badpkg1
  Ubuntu:
    - badpkg2
```

12.6.31 Running initdb

TPA は、最初に postgres_data_dir が存在しない場合は作成し、正しい所有権、権限、および SELinux コンテキストがあることを確認します。次に、ディレクトリに VERSION ファイルがまだ含まれていない限り、 initdb を実行して postgres_data_dir を初期化します。

pre-initdb hook を使用できます

postgres_data_dir が作成され、initdb が実行される前にタスクを実行します。フックが postgres_data_dir を初期化すると、TPA は VERSION ファイルを見つけ、 initdb 自分自身を実行する必要がないことを認識します。

オプションで postgres_initdb_opts を initdb に渡すオプションのリストに設定できます。

```
cluster_vars:
  postgres_locale: de_DE.UTF-8
  postgres_initdb_opts:
  - --data-checksums
```

常に--data-checksums オプションデフォルトで含まれるを含めることをお勧めします。

TPA は、 initdb の実行時に、環境に TZ=UTC を設定し、LC_ALL を指定した postgres_locale に設定します。

別の構成ディレクトリ

デフォルトでは、postgres_conf_dir は postgres_data_dir と等しく、Postgres 構成ファイル postgresql.conf、pg_ident.conf、pg_hba.conf、および conf.d のインクルードファイルはデータディレクトリ内に作成されます。postgres_conf_dir を変更すると、TPA は initdb を実行した後、生成された構成ファイルを新しい場所に移動します。

12.6.32 Installing Postgres-related packages

TPA は、展開の早い段階で非 Postgres 関連パッケージのバッチをインストールし、次にすべての Postgres 関連パッケージを一緒にインストールし、次にオプションコンポーネントのパッケージを個別にインストールします。このページは、Postgres 自分自身に依存する pglogical のようなパッケージのインストールに関するものです。

PostgresPostgis に依存する追加パッケージをインストールするには、config.yml の cluster_vars の extra_postgres_packages または特定のインスタンスの vars の下にリストします。

```
cluster_vars:
  extra_postgres_packages:
    common:
      - postgres-pkg1
      - postgres-pkg2
    Debian:
      - postgres-deb-pkg1
    RedHat:
      - postgres11-rhpkg1
      - postgres11-rhpkg2
    Ubuntu:
      - ubpkg1
    SLES:
      - slespkg1
```

packages.common の下にリストされているパッケージは、Postgres パッケージのデフォルトリスト、および指定したディストリビューション固有パッケージとともに、すべてのインスタンスにインストールされます。

Postgres source installation に関する別のページがあります。

12.6.33 PG Backup API

config.yml で enable_pg_backup_api: true を設定するか、configure 中に--enable-pg-backup-api コマンドラインオプションを使用すると、barman ロールを持つインスタンスは pg-backup-api をインストールし、クライアント証明書認証用に Apache プロキシを設定します。この Apache プロキシは、クラスター用に生成された SSL CA を使用して、サーバーとクライアントの証明書を生成します。

```
cluster_vars:
  enable_pg_backup_api: true
```

PG Backup API パッケージバージョン

デフォルトでは、TPA は pg-backup-api の利用可能な最新バージョンをインストールします。

config.yml ファイルの cluster_vars セクションの下の pg_backup_api_package_version: xxx を含めることにより、インストールされる pg-backup-api パッケージのバージョンを指定できます。

```
cluster_vars:
    …
    pg_backup_api_package_version: 2.0.0-1.el8
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

インストールオプション

pg-backup-api は、デフォルトでパッケージを介してインストールされますが、git ブランチまたはローカルディレクトリからインストールすることもできます。詳細については、*Building from source* および **ソースからのビルドとインストール** を参照してください。

セットアップの検証

pg-backup-api を実行している barman ノードで pg-backup-api status を実行します。「OK」が返された場合、pg-backup-api サービスが実行されています。

プロキシが動作していることをテストするには、次のコマンドを実行します。

```
curl --cert /etc/tpa/pg-backup-api/pg-backup-user.crt \
     --key /etc/tpa/pg-backup-api/pg-backup-user.key \
     -X GET https://{hostname}/diagnose
```

機能している場合は、大規模な json 出力を取得します。これを barman diagnose の出力と比較できます。それらは正確に一致するはずです。

SSL 証明書

ルート証明書は、デフォルトで/etc/tpa/pg-backup-api/ にコピーされます。

クライアント証明書とキー pg-backup-user.crt および pg-backup-user.key は、テストのために生成されます tpaexec test を介してまたは barman ホストからのコマンドライン。*tpaexec* **テストを実行する** を参照してください。

Apache プロキシサーバーの証明書とキー pg-backup-api.crt および pg-backup-api.key も生成されます

API を照会する必要がある各サービスは、独自のクライアント証明書を個別に生成する必要があります。たとえば、PEM エージェントロールは、--enable-pem および--enable-pg-backup-api または config.yml 同等の両方が使用される場合、セットアップ中にクライアント証明書を生成します。

tpaexec upgrade を使用した pg バックアップ API のマイナー更新

特定のパッケージバージョンにアップグレードしようとする場合は、config.yml の pg_backup_api_package_version が更新されて、目的のバージョンを反映していることを確認します。

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade <cluster_dir> -e pg_backup_api_package_version="<desired version>"
  --components=pg-backup-api
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードのために PG Backup API を選択するには、 tpaexec upgrade コマンドに渡される--components フラグに pg-backup-api または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.34 Setting sysctl values

デフォルトでは、TPA はターゲットインスタンスにさまざまな sysctl 値を設定し、それらを/etc/sysctl.conf に含めて、リブート後も持続するようにします。

オプションで sysctl_values に独自の値を指定できます。

```
cluster_vars:
  sysctl_values:
    kernel.core_pattern: core.%e.%p.%t
    vm.dirty_bytes: 4294967296
    vm.zone_reclaim_mode: 0
```

指定した値は、その変数の TPA のデフォルト値がある場合より優先されます。設定は最初に sysctl.conf 行ごとに追加され、最後に sysctl -p でロードされます。

Docker および lxd インスタンスは sysctl の設定をサポートしていないため、TPA はこれらのプラットフォームでこの手順を完全にスキップします。

Hugepages

デフォルトでは、TPA は Postgres が shared_buffers に使用するアーキテクチャのデフォルトサイズ x86_64 で 2MB の huge ページを予約します。これは、hugepages=N をカーネルコマンドラインに追加し、vm.nr_hugepages = N から/etc/sysctl.conf に書き込むことにより実行されます。

Postgres でより大きなヒュージページサイズ通常 x86_64 で 1GB を使用したい場合は、huge_page_size を Postgres スタイルのメモリ文字列に設定します。

```
instances:
  - Name: primary
```

(次のページに続く)

(前のページからの続き)

```
    vars:
      huge_page_size: 1GB
```

huge_page_size が設定されている場合、TPA は次のことを行います。

- カーネルコマンドラインでそのサイズのページを予約します。例 hugepagesz=1G hugepages=N ;
- /etc/sysctl.conf から vm.nr_hugepages を省略しますその sysctl はアーキテクチャのデフォルトサイズのプールにのみ適用され、他のサイズにはサイレントに無効です。そして
- Postgres が選択したプールからのページを使用するように、postgresql.conf で huge_page_size GUC を設定します。

cluster_vars ではなく、個々の Postgres インスタンスで huge_page_size を設定するのがほとんどの場合、正しいです。カーネルコマンドラインで行われた予約は、ブート時に通常のメモリから取得されるため、同じクラスター内の Barman サーバーまたは他の非 Postgres インスタンスは Postgres ノードと同じ割り当てが必要です。

デフォルトでは、TPA は shared_buffers プラス合計 RAM の 3%を巨大ページサイズで割ったものとして、予約する巨大ページ数を計算します。nr_hugepages を直接設定することにより、この計算をオーバーライドできます。

```
instances:
  - Name: primary
    vars:
      huge_page_size: 1GB
      nr_hugepages: 16
```

注釈

下位互換性のため、*nr_hugepages* が明示的に設定されていない場合、既存の `sysctl_values['vm.nr_hugepages']` がページ数として受け入れられます。

デフォルトでは、TPA はカーネルコマンドラインで transparent_hugepage を「never」に設定し、スクリプト/etc/tpa/rc.local をインストールして、sys ファイルシステムを介してブート時に設定します。別の設定を使用するには、インスタンス vars または cluster_vars で transparent_hugepage を always または madvise に設定して、すべてのインスタンスを制御します。

12.6.35 Creating Postgres databases

デプロイ中に Postgres データベースを作成するには、config.yml の cluster_vars または特定のインスタンスの vars の下の postgres_databases のリストにエントリを追加します。

```
cluster_vars:
  postgres_databases:
  - name: exampledb
```

(次のページに続く)

(前のページからの続き)

```
- name: complexdb
  owner: example
  encoding: UTF8
  lc_collate: de_DE.UTF-8
  lc_ctype: de_DE.UTF-8
  template: template0
  extensions:
  - name: hstore
  - name: dblink
  languages:
  - name: plperl
  - name: plpython
  tablespace: exampletablespace
```

上記の例では、2 つのデータベースを作成します bdr_database など、TPA 自分自身が作成することを決定したデータベースとは別に。

各エントリは、作成するデータベースの name を指定する必要があります。他のすべての属性はオプショナル。

owner はデフォルトで postgres ですが、任意の有効なユーザー名に設定できます *Creating Postgres users* のユーザー

この時点で作成されています)。

encoding 、lc_collate 、および lc_ctype 値は、デフォルトで initdb の実行時に設定された postgres_locale になりますデフォルトは、ターゲットシステムの LC_ALL または LANG 設定を使用します。デフォルト以外のロケール設定でデータベースを作成する場合、template: template0 も指定する必要があります。

オプションで、データベースのデフォルトの tablespace を指定できます。表領域は既に存在する必要があります *Creating Postgres tablespaces* を参照してください。

extensions および languages のオプションのリストを指定して、各データベース内で作成することができますテンプレートデータベースから継承した拡張機能または言語に加えて。必要なパッケージは、たとえば *Installing Postgres-related packages* に含めることにより、既にインストールされている必要があります。

TPA は、postgres_databases で言及されていない既存のデータベースをドロップせず、必要に応じて追加のデータベースを作成する場合があります BDR など。

12.6.36 Creating Postgres tablespaces

デプロイメント中に Postgres テーブルスペースを作成するには、config.yml の cluster_vars または特定のインスタンスの vars の下の postgres_tablespaces に名前と場所を定義します。

volume_for: postgres_tablespace セットと tablespace_name を定義して *Filesystem configuration* を実行すると、それらはデフォルトエントリとして postgres_tablespaces に追加されます。

```
cluster_vars:
  postgres_tablespaces:
    explicit:
      location: /some/path

instances:

- Name: example
  …
  volumes:
  - device_name: /dev/xvdh
    …
    vars:
      volume_for: postgres_tablespace
      tablespace_name: implicit
```

上記の例では、2 つのテーブルスペースを作成します。明示的 /some/path と暗黙的 ボリュームの別のマウントポイントを指定しない限り、デフォルトで/opt/postgres/tablespaces/implicit/tablespace_data に。

すべての postgres_tablespace ボリュームには tablespace_name を定義する必要があります。テーブルスペースの場所は、ボリュームのマウントポイントから取得されます。

postgres_tablespaces のすべてのエントリは、表領域名をキーとしてとその location を指定する必要があります。テーブルスペースの場所を明示的に指定する場合は、PGDATA 内にテーブルスペースを置かず、ボリュームマウントポイントをテーブルスペースの場所として直接使用しないでください。lost+found は、一部のツールを混乱させて、ディレクトリが空でないと判断します。

デフォルトでは、テーブルスペース owner は postgres ですが、任意の有効なユーザー名に設定できます *Creating Postgres users* のユーザー

この時点で作成されています)。

ストリーミングレプリカには、上流のインスタンスと同じ postgres_tablespace ボリュームと postgres_tablespaces 設定が必要です

Creating Postgres databases でデータベースのデフォルトの表領域を設定できます。

12.6.37 Configuring .pgpass

TPA は、クラスターインスタンス間で使用するために、デフォルトで~postgres/.pgpass を作成し、その中の postgres_user postgres_flavour に応じてデフォルトで postgres または entreprisedb のパスワードを使用します。

pgpass_users を設定して、ユーザーの別のリストのエントリを作成できます。pgpass_users リストはデフォルト値をオーバーライドするため、postgres_user postgres /enterprisedb は、 pgpass_users リストに明示的に含めない限り含まれないことに注意してください。

フックスクリプトから postgres/pgpass ロールを含めて、独自の.pgpass ファイルを作成することもできます。

```
- include_role: name=postgres/pgpass
  vars:
    pgpassfile: ~otheruser/.pgpass
    pgpass_owner: otheruser
    pgpass_group: somegroup
    pgpass_users:
    - xyzuser
    - pqruser
```

12.6.38 The postgres Unix user

このページには、postgres ユーザーとそのホームディレクトリがどのように構成されているかについて文書化されています。

Creating Postgres users の作成方法に関する別のページがあります。

シェル構成

TPA は.bashrc ファイルをインストールし、それが.profile または.bash_profile ファイルにも含まれることを確認します。

ユーザー名、ホスト名、作業ディレクトリを含むプロンプトを設定し、postgres_bin_dir が PATH にあることを確認し、PGDATA を postgres_data_dir の場所に設定します。

オプションで extra_bashrc_lines を指定して、.bashrc に任意の行を追加できます。 YAML 複数行文字列構文 >- を使用して、シェルメタキャラクタの引用符とエスケープについて心配する必要を回避します。

```
cluster_vars:
  extra_bashrc_lines:
  - alias la=ls\ -la
  - >-
    export PATH="$PATH":/some/other/dir
```

sudoers を編集して sudo systemctl start/stop/reload/restart/status postgres を許可し、また ulimits を変更して無制限のコアダンプを許可し、ファイル記述子の制限を引き上げます。

SSH キー

TPA は ssh-keygen を使用して postgres ユーザーの SSH キーペアを生成およびインストールし、クラスター内のインスタンスが postgres として互いに ssh できるように.ssh/authorized_keys を編集します。

TLS 証明書

デフォルトでは、TPA はプライベートキーと自己署名 TLS 証明書を生成します。これらは Postgres がそれぞれ ssl_key_file および ssl_cert_file として使用します。ファイルは、TPA クラスター名 cluster_name.key および cluster_name.crt を使用して名前付けられ、/etc/tpa にあります。独自のキーと証明書を提供する方法を含む詳細については、*SSL configuration* のドキュメントを参照してください。

自己署名 TLS キーのサイズは、変数 postgres_rsa_key_size を cluster_vars セクションに追加して変更できます。

```
(...)
cluster_vars:
  postgres_rsa_key_size: 4096
```

デフォルトでは、自己署名 TLS 証明書の期間は 3650 日です。期間は、cluster_vars セクションで openssl_certificate_validity 日単位を定義することにより調整できます。証明書を置き換える必要がある場合は、tpaexec deploy tpaexec deploy (...) -e tpa_tls_renew_tls_certificates=true で tpa_tls_renew_tls_certificates 変数を使用します。

ユーザー名

postgres_user および postgres_group 設定デフォルトでは両方とも postgres は、どこでも一貫して使用されます。何らかの理由で Postgres を別のユーザーとして実行する必要がある場合、それらを変更できます。

12.6.39 Creating Postgres users

デプロイ中に Postgres ユーザーを作成するには、config.yml の cluster_vars または特定のインスタンスの vars の下の postgres_users のリストにエントリを追加します。

```
cluster_vars:
  postgres_users:
  - username: example

  - username: otheruser
    generate_password: true
    role_attrs:
    - superuser
    - replication
    granted_roles:
    - r1
    - r2
```

上記の例では、2 つのユーザー repmgr や barman など、TPA 自分自身が作成することにしたユーザーを除きます。

各エントリーは、作成する username を指定する必要があります。

granted_roles リストのロールは、新しく作成したユーザーに付与されます。

role_attrs リストには、特定の CREATE ROLE options が含まれる場合があります

[NO]SUPERUSER 、[NO]CREATEDB 、[NO]LOGIN ユーザーまたはロールを作成するためなど。

パスワードの生成

デフォルトでは、TPA はユーザーのランダムなパスワードを生成し、クラスターのインベントリの<username>_password という名前のボールトで暗号化された変数に保存します。後で値を取得できます。

```
tpaexec show-password ~/clusters/speedy example
```

```
beePh~iez6lie4thi5KaiG%eghaeT]ai
```

config.yml でパスワードを明示的に指定することはできませんが、代わりに別の<username>_password をインベントリに保存できます。

```
tpaexec store-password ~/clusters/speedy example --random
tpaexec show-password ~/clusters/speedy example
```

```
)>tkc}}k1y4&epaJ?;NJ:luT{C7D*&lt;p
```

```
tpaexec store-password ~/clusters/speedy example
```

プロンプトにパスワードを入力します

```
Password:
```

```
tpaexec show-password ~/clusters/speedy example
```

```
terrible insecure password
```

ユーザーにパスワードをまったく与えたくない場合は、generate_password: false を設定できます。

12.6.40 tpaexec archive-logs

ログディレクトリを作成し、インスタンスからログをアーカイブするには、次のコマンドを実行します。

```
tpaexec archive-logs <cluster-dir>
```

これにより、クラスターディレクトリに logs/YYYYMMDDHHMMss/ディレクトリが作成され、クラスター内の各インスタンスの/var/log の下のすべてのファイルの tar.gz アーカイブが別のディレクトリにダウンロードされます。

前提条件

既存のクラスターがある場合は、tpaexec archive-logs をすぐに実行できます。ただし、新しいクラスターを構成する場合は、少なくともクラスターを *tpaexec provision* する必要があります。クラスターを *tpaexec deploy* する場合、より多くのログを取得します。

クイックスタート

```
tpaexec archive-logs ~/clusters/speedy
```

```
PLAY [Prepare local host archive]  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ***

TASK [Collect facts]  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ****  ****
ok: [localhost]

TASK [Set time stamp]  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ****  ***
ok: [localhost]

TASK [Create local log archive directory]  ****  ****  ****  ****  ****  ****  ****
↪****  ***
changed: [localhost]

PLAY [Archive log files from target instances]  ****  ****  ****  ****  ****  ****
↪**** **

...

TASK [Remove remote archives]  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ***
changed: [kinship]
changed: [khaki]
changed: [uncivil]
changed: [urchin]

PLAY RECAP  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****  ****
↪****  ****  ****  **** **
khaki                      : ok=3    changed=3    unreachable=0    failed=0
kinship                    : ok=3    changed=3    unreachable=0    failed=0
localhost                  : ok=3    changed=1    unreachable=0    failed=0
uncivil                    : ok=3    changed=3    unreachable=0    failed=0
urchin                     : ok=3    changed=3    unreachable=0    failed=0
```

より詳細な出力が必要な場合は、コマンドに-v 、-vv などを追加できます。

生成されたファイル

クラスターディレクトリの下で、各インスタンスのログを見つけることができます。

```
~/clusters/speedy/logs/
`-- 220220306T185049
    |-- khaki-logs-20220306T185049.tar.gz
    |-- kinship-logs-20220306T185049.tar.gz
    |-- uncivil-logs-20220306T185049.tar.gz
    `-- urchin-logs-20220306T185049.tar.gz
```

アーカイブコンテンツの例

```
khaki-logs
|-- anaconda
|   |-- anaconda.log
|   |-- dbus.log
|   |-- dnf.librepo.log
|   |-- hawkey.log
|   |-- journal.log
|   |-- ks-script-ipdkisn0.log
|   |-- ks-script-jr03uzns.log
|   |-- ks-script-mh2iidvh.log
|   |-- lvm.log
|   |-- packaging.log
|   |-- program.log
|   |-- storage.log
|   |-- syslog
|   `-- X.log
|-- btmp
|-- dnf.librepo.log
|-- dnf.log
|-- dnf.rpm.log
|-- hawkey.log
|-- lastlog
|-- private
|-- tpaexec.log
`-- wtmp
```

12.6.41 tpaexec download-packages

ダウンローダーの目的は、既存の構成から TPA クラスターの完全なインストールを実行するために必要なパッケージを提供することです。これは、インターネットアクセスのない安全なクラスターにパッケージを出荷する場合、またはテストクラスター用にパッケージを繰り返しダウンロードすることを避けたい場合に役立ちます。

ダウンローダーは、必要なパッケージの完全な依存関係ツリーをダウンロードし、結果のパッケージリポジトリには、ターゲットディストリビューションパッケージマネージャーのメタデータファイルが含まれるため、クラスターを構築するためにのみ使用できます。現時点では、パッケージマネージャー Apt および YUM がサポートされています。

注釈

ダウンロードパッケージ機能を使用するには、TPA ホストに Docker をインストールする必要があります。これは、ダウンローダーがターゲットオペレーティングシステムのコンテナを作成することにより動作し、そのシステムのパッケージマネージャーを使用して依存関係を解決し、必要なパッケージをすべてダウンロードするためです。ダウンロードパッケージに必要な Docker セットアップは、*Docker のインストール* の場合と同じです。

使用法

既存のクラスター構成が存在する必要があります。これは、`tpaexec configure` コマンドを使用して実現できます。ダウンローダーを使用するために特定のオプションは必要ありません。**クラスター構成** を参照

。

download-packages サブコマンドを実行して、ダウンロードプロセスを開始します。ダウンローダーが使用する OS と OS バージョンを提供します。

```
tpaexec download-packages cluster-dir --os RedHat --os-version 8
```

これは、特定の docker イメージとして表現することもできます。次の例のように、接頭辞がついた tpa イメージのいずれかを使用することを強くお勧めします。

```
tpaexec download-packages cluster-dir --docker-image tpa/redhat:8
```

ダウンローダーは、ダウンロードしたファイルをデフォルトで `local-repo` ディレクトリに配置します。オプション `--download-dir path` を使用して、別のディレクトリにダウンロードすることができます。

結果の使用

`local-repo` ディレクトリの内容には、docker イメージに含まれる OS に従って、ansible によって決定された構造が設定されます。たとえば、docker イメージ `tpa/redhat:8` には次のものが含まれます。

```
cluster-dir/
`-- local-repo
    `-- RedHat
        `-- 8
            |-- *.rpm
            `-- repodata
                `--  *repodata-files*
```

これをクラスターでそのまま使用するか、ターゲット制御ノードにコピーできます。*Managing clusters in a disconnected or air-gapped environment* を参照してください。*Creating and using a local repository* は、TPA によって自動的に検出および使用されます。

ローカルリポジトリのメタデータを再生成

アップグレードの前に、 download-packages コマンドを--refresh-repository オプションとともに使用して、ローカルリポジトリのメタデータを再生成し、新しいパッケージまたは既存のパッケージの新しいバージョンを確認できます。新しいコンテンツを local-repo ディレクトリに配置し、各 config.yml ファイルのパッケージの新しいバージョンを更新した後、次のようにコマンドを実行する必要があります。

```
tpaexec download-packages cluster-dir --os <OS> --os-version <version> --refresh-
↪repository
```

--refresh-repository オプションは、新しい Docker コンテナを作成し、リポジトリのメタデータを再生成し、自分自身を削除します。

> **注釈**
>
> 上記のコマンドを実行すると、各ノードのローカルリポジトリのキャッシュは自動的に更新されません。これはアップグレードステージで自動的に実行されるか、 *tpaexec cmd* を使用して手動で行う必要があります。

ダウンロードプロセス中にエラーが発生した場合、コマンドはデバッグを支援するために実行中のダウンローダーコンテナを残します。たとえば、障害が発生したダウンローダーコンテナにログインして、ログまたはネットワークを検査することができます。ダウンローダーコンテナの名前は、コンテナ名の許可制限 64 文字を超えない限り、通常$cluster_name-downloader になります。docker ps を実行して実行中のコンテナをリストし、クラスター名と一致するコンテナ名を探すことにより、正確な名前を確認できます。ほとんどの場合、docker exec -it $cluster_name-downloader /bin/bash を実行することにより、実行中のコンテナにログインできます。検査後、--tags cleanup で download-packages コマンドを実行することにより、残ったコンテナをクリーンアップできます。例

```
tpaexec download-packages cluster-dir --docker-image tpa/redhat:8 --tags cleanup
```

12.6.42 TPA custom commands

TPA クラスター内のインスタンスで環境に固有のタスクを実行するカスタムコマンドを定義できます。

このメカニズムを使用して、クラスターに適用されるプロセスを自動化できます。これらのコマンドは、ビルトインのクラスター管理コマンドと同様に、クラスターディレクトリに対して呼び出すことができます。このようなプロセスを定義および実行する統一の方法があると、誤解やオペレーターエラー、または過去に正しかったが、それ以降現実から乖離したプロセスドキュメントによって引き起こされるエラーの可能性が減少します。

Ansible プレイブックを書くということは、任意に複雑なタスクを実装できることを意味します。カスタムコマンド規約に従うということは、config.yml と、TPA が実行するクラスター検出タスクに基づいて設定されるさまざまなファクトを利用でき、接続、認証、その他の基本機能などの詳細を考える必要がないことを意味します。

これにより、アドホックシェルスクリプトは可能ですが、通常はそうではない方法で、復元力の高いべき等コマンドを書くのがはるかに簡単になります。

クイックスタート

- クラスターディレクトリ内に commands/mycmd.yml を作成します
- tpaexec mycmd /path/to/cluster を実行します

例

次に、クラスター内のすべてのインスタンスで単一のコマンドを実行するコマンドの例を示します。ユースケースに応じて、さまざまなホストをターゲットとするコマンドたとえば、Postgres インスタンスでのみ実行する hosts: role_postgres を書くか、追加のタスクを実行して条件を評価して、何を行うかを正確に決定できます。

```
- --

#  Always start with this


-  import_playbook: "{{ tpa_dir }}/architectures/lib/init.yml"
  tags: always


-  name: Perform custom command tasks
  hosts: all
  tasks:
  - name: Display last five lines of syslog
    command: tail -5 /var/log/syslog
    become_user: root
    become: yes
```

12.6.43 TPA custom tests

環境とアプリケーションに固有の詳細なテストを簡単に定義して、TPA の *tpaexec* テストを実行する を強化できます。

どんなに単純なタスクでも、クラスターで実行するテストを作成して、すべてが期待どおりに動作していることを確認することを強くお勧めします。このようなテストを実行する統一的で反復可能な方法があると、危機に対処する場合でも、日常的なクラスター管理を行う場合でも、重要な点を見逃すことがありません。

構成されたロールまたはその他のプロパティによってクラスターインスタンスをターゲットとするテストを記述する場合、該当するすべてのテストが適切なインスタンスで実行されることを保証できます。レプリケーションステータスを確認するレプリカの数、pgbouncer を実行しているサーバー、または手動で確認するときに間違いを招きやすいその他の詳細を調べたり覚えたりする必要はありません。

テストはクラスターに重要な変更を加えてはなりません。実稼働サーバーで実行することを考えない場合、おそらくテストすべきではありません。

クイックスタート

- クラスターディレクトリ内に tests/mytest.yml を作成します
- tpaexec test /path/to/cluster mytest を実行します

tpaexec test への--include-tests-from /other/path オプションを使用して、他の場所にテストを作成し、クラスター全体で使用することもできます。

使用方法については、 tpaexec help test を実行してください。

例

次に、すべての Postgres インスタンスで実行されるテストを記述する方法を示します。hosts: all の代わりに hosts: role_postgres に注意してください。

任意の Ansible タスクを使用して、クラスターから情報を収集し、テストを実行できます。いくつかの期待が満たされない場合に失敗するタスクを書くだけです assert 、fail … when など。

```
- --

-  name: Perform my custom tests
  hosts: role_postgres
  tasks:

  # Always start with this
  - include_role:
      name: test
      tasks_from: prereqs.yml

  # Make sure that the PGDATA/PG_VERSION file exists. (This is just a
  # simplified example, not something that actually needs testing.)
  - name: Perform simple test
    command: "test -f {{ postgres_data_dir }}/PG_VERSION"
    become_user: "{{ postgres_user }}"
    become: yes

  - name: Run pg_controldata
    command: >
      {{ postgres_bin_dir }}/pg_controldata {{ postgres_data_dir }}
    register: controldata
    become_user: "{{ postgres_user }}"
    become: yes

  # Write output to clusterdir/$timestamp/$hostname/pg_controldata.txt
  - name: Record pg_controldata output
    include_role:
        name: test
        tasks_from: output.yml
```

(次のページに続く)

(前のページからの続き)

```
  vars:
    output_file: pg_controldata.txt
    content: |
      {{ controldata.stdout }}
```

上記のようにビルトイン output.yml を使用して、クラスターディレクトリのタイムスタンプ付きテストディレクトリに任意のテスト出力を記録できます。

各テストは、完全な Ansible プレイブックつまり、タスクのリストではなくプレイのリストである必要があります。これは、基本的な TPA セットアップタスクの後にインポートおよび実行されます。

破壊的テスト

デフォルトでは、テストはクラスターに重要な変更を加えないでください。レプリケーションをテストするテーブルを作成するようなことをする場合でも、自分自身の後のクリーンアップに注意する必要があります。

実稼働クラスターでは受け入れられないクラスターに変更を加えるテストは、destructive としてマークされなければなりません。これらは、開発時、または初期クラスターの「バーンイン」プロセス中にのみ実行するテストである場合があります。

テストに prereqs.yml を含める場合、destructive: yes を設定することにより、「破壊的」テストを定義できます。

```
- hosts: …
  tasks:
  - include_role:
      name: test
      tasks_from: prereqs.yml
    vars:
      destructive: yes
```

誰かが tpaexec test /path/to/cluster mytest を実行すると、--destroy-this-cluster オプションを使用して実行を確認するように求めるエラーが発生します。

--destroy-this-cluster の使用は、コマンドを実行するリスクを認識することを示します。テストが実際にクラスターを破壊することを保証するものではありません。

12.6.44 Configuring the PGAI agent

> **注釈**
>
> TPA は現在、PGAI エージェントを使用してクラスターを EDB Postgres AI Cloud Service と統合することのみをサポートしています。 EDB Postgres AI Hybrid Manager と統合するためのエージェントの使用、または使用状況レポートは、TPA の将来のリリースでサポートされる予定です。

PGAI エージェントパッケージバージョン

デフォルトでは、TPA は beacon-agent の利用可能な最新バージョンをインストールします。

config.yml ファイルの cluster_vars セクションの下に beacon_agent_package_version: xxx を含めることにより、インストールされる beacon-agent パッケージのバージョンを指定できます。

```
cluster_vars:
    …
    beacon_agent_package_version: 1.56.2-1
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

PGAI エージェント構成

PGAI エージェント構成には、クラスターごとに設定する必要がある 2 つのパラメーター、アクセスキーとプロジェクト ID が含まれています。

アクセスキーはクラスターディレクトリで暗号化されたままで、tpa の store-password および show-password コマンドを使用して設定または読み取ることができます。

```
tpaexec store-password . beacon_agent_access_key
```

```
Password:
```

```
tpaexec show-password . beacon_agent_access_key
```

tpaexec provision の実行時に環境変数 BEACON_AGENT_ACCESS_KEY が設定されている場合、アクセスキーはその値から設定されます。

プロジェクト ID は、cluster_vars の beacon_agent_project_id キーの下の config.yml に保存されます。--beacon_agent_project_id 引数が tpaexec configure に渡される場合、その値は config.yml に適切に書き込まれます。

PGAI エージェントのインストール

TPA は EDB のリポジトリから PGAI エージェントをインストールし、beacon というオペレーティングシステムユーザーと beacon というデータベースユーザーを作成します。エージェントの構成ファイルは、ビーコンユーザーのホームディレクトリの.beacon/beacon_agent.yaml に書き込まれます。

PGAI エージェントの実行

TPA は、systemd サービスユニットファイルをインストールして、ブート時にエージェントを起動し、ビーコンユーザーとして実行します。

12.6.45 Compliance

TPA は、クラスターが STIG または CIS 標準に準拠することを簡単にするように設計された構成を生成できます。`--compliance stig` または`--compliance cis` を `tpaexec configure` に渡すと、TPA は次のことを行います。

- 他のオプションが適切な規格と互換性があることを確認します。
- これが特定の標準に準拠するクラスターであることのマーキングや、標準の要求に応じて Postgres 構成の設定を含む、生成された `config.yml` にさまざまなエントリを追加します。
- コンプライアンスを適用するように一部の展開タスクを調整します。
- 展開の最後にチェックを実行します。

オプション`--excluded_tasks=compliance` 〜`tpaexec deploy` を指定することにより、展開時チェックをスキップできます。この機能は、完全なコンプライアンスが不可能なテストシステムを使用する場合、SSL 証明書が使用できない場合など、テストのみを対象としています。

TPA が選択した標準に意図的に準拠しない場合があります。これらは以下の例外に文書化されています。

STIG

STIG 準拠は、 `tpaexec configure` への`--compliance stig` オプションによって示されます。

オプションの互換性

STIG コンプライアンスには、`bare` プラットフォームと `epas` フレーバーが必要です。バージョン 8 または 9 の RedHat OS が必要です。

config.yml の設定

EDB Postgres Advanced Server の SQL /Protect 機能を使用するために、次のエントリが `cluster_vars` に追加されます。

```
extra_postgres_extensions: [ sql_protect ]
```

次のエントリが `cluster_vars` に追加されて、クライアントに SSL 認証を使用するように強制します。

```
hba_force_hostssl: True
hba_force_certificate_auth: True
hba_cert_authentication_map: sslmap
```

次のエントリが `cluster_vars` に追加されて、postgresql.conf で GUC が設定されます。

```
tcp_keepalives_idle: 10
tcp_keepalives_interval: 10
tcp_keepalives_count: 10
log_destination: "stderr"
postgres_log_file_mode: "0600"
```

次のエントリが cluster_vars の postgres_conf_settings に追加されて、postgresql.conf で GUC を設定します。

```
edb_audit: "xml"
edb_audit_statement: "all"
edb_audit_connect: "all"
edb_audit_disconnect: "all"
statement_timeout: 1000
client_min_messages: "ERROR"
```

デプロイメントの違い

TPA は、展開中に、通常の操作に必要な接続数に対応して、作成するデータベースユーザーの接続制限を設定します。各ユーザーが設定されると、SSL クライアント証明書が提供されたことも確認します。

クライアント SSL 証明書の提供

STIG は、クライアント接続に国防総省承認の ssl 証明書を必要とします。これらの証明書は TPA では生成できないため、提供する必要があります。クラスター内のノードからユーザーの認証を設定すると、TPA はノードで証明書/キーペアを検索します。証明書とキーは、ssl_client_cert_dir 設定で指定されたディレクトリの<username>.crt および<username>.key というファイルにある必要があります。この設定のデフォルトは/ であるため、ファイルは、たとえば、barman ユーザーがセットアップされている場合の/barman.crt および/barman.key になります。

最終チェック

展開の最後に、TPA はサーバーで FIPS が有効になっていることを確認します。

例外

フェールオーバーマネージャーとして EFM を選択すると、TPA は EFM ユーザーのパスワード認証を構成します。これは、すべての TCP 接続が証明書認証を使用するという STIG 要件に反しています。この例外の理由は、EFM が証明書認証をサポートしていないためです。

CIS

CIS 準拠は、 tpaexec configure への--compliance cis オプションによって示されます。

config.yml の設定

次のエントリが cluster_vars に追加されて、postgresql.conf で GUC が設定されます。

```
log_connections: "on"
log_disconnections: "on"
```

次のエントリーが cluster_vars に追加されて、必要な拡張機能を有効にします。

```
extra_postgres_extensions: ["passwordcheck", "pgaudit"]
```

EPAS にはビルトイン監査ログがあるため、EPAS クラスターでは、pgaudit はこのリストから省略されます。

次のエントリーが cluster_vars に追加されて、postgres OS ユーザーの umask を設定します。

```
extra_bash_rc_lines: "umask 0077"
```

次のエントリが cluster_vars の postgres_conf_settings に追加されて、postgresql.conf で GUC を設定します。

```
log_error_verbosity: "verbose"
log_line_prefix: "%m [%p]: [%l-1] db=%d,user=%u,app=%a,client=%h "
log_replication_commands: "on"
temp_file_limit: "1GB"
```

最終チェック

展開の最後に、TPA はサーバーで FIPS が有効になっていることを確認します。

例外

CIS 仕様で記載されているように、TPA は pgBackRest をサポートしていません。代わりに、TPA は Barman をインストールします。

TPA は、CIS 仕様で必要とされる set_user のインストールと構成を行いません。これは、Postgres ユーザーによるログオンを防止すると、TPA がデータベースに接続および構成できなくなるためです。

12.6.46 etcd

etcd は、分散調整に一般的に使用される分散された信頼性の高いキー-値ストアです。TPA は、PostgreSQL クラスターのフェールオーバーマネージャーとして Patroni を使用する場合、主に分散構成ストア DCS として etcd を構成します。

M1 アーキテクチャで Patroni を有効にすると、TPA はデフォルトで 3 ノード etcd クラスターを自動的に展開および構成します。

インストール

TPA は、構成されたシステムリポジトリから入手可能な etcd パッケージをインストールします。SLES および RHEL ベースのシステムでは、TPA は PGDG extras リポジトリを自動的に有効にして etcd パッケージを提供します。

etcd パッケージバージョン

デフォルトでは、TPA は etcd の利用可能な最新バージョンをインストールします。

インストールされる etcd パッケージのバージョンは、 config.yml ファイルの cluster_vars セクションの下の etcd_package_version: xxx を含めることにより指定できます。

```
cluster_vars:
    …
    etcd_package_version: 3.6.2*
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを先頭に追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

構成

TPA は、config.yml 設定とクラスタートポロジに基づいて/etc/etcd/etcd.conf ファイルを生成します。ピアおよびクライアントの通信 URL、データディレクトリ、および SSL 証明書を構成します。

セキュリティ

TPA は、 etcd クラスターとの通信および etcd クラスター内の通信を保護するための堅牢なオプションを提供します。これは、主に etcd_ssl_enabled および etcd_authentication_mode 変数によって制御されます。

TLS 暗号化

etcd_ssl_enabled: true を設定すると、すべての etcd トラフィックの TLS 暗号化が有効になります。TPA は自動的に次のことを行います。

- 各 etcd ノードのクラスター CA とサーバー証明書適切な IP および DNS サブジェクト代替名を含むを生成します。
- ピアツーピア通信とクライアントサーバー通信の両方に HTTPS を使用するように etcd を構成します。
- HTTPS を介して接続し、CA を使用してサーバー証明書を検証するようにクライアント etcdctl 、Patroni を構成します。

クライアント認証

TLS 暗号化が有効になる etcd_ssl_enabled: true すると、etcd_authentication_mode 変数を使用して認証モードを選択できます。

- **``none`` デフォルト** クライアント認証は実行されません。etcd_ssl_enabled が true の場合、通信は TLS 暗号化にのみ依存します。
- **``basic`` :** クライアントにユーザー名とパスワードを使用して認証する必要があります。自動的に TPA

- 生成されたパスワードで管理用 `root` ユーザーを作成します。
- Patroni を使用する場合、特定のキープレフィックス/`tpa/cluster_name` などにのみ `readwrite` アクセスを持つ Patroni の専用の最小特権ユーザー `patroni_etcd_user` 経由で構成可能。
- これらの専用の資格情報を使用するように Patroni を構成します。
- **``mtls`` :** クライアントピアノードを含むに、信頼できるクラスター CA によって署名された有効な TLS 証明書を提示する必要があります。TPA は自動的に
- ピア接続の両方のクライアント証明書を要求および検証するように `etcd` を構成します。
- 認証のためにクライアント証明書を提示するようにクライアント `etcdctl` 、Patroni を構成します。

注釈

TPA には、展開または再構成中に異なる認証モード間の移行を処理する堅牢なロジックが含まれており、クラスターの安定性を保証します。構成検証チェックは、無効な組み合わせを防ぐために早期に実行されますたとえば、*mtls* には`etcd_ssl_enabled: true` が必要です。ただし、現在`etcd_ssl_enabled` の値の移行つまり `false` から `true` またはその逆を処理することはできません。この制限により、*etcd_ssl_enabled* の別の値を必要とする `etcd_authentication_mode` 値を切り替えることもできなくなります。例、*basic* から `mtls` 、または `mtls` から `basic` 。 23.41.0 より前のバージョンの TPA を使用して展開されたクラスターはデフォルトで `etcd_ssl_enabled: false` に設定されているため、 *etcd_ssl_enabled* を変更することはできません。

`etcd` には、次の変数を設定できます。

Variable	Default value	Description
etcd_peer_port	*2380*	`etcd`がピアツーピアサーバー間通信に使用する TCP ポート。
etcd_client_port	*2379*	`etcd`がクライアント通信に使用する TCP ポート。
etcd_ssl_enabled	*false**	すべての `etcd`通信の SSL/TLS 暗号化を有効にします。セキュリティを参照してください。 * *tpaexec configure* `を介した新しいクラスターの場合は` *true*。
etcd_authentication_mode	*none**	クライアント認証モード *none*、*basic*、*mtls* を定義します。 *none* `以外のモードの場合は` *etcd_ssl_enabled: true* `が必要です。セキュリティを参照してください。` * *tpaexec configure* `を介した新しいクラスターの場合は` *mtls*。
etcd_compaction_mode	*revision*	自動圧縮モード *revision* `または` *periodic* 。

次のページに続く

表 10 – 前のページからの続き

Variable	Default value	Description
etcd_compaction_retention	*10*	自動圧縮のリテンション値。 *revision* `モードの場合、これは保持するリビジョンの数です。 `*periodic* `モードの場合、これは時間間隔たとえば `*1h* 。

tpaexec upgrade を使用した etcd のマイナー更新

etcd v3.6 は--enable-v2 フラグを完全に削除し、v2 API をまったく使用できません。

TPA アップグレードは、v3 API を使用した etcd バージョンのアップグレードのみをサポートしています。

TPA を使用してクラスター内の etcd をアップグレードする前に、migrated from the v2 API to the v3 API があることを確認してください

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade -e etcd_package_version="<desired version>" --components=etcd
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードのために etcd を選択するには、 tpaexec upgrade コマンドに渡される--components フラグに etcd または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.47 Locale

一部のプラットフォームイメージと環境では、地域と言語設定を設定することが望ましい場合があります。

デフォルトでは、TPAexec は en_US.UTF-8 ロケールシステムファイルをインストールします。 config.yml で目的のロケールを設定できます。

```
user_locale: en_GB.UTF-8
```

サポートされているロケールを見つけるには、次のコマンドの出力を参照します。

```
localectl list-locales
```

または、Debian または Ubuntu のファイル/etc/locales.defs の内容。

12.6.48 Patroni cluster management commands

Patroni は、次のコマンドオプションを使用して、M1 アーキテクチャでシングルマスターフェールオーバーマネージャーとして使用できます。

```
tpaexec configure cluster_name -a M1 --enable-patroni --postgresql 14
```

または config.yml オプションを設定することにより

```
cluster_vars:
  failover_manager: patroni
```

TPA は、 patroni パッケージ PGDG リポジトリからまたは edb-patroni パッケージ EDB リポジトリからを使用して Patroni クラスターを展開できます。これは、config.yml の cluster_vars の下の patroni_package_flavour オプションを介して構成できます。これは、 --patroni-package-flavour コマンドライン引数を介して設定することもできます。patroni_package_flavour が明示的に設定されていない場合、TPA は、構成されたリポジトリに基づいてフレーバーを推論しようとします。EDB リポジトリが構成されている場合、暗黙的に edb フレーバーを選択し、それ以外の場合は暗黙的に community フレーバーを選択します。

TPA configure は 3 つの etcd ノードを追加し、オプション--enable-haproxy を指定する場合 2 つの haproxy ノードを追加する場合があります。etcd は、分散構成ストア DCS に使用されます。Patroni は他の DCS バックエンドをサポートしていますが、現在 EDB または TPA ではサポートされていません。

HAProxy の代替として、--enable-pgbouncer オプションを使用して、Postgres ノードで PgBouncer を構成できます。PgBouncer は、プライマリの接続をプールするように構成されます。Patroni は、クラスター内のフェールオーバーまたはスイッチオーバー時に PgBouncer を再構成するように構成されるため、PgBouncer は新しいプライマリ Postgres インスタンスに従います。

TPA は、既存の PostgreSQL クラスターを変換する Patroni の機能を使用します。これにより、TPA は構成を初期化および管理できます。PostgreSQL クラスターが作成されると、Patroni が管理を引き継ぎます。TPA は、セットアップ中に使用された postgres 構成ファイルを削除します。

セットアップが完了すると、クラスターの TPA と config.yml の設定を使用して、Postgres を引き続き管理できます。コマンドライン patronictl や REST API などの Patroni インターフェイスを使用することもできますが、可能な限り TPA メソッドを使用することをお勧めします。

Patroni パッケージバージョン

デフォルトでは、TPA は Patroni の最新バージョンをインストールします。

インストールされる Patroni パッケージのバージョンは、 config.yml ファイルの cluster_vars セクションの下の patroni_package_version: xxx を含めることにより指定できます。

```
cluster_vars:
    …
    patroni_package_version: 4.0.0-1PGDG.rhel8`
    …
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

TPA は、Patroni REST API を保護するための堅牢なオプションを提供します。これは、主に patroni_ssl_enabled および patroni_authentication_mode 変数によって制御されます。

TLS 暗号化

patroni_ssl_enabled: true を設定すると、REST API の TLS 暗号化が有効になります。 TPA は自動的に次のことを行います。

- 各 Patroni ノードの TPA クラスター CA とサーバー証明書を生成します。
- HTTPS でトラフィックを処理するように Patroni REST API を構成します。
- HTTPS を介して接続し、CA を使用してサーバー証明書を検証するようにクライアント patronictl や HAProxy などを構成します。

クライアント認証

TLS 暗号化が有効になる patroni_ssl_enabled: true すると、patroni_authentication_mode 変数を使用して認証モードを選択できます。

- **``basic`` デフォルト** クライアントは、設定済みの Patroni REST API のユーザー名とパスワードを提供する必要があります。TPA は、これらの資格情報を自動的に生成します。
- **``mtls`` :** クライアントは、信頼できるクラスター CA によって署名された有効な TLS 証明書を提示する必要があります。TPA は自動的に
- クライアント証明書 verify_client: required を要求および検証するように Patroni REST API を構成します。
- 必要なクライアント証明書とキーを使用してクライアント patronictl 、HAProxy を構成します。

API アクセス制御許可リスト

セキュリティを強化するために、Patroni の REST API は allowlist で自動的に構成されます。このリストは、TPA インベントリで定義されたホスト名に基づいて、有害な可能性のある API エンドポイント POST 、PUT 、PATCH 、DELETE を使用するエンドポイントへのアクセスをクラスター内の他の Patroni ノードにのみ制限します。読み取り専用エンドポイントヘルスチェックなどは、他のソース HAProxy などからアクセスしたままです。

これらの構成変数は、TPA での Patroni の展開での特定の動作を制御するために使用できます。

Variable	Default value	Description
patroni_super_user	postgres	スーパーユーザーロールの postgres で作成するユーザー。
patroni_replication_user	replicator	レプリケーションロールの postgres で作成するユーザー名。
patroni_restapi_user	patroni	patroni REST API 用に構成するユーザー名。
patroni_rewind_user	rewind	pg_rewind ファンクションの postgres で作成するユーザー名。

次のページに続く

表 11 – 前のページからの続き

Variable	Default value	Description
patroni_installation_method	pkg	パッケージまたはソースたとえば git リポジトリまたは docker の場合はローカルソースディレクトリなどから patroni をインストールします。
patroni_package_flavour	community if no EDB repository is configured, else edb	`edb-patroni`パッケージ `edb`フレーバー、EDB リポジトリが必要 または`patroni`パッケージ `community`フレーバー、PGDG および EPEL RedHat ベースのみのリポジトリが必要 をインストールするかどうか。
patroni_ssl_enabled	no	REST API および ctl 接続の SSL を有効にするかどうか。可能な場合は、クラスター SSL 証明書と CA を使用します。
patroni_authentication_mode	*basic*	REST API のクライアント認証モード `basic`または `mtls`を定義します。 `patroni_ssl_enabled: true`が必要です。
patroni_rewind_enabled	yes	postgres rewind を有効にするかどうか、patroni_rewind_user で定義されたユーザーを作成し、config セクションを追加します。
patroni_watchdog_enabled	no	追加のスプリットブレイン防止のためにカーネルウォッチドッグを構成するかどうか。
patroni_dcs	etcd	DCS に使用するバックエンド。現時点では etcd のみがオプションです。
patroni_dcs_namespace	*/tpa*	Patroni がクラスター状態を保存する DCS 内の名前空間。
patroni_listen_port	8008	REST API TCP ポート番号
patroni_etcd_user	etcd_patroni	`etcd_authentication_mode`が `basic`の場合に Patroni が使用する専用 etcd ユーザーのユーザー名。
patroni_conf_settings	{}	パトローニ構成のオーバーライドを含む構造化データオブジェクト。部分的なデータを提供でき、生成された構成とマージされます。実行時に既知のインスタンス情報に基づいて生成された値をオーバーライドしないように注意してください。
patroni_dynamic_conf_settings	{}	DCS 設定専用のオプションの構造化データ。これは`patroni_conf_settings`にマージされます。

次のページに続く

表 11 – 前のページからの続き

Variable	Default value	Description
patroni_repl_max_lag	None	これは、*haproxy_read_only_load_balancer_enabled* が *true* の場合にのみ *haproxy* バックエンドヘルスチェックで使用されます。*`/replica?lag=`* の可能な値については、*:ref:`Ansible Tower <Patroni cluster management commands>`* を参照してください

注釈

Ansible Tower を介して展開する場合、*patroni_etcd_user* をデフォルト値から変更すると、展開が失敗します。これは、TPA がインベントリ内の動的なシークレット名を処理できないために発生します。この問題を回避するには、プロビジョニング後および展開前に、パスワードをローカルで手動生成し、構成リポジトリにプッシュする必要があります。

```
tpaexec store-password <your_custom_username> .
```

patroni の構成は 3 つのレイヤーから構築され、Patroni デーモンによって設定されたデフォルトから始まり、DCS からロードされた構成、最後にローカル構成から構築されます。最後は、構成ファイルから制御でき、環境を介してオーバーライドできます。TPA は構成ファイルを制御し、値はこの順序で構築されます。

API に送信され、構成ファイルのブートストラップセクションに保存される DCS 構成。

- `postgres` の TPA 変数は DCS 設定にロードされます。*postgresql.conf* を参照してください。一部の機能はサポートされていません。以下の注意事項を参照してください。
- DCS 設定の Patroni のデフォルト
- `patroni_dynamic_conf_settings` のユーザー指定のデフォルト。DCS 設定をオーバーライドする場合は、ここで行うことができます。

YAML 構成ファイルに保存されるローカル構成

- 上記の前の手順からロードされた `bootstrap.dcs` 。
- `patroni_ssl_enabled` などの機能フラグによって有効になる構成。上記の表を参照してください。
- 次に、最後にユーザー指定設定からオーバーロードされます `patroni_conf_settings` オプション。機能フラグによって制御されない構成を変更または追加する場合、これはそれに最適な場所です。

構成は、IP アドレス、ポート番号、クラスターロールなどのクラスター情報から TPA によって生成された構成に_merged_されることに注意してください。これは、クラスターの安定したオペレーションに影響を与える可能性があるため、オーバーライドするものに注意してください。

Patroni はすべての postgres 構成を DCS に保存し、これが postgres に配布される方法と時期を制御するため、TPA の一部の機能は patroni と互換性がありません。

- postgres_conf_template 設定で postgresql.conf の生成に使用されたテンプレートを変更することはできません。
- 設定 postgres_conf_dir では、Postgres 構成ファイルの場所を変更することはできません。

TPA の Patroni 構成 config.yml

単一の値をオーバーライドできます。

```
cluster_vars:
  patroni_conf_settings:
    bootstrap:
      dcs:
        ttl: 120
```

またはフルブロック Patroni ドキュメントからの例を使用して

```
cluster_vars:
  patroni_conf_settings:
    restapi:
      http_extra_headers:
        "X-Frame-Options": "SAMEORIGIN"
        "X-XSS-Protection": "1; mode=block"
        "X-Content-Type-Options": "nosniff"
      https_extra_headers:
        "Strict-Transport-Security": "max-age=31536000; includeSubDomains"
```

デフォルトの TPA 構成変数に存在する値またはセクションを無効にする場合は、値を null に設定できます。これにより、パトロニは、構成ファイルをロードするときにこのセクションを無視します。

たとえば、log のデフォルトの TPA 構成は次のとおりです。

```
log:
  dir: /var/log/patroni
```

ロギングをオフにするには、これを config.yml に追加します。

```
cluster_vars:
  patroni_conf_settings:
    log: null
```

Eetcd 統合

etcd が DCS として使用される場合、TPA は etcd セキュリティ構成に基づいて、それに接続するように Patroni を自動的に構成します。

- **プロトコル** etcd_ssl_enabled に基づいて http または https を使用します。
- **CA 証明書** TLS が有効になっている場合に、クラスター CA 証明書パスを提供します。

- 認証
- etcd_authentication_mode が basic の場合、Patroni はユーザー名 patroni_etcd_user とその生成されたパスワードで構成されます。
- etcd_authentication_mode が mtls の場合、Patroni はクライアント証明書とキーへのパスで構成されます。

TPA は、Patroni クラスターを管理するためのこれらの最小限のツールセットを提供します。

ステータス

Patroni 実行に従って TPA クラスターの現在のステータスを確認するには

```
tpaexec status cluster_name
```

スイッチオーバー

レプリカノードへのスイッチオーバーを実行するにはメンテナンスを実行するには、次のコマンドを実行します。

```
tpaexec switchover cluster_name new_primary
```

new_primary 引数は、現在正常なレプリカとして実行されている既存のクラスターノードの名前である必要があります。スイッチオーバーが実行される前に、これが true を確認するためにチェックが実行されます。

スイッチオーバーが実行されたら、 deploy および test を実行して、正常なクラスターを確認することをお勧めします。

```
tpaexec deploy cluster_name
tpaexec test cluster_name
```

TPA は、config.yml の内容に関係なく、展開中にノードの現在のロールを検出しますたとえば、別のノードがリーダーである場合。

tpaexec upgrade を使用したパトローニのマイナー更新

特定のパッケージバージョンにアップグレードしようとする場合は、config.yml の patroni_package_version が更新されて、目的のバージョンを反映していることを確認します。

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade <cluster_dir> -e patroni_package_version="<desired version>" --
↪components=patroni
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードのために Patroni を選択するには、 tpaexec upgrade コマンドに渡される--components フラグに patroni または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.49 pg_hba.conf

Postgres ドキュメントでは、pg_hba.conf で使用可能なさまざまなオプションについて説明しています。

デフォルトでは、TPA はクラスター用に適切な pg_hba.conf を生成し、インスタンス間のレプリケーション、および認証されたクライアントからの接続を許可します。

postgres_hba_settings のリストを提供することにより、デフォルト構成にエントリを追加できます。

```
cluster_vars:
  postgres_hba_settings:
  - "# let authenticated users connect from anywhere"
  - hostssl all all 0.0.0.0/0 scram-sha-256
```

postgres_hba_local_auth_method: md5 を設定することにより、pg_hba.conf のデフォルトの local all all peer 行をオーバーライドできます。

デフォルトのエントリが不要な場合は、postgres_hba_template を変更できます。

```
cluster_vars:
  postgres_hba_template: pg_hba.lines.j2
  postgres_hba_settings:
  - "# my lines of text"
  - "# and nothing but my lines"
  - "# …not even any clients!"
  - hostssl all all 0.0.0.0/0 reject
```

クラスターディレクトリに templates/my_hba.conf.j2 を作成し、次のように設定することもできます。

```
cluster_vars:
  postgres_hba_template: my_hba.conf.j2
```

クラスターディレクトリの templates サブディレクトリの外部にテンプレートファイルを配置する場合、ファイルへの絶対パスを指定してください。

```
#  in the root of the cluster directory

cluster_vars:
  postgres_hba_template: "{{ cluster_dir }}/my_hba.conf.j2"
```

```
#  in a subdirectory of the cluster directory that is NOT templates

cluster_vars:
  postgres_hba_template: "{{ cluster_dir }}/subdirectory/my_hba.conf.j2"
```

```
#  in a directory outside of the cluster directory

cluster_vars:
  postgres_hba_template: /path/to/file/outside/cluster_dir/my_hba.conf.j2
```

既存の pg_hba.conf だけを残したい場合は、それもできます。

```
cluster_vars:
  postgres_hba_template:
```

さまざまなインスタンスで異なるように pg_hba.conf を構成することは可能ですが、スイッチオーバーやフェイルオーバーなどのトポロジを変更するイベント後のアクセスとレプリケーションの問題を回避するために、一般的に均一な構成を推奨します。

12.6.50 pg_ident.conf

通常は pg_ident.conf を変更する必要はありません。デフォルトでは、TPA はそれを変更しません。

pg_ident.conf を好きなコンテンツに置き換えるように postgres_ident_template を設定できます。

```
cluster_vars:
    pg_ident_template: ident.j2
```

クラスターディレクトリに templates/ident.j2 も作成する必要があります。

```
{% for u in [unixuser1, unixuser2] %}
mymap {{ u }} dbusername
{% endfor %}
```

12.6.51 pgd-cli

pgd-cli パッケージバージョン

デフォルトでは、TPA は pgd-cli の利用可能な最新バージョンをインストールします。

```
cluster_vars:
  pgdcli_package_version: "5.0.0-1"
```

apt または yum が受け入れるバージョン指定子を使用できます。

バージョンが一致しない場合は、* ワイルドカードを先頭に追加してみてください。これは、パッケージバージョンに 2:... のようなエポック修飾子がある場合に必要になることがよくあります。

tpaexec upgrade を使用した pgd-cli の更新

> **注釈**
>
> ドキュメントではこのコンポーネントを `pgd-cli` と参照していますが、 *--components* リストに渡す名前は `pgdcli` です。

目的のバージョンを追加の引数として tpaexec upgrade コマンドに渡すこともできます。

```
tpaexec upgrade <cluster_dir> -e pgdcli_package_version="<desired version>" \
    --components=pgdcli
```

詳細については、 *package version selection and upgrade* のセクションを参照してください。

アップグレードのために pgd-cli を選択するには、 tpaexec upgrade コマンドに渡される--components フラグに pgdcli または all が含まれていることを確認します

component selection for upgrade のセクションを参照してください

詳細については、

12.6.52 Adding Postgres extensions

デフォルトの Postgres 拡張機能

デフォルトでは、TPA は次の拡張機能をすべての Postgres データベースに追加しますおよび必要に応じて、対応するエントリを共有プリロードライブラリに自動的に追加します

- pg_stat_statements
- pg_freespacemap
- pg_visibility
- pageinspect
- pgstatus

ユーザー定義の拡張機能

拡張機能名、必要な共有プリロードエントリ、および拡張機能を含むパッケージを指定することにより、config.yml 内で追加の拡張機能を構成できます。

拡張機能を追加するときは、パッケージ名を extra_postgres_packages に含め、拡張機能名の両方を extra_postgres_extensions または postgres_databases の下で定義されたデータベースの extensions リストに含めてください。

拡張機能が必要な場合は、拡張機能の共有プリロードエントリ名を preload_extensions リストに追加します。この名前は拡張機能名自分自身とは異なる場合があることに注意してください、拡張機能自分自身のドキュメントを必ず確認してください。

次に、拡張機能とライブラリ名の別のエントリを使用して共有プリロード拡張機能リストに追加する必要がある拡張機能の簡単な例を示します。

```
cluster_vars:
  [...]
  extra_postgres_packages:
    common:
      - postgresql-17-my-extension
  extra_postgres_extensions:
    - my-extension
  preload_extensions:
    -  my_extension
```

以下のセクションでは、詳細な情報を提供します。

- *Reconciling changes made outside of TPA*
- *Creating Postgres databases*
- *shared_preload_libraries*
- *Installing Postgres-related packages*

TPA 認識される拡張機能

次の拡張機能のリストでは、 `config.yml` `extra_postgres_extensions` または `postgres_databases` で指定されたデータベースの `extensions` リストのいずれかに拡張機能名を追加するだけでよく、TPA は自動的に正しいパッケージと必要なエントリを shared_preload_libraries に含めます。

- edb_pg_tuner
- query_advisor
- edb_wait_states
- sql_profiler
- pg_failover_slots
- sql_protect
- edb_stat_monitor
- オートクラスター
- refdata
- ブルーフィン
- postgis
- pgaudit
- パスワードチェック

12.6.53 postgresql.conf

TPA は、その下にさまざまな.conf ファイルを含む conf.d ディレクトリを作成し、メイン postgresql.conf の include_dir を使用してこれらの追加構成ファイルを使用します。

Postgres 構成ファイル postgresql.conf、pg_ident.conf、および pg_hba.conf および conf.d の下のインクルードファイルは、常に postgres_conf_dir に保存されます。これはデフォルトで postgres_data_dir と同じですが、構成をデータディレクトリとは別にしておきたい場合は、別の場所に設定できます。

主な構成メカニズムは、変数を直接設定することです。

```
cluster_vars:
  temp_buffers: 16MB
  log_connections: on
  autovacuum_vacuum_cost_limit: -1
  effective_cache_size: 4GB
  max_connections: 300
  max_wal_senders: 32
```

TPA は、構成を複数のファイルに分割します。2 つのメインファイルは、0000-tpa.conf および 0001-tpa_restart.conf です。これらには、それぞれサーバーのリロードまたは再起動を必要とする設定が含まれています。展開中に、TPA は正しいファイルに変更を書き込み、必要に応じて Postgres をリロードまたは再起動します。

TPA は、特定の状況オプションの拡張機能を構成する場合などに他のファイルを使用する場合がありますが、通常、特定のパラメーターが正確にどこに設定されているかを気にする必要はありません。

次回 tpaexec deploy を実行したときに変更が上書きされる可能性があるため、 conf.d の下のファイルは決して編集しないでください。

postgres_port

Postgres ポートは、デフォルト値から別の有効なポート番号に上書きできます。

```
cluster_vars:
  postgres_port: 5433
```

postgres_port は、値が正しく共有され、この値に依存するすべてのコンポーネントで使用されることを保証ため、postgres_conf_settings を介してポートを設定するよりも優先されます。

postgres_conf_settings

TPA は、多くの使用可能な postgresql.conf 設定すべてではありませんが、直接設定できる temp_buffers や maintenance_work_mem のような変数を提供します。

postgres_conf_settings を使用して、TPA で認識されるかどうかにかかわらず、パラメーターを設定できます。postgresql.conf に表示されるのとまったく同じように値を引用符で囲む必要があります。

```
cluster_vars:
  effective_cache_size: 2GB
  postgres_conf_settings:
    effective_cache_size: 4GB
    authentication_timeout: 1min
    synchronous_standby_names: >-
      any 2 ("first", "second", "third")
    bdr.global_lock_statement_timeout: 60s
```

これは、TPA がネイティブに認識しない設定で最も役立ちますが、任意のパラメーターに使用できますたとえば、effective_cache_size は変数として設定できますが、authentication_timeout はできません。

これらの設定は conf.d/9900-role-settings.conf に書き込まれるため、他の方法で設定された変数より優先されます。

postgres_conf_settings の下の値を変更した場合、TPA は、リロードが変更を有効にするのに十分であるかどうか、または再起動が必要かどうかを知る方法がありません。したがって、常にサーバーを再起動して変更をアクティブにします。これが、可能な場合は常に変数を直接使用することをお勧めする理由です。

shared_buffers

デフォルトでは、TPA は shared_buffers を使用可能なメモリの 25%に設定しますこれは単なる経験則であり、推奨ではありません。別の比率を使用するように shared_buffers_ratio: 0.35 を設定するか、shared_buffers_mb: 796 を特定の MB 数に設定するか、shared_buffers: "2GB" などの正確な値を直接指定することにより、このデフォルトをオーバーライドできます。

huge_page_size

TPA は、Postgres が使用するアーキテクチャのデフォルトサイズの huge ページを予約します。別のサイズを使用するには、*Hugepages* を参照してください。 huge_page_size 設定は、カーネルコマンドラインでそのサイズのページを予約し、それに応じて postgresql.conf に huge_page_size を設定します。

effect_cache_size

デフォルトでは、TPA は effective_cache_size を使用可能なメモリの 50%に設定します。別の比率を使用するように effective_cache_size_ratio: 0.35 を設定するか、effective_cache_size_mb: 796 を特定の MB 数に設定するか、effective_cache_size: "8GB" などの正確な値を直接指定することにより、このデフォルトをオーバーライドできます。

max_active_replication_origins

PostgreSQL 18 では、同時にアクティブなレプリケーションオリジンの数を制御するために max_active_replication_origins GUC が導入されました。 PostgreSQL 18 より前では、これは max_replication_slots によって暗黙的に制限されていました。

PostgreSQL 18 以降の PGD クラスターの場合、TPA は max_active_replication_origins を 3 * number_of_nodes + 3 に設定します。これは、「ピアノードごとに 3 つ」の最小 recommended by the EDB

PGD documentation を余裕で超えて、ノード参加または同期操作中に作成される一時的なオリジンの安全マージンを残します。

非 PGD クラスターの場合、TPA は PostgreSQL のデフォルト 10 をそのままにします。

変数を直接設定することにより、これをオーバーライドできます。

```
cluster_vars:
  max_active_replication_origins: 50
```

この設定は、18 より前の PostgreSQL バージョンでは無視されます。

shared_preload_libraries

TPA は、動作するために shared_preload_libraries のエントリを必要とする拡張機能の内部リストを維持しており、このような拡張機能を postgres_extensions に含めると、shared_preload_libraries が自動的に更新されます。

プリロードが必要な認識されない拡張機能を使用している場合は、preload_extensions に追加できます。

```
cluster_vars:
  preload_extensions:
    - myext
    - otherext
```

myext を postgres_extensions に追加すると、shared_preload_libraries には myext が含まれます。

デフォルトでは、conf.d/8888-shared_preload_libraries.conf に shared_preload_libraries が設定されます。

shared_preload_libraries を変数として直接設定することはサポートされていません。通常は設定する必要はありませんが、やむを得ない場合は、*postgres_conf_settings* の下に完全に引用符で囲まれた値を設定できます。この場合、値は conf.d/9900-tpa_postgres_conf_settings.conf に設定されます。

Postgres ロギング

デフォルトでは、TPA は Postgres log_destination GUC を syslog として構成し、Postgres ログを/var/log/postgres/postgres.log に書き込むように rsyslog を構成します。

次のクラスター変数を使用して、これらのデフォルトを変更できます。

- postgres_log_file は、ログファイルへのパスです。デフォルト/var/log/postgres/postgres.log
- postgres_log_file_mode は、ログファイルのモードです。デフォルト 0640
- postgres_log_directory_mode は、ログディレクトリのモードです。デフォルト 0700
- log_destination は、同じ名前の Postgres GUC を設定します。デフォルト syslog
- logging_collector は、同じ名前の Postgres GUC を設定します。log_destination が syslog の場合はデフォルト off 、それ以外の場合は on 。

syslog 以外の log_destination を選択すると、TPA はログを書き込むように Postgres ロギングコレクターを設定します。すべての場合、TPA はディレクトリの作成とログローテーションの構成を行います。

次の例は、 rsyslog を使用して選択した場所にログを記録するように TPA に指示します。

```
cluster_vars:
  [...]
  postgres_log_file: /srv/fantastic_logs/pg_server.log
```

この例では、ログは同じ場所にありますが、Postgres ロギングコレクターを使用して JSON ログを書き込み、グループメンバーのログディレクトリでの読み取りと実行を許可しています。

```
cluster_vars:
  [...]
  postgres_log_file: /srv/fantastic_logs/pg_server.log
  postgres_log_directory_mode: 0750
  log_destination: jsonlog
```

TPA からではなく。

最終的な拡張子を含むログファイルの正確なパスにアクセスする必要がある場合たとえばフックの一部として、これは Ansible ファクト postgres_log_file_with_extension に保存されます。直接設定することはできません。

デフォルトでは、TPA はプライベートキーと自己署名 TLS 証明書を生成します。これらは Postgres がそれぞれ ssl_key_file および ssl_cert_file として使用します。ファイルは TPA クラスター名 cluster_name.key および cluster_name.crt を使用して名前付けられ、/etc/tpa にあります。その結果、0001-tpa_restart.conf では次のデフォルト構成になります。

```
ssl_key_file=/etc/tpa/cluster_name.key
ssl_cert_file=/etc/tpa/cluster_name.crt
```

これは、クライアントとサーバー間のトラフィックが転送中に暗号化されることを保証には十分です。

独自の証明書を提供するには、証明書をターゲットノードに *Uploading artifacts* としてアップロードし、次のクラスター変数を指定してパスを設定します。

```
cluster_vars:
  ...
  artifacts:
  - type: file
    dest: /path/to/your_key.key
    src: /local/path/to/your_key.key
    owner: root
    group: root
    mode: "0644"
  - type: file
    dest: /path/to/your_cert.crt
    src: /local/path/to/your_cert.crt
```

(次のページに続く)

(前のページからの続き)

```
    owner: root
    group: root
    mode: "0600"
  ssl_key_file: /path/to/your_key.key
  ssl_cert_file: /path/to/your_cert.crt
```

または、キーと証明書をデフォルトの場所にアップロードすると、TPA は独自の生成の代わりにそれらを使用し、ssl_key_file または ssl_cert_file を指定する必要はありません。ただし、 /etc/tpa は、アーティファクトのアップロード時に存在しないため、明示的に作成する必要があることに注意してください。これらのファイルの権限と所有権は、展開中に postgres ユーザーが作成されるときに TPA によって調整されます。

```
cluster_vars:
  ...
  artifacts:
  - type: path
    path: /etc/tpa
    state: directory
    owner: root
    group: root
    mode: "0755"
  - type: file
    dest: /etc/tpa/cluster_name.key
    src: /local/path/to/your_key.key
    owner: root
    group: root
    mode: "0644"
  - type: file
    dest: /etc/tpa/cluster_name.crt
    src: /local/path/to/your_cert.crt
    owner: root
    group: root
    mode: "0600"
```

TPA で生成された構成をオーバーライドするには 2 つの方法があります。

最初のおよび推奨されるオプションは、ALTER SYSTEM を使用することです。これは、常に構成ファイルのすべてより優先されます。

```
ALTER SYSTEM SET bdr.global_lock_statement_timeout TO 60s;
```

conf.d/9999-override.conf を編集することもできます。

```
echo "bdr.global_lock_statement_timeout=60s" >> conf.d/9999-override.conf
```

conf.d の下の他のすべてのファイルは、展開中に構成が変更された場合に上書きされる可能性がありますが、TPA は、最初に空のファイルを作成した後、9999-override.conf を変更することはありません。

変更する設定によっては、SELECT pg_reload_conf() を実行するか、サーバーを再起動して変更を有効にする必要がある場合があります。

postgresql.conf をスクラッチから生成する

デフォルトでは、TPA は include_dir を追加する以外に、デフォルトの initdb 生成された postgresql.conf ファイルをそのまま残します。通常、この動作をオーバーライドする必要はありませんが、そうするように postgres_conf_template を設定できます。

```
cluster_vars:
  postgres_conf_template: "pgconf.j2"
```

これで、クラスターディレクトリの templates/pgconf.j2 を使用して postgresql.conf が生成されます。

12.6.54 tpaexec deprovision

プロビジョニングを解除すると、クラスターと関連するリソースが破壊されます。

aws プラットフォームを使用するクラスターの場合、クラスターに設定されたインスタンスとすべてのキーペア、ポリシー、ボリューム、セキュリティグループ、ルートテーブル、VPC サブネット、インターネットゲートウェイ、および VPC が削除されます。

docker プラットフォームを使用するクラスターの場合、コンテナ、コンテナでソースビルド用に設定された ccache ディレクトリ、およびクラスター用に設定された Docker ネットワークを削除します。

すべてのプラットフォームで、ssh キー、保存されたパスワード、Ansible インベントリ、ログを含む、tpaexec provision によってローカルに作成されたすべてのファイルが削除されます。

12.6.55 tpaexec info

info コマンドを使用して、TPA インストールに関する情報を出力できます。この情報の提供は、トラブルシューティングに役立ちます。

使用法

- tpaexec info を実行します

サブコマンド

- tpaexec info version

現在の TPA バージョンを表示します

- tpaexec info platforms

使用可能な展開プラットフォームを表示します

- tpaexec info architectures

使用可能な展開アーキテクチャを表示します

- tpaexec info platforms/<name>

特定のプラットフォームに関する情報を表示します

- tpaexec info architectures/<name>

特定のアーキテクチャに関する情報を表示します

出力例

tpaexec info コマンドは次を出力します。

```
#  TPAexec 23.29

tpaexec=/opt/EDB/TPA/bin/tpaexec
TPA_DIR=/opt/EDB/TPA
PYTHON=/opt/EDB/TPA/tpa-venv/bin/python3 (v3.12.18, venv)
TPA_VENV=/opt/EDB/TPA/tpa-venv
ANSIBLE=/opt/EDB/TPA/tpa-venv/bin/ansible (v2.16.3)
Validated: ea844d1b90295597d080bbf824dbbc6954886cb54ffdb265c7c71b99bedee67b [OK]
```

12.6.56 tpaexec reconfigure

tpaexec reconfigure コマンドは、config.yml を読み取り、引数に応じてさまざまな方法でクラスターを変更する変更バージョンを生成します。

引数

他の tpaexec コマンドと同様に、クラスターディレクトリは常に指定する必要があります。

クラスターのアーキテクチャの変更

次の引数により、クラスターのアーキテクチャを変更できます。

- --architecture <architecture> (必須) クラスターの新しいアーキテクチャ。 PGD-Always-ON および PGD-X を有効な引数として受け入れます
- --pgd-proxy-routing <global|local> PGD-Always-ON に必要 PGD-Proxy が接続をルーティングする方法。この引数の意味の詳細については、*PGD-Always-ON* を参照してください。
- --edb-repositories <repositories> オプショナル EDB パッケージリポジトリのスペース区切りリスト。通常、これを指定する必要はありません。tpaexec configure は、クラスターで使用されている postgres フレーバーに基づいて、適切なリポジトリを選択します。

アーキテクチャを変更した後、*Upgrading your cluster* を実行して、クラスターに必要な変更を加えます。

クラスターを PGD-プロキシから PGD-Always-ON の接続マネージャーに変更する

- --enable-connection-manager(required) PGD 5.9+クラスター上の PGD-プロキシから新しいビルトイン接続マネージャーへの移行を許可するオプション。

クラスターを 2q から EDB リポジトリへの変更

`--replace-2q-repositories` 引数は、クラスターが使用する 2ndQuadrant リポジトリを削除し、必要に応じて EDB リポジトリを追加して、それらを置き換えます。

この引数で再構成した後、 [tpaexec promote)(tpaexec-deploy.md) を実行して、クラスターに必要な変更を加えます。

出力フォーマット

次のオプションは、出力の形式を制御します。

- `--describe` 何も変更せずに、変更される内容の説明を表示します。
- `--check` 加えられる変更を検証し、エラーがあれば何も変更せずに、検証の結果発生したエラーまたは警告を表示します。
- `--output <filename>` config.yml 以外のファイルに出力を書き込みます。

サンプル呼び出し

```
tpaexec reconfigure ~/clusters/speedy\
  --architecture PGD-Always-ON\
  --pgd-proxy-routing local
```

12.7 tpaexec eval

eval コマンドを使用して、クラスターのインベントリのコンテキストで任意の Jinja/Ansible 式を評価できます。情報は、変数、インベントリ、収集された OS ファクト、計算されたパス、パッケージ名などに関連する場合があります。

12.7.1 使用法

- `tpaexec eval ~/clusters/speedy '<expression>'` を実行します

オプション

- `tpaexec eval ~/clusters/speedy '<expression>' --hosts <pattern>`

評価をホストパターンに制限します。パターンは、単一のインスタンス名 `unreal` 、コロン区切りリスト `unreal:ultra` 、`localhost` 、またはグループ名 `role_primary` 、`role_etcd` です。

- `tpaexec eval ~/clusters/speedy '<expression>' --no-init`

クラスター初期化プレイブックをスキップします。 `config.yml` および生成されたインベントリから派生した変数のみが使用できます。`ansible_*` ホストファクトは未定義になります。スピーディーな在庫確認におすすめです。

12.7.2 出力例

単一ホストの config.yml から値を検索します。

```
tpaexec eval ~/clusters/speedy {{ postgres_version }} --no-init --hosts unreal
```

```
  unreal => {
    "msg": "17"
}
```

クラスター内のすべてのホストの値を出力します。

```
tpaexec eval ~/clusters/speedy {{ ip_address }} --no-init
```

```
  unreal => {
    "msg": "10.33.155.2"
}
  ultra => {
    "msg": "10.33.155.3"
}
  quondam => {
    "msg": "10.33.155.4"
}
```

localhost をターゲティングしてインベントリ全体のデータを検査します。

```
tpaexec eval ~/clusters/speedy {{ groups.keys() | list }} --no-init --hosts localhost
```

```
  localhost => {
    "msg": [
        "all",
        "ungrouped",
        "tag_Cluster_speedy"
    ]
}
```

ホスト用に定義されたすべての変数をリストします必要な変数の名前がまだわからない場合に役立ちます。

```
tpaexec eval ~/clusters/speedy {{ hostvars[inventory_hostname].keys() | list | sort }}
↪ --no-init --hosts unreal
```

```
  unreal => {
    "msg": [
        "ansible_user",
        "backup",
        "cluster_name",
```

(次のページに続く)

(前のページからの続き)

```
        "failover_manager",
        "ip_address",
        "postgres_version",
        "role",
        "..."
    ]
}
```

ホスト自分自身から収集されたファクトを読み取ります --no-init を省略すると、初期化プレイブックが実行され、ansible_* ファクトが設定されます。

```
tpaexec eval ~/clusters/speedy {{ ansible_distribution }} --hosts unreal
```

```
  unreal => {
    "msg": "Debian"
}
```